EW Library
4.20100927

Contents

1 Main Page

The EW library contains C++ classes that implement various visualization and analysis facilities for geometric morphometrics. It replaces the **edgewarp** program, written in C and Tcl/Tk.

It was written by Bill Green for the EVAN Geometric Morphometric Toolkit Consortium. Copyright and license information are in the page Copyright . The changelog is in the page Changelog .

All functions and variables documented here are defined in `libew.a`. The following libraries are dependencies of modules of `libew.a`:

- the standard math library
- lapack
- libxml2
- opengl

Various example files are in the page Examples.

2 Examples

Example surface `file:`

- tetrahedron.obj

An example ew::Form3 file, a simple unstructured set of unnamed landmarks:

- simple_landmarks.form

An example ew::Form3 file, a structured landmark template configuration form with surfaces, curves, named landmarks and named semi-landmark sets.

- mandible_landmarks.form

An example ew::Form3 file, a structured landmark specimen configuration form with surfaces, curves, named landmarks and named semi-landmark sets:

- mandible_case.form

An example ew::Dig3Tableau file.

- example.sav

3 tetrahedron.obj

```
v 0 0 0
v 0 0 0
v 0 0 0
v 0 0 1
```

```
v 0 0 1
v 0 0 1
v 0 1 1
v 0 1 1
v 0 1 1
v 1 1 1
v 1 1 1
v 1 1 1
f 7 4 10
f 1 5 8
f 9 11 2
f 6 3 12
```

4 simple_landmarks.form

```
<ew_form3 version="1.0">

<pointset n="5">
<locations>
0.1 0.1 0.2
0.2 1.3 0.3
1.3 0.2 0.4
1.4 1.3 0.4
0.5 0.5 0.4
</locations>
</pointset>

</ew_form3>
```

5 mandible_landmarks.form

```
<ew_form3 version="1.0">

<surface id="Mand" file="template.sur"/>

<curve id="Bord" file="template_lm.cur"/>
<curve id="RamL" file="template_lar.cur"/>
<curve id="RamR" file="template_rar.cur"/>
<curve id="AlvB" file="template_lia.cur"/>
<curve id="AlvL" file="template_loa.cur"/>
<curve id="Sym" file="template_sym.cur"/>

<pointset id="MSpl" type="plane">
<locations>1.2 3.4 5.6</locations>
<orientations>1 0 0 0 1 0 0 0 1</orientations>
<sizes>86.3</sizes>
</pointset>

<pointset id="CCpl" type="plane">
<locations>1.2 3.4 5.6</locations>
<orientations>1 0 0 0 0 1 0 -1 0</orientations>
<sizes>86.3</sizes>
```

```
</pointset>

<pointset id="ConRT" type="landmark">
<locations>9.8 7.6 5.4</locations>
</pointset>

<pointset id="ConLT" type="landmark">
<locations>9.8 7.6 5.4</locations>
</pointset>

<pointset id="ConRM" type="landmark">
<locations>9.8 7.6 5.4</locations>
</pointset>

<pointset id="ConLM" type="landmark">
<locations>9.8 7.6 5.4</locations>
</pointset>

<pointset id="ConRL" type="landmark">
<locations>9.8 7.6 5.4</locations>
</pointset>

<pointset id="ConLL" type="landmark">
<locations>9.8 7.6 5.4</locations>
</pointset>

<pointset id="CorR" type="landmark">
<locations>9.8 7.6 5.4</locations>
</pointset>

<pointset id="CorL" type="landmark">
<locations>9.8 7.6 5.4</locations>
</pointset>

<pointset id="MenR" type="landmark">
<locations>9.8 7.6 5.4</locations>
<orientations>1 0 0 0 0 1 0 -1 0</orientations>
</pointset>

<pointset id="MenL" type="landmark">
<locations>9.8 7.6 5.4</locations>
<orientations>1 0 0 0 0 1 0 -1 0</orientations>
</pointset>

<pointset id="MandR" type="landmark">
<locations>9.8 7.6 5.4</locations>
<orientations>1 0 0 0 0 1 0 -1 0</orientations>
</pointset>

<pointset id="MandL" type="landmark">
<locations>9.8 7.6 5.4</locations>
<orientations>1 0 0 0 0 1 0 -1 0</orientations>
</pointset>

<pointset id="Inf1" type="landmark">
<locations>9.8 7.6 5.4</locations>
</pointset>
```

```
<pointset id="Inf2" type="landmark">
<locations>9.8 7.6 5.4</locations>
</pointset>

<pointset id="Symph" type="landmark">
<locations>9.8 7.6 5.4</locations>
</pointset>

<pointset id="slMandL" type="semi-landmark" n="2">
<locations>
1.23 4.56 7.89
2.13 5.46 8.79
</locations>
</pointset>

<pointset id="slMandR" type="semi-landmark" n="2">
<locations>
1.23 4.56 7.89
2.13 5.46 8.79
</locations>
</pointset>

<pointset id="slBordL" type="semi-landmark" n="2">
<locations>
1.23 4.56 7.89
2.13 5.46 8.79
</locations>
</pointset>

<pointset id="slBordR" type="semi-landmark" n="2">
<locations>
1.23 4.56 7.89
2.13 5.46 8.79
</locations>
</pointset>

<pointset id="slRamL" type="semi-landmark" n="2">
<locations>
1.23 4.56 7.89
2.13 5.46 8.79
</locations>
</pointset>

<pointset id="slRamR" type="semi-landmark" n="2">
<locations>
1.23 4.56 7.89
2.13 5.46 8.79
</locations>
</pointset>

<pointset id="slAlvBL" type="semi-landmark" n="2">
<locations>
1.23 4.56 7.89
2.13 5.46 8.79
</locations>
</pointset>
```

```
<pointset id="slAlvBR" type="semi-landmark" n="2">
<locations>
1.23 4.56 7.89
2.13 5.46 8.79
</locations>
</pointset>

<pointset id="slAlvLL" type="semi-landmark" n="2">
<locations>
1.23 4.56 7.89
2.13 5.46 8.79
</locations>
</pointset>

<pointset id="slAlvLR" type="semi-landmark" n="2">
<locations>
1.23 4.56 7.89
2.13 5.46 8.79
</locations>
</pointset>

<pointset id="slSym" type="semi-landmark" n="2">
<locations>
1.23 4.56 7.89
2.13 5.46 8.79
</locations>
</pointset>

<embedding subset="Bord" superset="Mand"/>
<embedding subset="RamL" superset="Mand"/>
<embedding subset="RamR" superset="Mand"/>
<embedding subset="AlvBü" superset="Mand"/>
<embedding subset="AlvLi" superset="Mand"/>
<embedding subset="Sym" superset="Mand"/>
<embedding subset="Sym" superset="MSpl"/>
<embedding subset="ConLT" superset="Mand"/>
<embedding subset="ConRT" superset="Mand"/>
<embedding subset="ConLM" superset="Mand"/>
<embedding subset="ConRM" superset="Mand"/>
<embedding subset="ConLL" superset="Mand"/>
<embedding subset="ConRL" superset="Mand"/>
<embedding subset="CorL" superset="Mand"/>
<embedding subset="CorR" superset="Mand"/>
<embedding subset="Inf1" superset="Bord"/>
<embedding subset="Inf1" superset="MSpl"/>
<embedding subset="Inf2" superset="Borf"/>
<embedding subset="Inf2" superset="MSpl"/>
<embedding subset="Symph" superset="Bord"/>
<embedding subset="Symph" superset="Sym"/>
<embedding subset="slMandL" superset="Mand"/>
<embedding subset="slMandR" superset="Mand"/>
<embedding subset="slBordL" superset="Bord"/>
<embedding subset="slBordR" superset="Bord"/>
<embedding subset="slRamL" superset="RamL"/>
<embedding subset="slRamR" superset="RamR"/>
<embedding subset="slAlvBL" superset="ALvBü"/>
```

```
<embedding subset="slAlvBR" superset="ALvBü"/>
<embedding subset="slAlvLL" superset="AlvLi"/>
<embedding subset="slAlvLR" superset="AlvLi"/>
<embedding subset="slSym" superset="Sym"/>

<reflection left="ConLT" right="ConRT"/>
<reflection left="ConLM" right="ConRM"/>
<reflection left="ConLL" right="ConRL"/>
<reflection left="CorL" right="CorR"/>
<reflection left="MandL" right="MandR"/>
<reflection left="MenL" right="MenR"/>
<reflection left="slMandL" right="slMandR"/>
<reflection left="slRamL" right="slRamR"/>
<reflection left="slAlvBL" right="slAlvBR"/>
<reflection left="slAlvLL" right="slAlvLR"/>
<reflection left="Inf1" right="Inf1"/>
<reflection left="Inf2" right="Inf2"/>
<reflection left="Symph" right="Symph"/>
<reflection left="slSym" right="slSym"/>

</ew_form3>
```

6 mandible_case.form

```
<ew_form3 version="1.0">

<surface id="Mand" file="case07.sur"/>

<curve id="Bord"/>
<curve id="RamL"/>
<curve id="RamR" state="unset"/>
<curve id="AlvBü" file="case07_lia.cur" state="provisional"/>
<curve id="AlvLi" file="case07_loa.cur" state="warped"/>
<curve id="Sym" file="case07_sym.cur"/>

<pointset id="MSpl" type="plane" state="provisional">
<locations>1.2 3.4 5.6</locations>
<orientations>1 0 0 0 1 0 0 0 1</orientations>
<sizes>86.3</sizes>
</pointset>

<pointset id="CCpl" type="plane">
<locations>0 0 0</locations>
</pointset>

<pointset id="ConRT">
<locations>9.8 7.6 5.4</locations>
<relax_dims>3</relax_dims>
</pointset>

<pointset id="ConLT">
<locations>9.8 7.6 5.4</locations>
<relax_dims>2</relax_dims>
<relax_params>1.2 3.4 5.6</relax_params>
</pointset>
```

```
<pointset id="ConRM">
<locations>9.8 7.6 5.4</locations>
<relax_dims>1</relax_dims>
<relax_params>1.2 3.4 5.6</relax_params>
</pointset>

<pointset id="ConLM">
<locations>0 0 0</locations>
</pointset>

<pointset id="ConRL">
<locations>0 0 0</locations>
</pointset>

<pointset id="ConLL">
<locations>0 0 0</locations>
</pointset>

<pointset id="CorR">
<locations>0 0 0</locations>
</pointset>

<pointset id="CorL">
<locations>0 0 0</locations>
</pointset>

<pointset id="MenR">
<locations>9.8 7.6 5.4</locations>
<orientations>1 0 0 0 0 1 0 -1 0</orientations>
</pointset>

<pointset id="MenL">
<locations>9.8 7.6 5.4</locations>
<orientations>1 0 0 0 0 1 0 -1 0</orientations>
</pointset>

<pointset id="MandR">
<locations>0 0 0</locations>
</pointset>

<pointset id="MandL">
<locations>0 0 0</locations>
</pointset>

<pointset id="Inf1">
<locations>9.8 7.6 5.4</locations>
</pointset>

<pointset id="Inf2">
<locations>9.8 7.6 5.4</locations>
</pointset>

<pointset id="Symph">
<locations>9.8 7.6 5.4</locations>
</pointset>
```

```
<pointset id="slMandL" type="semi-landmark" state="projected" n="2">
<locations>
0 0 0
0 0 0
</locations>
</pointset>

<pointset id="slMandR" type="semi-landmark" state="optimized" n="2">
<locations>
0 0 0
0 0 0
</locations>
</pointset>

<pointset id="slBordL" type="semi-landmark" n="2">
<locations>
0 0 0
0 0 0
</locations>
</pointset>

<pointset id="slBordR" type="semi-landmark" n="2">
<locations>
0 0 0
0 0 0
</locations>
</pointset>

<pointset id="slRamL" type="semi-landmark" n="2">
<locations>
0 0 0
0 0 0
</locations>
</pointset>

<pointset id="slRamR" type="semi-landmark" n="2">
<locations>
0 0 0
0 0 0
</locations>
</pointset>

<pointset id="slAlvBL" type="semi-landmark" n="2">
<locations>
1.23 4.56 7.89
2.13 5.46 1.4
</locations>
<relax_dims>
0
3
</relax_dims>
</pointset>

<pointset id="slAlvBR" type="semi-landmark" n="2">
<locations>
1.23 4.56 7.89
2.13 5.46 8.79
```

```
</locations>
<relax_dims>
0
2
</relax_dims>
<relax_params>
0 0 0
2.13 5.46 8.79
</relax_params>
</pointset>

<pointset id="slAlvLL" type="semi-landmark" n="2">
<locations>
1.23 4.56 7.89
2.13 5.46 8.79
</locations>
</pointset>

<pointset id="slAlvLR" type="semi-landmark" n="2">
<locations>
1.23 4.56 7.89
2.13 5.46 8.79
</locations>
</pointset>

<pointset id="slSym" type="semi-landmark" n="2">
<locations>
1.23 4.56 7.89
2.13 5.46 8.79
</locations>
</pointset>

<embedding subset="Bord" superset="Mand"/>
<embedding subset="RamL" superset="Mand"/>
<embedding subset="RamR" superset="Mand"/>
<embedding subset="AlvBü" superset="Mand"/>
<embedding subset="AlvLi" superset="Mand"/>
<embedding subset="Sym" superset="Mand"/>
<embedding subset="Sym" superset="MSpl"/>
<embedding subset="ConLT" superset="Mand"/>
<embedding subset="ConRT" superset="Mand"/>
<embedding subset="ConLM" superset="Mand"/>
<embedding subset="ConRM" superset="Mand"/>
<embedding subset="ConLL" superset="Mand"/>
<embedding subset="ConRL" superset="Mand"/>
<embedding subset="CorL" superset="Mand"/>
<embedding subset="CorR" superset="Mand"/>
<embedding subset="Inf1" superset="Bord"/>
<embedding subset="Inf1" superset="MSpl"/>
<embedding subset="Inf2" superset="Borf"/>
<embedding subset="Inf2" superset="MSpl"/>
<embedding subset="Symph" superset="Bord"/>
<embedding subset="Symph" superset="Sym"/>
<embedding subset="slMandL" superset="Mand"/>
<embedding subset="slMandR" superset="Mand"/>
<embedding subset="slBordL" superset="Bord"/>
<embedding subset="slBordR" superset="Bord"/>
```

```
<embedding subset="slRamL" superset="RamL"/>
<embedding subset="slRamR" superset="RamR"/>
<embedding subset="slAlvBL" superset="ALvBü"/>
<embedding subset="slAlvBR" superset="ALvBü"/>
<embedding subset="slAlvLL" superset="AlvLi"/>
<embedding subset="slAlvLR" superset="AlvLi"/>
<embedding subset="slSym" superset="Sym"/>

<reflection left="ConLT" right="ConRT"/>
<reflection left="ConLM" right="ConRM"/>
<reflection left="ConLL" right="ConRL"/>
<reflection left="CorL" right="CorR"/>
<reflection left="MandL" right="MandR"/>
<reflection left="MenL" right="MenR"/>
<reflection left="slMandL" right="slMandR"/>
<reflection left="slRamL" right="slRamR"/>
<reflection left="slAlvBL" right="slAlvBR"/>
<reflection left="slAlvLL" right="slAlvLR"/>
<reflection left="Inf1" right="Inf1"/>
<reflection left="Inf2" right="Inf2"/>
<reflection left="Symph" right="Symph"/>
<reflection left="slSym" right="slSym"/>

</ew_form3>
```

7 example.sav

```
<?xml version="1.0" encoding="UTF-8"?>
<ew_dig3>

<tableau>
<template_form>test_dig3_tem000.out</template_form>
<template_view>[0 0 1.5] [1 0 0 0 1 0 0 0 1] .1</template_view>
<template_slice>[0 0 1.5] [1 0 0 0 1 0 0 0 1] .1</template_slice>
<template_surface id="Mand"/>
<specimen_form>test_dig3_tar000.out</specimen_form>
<specimen_view>[0 0 1.5] [1 0 0 0 1 0 0 0 1] .1</specimen_view>
<specimen_slice>[0 0 1.5] [1 0 0 0 1 0 0 0 1] .1</specimen_slice>
<specimen_surface id="Mand"/>
<specimen_surface id="T1"/>
</tableau>

<tableau>
<template_form>test_dig3_tem001.out</template_form>
<template_view>[0 0 1.5] [1 0 0 0 1 0 0 0 1] .1</template_view>
<template_slice>[0 0 1.5] [1 0 0 0 1 0 0 0 1] .1</template_slice>
<template_surface id="Mand"/>
<specimen_form>test_dig3_tar001.out</specimen_form>
<specimen_view>[0 0 1.5] [1 0 0 0 1 0 0 0 1] .1</specimen_view>
<specimen_slice>[0 0 1.5] [1 0 0 0 1 0 0 0 1] .1</specimen_slice>
<specimen_surface id="Mand"/>
<specimen_surface id="T1"/>
</tableau>

<tableau>
```

```
<template_form>test_dig3_tem002.out</template_form>
<template_view>[-.7876785663729557 -.7859725280754518 1.0058874992698374] [.8364415926970159 -.15689
<template_slice>[.30521663602401067 .3155556729796402 1.4343177550126587] [.9786488421258913 -.02903
<template_surface id="Mand"/>
<specimen_form>test_dig3_tar002.out</specimen_form>
<specimen_view>[-.7041977804372336 -.9846155272715682 .8857978039502046] [.8601678909243633 -.199282
<specimen_slice>[.5742790163936046 .5909587618098809 1.253383960791795] [.9174371470618669 -.1083185
<specimen_surface id="Mand"/>
<specimen_surface id="T1"/>
</tableau>

<tableau>
<template_form>test_dig3_tem003.out</template_form>
<template_view>[-.7876785663729557 -.7859725280754518 1.0058874992698374] [.8364415926970159 -.15689
<template_slice>[.30521663602401067 .3155556729796402 1.4343177550126587] [.9786488421258913 -.02903
<specimen_form>test_dig3_tar003.out</specimen_form>
<specimen_view>[-.7041977804372336 -.9846155272715682 .8857978039502046] [.8601678909243633 -.199282
<specimen_slice>[.5742790163936046 .5909587618098809 1.253383960791795] [.9174371470618669 -.1083185
</tableau>

</ew_dig3>
```

8 Copyright

```
The EW library was written by Bill Green in 2008-2010.
It was written for the European Virtual Anthropology Network and for the
University of Vienna.
It is copyright 2008-2010 Bill Green and the University of Vienna.
It is licensed for distribution under the terms of the GNU Lesser General
Public License version 2, as published by the Free Software Foundation.
It is also licensed for distribution under the terms of the GNU General Public
License version 2, as published by the Free Software Foundation.

The EW library replaces much of the Edgewarp version 3 application,
written by Bill Green 1998-2007.
However, it is a completely new implementation.
Edgewarp version 3 is copyright 1998-2008 the University of Michigan and the
University of Washington.

The EW library file Gdtoa.cpp contains code adapted from the file gdtoa.tgz at
www.netlib.org, written by David. M. Gay.
This is the copyright notice and license from gdtoa.tgz:

    Copyright (C) 1997-2001 Lucent Technologies
    All Rights Reserved

    Permission to use, copy, modify, and distribute this software and
    its documentation for any purpose and without fee is hereby
    granted, provided that the above copyright notice appear in all
    copies and that both that the copyright notice and this
    permission notice and warranty disclaimer appear in supporting
    documentation, and that the name of Lucent or any of its entities
    not be used in advertising or publicity pertaining to
    distribution of the software without specific, written prior
    permission.
```

```
LUCENT DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS.
IN NO EVENT SHALL LUCENT OR ANY OF ITS ENTITIES BE LIABLE FOR ANY
SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER
IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION,
ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF
THIS SOFTWARE.
```

9 Changelog

```
2010-10-12  Bill Green  <wdkg@wdkg.org>

    * Renamed all classes, e.g:
        EW_Dataflow_Node -> ew::DataflowNode
    * Renamed all files, e.g:
        dataflow_node.h -> DataflowNode.h
    * Renamed
        ew::DataflowArray3Explicit -> ew::DataflowArray3E
        ew::DataflowCurve3Explicit -> ew::DataflowCurve3E
        ew::DataflowSurface3Explicit -> ew::DataflowSurface3E
        ew::Dig3TableauCurve -> ew::Dig3SetCurve
        ew::Dig3TableauSurface -> ew::Dig3SetSurface
    * Added
        ew::Curve3::write_file
        ew::Curve3::read_points
        ew::Dig3Space::project
        ew::Form3::search_superset
        ew::Form3::set_superset
        ew::DataflowForm3::set_superset
        ew::Dig3Space::set_superset


2010-07-04  Bill Green  <wdkg@wdkg.org>

    * Replaced EW_Dig3_Space::get_index() with constant EW_Dig3_Space::index.
    * Replaced EW_Form3_IOError, EW_Dig3_Tableau_IOError, EW_Curve3_IOError and
    EW_Surface3_IOError with EW_Error_IO.
    Added EW_Error_Runtime.
    * Added EW_Dig3::SPACE_TEMPLATE and EW_DIG3::SPACE_SPECIMEN.
    * EW_Dig3_View now inherits EW_View3_Widget instead of having to be
    associated with a separately created EW_View3_Widget.
    * Added EW_View3_Pick and EW_View3_Widget::pickv.
    EW_View3_Widget::pick is deprecated and will be removed later.
    * Added a member EW_View3_Item::type and an optional argument in
    the constructors of classes derived from EW_View3_Item to initialize it.
    Also added a corresponding enum EW_Dig3_View::item_t.
    * Removed EW_Dataflow_Curve3_Explicit::get_filename and
    EW_Dataflow_Surface3_Explicit::get_filename.
    * Replaced EW_Dataflow_Netork::cycle_t with unsigned long, which is
    what it was a typedef for.


2010-03-24  Bill Green  <wdkg@wdkg.org>
```

```
   * Renamed
       EW_Dig3::TableauFlags -> EW_Dig3::tableau_flags_t
       EW_Form3::StateType -> EW_Form3::state_t
       EW_Form3::PointType -> EW_Form3::point_t
       EW_View3_Landmarks::SymbolType -> EW_View3_Landmarks::symbol_t


2010-02-24  Bill Green  <wdkg@wdkg.org>

   * Added, to support curves:
       EW_Curve3
       EW_Curve3_IOError
       EW_Dataflow_Curve3
       EW_Dataflow_Curve3_Explicit
       EW_Dataflow_Network::cached_curve
       EW_View3_Curve
       EW_Dig3_Tableau_Curve
       EW_Dig3_Tableau_Space::curve_settings
       EW_Dig3_Space::get_curve_nodes
       EW_Dig3_Space::get_curve_of_pointset
       EW_Dig3_Space::set_form_curve
       EW_Dig3_Space::remove_form_curve
       EW_Dig3_View::get_curve_items
    Modified, to support curves:
       EW_Dig3
       EW_Dig3_Tableau
       EW_Dig3_Space
       EW_Dig3_View


2010-02-03  Bill Green  <wdkg@wdkg.org>

   * Debugged a subtle problem on Mac OS X: the bundled lapack routines
   occasionally crash if their array arguments are not aligned to 16 bytes.
   This can be ensured by new'ing them or by making them std::vector's.


2010-01-27  Bill Green  <wdkg@wdkg.org>

   * Added EW_Dig3_Space::get_surface_of_pointset.


2010-01-25  Bill Green  <wdkg@wdkg.org>

   * Added:
       EW_Dataflow_Spline3::get_nonsingular
       EW_Dataflow_Surface3::make_index
       EW_Dataflow_Surface3::index_is_made
   Changed and completed:
       EW_Dataflow_Surface3::project
   Renamed:
       EW_Dataflow_Surface3::optimized_lmk_images ->
        EW_Dataflow_Surface3::get_optimized_lmk_images


2010-01-20  Bill Green  <wdkg@wdkg.org>
```

```
    * Renamed
        EW_Dataflow_Surface3::get_triangles -> EW_Dataflow_Surface3::get_faces
        EW_Dataflow_Surface3::get_num_triangles ->
         EW_Dataflow_Surface3::get_num_faces


2010-01-19  Bill Green  <wdkg@wdkg.org>

    * Made EW_Dataflow_Network::surface_cached surfaces persist until the
    network is destroyed.

    * Support const EW_Dataflow_Nodes: added a lot of const's to method
    signatures.
    The nodes created by EW_Dig3 are now accessible as pointers to const, so
    they may only be changed using EW_Dig3_Space wrappers.
    This is to clarify what may be done to these nodes.
    Added:
        EW_Dig3_Space::set_form_pointset
        EW_Dig3_Space::set_form_pointset_location
        EW_Dig3_Space::set_form_pointset_relax
        EW_Dig3_Space::remove_form_pointset
        EW_Dig3_Space::set_form_embedding
        EW_Dig3_Space::remove_form_embedding
    and changed:
        EW_Dig3_Space::set_surface -> EW_Dig3_Space::set_form_surface
        EW_Dig3_Space::remove_surface -> EW_Dig3_Space::remove_form_surface
    These functions have the same signatures as the EW_Dataflow_Form3
    equivalents.
    These replace methods removed on 2010-01-07.

    * Changed EW_Dataflow_Form3::set_surface and EW_Dig3_Space::set_surface to
    use a EW_Form3_Surface pointer argument for consistency.

    * Added:
      EW_Dataflow_Form3::set_volume
      EW_Dataflow_Form3::remove_volume
      EW_Dataflow_Form3::set_curve
      EW_Dataflow_Form3::remove_curve
      EW_Dataflow_Form3::set_embedding
      EW_Dataflow_Form3::remove_embedding
      EW_Dataflow_Form3::set_reflection
      EW_Dataflow_Form3::remove_reflection


2010-01-14  Bill Green  <wdkg@wdkg.org>

    * Various bug fixes.

    * Made EW_View3_Item destructors protected and documented that items
    belong to their widget after creation.
    Added EW_View3_Item::destroy() to use instead of the destructor.

    * Replaced
        EW_Dataflow_Surface3_Explicit::cached ->
         EW_Dataflow_Network::surface_cached
```

```
2010-01-12  Bill Green  <wdkg@wdkg.org>

    * Renamed, for consistency:
        EW_Bbox3::set_empty -> EW_Bbox3::set_to_empty


2010-01-07  Bill Green  <wdkg@wdkg.org>

    * Added EW_Dig3_Space::get_form_node.
    Removed the following, which this makes redundent:
        EW_Dig3_Space::set_pointset
        EW_Dig3_Space::set_pointset_location
        EW_Dig3_Space::remove_pointset
        EW_Dig3_Space::set_embedding
        EW_Dig3_Space::remove_embedding
    Added EW_Dataflow_Form3::set_pointset_relax.
    Changed:
        EW_Dataflow_Spline3::warp_point -> EW_Dataflow_Spline3::warp_points
    Changed EW_Dataflow_Spline3::optimized_lmk_images to return a pointer to
    the array of coordinates of all optimized landmarks.


2010-01-05  Bill Green  <wdkg@wdkg.org>

    * All classes now have their own header and source files.
    Client code may need additional include statements.


2009-12-15  Bill Green  <wdkg@wdkg.org>

    * Improved playback interpolation algorithm.


2009-12-14  Bill Green  <wdkg@wdkg.org>

    * Changed signature of EW_Transform3::format and EW_Transform3::scan,
    to be less awkward.


2009-12-11  Bill Green  <wdkg@wdkg.org>

    * Started to add support for more landmark operations.
    These operations can already be performed using existing methods:
        create a EW_Form3_Pointset in the template or specimen
        delete a EW_Form3_Pointset from the template or specimen
        delete all EW_Form3_Pointset's in the template or in the specimen
        add a point to a EW_Form3_Pointset
        delete a point from a EW_Form3_Pointset
        select a point in the template or specimen
        set or change the following attributes of a EW_Form3_Pointset
          id
          type (currently landmark or semi-landmark)
          state
          n
        set or change the following attributes of a point in a EW_Form3_Pointset
          location
```

```
        relax_params
        relax_dims (including whether the point should be ignored)
      set all semi-landmarks to be ignored
      set a landmark or semi-landmark to be ignored
      set a landmark or semi-landmark to be not ignored
      copy a EW_Form3_Pointset from the template to the specimen or vice versa
      copy all EW_Form3_Pointset's from the template to the specimen or vice
       versa
      move a point in the specimen to the location it occupies in the
       template or vice versa
      move all points in a EW_From3_Pointset in the specimen to the locations
       they occupy in the template or vice versa
      move all points in the specimen to the locations they occupy in the
       template or vice versa
  These methods have been partially implemented:
      EW_Dig3::get_spline_node
      EW_Dig3_Space::get_surface_nodes
      EW_Dig3_Space::set_embedding
      EW_Dig3_Space::remove_embedding
      EW_Dataflow_Spline::get_energy
      EW_Dataflow_Spline::get_n_lmks
      EW_Dataflow_Spline::get_f_size
      EW_Dataflow_Spline::warp_point
      EW_Dataflow_Spline::lmk_index
      EW_Dataflow_Spline::lmk_pointset
      EW_Dataflow_Spline::lmk_pointset_i
      EW_Dataflow_Spline::optimized_lmk_image
      EW_Dataflow_Surface3_Explicit::project
  They will allow the following additional operations to be performed:
      display information about the spline (number of (semi-)landmarks,
       algebraic dimension, singularity, energy)
      move an ignored landmark or semi-landmark in the specimen to the
       where it is warped from the template
      move all ignored landmarks or semi-landmarks in the specimen to the
       where they are warped from the template
      change a EW_Form3_Pointset in the specimen from "unset" to "warped",
       setting all points to where they are warped from the template
      set or change the surface an EW_Form3_Pointset is embedded in
       move an embedded landmark to its surface
      move a semi-landmark to its surface, display the distance moved, and/or
       set the relax_params and relax_dims of a semi-landmark to the surface
       tangent space
      move all semi-landmarks of a EW_Form3_Pointset to their surface,
       display the average and maximum distance moved, and/or set the
       relax_params and relax_dims of the semi-landmarks to the surface
       tangent spaces
      move all semi-landmarks to their surfaces, display the average and
       maximum distance moved, and/or set the relax_params and relax_dims of
       the semi-landmarks to the surface tangent spaces.
      slide a semi-landmark to the position in its relaxation space that
       minimizes the spline's energy, and display the distance moved
      slide all semi-landmarks to the positions in their relaxation spaces
       that minimize the spline's energy, and display the average and
       maximum distance moved


2009-12-10  Bill Green  <wdkg@wdkg.org>
```

```
    * Restructured exception handling to improve exception safety and allow the
    elimination of the warn_stream optional argument of the
    EW_Dataflow_network constructor and of EW_Dataflow_network::get_warned.


2009-11-29  Bill Green  <wdkg@wdkg.org>

    * Fixed misbehviour in the filename handling code.

    * Converted surface faces and points from an array to a vector.
    The efficiency gained from using arrays (around 10%) did not justify the
    amount of code.


2009-11-05  Bill Green  <wdkg@wdkg.org>

    * All filename fields and arguments are now expected to be canonical
    utf-8 filenames.
    EW_Form3::read_file and EW_Form3::write_file will convert filenames in
    the input/output from/to simple filenames, when appropriate.


2009-11-04  Bill Green  <wdkg@wdkg.org>

    * Removed the form element label field.
    The id will now be used as the label.
    Id's can now contain whitespace.
    This results in a change to the form file format.

    * Included a version number in the form3 and dig3 file formats.


2009-09-17  Bill Green  <wdkg@wdkg.org>

    * Changed "target" to "specimen" in code and tableau file formats.


2009-09-10  Bill Green  <wdkg@wdkg.org>

    * Added:
      EW_View3_Widget::set_highlight_color
      EW_View3_Widget::get_highlight_color
      EW_View3_Widget::get_highlight_item
      EW_View3_Widget::clear_highlight
      EW_View3_Landmarks::set_highlight
      EW_View3_Landmarks::get_highlight_ps
      EW_View3_Landmarks::get_highlight_i

    * Added EW_View3_Widget::pick.

    * Changed signature of EW_Dig3_Space::set_pointset.


2009-09-02  Bill Green  <wdkg@wdkg.org>

    * Changed EW_View3_Item linked list into a vector of pointers in
```

```
    EW_View3_Widget.
    Removed:
        EW_View3_Widget::get_first_item
        EW_View3_Widget::get_last_item
        EW_View3_Widget::move_item_before
        EW_View3_Widget::move_item_after
        EW_View3_Item::get_next_item
        EW_View3_Item::get_prev_item
    Added:
        EW_View3_Widget::get_items
        EW_View3_Widget::get_n_items
        EW_View3_Widget::move_item
        EW_View3_Item::get_index


2009-08-27  Bill Green  <wdkg@wdkg.org>

    * Added EW_Dig3::interpolate_tableau.


2009-08-26  Bill Green  <wdkg@wdkg.org>

    * Added
        EW_BBox3
        EW_Dataflow_Surface3::get_bbox
        EW_Dataflow_Landmarks3::get_bbox
        EW_View3_Item::get_bbox
        EW_View3_Widget::get_bbox
        EW_View3_Widget::get_bbox_tr
        EW_Dig3_Space::get_bbox


2009-08-25  Bill Green  <wdkg@wdkg.org>

    * Added:
        EW_View3_Widget::get_center_location
        EW_View3_Widget::get_jump_tr

    * Fixed omission in EW_Dig3::load_tableau.


2009-08-24  Bill Green  <wdkg@wdkg.org>

    * Fixed crashing caused by EW_View3_Widget::move_item_before.

    * Fixed failure to load obj files with multiple index f form.

    * Reverted to rendering slice view without depth buffer, as the only
    reliable way to get landmarks to always show up.


2009-08-24  Bill Green  <wdkg@wdkg.org>

    * Added, to support landmarks:
        EW_Form3::search_pointset
        EW_Dataflow_Form3::set_pointset
        EW_Dataflow_Form3::set_pointset_location
```

```
        EW_Dataflow_Form3::remove_pointset
        EW_View3_Landmarks
        EW_Dig3_Tableau_Space::bool show_lmks_in_main
        EW_Dig3_Tableau_Space::bool show_lmks_in_slice
        EW_Dig3_Tableau_Space::int lmks_symbol
        EW_Dig3_Tableau_Space::unsigned char lmks_col
        EW_Dig3_View::get_landmarks_item
        EW_View3_Widget::get_pointer_location
    and updated:
        EW_Dig3::save_tableau
        EW_Dig3::load_tableau

    * Renamed, to be a little less confusing:
        EW_Dataflow_Form3::unset_surface -> EW_Dataflow_Form3::remove_surface


2009-08-20  Bill Green  <wdkg@wdkg.org>

    * Simplified the form3 file format by merging the very similar point
    and pointset fields.


2009-08-18  Bill Green  <wdkg@wdkg.org>

    * Added EW_Dig3_View::get_slice_index()


2009-08-06  Bill Green  <wdkg@wdkg.org>

    * Added class EW_Dig3_Tableau for save files, and related methods in
    EW_Dig3.

    * Added:
        EW_Transform3::scan
        EW_Transform3::format

    * Renamed, to work better with debug message selection:
        EW_Dataflow -> EW_Dataflow_Network
        EW_View3 -> EW_View3_Widget

    * Replaced EW_Dig3_Geom and EW_Dig3_Flat with EW_Dig3, EW_Dig3_Space and
    EW_Dig3_View, to reduce the number of classes an application needs to
    implement and to avoid requiring virtual inheritance.

    * Renamed, to better reflect their purpose:
        EW_Transform2::set_identity -> EW_Transform2::set_to_identity
        EW_Transform2::set_inverse -> EW_Transform2::set_to_inverse
        EW_Transform2::set_composition -> EW_Transform2::set_to_composition
        EW_Transform2::set_interpolation -> EW_Transform2::set_to_interpolation
        EW_Transform2::set_normalization -> EW_Transform2::set_to_normalization
        EW_Transform3::set_identity -> EW_Transform3::set_to_identity
        EW_Transform3::set_inverse -> EW_Transform3::set_to_inverse
        EW_Transform3::set_composition -> EW_Transform3::set_to_composition
        EW_Transform3::set_interpolation -> EW_Transform3::set_to_interpolation
        EW_Transform3::set_normalization -> EW_Transform3::set_to_normalization

    * Renamed in EW_View3_Widget
```

```
    EW_View3_Widget::schedule_idle_handler ->
     EW_View3_Widget::schedule_idle_handler_cb
    EW_View3_Widget::redraw -> EW_View3_Widget::redraw_cb

* Replaced the exception in EW_Transform2/3::set_to_interpolation with a
return code.

* Replaced GLuint with unsigned int to avoid exposing OpenGL in interface.

* Made a depth buffer a requirement in view3_widget.cpp.

* Replaced use of isspace, printf, sscanf, strtol and strtod for data files
to avoid locale and library implementation complications.
Used gdtoa for floating point conversions.

* Made use of XML_PARSE_COMPACT in dataflow_form3.cpp dependent on
LIBXML_VERSION for Mac OS X, where the default libxml2 is too old to
support this.
```

10 Thin-Plate Spline with Semi-landmarks on Affine Subspaces

Suppose we have *n_lmks* landmarks in $\mathbb{R}^D$, where D is 2 or 3. Let *lmks* be the *n_lmks* x D matrix with rows the landmark coordinates.

A thin-plate spline is a linear combination of these functions $\mathbb{R}^D \to \mathbb{R}$:

- the constant function 1
- the linear function x
- the linear function y
- (if D = 3) the linear function z
- the radial basis function at landmark 1
- ...
- the radial basis function at landmark *n_lmks*

A D-valued thin-plate spline is a function $\mathbb{R}^D \to \mathbb{R}^D$, consisting of D single-valued thin-plate splines, one in each of the x, y and (if D = 3) z directions. It can be represented as *spline*, the (1 + D + *n_lmks*) x D matrix of the coefficients of the corresponding linear combinations.

Let *lmk_images* be the *n_lmks* x D matrix with rows the images of the landmarks under the spline. Let *aff_lmk_images* denote the (1 + D + *n_lmks*) x D matrix with first 1 + D rows zero (corresponding to the affine spline components) and remaining rows the rows of *lmk_images*. Then

$$L * spline = aff\_lmk\_images$$

where L is a (1 + D + *n_lmks*) x (1 + D + *n_lmks*) matrix that is calculated from *lmks*. L can be viewed as the matrix of landmark interactions.

A semi-landmark is a landmark for which, instead of an image, an affine subspace of $\mathbb{R}^D$ is specified. A normal landmark can be considered a semi-landmark with an affine space of dimension 0. A D-valued thin-plate spline with semi-landmarks is a thin-plate spline that minimizes bending energy while mapping every semi-landmark into its corresponding affine subspace. The spline is allowed to 'relax' (minimize energy) by sliding the semi-landmark images along their 'relaxation' subspaces.

These affine subspaces will be specified here as linear spaces about the points that are the rows of *lmk_images*. Let *relax_dims* be the integer vector of length *n_lmks*, consisting of the dimensions of these affine spaces. It's elements can be

- 0 The semi-landmark is a regular landmark.
- 1 The semi-landmark is a semi-landmark relaxed along a line.
- 2 The semi-landmark is a semi-landmark relaxed along a plane (in 3D) or is ignored (in 2D).
- 3 (3D only) The semi-landmark is ignored.

The relaxation subspaces can be defined by a *n_lmks* x D matrix, *relax_params*. For regular landmarks and ignored semi-landmarks, the corresponding row of *relax_params* is unused. For codimension 1 affine spaces, the corresponding row of *relax_params* should be a unit vector normal to the space. For semi-landmarks along a line in 3D, the corresponding row of *relax_params* should be a unit vector normal in the direction of the line.

Our problem, then, is to find the spline *spline* that minimizes energy, given *lmks*, *lmk_images*, *relax_dims* and *relax_params*. Let *relax_lmk_images* be the *n_lmks* x D matrix of the coordinates of the images of the semi-landmarks under the relaxed spline.

Rewrite

$$L * spline = aff\_relax\_lmk\_images$$

as a simple equation in (1 + D + *n_lmks*) $*$ D variables

$$L3 * spline\_flat = aff\_relax\_lmk\_images\_flat$$

where *spline_flat* is the vector formed by concatenating the rows of *spline*, *aff_relax_lmk_images_flat* is the vector formed by concatenating the rows of *aff_relax_lmk_images*, and *L3* is the matrix formed by replacing each element of L by the D x D identity matrix multiplied by that element.

The constraint on the relaxed spline is that

aff_relax_lmk_images_flat - aff_lmk_images_flat

must be in the direct sum of the linear parts of the relaxation spaces. The energy of the spline is the dot product of *spline_flat* and *aff_relax_lmk_images_flat*. Hence, the energy will be minimized when *spline_flat* is orthogonal to this direct sum.

Construct a matrix *R* as follows. *R* is a block matrix, with (1 + D + *n_lmks*) x (1 + D + *n_lmks*) blocks and all off diagonal blocks zero. The first 1 + D on-diagonal blocks are D x D identity matrices The remaining on-diagonal blocks depend on the dimension of the relax space of the corresponding semi-landmark. If D = 2:

- 0: a D x D identity matrix
- 1: a D x 1 column vector equal to the corresponding part of *relax_params*
- 2: a D x 0 empty matrix

If D = 3:

- 0: a D x D identity matrix
- 1: a D x 2 matrix with 2 independent column vectors perpendicular to the corresponding part of *relax_params*
- 2: a D x 1 column vector equal to the corresponding part of *relax_params*
- 3: a D x 0 empty matrix

Then the columns of *R* form a basis of the orthogonal complement to the direct of the linear parts of the relaxation spaces. Hence *spline_flat* is a linear combination of these columns, and we can write

$$spline\_flat = R * spline\_flat\_reduced$$

for some vector *spline_flat_reduced*. Also

$$R' * aff\_relax\_lmk\_images\_flat = R' * aff\_lmk\_images\_flat$$

where *R'* is the transpose of *R*. The spline equation then becomes

$$(R' * L3 * R) * spline\_flat\_reduced = aff\_lmk\_images\_flat\_reduced$$

where

$$aff\_lmk\_images\_flat\_reduced = R' * aff\_lmk\_images\_flat$$

In the general case, this is the equation factored by ew::Tps2::factorize or ew::Tps3::factorize, and solved by ew::Tps2::solve or ew::Tps3::solve.

If there are only trivial semi-landmarks (the relax space has dimension 0 or D), the matrix *R'* ∗ *L3* ∗ *R* is, like *L3*, block diagonal. This is the case *is_mixed* = `false`. In this case, we don't need to invert (*R'* ∗ *L3* ∗ *R*), rather just one of the diagonal blocks, which is just *L* with the rows and columns of the ignored landmarks deleted. This is the matrix factored by ew::Tps2::factorize or ew::Tps3::factorize. ew::Tps2::solve or ew::Tps3::solve then solve the 3 sets of equations. With no semi-landmarks, this reverts to the original equation.

Non-trivial semi-landmarks introduce an interaction between the components of the D-valued spline, and we cannot factor the equation as we've presented it.

Let *r* be the sum of the dimensions of the relax spaces. Then *f_size*, the size of the matrix the algorithm factors is as follows:

- if *is_reduced* is `false`, *f_size* is 1 + D + *n_lmks*,
- if *is_mixed* is `false`, *f_size* is 1 + D + *n_lmks* - *r* / D,
- otherwise, *f_size* is D ∗ (1 + D + *n_lmks*) - *r*.

An alternative algorithm would be to invert *L*, calculate the bending energy relative to a basis of the relaxation space as a quadratic function, and then minimize this quadratic function. This would involve inverting a matrix of size 1 + D + *n_lmks* and then diagonalizing a quadratic form of size *r*. Clearly, for small non-zero *r*, this would be faster (none of this applies if *r* is zero, or more generally if *is_mixed* is `false`). On the other hand, this would be slower for a spline in 3D with mostly plane semi-landmarks, where *r* approached (D - 1) ∗ *n_lmks*. This is exactly the case where we are likely to encounter a very large number of landmarks, and it is the case for which this code has been optimized.

11 Canonical Uniform Warp Basis

For landmark configurations in 2D, this is the basis of the space of shape variables described in

```
BOOKSTEIN, F L. 1996b. A standard formula for the uniform shape
component in landmark data. Pages 153–168 in Advances in morphometrics
(L. F Marcus, M. Corti, A. Loy, G. J. P. Naylor, and D.
E. Slice, eds.). Plenum, New York
```

and in

```
Computing the Uniform Component of Shape Variation
F. JAMES ROHLF AND FRED L. BOOKSTEIN
Syst. Biol. 52(1):66–69, 2003
```

These papers describe the basis in the case of a landmark configuration that has been centered, scaled to centroid size one and had its principal axes oriented to the coordinate axes.

The space in question is 2 dimensional, and the 2 basis elements are derived by applying linear shears

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$$

to the landmarks of the centered, scaled and oriented landmark configuration. The resulting shape variables are projected onto the shape space tangent space.

ew::Tps2::principal_axes() calculates such an arrangement of axes, and ew::Tps2::uniform_basis() calculates the same basis for an arbitrary configuration of landmarks, one which hasn't necessarily been centered, scaled or aligned.

ew::Tps3::principal_axes() calculates the analagous arrangement of axes in 3D. ew::Tps3::uniform_basis() calculates an unpublished generalization of the above basis. In 3D the space of uniform warps is 5 dimensional. If we apply the same procedure in 3D, as we did in 2D, but with the following shears

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

the last 3 resulting shape variables are linearly dependent. ew::Tps3::uniform_basis() calculates the first 4 resulting shape variables and a linear combination of the last 2 which result in a basis.

In a centered and scaled landmark configuration of n landmarks, the results of ew::Tps2::uniform_basis() are orthonormal vectors of $\mathbb{R}^{2n}$ that are othogonal to the landmark displacements resulting from infinitesimal rotations, translations and scalings, and similarly for the results of ew::Tps3::uniform_basis() in $\mathbb{R}^{3n}$.

12 Class Index

12.1 Class Hierarchy

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

13 Class Index

13.1 Class List

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

14 Class Documentation

14.1 ew::Bbox3 Class Reference

3D Bounding Box

`#include <ew/Bbox3.h>`

Public Member Functions

- bool get_empty () const
- double get_radius_center (double *c) const
- void set_to_empty ()
- void set_to_point (const float *pt)
- void set_to_point (const double *pt)
- void set_to_points (const float *pt, int n)
- void set_to_points (const double *pt, int n)
- void add (const float *pt)
- void add (const double *pt)
- void add (const float *pt, int n)
- void add (const double *pt, int n)
- void set_to_union (const ew::Bbox3 *b1, const ew::Bbox3 *b2)
- bool operator== (const ew::Bbox3 &a) const
- bool operator!= (const ew::Bbox3 &a) const

Public Attributes

- double min [3]
- double max [3]

Static Public Attributes

- static const ew::Bbox3 empty_box

14.1.1 Detailed Description

ew::Bbox3 represents bounding boxes in $\mathbb{R}^3$.

ew::Bbox3 is a POD type class.

14.1.2 Member Function Documentation

14.1.2.1 bool ew::Bbox3::get_empty ( ) const [inline]

Returns

`true` if the minima are infinity and the maxima are negative infinity

14.1.2.2 double ew::Bbox3::get_radius_center (double * *c*) const

Calculates the center and circumradius of a box.

Precondition

The box is not empty.

Parameters

`[out]` ***c*** the address of an array of size 3 where the center coordinates should be stored.

Returns

the radius

14.1.2.3 void ew::Bbox3::set_to_empty ( ) [inline]

Set this box to be the empty box.

14.1.2.4 void ew::Bbox3::set_to_point (const float * *pt*) [inline]

Set this box to be the bounding box of a point.

Parameters

`[in]` ***pt*** the address of the array of size 3 containing the point coordinates

14.1.2.5 void ew::Bbox3::set_to_point (const double * *pt*) [inline]

Set this box to be the bounding box of a point.

Parameters

`[in]` ***pt*** the address of the array of size 3 containing the point coordinates

14.1.2.6 void ew::Bbox3::set_to_points (const float * *pt*, int *n*) [inline]

Set this box to be the bounding box of an array of points.

Parameters

[in] ***pt*** the address of an array of size $3 * n$ containing the point coordinates, ordered by point

n the number of points

14.1.2.7 void ew::Bbox3::set_to_points (const double * *pt*, int *n*) [inline]

Set this box to be the bounding box of an array of points.

Parameters

[in] ***pt*** the address of an array of size $3 * n$ containing the point coordinates, ordered by point

n the number of points

14.1.2.8 void ew::Bbox3::add (const float * *pt*) [inline]

Extend this box, if necessary, to include a point.

Parameters

[in] ***pt*** the address of the array of size 3 containing the point coordinates

14.1.2.9 void ew::Bbox3::add (const double * *pt*) [inline]

Extend this box, if necessary, to include a point.

Parameters

[in] ***pt*** the address of the array of size 3 containing the point coordinates

14.1.2.10 void ew::Bbox3::add (const float * *pt*, int *n*) [inline]

Extend this box, if necessary, to include an array of points.

Parameters

[in] ***pt*** the address of an array of size $3 * n$ containing the point coordinates, ordered by point

n the number of points

14.1.2.11 void ew::Bbox3::add (const double * *pt*, int *n*) [inline]

Extend this box, if necessary, to include an array of points.

Parameters

[in] ***pt*** the address of an array of size $3 * n$ containing the point coordinates, ordered by point

n the number of points

14.1.2.12 void ew::Bbox3::set_to_union (const ew::Bbox3 * *b1*, const ew::Bbox3 * *b2*)

Set the bounding box to the union of the specified bounding boxes.

Parameters

b1,b2 the boxes to combine

14.1.2.13 bool ew::Bbox3::operator== (const ew::Bbox3 & *a*) const

Compares this box with another, member by member.

Parameters

a the other box

14.1.2.14 bool ew::Bbox3::operator!= (const ew::Bbox3 & *a*) const [inline]

Compares this box with another, member by member.

Parameters

a the other box

14.1.3 Member Data Documentation

14.1.3.1 double ew::Bbox3::min[3]

These are the minima in the 3 coordinate directions. Except for the empty box, these should all be finite and no greater than the corresponding maxima.

14.1.3.2 double ew::Bbox3::max[3]

These are the maxima in the 3 coordinate directions. Except for the empty box, these should all be finite and no less than the corresponding minima.

14.1.3.3 const ew::Bbox3 ew::Bbox3::empty_box [static]

Initial value:

```
{
 {
   std::numeric_limits<double>::infinity(),
   std::numeric_limits<double>::infinity(),
   std::numeric_limits<double>::infinity()
 }, {
   -std::numeric_limits<double>::infinity(),
   -std::numeric_limits<double>::infinity(),
   -std::numeric_limits<double>::infinity()
 }
}
```

This is an empty box.

14.2 ew::Curve3 Class Reference

Piecewise-linear Curve.

`#include <ew/Curve3.h>`

Public Member Functions

- void reset ()
- void read_file (const char *file)
- void read_points (const std::vector< double > &coords)
- void write_file (const char *file) const
- void swap (ew::Curve3 &s)
- bool operator== (const ew::Curve3 &a) const
- bool operator!= (const ew::Curve3 &a) const

Public Attributes

- std::vector< float > points
- std::vector< int > edges

14.2.1 Detailed Description

ew::Curve3 encapsulates data for piecewise-linear curves in $\mathbb{R}^3$.

The data consists of a vector of points and a vector of edges.

ew::Curve3 is a data structure that supports default construction, copy construction and assignment.

14.2.2 Member Function Documentation

14.2.2.1 void ew::Curve3::reset ( )

This frees any current data and sets the object to the initial state.

14.2.2.2 void ew::Curve3::read_file (const char * *filename*)

This reads the surface contained in the *file*. The following surface formats are supported:

- obj format Only these types of record are currently supported:
  - v
  - l with non-negative arguments

Parameters

filename the name of the file to read

Exceptions

std::bad_alloc

ew::ErrorIO

14.2.2.3 void ew::Curve3::read_points (const std::vector< double > & *coords*)

This reads a curve as a list of points.

Parameters

coords the vector of coordinates

Exceptions

std::bad_alloc

14.2.2.4 void ew::Curve3::write_file (const char * *filename*) const

This writes the surface to the *file* in obj format.

Parameters

filename the name of the file to write

Exceptions

ew::ErrorIO

14.2.2.5 void ew::Curve3::swap (ew::Curve3 & *s*) [inline]

Swap data between 2 curves without copying.

Parameters

s the surface to swap `this` with

14.2.2.6 bool ew::Curve3::operator== (const ew::Curve3 & *a*) const

Compares this curve with another, member by member.

Parameters

a the other curve

14.2.2.7 bool ew::Curve3::operator!= (const ew::Curve3 & *a*) const [inline]

Compares this curve with another, member by member.

Parameters

a the other curve

14.2.3 Member Data Documentation

14.2.3.1 ew::Curve3::points

This vector has size 3 times the number of points. Each consecutive triple defines a point.

14.2.3.2 ew::Curve3::edges

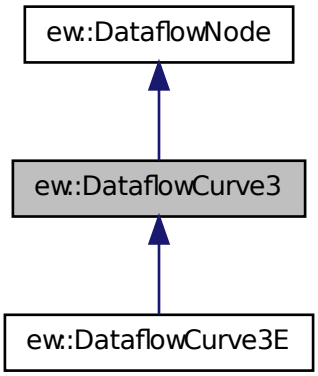
This vector has size 2 times the number of edges. The elements are indices of the points list, and are in the range [0, number of points). Each consecutive pair defines an edge.

14.3 ew::DataflowCurve3 Class Reference

Curve Node Base Class.

`#include <ew/DataflowCurve3.h>`

Inheritance diagram for ew::DataflowCurve3:



Public Member Functions

- int get_num_points () const

- int get_num_edges () const
- const float * get_points () const
- const int * get_edges () const
- const ew::Bbox3 * get_bbox () const
- void make_index () const
- bool index_is_made () const
- double project (int *edge, double *coeffs, double *tangent, double *proj, const double *inp) const

14.3.1 Detailed Description

ew::DataflowCurve3 is the base class for nodes containing piecewise-linear curves in $\mathbb{R}^3$.

ew::DataflowCurve3 is a class without assignment or comparison. There are private member variables.

For efficient rendering of the same curve in multiple windows, an OpenGL display list is constructed containing the basic curve geometry commands. All windows displaying a given curve must share display lists.

14.3.2 Member Function Documentation

14.3.2.1 int ew::DataflowCurve3::get_num_points ( ) const [inline]

This will force the network into an updating phase.

Returns

The number of points used in the curve.

14.3.2.2 int ew::DataflowCurve3::get_num_edges ( ) const [inline]

This will force the network into an updating phase.

Returns

The number of triangular edges in the curve.

14.3.2.3 const float * ew::DataflowCurve3::get_points ( ) const [inline]

This will force the network into an updating phase. The pointer is valid until the node or a dependency of the node is changed.

Returns

A pointer to the coordinates of the points used in the curve.

14.3.2.4 const int * ew::DataflowCurve3::get_edges ( ) const [inline]

This will force the network into an updating phase. The pointer is valid until the node or a dependency of the node is changed.

Returns

A pointer to the indices of the points used in the curve.

14.3.2.5 const ew::Bbox3 * ew::DataflowCurve3::get_bbox ( ) const [inline]

The pointer is valid until the node or a dependency of the node is changed.

Returns

A pointer to the bounding box of the curve.

14.3.2.6 void ew::DataflowCurve3::make_index ( ) const [inline]

This makes the curve spatial index if it has not already been made since the curve was last changed. The spatial index is used by project.

14.3.2.7 bool ew::DataflowCurve3::index_is_made ( ) const [inline]

Returns

`true` if the curve spatial index is up to date.

14.3.2.8 double ew::DataflowCurve3::project (int * *edge*, double * *coeffs*, double * *tangent*, double * *proj*, const double * *inp*) const

This finds the nearest point on the curve to a given point. The curve spatial index will be made if it has not already been made.

Parameters

`[out]` ***edge*** The index of the edge of the curve containing the nearest point.

[out] ***coeffs*** The coefficients of the nearest point when expressed as a linear combination of the vertices of the edge.

[out] ***tangent*** The interpolated tangent at the nearest point. If a sensible tangent cannot be calculated, an arbitrary unit vector is returned.

[out] ***proj*** The coordinates of the nearest point on the curve.

[in] ***inp*** The coordinates of the original point.

Returns

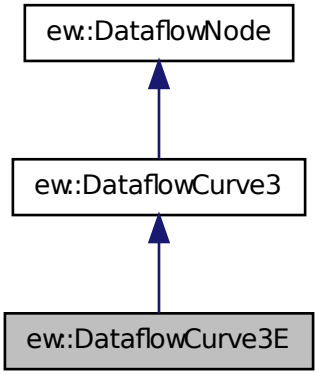
The distance from the original point to the projected point.

14.4 ew::DataflowCurve3E Class Reference

Explicit Curve Node.

`#include <ew/DataflowCurve3E.h>`

Inheritance diagram for ew::DataflowCurve3E:



Public Member Functions

- DataflowCurve3E (ew::DataflowNetwork *i_network)
- void set_data (ew::Curve3 *data)
- const ew::Curve3 * get_data () const

14.4.1 Detailed Description

ew::DataflowCurve3E is a node containing an explicitly specified curve.

ew::DataflowCurve3E objects may not be copied or compared. There are private member variables.

Nodes of this class do not depend on other nodes, but other nodes can depend on them. They are valid if they have some faces.

Initially and when reset, the node is empty.

14.4.2 Constructor & Destructor Documentation

14.4.2.1 ew::DataflowCurve3E::DataflowCurve3E (ew::DataflowNetwork * *i_network*) [explicit]

This creates an explicit curve node.

Parameters

i_network The network this node should belong to.

14.4.3 Member Function Documentation

14.4.3.1 void ew::DataflowCurve3E::set_data (ew::Curve3 * *data*)

This transfers the curve data from the ew::Curve3 object to the node. Afterwards, the ew::Curve3 object is reset.

Precondition

This node must not be a cached node.

Parameters

data The object containing the curve data.

14.4.3.2 const ew::Curve3 * ew::DataflowCurve3E::get_data ( ) const [inline]

Returns

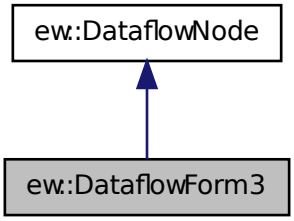
A pointer to the internal curve data.

14.5 ew::DataflowForm3 Class Reference

3D Form Node

`#include <ew/DataflowForm3.h>`

Inheritance diagram for ew::DataflowForm3:



Public Member Functions

- DataflowForm3 (ew::DataflowNetwork *i_network)
- void set_data (const ew::Form3 *data)
- const ew::Form3 * get_data () const
- int set_volume (bool *replaced, const ew::Form3Volume *vl)
- void remove_volume (int n)
- int set_surface (bool *replaced, const ew::Form3Surface *sr)
- void remove_surface (int n)
- int set_curve (bool *replaced, const ew::Form3Curve *cr)
- void remove_curve (int n)
- int set_pointset (bool *replaced, const ew::Form3PointSet *ps)
- void set_pointset_location (int n, int i, const double *loc)
- void set_pointset_relax (int n, int i, int rdim, const double *rparam)
- void remove_pointset (int n)
- int set_embedding (bool *replaced, const ew::Form3Embedding *em)
- void set_superset (const char *subset_id, const char *superset_id)
- void remove_embedding (int n)
- int set_reflection (bool *replaced, const ew::Form3Reflection *rf)
- void remove_reflection (int n)
- unsigned long get_change_cycle_association () const

- unsigned long get_change_cycle_coords () const
- unsigned long get_change_cycle_relax () const
- const ew::Bbox3 * get_bbox () const

14.5.1 Detailed Description

ew::DataflowForm3 is a node that manages an ew::Form3.

ew::DataflowForm3 is a class without assignment or comparison. There are private member variables.

Nodes of this class do not depend on other nodes, but other nodes can depend on them. They are always considered valid.

Initially and when reset, the node is empty.

14.5.2 Constructor & Destructor Documentation

14.5.2.1 ew::DataflowForm3::DataflowForm3 (ew::DataflowNetwork * *i_network*) [explicit]

This creates a form3 node.

Parameters

i_network The network this node should belong to.

14.5.3 Member Function Documentation

14.5.3.1 void ew::DataflowForm3::set_data (const ew::Form3 * *data*)

This sets the contents of the node.

Parameters

data A pointer to the ew::Form3 data to copy.

14.5.3.2 const ew::Form3 * ew::DataflowForm3::get_data ( ) const [inline]

Returns

A pointer to the current ew::Form3 data contained in the node.

14.5.3.3 int ew::DataflowForm3::set_volume (bool * *replaced*, const ew::Form3Volume * *vl*)

This adds or replaces a volume in the ew::Form3 data managed by the node.

Parameters

`[out]` ***replaced*** `true` if the volume replaced an existing volume.

vl A pointer to the volume data to copy.

Returns

The index of the new volume in ew::Form3::volumes.

14.5.3.4 void ew::DataflowForm3::remove_volume (int *n*)

This deletes a volume in the ew::Form3 managed by the node.

Parameters

n The index of the volume to delete.

14.5.3.5 int ew::DataflowForm3::set_surface (bool * *replaced*, const ew::Form3Surface * *sr*)

This adds or replaces a surface in the ew::Form3 data managed by the node.

Parameters

`[out]` ***replaced*** `true` if the surface replaced an existing surface.

sr A pointer to the surface data to copy.

Returns

The index of the new surface in ew::Form3::surfaces.

14.5.3.6 void ew::DataflowForm3::remove_surface (int *n*)

This deletes a surface in the ew::Form3 managed by the node.

Parameters

n The index of the surface to delete.

14.5.3.7 int ew::DataflowForm3::set_curve (bool * *replaced*, const ew::Form3Curve * *cr*)

This adds or replaces a curve in the ew::Form3 data managed by the node.

Parameters

`[out]` ***replaced*** `true` if the curve replaced an existing curve.

cr A pointer to the curve data to copy.

Returns

The index of the new curve in ew::Form3::curves.

14.5.3.8 void ew::DataflowForm3::remove_curve (int *n*)

This deletes a curve in the ew::Form3 managed by the node.

Parameters

n The index of the curve to delete.

14.5.3.9 int ew::DataflowForm3::set_pointset (bool * *replaced*, const ew::Form3PointSet * *ps*)

This adds or replaces a point set in the ew::Form3 data managed by the node.

Parameters

`[out]` ***replaced*** `true` if the point set replaced an existing point set.

ps A pointer to the point set data to copy.

Returns

The index of the new or replaced pointset.

14.5.3.10 void ew::DataflowForm3::set_pointset_location (int *n*, int *i*, const double * *loc*)

This changes the coordinates of an element of the point set.

Parameters

n The index of the point set.

i The index of the element in the point set.

`[in]` ***loc*** A pointer to the new coordinates of this entry.

14.5.3.11 void ew::DataflowForm3::set_pointset_relax (int *n*, int *i*, int *rdim*, const double * *rparam*)

This changes the relaxation parameters of an element of the point set.

Parameters

n The index of the point set.

i The index of the element in the point set.

`[in]` ***rdim*** The new relax_dims of this entry.

`[in]` ***rparam*** A pointer to the new relax_params of this entry.

14.5.3.12 void ew::DataflowForm3::remove_pointset (int *n*)

This deletes a point set in the ew::Form3 managed by the node.

Parameters

n The index of the point set to delete.

14.5.3.13 int ew::DataflowForm3::set_embedding (bool * *replaced*, const ew::Form3Embedding * *em*)

This adds or replaces a embedding in the ew::Form3 data managed by the node.

Parameters

`[out]` ***replaced*** `true` if the embedding replaced an existing embedding.

em A pointer to the embedding data to copy.

Returns

The index of the new embedding in ew::Form3::embeddings.

14.5.3.14 void ew::DataflowForm3::set_superset (const char * *subset_id*, const char * *superset_id*)

This makes one element of the form the unique superset of another element.

Parameters

subset_id The id of the element that should have a unique superset.

superset_id The id of the element that be the unique superset.

14.5.3.15 void ew::DataflowForm3::remove_embedding (int *n*)

This deletes a embedding in the ew::Form3 managed by the node.

Parameters

n The index of the embedding to delete.

14.5.3.16 int ew::DataflowForm3::set_reflection (bool * *replaced*, const ew::Form3Reflection * *rf*)

This adds or replaces a reflection in the ew::Form3 data managed by the node.

Parameters

`[out]` ***replaced*** `true` if the reflection replaced an existing reflection.

rf A pointer to the reflection data to copy.

Returns

The index of the new reflection in ew::Form3::reflections.

14.5.3.17 void ew::DataflowForm3::remove_reflection (int *n*)

This deletes a reflection in the ew::Form3 managed by the node.

Parameters

n The index of the reflection to delete.

14.5.3.18 unsigned long ew::DataflowForm3::get_change_cycle_association ( ) const [inline]

Returns

The last cycle the form was changed in a way that effects landmark matching.

14.5.3.19 unsigned long ew::DataflowForm3::get_change_cycle_coords ( ) const [inline]

Returns

The last cycle that any location of a landmark was changed.

14.5.3.20 unsigned long ew::DataflowForm3::get_change_cycle_relax ( ) const [inline]

Returns

The last cycle that any relaxation of a landmark was changed.

14.5.3.21 const ew::Bbox3 * ew::DataflowForm3::get_bbox ( ) const [inline]

The pointer until the node or a dependency of the node is changed.

Returns

A pointer to the bounding box of the points in the form.

14.6 ew::DataflowNetwork Class Reference

Lightweight Dataflow Network.

`#include <ew/DataflowNetwork.h>`

Public Member Functions

- DataflowNetwork ()
- ~DataflowNetwork ()
- unsigned long get_cycle () const
- const ew::DataflowCurve3E * cached_curve (const char *filename)
- const ew::DataflowSurface3E * cached_surface (const char *filename)

14.6.1 Detailed Description

EW::DataflowNetwork provides the infrastructure for a lightweight dataflow network.

To keep track of what calculations done by nodes in the network are up to date, and which need to be redone, EW::DataflowNetwork keeps a cycle counter. A cycle consists of an initial phase, when changes are made to nodes of the network, followed by a second phase, when updates of various calculations are made. Requesting updated

output from a node puts the network into the update phase of the same cycle, if it is not already in an update phase. Changing a node puts the network into the change phase of the next cycle, if it is not already in a change phase. Initially the network is in the change phase of cycle 1.

Operations on a network must not be performed simultaneously in different threads. However, operations on different networks may be.

This class does not support assignment or comparison. It contains private members and undocumented members.

14.6.2 Constructor & Destructor Documentation

14.6.2.1 ew::DataflowNetwork::DataflowNetwork ( )

This creates an empty network.

14.6.2.2 ew::DataflowNetwork::∼DataflowNetwork ( )

This destroys the network. All nodes and other objects using this network must already have been destroyed.

14.6.3 Member Function Documentation

14.6.3.1 unsigned long ew::DataflowNetwork::get_cycle ( ) const [inline]

Returns

The current cycle number.

14.6.3.2 const ew::DataflowCurve3E ∗ ew::DataflowNetwork::cached_curve (const char ∗ *filename*)

The first time this is called with a given filename, it reads the curve from the file, creates a new explicit curve node using it, and returns a pointer to it. Subsequently it increments the same node's reference count and returns a pointer to it. When the pointer to the node is no longer needed, the reference should be released with EW::DataflowNode::decr_ref_count. The cache retains a reference to the node, so it will not be deleted until all other references are released and the network is destroyed.

Parameters

filename is the canonical name of the file to read.

Exceptions

std::bad_alloc

EW::ErrorIO

14.6.3.3 const ew::DataflowSurface3E * ew::DataflowNetwork::cached_surface (const char * *filename*)

The first time this is called with a given filename, it reads the surface from the file, creates a new explicit surface node using it, and returns a pointer to it. Subsequently it increments the same node's reference count and returns a pointer to it. When the pointer to the node is no longer needed, the reference should be released with EW::DataflowNode::decr_ref_count. The cache retains a reference to the node, so it will not be deleted until all other references are released and the network is destroyed.

Parameters

filename The name of the file to read.

Exceptions

std::bad_alloc

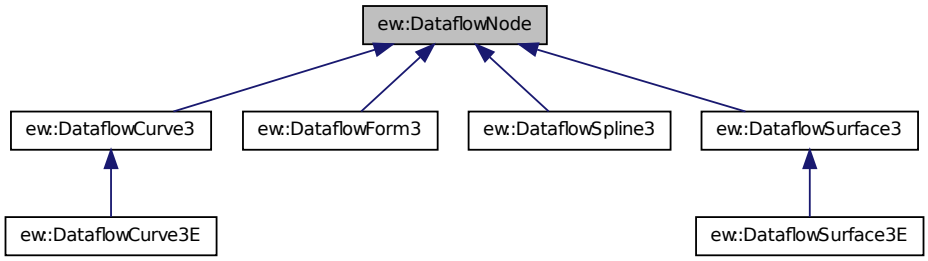
EW::ErrorIO

14.7 ew::DataflowNode Class Reference

Node Base Class.

`#include <ew/DataflowNode.h>`

Inheritance diagram for ew::DataflowNode:



Public Member Functions

- virtual void reset ()=0
- void incr_ref_count () const
- void decr_ref_count () const
- unsigned long get_change_cycle () const
- unsigned long get_version () const
- bool get_valid () const

Public Attributes

- ew::DataflowNetwork ∗const network

14.7.1 Detailed Description

EW::DataflowNode is the base class for nodes in a EW::DataflowNetwork.

EW::DataflowNode does not support assignment or comparison. It contains private members, protected members and undocumented members.

Nodes are referenced counted and can only be deleted indirectly by calling decr_ref_count. The reference count is set to 1 by the constructor.

Depending on its parameters, a node is considered valid or invalid. What constitutes a valid state depends on the specific node class.

A node can depend on other nodes. The possible dependencies are defined by the specific node class. It is a precondition of methods that set these dependencies that they do not create dependency loops.

After the EW::DataflowNetwork has been destroyed, the only EW::DataflowNode method that may be called is decr_ref_count.

14.7.2 Member Function Documentation

14.7.2.1 virtual void ew::DataflowNode::reset ( ) [pure virtual]

This resets a node to the state it had immediately after construction.

14.7.2.2 void ew::DataflowNode::incr_ref_count ( ) const

This increases the reference count.

14.7.2.3 void ew::DataflowNode::decr_ref_count ( ) const

This decreases the reference count. If the reference count becomes zero, the node is deleted.

14.7.2.4 unsigned long ew::DataflowNode::get_change_cycle ( ) const [inline]

Returns

The last cycle on which a change to the node was made.

14.7.2.5 unsigned long ew::DataflowNode::get_version ( ) const [inline]

The version of a node is the greater of the last network cycle that any parameter of the node was changed, and the versions of any nodes that the node depends on. This will force the network into an updating phase.

Returns

The version of the node.

14.7.2.6 bool ew::DataflowNode::get_valid ( ) const [inline]

Returns

`true` if the node is in a valid state.

Postcondition

The network will be in an update phase.

14.7.3 Member Data Documentation

14.7.3.1 ew::DataflowNetwork* const ew::DataflowNode::network

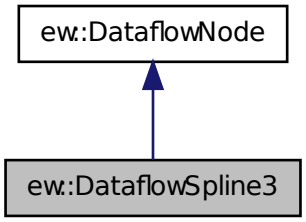
This points to the EW::Dataflow_Network that this EW::Dataflow_Node was created with.

14.8 ew::DataflowSpline3 Class Reference

3D Spline Node

`#include <ew/DataflowSpline3.h>`

Inheritance diagram for ew::DataflowSpline3:



Public Member Functions

- void set_template (const ew::DataflowForm3 *f)
- const ew::DataflowForm3 * get_template () const
- unsigned long get_change_cycle_template () const
- void set_specimen (const ew::DataflowForm3 *f)
- const ew::DataflowForm3 * get_specimen () const

- unsigned long get_change_cycle_specimen () const
- int get_n_lmks () const
- int get_f_size () const
- int lmk_index (int side, int point_set, int i) const
- int lmk_pointset (int side, int lmk_index) const
- int lmk_pointset_i (int side, int lmk_index) const
- bool get_nonsingular () const
- double get_energy () const
- const double * get_optimized_lmk_images () const
- void warp_points (double *outp, const double *inp, int l) const

Private Member Functions

- unsigned long get_version_association () const
- unsigned long get_version_interaction () const
- unsigned long get_version_factorization () const
- unsigned long get_version_spline () const

14.8.1 Detailed Description

ew::DataflowSpline3 is a node that manages a thin-plate spline.

ew::DataflowSpline3 is a class without assignment or comparison. There are private member variables.

Nodes of this class depend on 2 nodes, the template and specimen ew::DataflowForm3 nodes. Nodes representing warped geometric objects will depend on a ew::DataflowSpline3 node. A ew::Dataflow_Spline3 is considered valid if it's template and specimen dependencies have been set and if there are 4 or more associated landmarks.

Initially and when reset, the node has neither template or specimen set.

14.8.2 Member Function Documentation

14.8.2.1 void ew::DataflowSpline3::set_template (const ew::DataflowForm3 * *f*)

Parameters

f A pointer to the new template form node.

14.8.2.2 const ew::DataflowForm3 * ew::DataflowSpline3::get_template ( ) const [inline]

Returns

A pointer to the template node.

14.8.2.3 unsigned long ew::DataflowSpline3::get_change_cycle_template ( ) const [inline]

Returns

The last cycle the template pointer was changed.

14.8.2.4 void ew::DataflowSpline3::set_specimen (const ew::DataflowForm3 * *f*)

Parameters

f A pointer to the new specimen form node.

14.8.2.5 const ew::DataflowForm3 * ew::DataflowSpline3::get_specimen ( ) const [inline]

Returns

A pointer to the specimen node.

14.8.2.6 unsigned long ew::DataflowSpline3::get_change_cycle_specimen ( ) const [inline]

Returns

The last cycle the specimen pointer was changed.

14.8.2.7 int ew::DataflowSpline3::get_n_lmks ( ) const [inline]

Returns

The number of landmarks and semi-landmarks matched between template and specimen.

14.8.2.8 int ew::DataflowSpline3::get_f_size ( ) const [inline]

Returns

The algebraic dimension of the spline, or -1.

14.8.2.9 int ew::DataflowSpline3::lmk_index (int *side*, int *point_set*, int *i*) const [inline]

To construct the spline, landmarks in the template and specimen are matched by id and by position within the point_set. This calculates the index in the spline of a point in a point_set.

Parameters

side 0 if the point is in the template, 1 if it is in the specimen.

point_set The index of the point_set.

i The position of the point within the point_set.

Returns

The index, if the point is matched, otherwise -1.

14.8.2.10 int ew::DataflowSpline3::lmk_pointset (int *side*, int *index*) const [inline]

This is the inverse of lmk_index.

Parameters

side 0 for the pointset in the template, 1 for the pointset in the specimen.

index The index as returned by lmk_index.

Returns

The pointset of the specified landmark in the spline.

14.8.2.11 int ew::DataflowSpline3::lmk_pointset_i (int *side*, int *index*) const [inline]

This is the inverse of lmk_index.

Parameters

side 0 for the pointset in the template, 1 for the pointset in the specimen.

index The index as returned by lmk_index.

Returns

The index of the landmark in the pointset of the specified landmark in the spline.

14.8.2.12 bool ew::DataflowSpline3::get_nonsingular ( ) const [inline]

Returns

`true` if the spline is non-singular.

14.8.2.13 double ew::DataflowSpline3::get_energy ( ) const [inline]

Returns

The bending energy of the spline.

Exceptions

If the spline is singular, a std::runtime_error is thrown.

14.8.2.14 const double * ew::DataflowSpline3::get_optimized_lmk_images ( ) const [inline]

This returns the positions of the landmarks after being allowed to slide in their relaxation spaces to the positions that minimize bending energy. The order of the landmarks is as defined by lmk_index. If the spline is singular, a std::runtime_error is thrown. The pointer is valid until the node or a dependency of the node is changed.

Returns

A pointer to an array of coordinates.

14.8.2.15 void ew::DataflowSpline3::warp_points (double * *outp*, const double * *inp*, int *l*) const

This applies the spline to an array of points. If the spline is singular, a std::runtime_error is thrown.

Parameters

[out] ***outp*** Where to store the warped point coordinates.

[in] ***inp*** Where the original point coordinates are stored.

[in] ***l*** The number of points.

14.8.2.16 unsigned long ew::DataflowSpline3::get_version_association ( ) const [inline, private]

Returns

The last cycle the XXX was changed.

14.8.2.17 unsigned long ew::DataflowSpline3::get_version_interaction ( ) const [inline, private]

Returns

The last cycle the XXX was changed.

14.8.2.18 unsigned long ew::DataflowSpline3::get_version_factorization ( ) const [inline, private]

Returns

The last cycle the XXX was changed.

14.8.2.19 unsigned long ew::DataflowSpline3::get_version_spline ( ) const [inline, private]

Returns

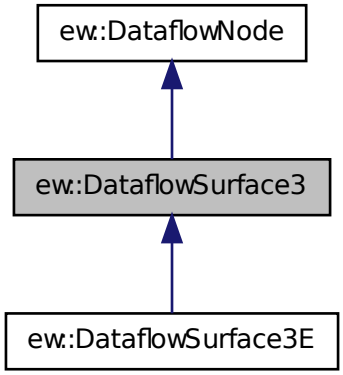
The last cycle the XXX was changed.

14.9 ew::DataflowSurface3 Class Reference

Surface Node Base Class.

`#include <ew/DataflowSurface3.h>`

Inheritance diagram for ew::DataflowSurface3:



Public Member Functions

- int get_num_points () const
- int get_num_faces () const
- const float * get_points () const
- const int * get_faces () const
- const ew::Bbox3 * get_bbox () const
- void make_index () const
- bool index_is_made () const
- double project (int *face, double *coeffs, double *normal, double *proj, const double *inp) const

14.9.1 Detailed Description

ew::DataflowSurface3 is the base class for nodes containing triangulated surfaces in $\mathbb{R}^3$.

ew::DataflowSurface3 is a class without assignment or comparison. There are private member variables.

For efficient rendering of the same surface in multiple windows, an OpenGL display list is constructed containing the basic surface geometry commands. All windows displaying a given surface must share display lists.

14.9.2 Member Function Documentation

14.9.2.1 int ew::DataflowSurface3::get_num_points ( ) const [inline]

This will force the network into an updating phase.

Returns

The number of points used in the surface.

14.9.2.2 int ew::DataflowSurface3::get_num_faces ( ) const [inline]

This will force the network into an updating phase.

Returns

The number of triangular faces in the surface.

14.9.2.3 const float * ew::DataflowSurface3::get_points ( ) const [inline]

This will force the network into an updating phase. The pointer is valid until the node or a dependency of the node is changed.

Returns

A pointer to the coordinates of the points used in the surface.

14.9.2.4 const int * ew::DataflowSurface3::get_faces ( ) const [inline]

This will force the network into an updating phase. The pointer is valid until the node or a dependency of the node is changed.

Returns

A pointer to the indices of the points used in the surface.

14.9.2.5 const ew::Bbox3 * ew::DataflowSurface3::get_bbox ( ) const [inline]

The pointer is valid until the node or a dependency of the node is changed.

Returns

A pointer to the bounding box of the surface.

14.9.2.6 void ew::DataflowSurface3::make_index ( ) const [inline]

This makes the surface spatial index if it has not already been made since the surface was last changed. The spatial index is used by project.

14.9.2.7 bool ew::DataflowSurface3::index_is_made ( ) const [inline]

Returns

`true` if the surface spatial index is up to date.

14.9.2.8 double ew::DataflowSurface3::project (int * *face*, double * *coeffs*, double * *normal*, double * *proj*, const double * *inp*) const

This finds the nearest point on the surface to a given point. The surface spatial index will be made if it has not already been made. This might be time consuming. The timing of this delay can be controlled by calling make_index beforehand.

Parameters

`[out]` ***face*** The index of the face of the surface containing the nearest point.

`[out]` ***coeffs*** The coefficients of the nearest point when expressed as a linear combination of the vertices of the face.

`[out]` ***normal*** The interpolated normal at the nearest point. If a sensible normal cannot be calculated, an arbitrary unit vector is returned.

`[out]` ***proj*** The coordinates of the nearest point on the surface.

[in] ***inp*** The coordinates of the original point.

Returns

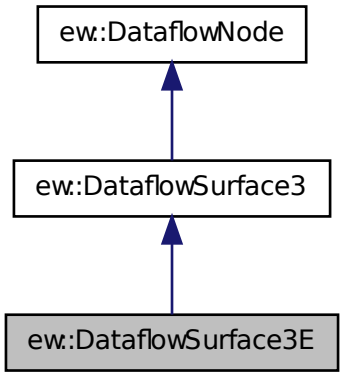
The distance from the original point to the projected point.

14.10 ew::DataflowSurface3E Class Reference

Explicit Surface Node.

`#include <ew/DataflowSurface3E.h>`

Inheritance diagram for ew::DataflowSurface3E:



Public Member Functions

- DataflowSurface3E (ew::DataflowNetwork *i_network)
- void set_data (ew::Surface3 *data)
- const ew::Surface3 * get_data () const

14.10.1 Detailed Description

ew::DataflowSurface3E is a node containing an explicitly specified surface.

ew::DataflowSurface3E objects may not be copied or compared. There are private member variables.

Nodes of this class do not depend on other nodes, but other nodes can depend on them. They are valid if they have some faces.

Initially and when reset, the node is empty.

14.10.2 Constructor & Destructor Documentation

14.10.2.1 ew::DataflowSurface3E::DataflowSurface3E (ew::DataflowNetwork * *i_network*) [explicit]

This creates an explicit surface node.

Parameters

i_network The network this node should belong to.

14.10.3 Member Function Documentation

14.10.3.1 void ew::DataflowSurface3E::set_data (ew::Surface3 * *data*)

This transfers the surface data from the ew::Surface3 object to the node. Afterwards, the ew::Surface3 object is reset.

Precondition

This node must not be a cached node.

Parameters

data The object containing the surface data.

14.10.3.2 const ew::Surface3 * ew::DataflowSurface3E::get_data ( ) const [inline]

Returns

A pointer to the internal surface data.

14.11 ew::Dig3 Class Reference

Digitizing Application.

`#include <ew/Dig3.h>`

Public Types

- enum space_index_t { SPACE_TEMPLATE = 0, SPACE_SPECIMEN = 1 }
- enum tableau_flags_t { TABLEAU_VIEW = 0x01, TABLEAU_SETTINGS = 0x02, TABLEAU_ALL = 0x03 }

Public Member Functions

- Dig3 (ew::DataflowNetwork *network)
- ~Dig3 ()
- int get_n_views () const
- ew::Dig3View *const * get_views () const
- ew::Dig3Space *const * get_spaces () const
- const ew::DataflowSpline3 * get_spline_node () const
- void save_tableau (ew::Dig3Tableau *outp, int template_main, int template_slice, int specimen_main, int specimen_slice)
- void load_tableau (const ew::Dig3Tableau *inp, int view, unsigned int flags)
- void interpolate_tableau (const ew::Dig3Tableau *inp1, const ew::Dig3Tableau *inp2, double e, int view)

Public Attributes

- ew::DataflowNetwork *const network

14.11.1 Detailed Description

ew::Dig3 implements a 3D digitizing application.

ew::Dig3 is a class without assignment or comparison. There are private member variables.

An ew::Dig3 maintains 2 space elements, the template and the specimen.

View elements can be added and removed by creating or destroying ew::Dig3View objects.

14.11.2 Member Enumeration Documentation

14.11.2.1 enum ew::Dig3::space_index_t

These are the indices of the 2 spaces in the array of spaces.

Enumerator:

SPACE_TEMPLATE The index of the template space.

SPACE_SPECIMEN The index of the specimen space.

14.11.2.2 enum ew::Dig3::tableau_flags_t

These are flags that can be or-ed and passed to load_tableau.

Enumerator:

TABLEAU_VIEW Set the view mapping.
TABLEAU_SETTINGS Set the settings.
TABLEAU_ALL Set everything.

14.11.3 Constructor & Destructor Documentation

14.11.3.1 ew::Dig3::Dig3 (ew::DataflowNetwork * *network*) [explicit]

This creates an empty digitizing application.

Parameters

network A pointer to a dataflow network.

14.11.3.2 ew::Dig3::~Dig3 ( )

This destroys the digitizing application. Any ew::Dig3View widgets associated with it are also destroyed.

14.11.4 Member Function Documentation

14.11.4.1 int ew::Dig3::get_n_views ( ) const [inline]

Returns

The number of views.

14.11.4.2 ew::Dig3View *const * ew::Dig3::get_views ( ) const [inline]

Returns

The view index.

14.11.4.3 ew::Dig3Space ∗const ∗ ew::Dig3::get_spaces ( ) const [inline]

Returns

The space index. There are always 2 spaces and the index is constant.

14.11.4.4 const ew::DataflowSpline3 ∗ ew::Dig3::get_spline_node ( ) const [inline]

Returns

A pointer to the spline node that this ew::Dig3 manages.

14.11.4.5 void ew::Dig3::save_tableau (ew::Dig3Tableau ∗ *outp*, int *template_main*, int *template_slice*, int *specimen_main*, int *specimen_slice*)

This creates a tableau record from the indicated views, specified by index. The form filename fields of the tableau are left blank, and must be explicitly set.

Parameters

outp Where to store the tableau record.

template_main The index of a template main view to record.

template_slice The index of a template slice view to record.

specimen_main The index of a specimen main view to record.

specimen_slice The index of a specimen slice view to record.

14.11.4.6 void ew::Dig3::load_tableau (const ew::Dig3Tableau ∗ *inp*, int *view*, unsigned int *flags*)

This applies a tableau record to the indicated view. The type of view and its space determine which part of the record is applied. The form filename fields are not applied. The forms must be explicitly set beforehand.

Parameters

inp The tableau to apply.

view The index of the view to apply it to.

flags Flags of type tableau_flags_t indicating which parts of the tableau record to apply.

14.11.4.7 void ew::Dig3::interpolate_tableau (const ew::Dig3Tableau * *inp1*, const ew::Dig3Tableau * *inp2*, double *e*, int *view*)

This interpolates between tableau records and applies the result to the indicated view. Only the views are applied. The settings of the tableau and the forms cannot be interpolated and need to be manually applied when a new frame is reached in a filmstrip.

Parameters

inp1,inp2 The tableaus to interpolate.

e The interpolation parameter, with 0.0 corresponding to *inp1* and 1.0 to *inp*. *e* is not restricted to [0.0, 1.0].

view The index of the view to apply it to.

14.11.5 Member Data Documentation

14.11.5.1 ew::DataflowNetwork *const ew::Dig3::network

This points to the ew::DataflowNetwork that this ew::Dig3 was created with.

14.12 ew::Dig3SetCurve Class Reference

Curve Element.

`#include <ew/Dig3SetCurve.h>`

Public Member Functions

- Dig3SetCurve ()
- bool operator== (const ew::Dig3SetCurve &a) const
- bool operator!= (const ew::Dig3SetCurve &a) const

Public Attributes

- std::string id
- bool show_in_main
- bool show_in_slice
- unsigned char col [3]

14.12.1 Detailed Description

ew::Dig3SetCurve contains data for a curve element of ew::Tableau.

ew::Dig3SetCurve is a data structure that supports default construction, copy construction, assignment and equality comparison.

14.12.2 Constructor & Destructor Documentation

14.12.2.1 ew::Dig3SetCurve::Dig3SetCurve ( ) [inline]

This sets intrinsic member variables to their default values.

14.12.3 Member Function Documentation

14.12.3.1 bool ew::Dig3SetCurve::operator== (const ew::Dig3SetCurve & *a*) const [inline]

Compares this element with another, member by member.

Parameters

a the other element

14.12.3.2 bool ew::Dig3SetCurve::operator!= (const ew::Dig3SetCurve & *a*) const [inline]

Compares this element with another, member by member.

Parameters

a the other element

14.12.4 Member Data Documentation

14.12.4.1 std::string ew::Dig3SetCurve::id

The ew::Form3Curve::id of the curve this field should apply to.

14.12.4.2 bool ew::Dig3SetCurve::show_in_main

`true` if the curve should be displayed in the main view. The default is `true`.

14.12.4.3 bool ew::Dig3SetCurve::show_in_slice

`true` if the curve should be displayed in the slice view. The default is `true`.

14.12.4.4 unsigned char ew::Dig3SetCurve::col[3]

The colour of the curve. The default is [128, 128, 128].

14.13 ew::Dig3SetSurface Class Reference

Surface Element.

`#include <ew/Dig3SetSurface.h>`

Public Member Functions

- Dig3SetSurface ()
- bool operator== (const ew::Dig3SetSurface &a) const
- bool operator!= (const ew::Dig3SetSurface &a) const

Public Attributes

- std::string id
- bool show_in_main
- bool show_in_slice
- unsigned char front_col [3]
- unsigned char back_col [3]

14.13.1 Detailed Description

ew::Dig3SetSurface contains data for a surface element of ew::Tableau.

ew::Dig3SetSurface is a data structure that supports default construction, copy construction, assignment and equality comparison.

14.13.2 Constructor & Destructor Documentation

14.13.2.1 ew::Dig3SetSurface::Dig3SetSurface ( ) [inline]

This sets intrinsic member variables to their default values.

14.13.3 Member Function Documentation

14.13.3.1 bool ew::Dig3SetSurface::operator== (const ew::Dig3SetSurface & *a*) const [inline]

Compares this element with another, member by member.

Parameters

a the other element

14.13.3.2 bool ew::Dig3SetSurface::operator!= (const ew::Dig3SetSurface & *a*) const [inline]

Compares this element with another, member by member.

Parameters

a the other element

14.13.4 Member Data Documentation

14.13.4.1 std::string ew::Dig3SetSurface::id

The ew::Form3Surface::id of the surface this field should apply to.

14.13.4.2 bool ew::Dig3SetSurface::show_in_main

`true` if the surface should be displayed in the main view. The default is `true`.

14.13.4.3 bool ew::Dig3SetSurface::show_in_slice

`true` if the surface should be displayed in the slice view. The default is `true`.

14.13.4.4 unsigned char ew::Dig3SetSurface::front_col[3]

The colour of the front side of the surface. The default is [128, 128, 128].

14.13.4.5 unsigned char ew::Dig3SetSurface::back_col[3]

The colour of the back side of the surface. The default is [128, 128, 128].

14.14 ew::Dig3Space Class Reference

Space Element.

`#include <ew/Dig3Space.h>`

Public Member Functions

- const ew::DataflowForm3 * get_form_node ()
- void reset_form ()
- void set_form_data (const ew::Form3 *form_data)
- const ew::Form3 * get_form_data () const
- int set_form_curve (bool *replaced, const ew::Form3Curve *cr)
- void remove_form_curve (int n)
- int set_form_surface (bool *replaced, const ew::Form3Surface *sr)
- void remove_form_surface (int n)
- int set_form_pointset (bool *replaced, const ew::Form3PointSet *ps)
- void set_form_pointset_location (int n, int i, const double *loc)
- void set_form_pointset_relax (int n, int i, int rdim, const double *rparam)
- void remove_form_pointset (int n)
- int set_form_embedding (bool *replaced, const ew::Form3Embedding *em)
- void set_form_superset (const char *subset_id, const char *superset_id)
- void remove_form_embedding (int n)
- int get_curve_of_pointset (int ps) const
- int get_surface_of_pointset (int ps) const
- void get_bbox (ew::Bbox3 *b) const
- const ew::DataflowCurve3E *const * get_curve_nodes () const
- const ew::DataflowSurface3E *const * get_surface_nodes () const
- bool project (int *rdim, double *rparam, double *proj, const double *p, const char *id) const

Public Attributes

- ew::DataflowNetwork *const network
- ew::Dig3 *const dig3
- const int index

14.14.1 Detailed Description

ew::Dig3Space implements a space element of ew::Dig3. There are 2 spaces, the template space and the specimen space.

14.14.2 Member Function Documentation

14.14.2.1 const ew::DataflowForm3 * ew::Dig3Space::get_form_node ( ) [inline]

This returns a pointer to the ew::DataflowForm3 node managed by the ew::Dig3Space. This node can be changed using ew::Dig3Space wrappers for the ew::DataflowForm3 methods.

14.14.2.2 void ew::Dig3Space::reset_form ( )

This sets the space to an empty form.

14.14.2.3 void ew::Dig3Space::set_form_data (const ew::Form3 * *form_data*)

This sets the space to a new form.

Parameters

form_data The form data to copy.

14.14.2.4 const ew::Form3 * ew::Dig3Space::get_form_data ( ) const [inline]

Returns

A pointer to the ew::From3 data of the space.

14.14.2.5 int ew::Dig3Space::set_form_curve (bool * *replaced*, const ew::Form3Curve * *cr*)

This adds or replaces a curve in the space.

Parameters

`[out]` ***replaced*** `true` if the curve replaced an existing curve.

cr A pointer to the curve data to copy.

Returns

The index of the curve.

14.14.2.6 void ew::Dig3Space::remove_form_curve (int *n*)

This deletes a curve from the space.

Parameters

n The index of the curve to delete.

14.14.2.7 int ew::Dig3Space::set_form_surface (bool * *replaced,* const ew::Form3Surface * *sr*)

This adds or replaces a surface in the space.

Parameters

`[out]` ***replaced*** `true` if the surface replaced an existing surface.

sr A pointer to the surface data to copy.

Returns

The index of the surface.

14.14.2.8 void ew::Dig3Space::remove_form_surface (int *n*)

This deletes a surface from the space.

Parameters

n The index of the surface to delete.

14.14.2.9 int ew::Dig3Space::set_form_pointset (bool * *replaced,* const ew::Form3PointSet * *ps*) `[inline]`

This adds or replaces a pointset in the space.

Parameters

`[out]` ***replaced*** `true` if the point set replaced an existing point set.

ps A pointer to the point set data to copy.

Returns

The index of the point set.

14.14.2.10 void ew::Dig3Space::set_form_pointset_location (int *n*, int *i*, const double * *loc*) [inline]

This changes the coordinates of an element of the point set.

Parameters

n The index of the point set.

i The index of the element in the point set.

[in] ***loc*** A pointer to the new coordinates of this entry.

14.14.2.11 void ew::Dig3Space::set_form_pointset_relax (int *n*, int *i*, int *rdim*, const double * *rparam*) [inline]

This changes the relaxation parameters of an element of the point set.

Parameters

n The index of the point set.

i The index of the element in the point set.

[in] ***rdim*** The new relax_dims of this entry.

[in] ***rparam*** A pointer to the new relax_params of this entry.

14.14.2.12 void ew::Dig3Space::remove_form_pointset (int *n*) [inline]

This deletes a point set from the space.

Parameters

n The index of the point set to delete.

14.14.2.13 int ew::Dig3Space::set_form_embedding (bool * *replaced*, const ew::Form3Embedding * *em*) [inline]

This adds or replaces a embedding in the space.

Parameters

[out] ***replaced*** `true` if the embedding replaced an existing embedding.

em A pointer to the embedding data to copy.

Returns

The index of the embedding.

14.14.2.14 void ew::Dig3Space::set_form_superset (const char * *subset_id*, const char * *superset_id*) [inline]

This makes one element of the form the unique superset of another element.

Parameters

subset_id The id of the element that should have a unique superset.

superset_id The id of the element that be the unique superset.

14.14.2.15 void ew::Dig3Space::remove_form_embedding (int *n*) [inline]

This deletes a embedding from the space.

Parameters

n The index of the embedding to delete.

14.14.2.16 int ew::Dig3Space::get_curve_of_pointset (int *ps*) const

This finds the curve a pointset is embedded in if any.

Parameters

ps The index of the pointset.

Returns

The index of the curve, or -1.

14.14.2.17 int ew::Dig3Space::get_surface_of_pointset (int *ps*) const

This finds the surface a pointset is embedded in if any.

Parameters

ps The index of the pointset.

Returns

The index of the surface, or -1.

14.14.2.18 void ew::Dig3Space::get_bbox (ew::Bbox3 * *b*) const

This calculates the bounding box of all elements of the space's form.

Parameters

[out] ***b*** Where to store the bounding box.

14.14.2.19 const ew::DataflowCurve3E *const * ew::Dig3Space::get_curve_nodes ( ) const [inline]

Returns

The curve node index. The i'th entry is a pointer to the curve node corresponding to the i'th curve, or zero if this curve has no data.

14.14.2.20 const ew::DataflowSurface3E *const * ew::Dig3Space::get_surface_nodes ( ) const [inline]

Returns

The surface node index. The i'th entry is a pointer to the surface node corresponding to the i'th surface, or zero if this surface has no data.

14.14.2.21 bool ew::Dig3Space::project (int * *rdim*, double * *rparam*, double * *proj*, const double * *p*, const char * *id*) const

This projects a point onto either a curve or surface.

Parameters

[out] ***rdim*** Where to store the relax_dim of the projection.

[out] ***rparam*** Where to store the relax_params of the projection. This should point to an array of size 3.

[out] ***proj*** Where to store the coordinates of the projected point. This should point to an array of size 3.

[in] ***p*** The coordinates of the original point. This should point to an array of size 3.

[in] ***id*** The id of the surface or curve to project onto.

Returns

Whether the operation could be performed. This is false if there is no surface or curve with the given id, or if this surface or curve is a placeholder (has no filename).

14.14.3 Member Data Documentation

14.14.3.1 ew::DataflowNetwork *const ew::Dig3Space::network

This points to the ew::DataflowNetwork that the dig3 was created with.

14.14.3.2 ew::Dig3 *const ew::Dig3Space::dig3

This points to the ew::Dig3 that this ew::Dig3Space is contained in.

14.14.3.3 const int ew::Dig3Space::index

This is 0 if this is the template space of dig3, 1 if the specimen space.

14.15 ew::Dig3Tableau Class Reference

Viewing State Record.

`#include <ew/Dig3Tableau.h>`

Public Member Functions

- Dig3Tableau ()

  This sets intrinsic member variables to their default values.

- void reset ()
- bool operator!= (const ew::Dig3Tableau &a) const

Static Public Member Functions

- static void read_file (std::vector< ew::Dig3Tableau > *outp, const char *file)
- static void write_file (const char *file, bool compress, const std::vector< ew::Dig3Tableau > *inp)

Public Attributes

- ew::Dig3TableauSpace space [2]
- double slice_clip_ratio
- unsigned char bg [3]

14.15.1 Detailed Description

ew::Dig3Tableau contains a partial record of the state of an ew::Dig3.

ew::Dig3Tableau is a data structure that supports default construction, copy construction, assignment and equality comparison.

ew::Dig3Tableau records the ew::Dig3 space forms indirectly as file names. It does not record the ew::Dig3 views, their widgets or their linkages. It records the settings of one each of:

- a template main view
- a template slice view
- a specimen main view
- a specimen slice view

Some settings will be assumed to be common to some or all of the windows. It does not record state that is not controlled by the library, such as the window sizes, file browser history or transient state such as the selected item, window configuration, or gui modal state.

ew::Dig3Tableau lists are represented in save files read by read_file and write_file. Here is an example:

- example.sav

The representation is exact except that filenames in the form files are abbreviated to relative filenames in common situations were possible, but in ew::Dig3Tableau they are always in canonical form.

14.15.2 Member Function Documentation

14.15.2.1 void ew::Dig3Tableau::reset ( )

This resets the object to its initial state.

14.15.2.2 void ew::Dig3Tableau::read_file (std::vector< ew::Dig3Tableau > * *outp,* const char * *file*) [static]

This reads a save file. The file can be compressed with gzip compression.

Parameters

[in] ***outp*** The vector to store the records read in.

file The file name.

Exceptions

std::bad_alloc

ew::IOError

14.15.2.3 void ew::Dig3Tableau::write_file (const char * *file,* bool *compress,* const std::vector< ew::Dig3Tableau > * *outp*) [static]

This writes a save file.

Parameters

file The file name.

compress If `true`, write the file compressed with gzip compression.

[in] ***outp*** The vector of records to write.

Exceptions

std::bad_alloc

EW::ErrorIO

14.15.2.4 bool ew::Dig3Tableau::operator!= (const ew::Dig3Tableau & *a*) const [inline]

Compares this record with another, member by member.

Parameters

a the other record

14.15.3 Member Data Documentation

14.15.3.1 ew::Dig3TableauSpace ew::Dig3Tableau::space[2]

The template and specimen space states.

14.15.3.2 double ew::Dig3Tableau::slice_clip_ratio

The slice clip ratio. The default is 0.01.

14.15.3.3 unsigned char ew::Dig3Tableau::bg[3]

The window background. The default is [0, 0, 0].

14.16 ew::Dig3TableauSpace Class Reference

Space Element.

`#include <ew/Dig3TableauSpace.h>`

Public Member Functions

- Dig3TableauSpace ()
- bool operator== (const ew::Dig3TableauSpace &a) const
- bool operator!= (const ew::Dig3TableauSpace &a) const

Public Attributes

- std::string form_filename
- ew::Transform3 main_view
- ew::Transform3 slice_view
- bool show_slice_in_main
- bool show_lmks_in_main
- bool show_lmks_in_slice
- int lmks_symbol
- unsigned char lmks_col [3]
- std::vector< ew::Dig3SetCurve > curve_settings
- std::vector< ew::Dig3SetSurface > surface_settings

14.16.1 Detailed Description

ew::Dig3TableauSpace contains data for a space element of ew::Dig3Tableau.

ew::Dig3TableauSpace is a data structure that supports default construction, copy construction, assignment and equality comparison.

14.16.2 Constructor & Destructor Documentation

14.16.2.1 ew::Dig3TableauSpace::Dig3TableauSpace ( ) [inline]

This sets intrinsic and POD member variables to their default values.

14.16.3 Member Function Documentation

14.16.3.1 bool ew::Dig3TableauSpace::operator== (const ew::Dig3TableauSpace & *a*) const [inline]

Compares this element with another, member by member.

Parameters

a the other element

14.16.3.2 bool ew::Dig3TableauSpace::operator!= (const ew::Dig3TableauSpace & *a*) const [inline]

Compares this element with another, member by member.

Parameters

a the other element

14.16.4 Member Data Documentation

14.16.4.1 std::string ew::Dig3TableauSpace::form_filename

This file containing the form data for the space.

14.16.4.2 ew::Transform3 ew::Dig3TableauSpace::main_view

The view mapping of the main view of this space. The default is the identity transformation.

14.16.4.3 ew::Transform3 ew::Dig3TableauSpace::slice_view

The view mapping of the slice view of this space. The default is the identity transformation.

14.16.4.4 bool ew::Dig3TableauSpace::show_slice_in_main

`true` if the slice should be displayed in the main view. The default is `true`.

14.16.4.5 bool ew::Dig3TableauSpace::show_lmks_in_main

`true` if the landmarks should be displayed in the main view. The default is `true`.

14.16.4.6 bool ew::Dig3TableauSpace::show_lmks_in_slice

`true` if the landmarks should be displayed in the slice view. The default is `true`.

14.16.4.7 int ew::Dig3TableauSpace::lmks_symbol

The symbol to use for landmarks. The default is 0.

14.16.4.8 unsigned char ew::Dig3TableauSpace::lmks_col[3]

The colour of the landmarks. The default is [0, 255, 0].

14.16.4.9 std::vector< ew::Dig3SetCurve > ew::Dig3TableauSpace::curve_settings

The settings for curves in the space.

14.16.4.10 std::vector< ew::Dig3SetSurface > ew::Dig3TableauSpace::surface_settings

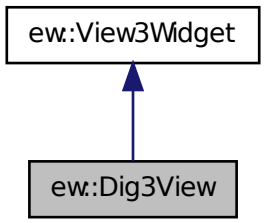
The settings for surfaces in the space.

14.17 ew::Dig3View Class Reference

View Element.

`#include <ew/Dig3View.h>`

Inheritance diagram for ew::Dig3View:



Public Types

- enum item_t { ITEM_SLICE, ITEM_LANDMARKS, ITEM_CURVE, ITEM_SURFACE }

Public Member Functions

- Dig3View (ew::Dig3 *dig3, int space, int slice_mode)
- ~Dig3View ()
- void set_slice_mode (int sm)
- int get_slice_mode () const
- void set_space (int sp)
- int get_space () const
- void set_link (ew::Dig3View *v, const ew::Transform3 *tr)
- void set_view_mapping (const ew::Transform3 *tr)
- void view_post_compose (const ew::Transform3 *tr)
- ew::View3Image *const * get_slice_items () const
- ew::View3Curve *const * get_curve_items () const
- ew::View3Surface *const * get_surface_items () const
- ew::View3Landmarks * get_landmarks_item () const
- int get_index () const

Public Attributes

- ew::Dig3 *const dig3

14.17.1 Detailed Description

ew::Dig3View implements a view element of ew::Dig3.

ew::Dig3View is a class without assignment or comparison. There are private member variables.

The settings and item settings of the base ew::View3Widget, may be directly manipulated, except for the view mapping. The toolkit interaction of this widget should be handled by inheriting this class and implementing the pure virtual methods of ew::View3Widget.

14.17.2 Member Enumeration Documentation

14.17.2.1 enum ew::Dig3View::item_t

These are the possible values for ew::View3Item::type for items created by this class.

Enumerator:

ITEM_SLICE Item that represents the primary slice.

ITEM_LANDMARKS Item that represents form landmarks and semi-landmarks.

ITEM_CURVE Item that represents form (note warped) curves.

ITEM_SURFACE Item that represents form (note warped) surfaces.

14.17.3 Constructor & Destructor Documentation

14.17.3.1 ew::Dig3View::Dig3View (ew::Dig3 * *dig3,* int *space,* int *slice_mode*)

This creates a view element of the *dig3*.

Parameters

dig3 The ew::Dig3 this view element will belong to.

space Which space of the *dig3* this view element will initially display. Either 0 or 1.

slice_mode Whether this will be a slice view. Either 0 or 1.

14.17.3.2 ew::Dig3View::~Dig3View ( )

This removes the view from dig3, and destroys it.

14.17.4 Member Function Documentation

14.17.4.1 void ew::Dig3View::set_slice_mode (int *sm*)

Parameters

sm Whether this view should now be a slice view. Either 0 or 1.

14.17.4.2 int ew::Dig3View::get_slice_mode ( ) const [inline]

Returns

Whether this is a slice view. Either 0 or 1.

14.17.4.3 void ew::Dig3View::set_space (int *sp*)

Parameters

sp The index of the space this view should now be associated with. Either 0 or 1.

14.17.4.4 int ew::Dig3View::get_space ( ) const [inline]

Returns

The index of the space this view is currently associated with. Either 0 or 1.

14.17.4.5 void ew::Dig3View::set_link (ew::Dig3View * *v*, const ew::Transform3 * *tr*)

This sets up a link between the view mapping of this view and of another one. Links must be disjoint cycles. The composition of the transforms should be the identity for sensible results.

Parameters

v The view to link to.

tr The transform that should be left-composed with this view's view mapping to get the link views view mapping.

14.17.4.6 void ew::Dig3View::set_view_mapping (const ew::Transform3 * *tr*)

This operates on the view mapping of the base ew::View3Widget, and synchronizes with other views and with slices as necessary. This method has the same name as a non-virtual method in the base class.

Parameters

tr As for ew::View3Widget::set_view_mapping().

Reimplemented from ew::View3Widget.

14.17.4.7 void ew::Dig3View::view_post_compose (const ew::Transform3 * *tr*)

This operates on the view mapping of the base ew::View3Widget, and synchronizes with other views and with slices as necessary. This method has the same name as a non-virtual method in the base class.

Parameters

tr As for ew::View3Widget::view_post_compose().

Reimplemented from ew::View3Widget.

14.17.4.8 ew::View3Image *const * ew::Dig3View::get_slice_items ( ) const [inline]

Returns

The slice item index. The i'th entry is the item displaying the slice of the i'th view of the dig3, or zero if the i'th view is not in slice mode, or is not associated with the same space.

14.17.4.9 ew::View3Curve *const * ew::Dig3View::get_curve_items ( ) const [inline]

Returns

The curve item index. The i'th entry is the item displaying the i'th curve of the space this view is currently associated with.

14.17.4.10 ew::View3Surface *const * ew::Dig3View::get_surface_items ( ) const [inline]

Returns

The surface item index. The i'th entry is the item displaying the i'th surface of the space this view is currently associated with.

14.17.4.11 ew::View3Landmarks * ew::Dig3View::get_landmarks_item ( ) const [inline]

Returns

The landmarks item.

14.17.4.12 int ew::Dig3View::get_index ( ) const [inline]

Returns

The current index of this view element in the view index of dig3.

14.17.5 Member Data Documentation

14.17.5.1 ew::Dig3 *const ew::Dig3View::dig3

This points to the ew::Dig3 that this ew::Dig3View was created with.

14.18 ew::ErrorIO Class Reference

I/O Exception.

`#include <ew/ErrorIO.h>`

Inherits std::runtime_error.

Public Member Functions

- ErrorIO (const std::string &desc)

14.18.1 Detailed Description

ew::ErrorIO is an exception thrown by EW library classes to report input or output errors or bad file formats.

ew::ErrorIO is a trivial extension of std::runtime_error.

14.18.2 Constructor & Destructor Documentation

14.18.2.1 ew::ErrorIO::ErrorIO (const std::string & *desc*) [explicit]

Construct an ew::ErrorIO object.

Parameters

desc the error description

14.19 ew::ErrorRuntime Class Reference

Runtime Exception.

`#include <ew/ErrorRuntime.h>`

Inherits std::runtime_error.

Public Member Functions

- ErrorRuntime (const std::string &desc)

14.19.1 Detailed Description

ew::ErrorRuntime is an exception thrown by EW library classes to report runtime errors.

ew::ErrorRuntime is a trivial extension of std::runtime_error.

14.19.2 Constructor & Destructor Documentation

14.19.2.1 ew::ErrorRuntime::ErrorRuntime (const std::string & *desc*) [explicit]

Construct an EW::Error_Runtime object.

Parameters

desc the error description

14.20 ew::Form3 Class Reference

Morphometric Form.

`#include <ew/Form3.h>`

Public Types

- enum state_t {
  STATE_SET, STATE_UNSET, STATE_PROVISIONAL, STATE_WARPED,
  STATE_PROJECTED, STATE_OPTIMIZED }
- enum point_t {
  TYPE_LANDMARK, TYPE_SEMI_LANDMARK, TYPE_POINT, TYPE_-
  LINE,
  TYPE_PLANE, TYPE_FRAME }

Public Member Functions

- void reset ()
- void read_file (const char *file)
- void write_file (const char *file, bool compress) const
- bool search_volume (int *position, const char *id) const
- bool search_surface (int *position, const char *id) const
- bool search_curve (int *position, const char *id) const
- bool search_pointset (int *position, const char *id) const
- bool search_embedding (int *position, const char *id1, const char *id2) const
- bool search_reflection (int *position, const char *id1, const char *id2) const
- const char * search_superset (const char *id) const
- void set_superset (const char *subset_id, const char *superset_id)
- void swap (ew::Form3 &f)
- bool operator== (const ew::Form3 &a) const
- bool operator!= (const ew::Form3 &a) const

Public Attributes

- std::vector< ew::Form3Volume > volumes
- std::vector< ew::Form3Surface > surfaces
- std::vector< ew::Form3Curve > curves
- std::vector< ew::Form3PointSet > pointsets
- std::vector< ew::Form3Embedding > embeddings
- std::vector< ew::Form3Reflection > reflections

14.20.1 Detailed Description

ew::Form3 represents morphometric forms in $\mathbb{R}^3$. These are landmark configurations, possibly including semi-landmarks, generalized to include curve, surface and volume data.

ew::Form3 is a data structure that supports default construction, copy construction, assignment and equality comparison.

ew::Form3 data is represented in form files read by read_file and written by write_file. These are example files:

- simple_landmarks.form
- mandible_landmarks.form
- mandible_case.form

The representation is exact except that filenames in the form files are abbreviated to relative filenames in common situations were possible, but in ew::Form3 they are always in canonical form.

All elements in the form have an id. These id's must be unique in the from.

14.20.2 Member Enumeration Documentation

14.20.2.1 enum ew::Form3::state_t

Some elements can have a state associated with them. These are the possible values. Not all states are appropriate for all element types. The default is STATE_SET.

Enumerator:

STATE_SET The element has been explicitly digitized (default).

STATE_UNSET The element has not been digitized.

STATE_PROVISIONAL The element has been provisionally digitized.

STATE_WARPED The element is the image under a warp.

STATE_PROJECTED The element has been projected.

STATE_OPTIMIZED The element has been optimized.

14.20.2.2 enum ew::Form3::point_t

Point sets can have different types. These are the possible values. The default is TYPE_LANDMARK.

Enumerator:

TYPE_LANDMARK The points are landmarks (default).

TYPE_SEMI_LANDMARK The points are semi-landmarks.

TYPE_POINT The point is a non-landmark point.

TYPE_LINE The point represents a line, such as a best-fit axis of symmetry.

TYPE_PLANE The point represents a plane, such as a best-fit multi-tangent or best-fit symmetry plane.

TYPE_FRAME The point represents a frame, such as a standard view.

14.20.3 Member Function Documentation

14.20.3.1 void ew::Form3::reset ( )

This resets the form to its initial state.

14.20.3.2 void ew::Form3::read_file (const char * *file*)

This reads a form from a file. The file must be a real filename, URL's are not supported. The file can be compressed with gzip compression. Relative filenames with initial path component "." inside the file are converted to canonical filenames using the directory part of *file*. If an exception is thrown, the form is unchanged.

Parameters

file The canonical name of the file to read.

Exceptions

std::bad_alloc

ew::ErrorIO

14.20.3.3 void ew::Form3::write_file (const char * *file*, bool *compress*) const

This writes a form to a file. Filenames written into the output are converted to relative filenames with initial path component "." if their path contains the directory part of *file* as an initial segment.

Parameters

file The canonical name of the file to write.

compress If `true`, write the file compressed with gzip compression.

Exceptions

std::bad_alloc

ew::ErrorIO

14.20.3.4 bool ew::Form3::search_volume (int * *position*, const char * *id*) const

This searches the volumes in the form for an id.

Parameters

`[out]` ***position*** The index in the vector that a volume with this id would be if inserted into the vector of volumes.

id The id to search for.

Returns

`true` if the id is an existing volume id.

14.20.3.5 bool ew::Form3::search_surface (int * *position*, const char * *id*) const

This searches the surfaces in the form for an id.

Parameters

`[out]` ***position*** The index in the vector that a surface with this id would be if inserted into the vector of surfaces.

id The id to search for.

Returns

`true` if the id is an existing surface id.

14.20.3.6 bool ew::Form3::search_curve (int * *position*, const char * *id*) const

This searches the curves in the form for an id.

Parameters

[out] ***position*** The index in the vector that a curve with this id would be if inserted into the vector of curves.

id The id to search for.

Returns

`true` if the id is an existing curve id.

14.20.3.7 bool ew::Form3::search_pointset (int * *position*, const char * *id*) const

This searches the point sets in the form for an id.

Parameters

[out] ***position*** The index in the vector that a point set with this id would be if inserted into the vector of point sets.

id The id to search for.

Returns

`true` if the id is an existing point set id.

14.20.3.8 bool ew::Form3::search_embedding (int * *position*, const char * *id1*, const char * *id2*) const

This searches the embeddings in the form for the id's.

Parameters

[out] ***position*** The index in the vector that a embedding with these id's would be if inserted into the vector of embeddings.

id1 The subset_id to search for.

id2 The superset_id to search for.

Returns

`true` if the id's belong to an existing embedding.

14.20.3.9 bool ew::Form3::search_reflection (int * *position*, const char * *id1*, const char * *id2*) const

This searches the reflections in the form for the id's.

Parameters

[out] ***position*** The index in the vector that a reflection with these id's would be if inserted into the vector of reflections.

id1 The left_id to search for.

id2 The right_id to search for.

Returns

`true` if the id's belong to an existing reflection.

14.20.3.10 const char * ew::Form3::search_superset (const char * *id*) const

This searches the embedding relations to see what, if any, element is a unique superset of a given element.

Parameters

[in] ***id*** The id of the original element.

Returns

The id of the unique superset of the original element, or 0. This pointer is valid until the form is changed.

14.20.3.11 void ew::Form3::set_superset (const char * *subset_id*, const char * *superset_id*)

This makes one element the unique superset of another element.

Parameters

subset_id The id of the element that should have a unique superset.

superset_id The id of the element that be the unique superset.

14.20.3.12 void ew::Form3::swap (ew::Form3 & *that*) [inline]

This swaps the data of 2 forms without any copying.

Parameters

that The form to swap with `this`.

14.20.3.13 bool ew::Form3::operator== (const ew::Form3 & *a*) const

Compares this form with another, member by member.

Parameters

a the other form

14.20.3.14 bool ew::Form3::operator!= (const ew::Form3 & *a*) const [inline]

Compares this form with another, member by member.

Parameters

a the other form

14.20.4 Member Data Documentation

14.20.4.1 std::vector< ew::Form3Volume > ew::Form3::volumes

An arbitrary number of volume elements. The volume data is stored in separate files. The files named are not expected to change during their use. These must be in alphabetical order by ew::Form3Volume::id in the POSIX sorting order.

14.20.4.2 std::vector< ew::Form3Surface > ew::Form3::surfaces

An arbitrary number of surface elements. The surface data is stored in separate files. The files named are not expected to change during their use. These must be in alphabetical order by ew::Form3Surface::id in the POSIX sorting order.

14.20.4.3 std::vector< ew::Form3Curve > ew::Form3::curves

An arbitrary number of curve elements. The curve data is stored in separate files. The files named can be created during the digitization, so can change during their use. These must be in alphabetical order by ew::Form3Curve::id in the POSIX sorting order.

14.20.4.4 std::vector< ew::Form3PointSet > ew::Form3::pointsets

An arbitrary number of point set elements. These must be in alphabetical order by id in the POSIX sorting order.

14.20.4.5 std::vector< ew::Form3Embedding > ew::Form3::embeddings

An arbitrary number of individual embedding relationships. These must be in alphabetical order by ew::Form3Embedding::subset_id, then ew::Form3Embedding::superset_id in the POSIX sorting order. They must refer to elements of the form of appropriate dimension. This is how the curve or surface a semi-landmark belongs to is indicated. The semi-landmarks must be embedded in a unique curve or surface of the appropriate dimension. Regular landmarks can be embedded in one or more curves or surfaces.

14.20.4.6 std::vector< ew::Form3Reflection > ew::Form3::reflections

An arbitrary number of individual reflection relationships. These must be in alphabetical order by ew::Form3Reflection::left_id, then ew::Form3Reflection::right_id in the POSIX sorting order. They must refer to elements of the form of appropriate dimension.

14.21 ew::Form3Curve Class Reference

Curve Element.

`#include <ew/Form3Curve.h>`

Public Member Functions

- Form3Curve ()
- bool operator== (const ew::Form3Curve &a) const
- bool operator!= (const ew::Form3Curve &a) const

Public Attributes

- std::string id
- std::string file
- int state

14.21.1 Detailed Description

ew::Form3Curve contains data for a curve element of ew::Form3.

ew::Form3Curve is a data structure that supports default construction, copy construction, assignment and equality comparison.

14.21.2 Constructor & Destructor Documentation

14.21.2.1 ew::Form3Curve::Form3Curve ( ) [inline]

This sets intrinsic member variables to their default values.

14.21.3 Member Function Documentation

14.21.3.1 bool ew::Form3Curve::operator== (const ew::Form3Curve & *a*) const [inline]

Compares this element with another, member by member.

Parameters

a the other element

14.21.3.2 bool ew::Form3Curve::operator!= (const ew::Form3Curve & *a*) const [inline]

Compares this element with another, member by member.

Parameters

a the other element

14.21.4 Member Data Documentation

14.21.4.1 std::string ew::Form3Curve::id

The id of this element used as a key to refer to this element and as a label.

14.21.4.2 std::string ew::Form3Curve::file

The canonical name of the file containing the data for this element, or an empty string indicating that there is no associated file.

14.21.4.3 int ew::Form3Curve::state

A code indicating the digitizing state of the curve. Its value is a ew::Form3::state_t. The default is 0.

14.22 ew::Form3Embedding Class Reference

Embedding Relation.

`#include <ew/Form3Embedding.h>`

Public Member Functions

- bool operator== (const ew::Form3Embedding &a) const
- bool operator!= (const ew::Form3Embedding &a) const

Public Attributes

- std::string subset_id
- std::string superset_id

14.22.1 Detailed Description

ew::Form3Embedding contains data for an embedding relation of ew::Form3. It is used to indicate subset relationships between point, curve and surface elements.

ew::Form3Embedding is a data structure that supports default construction, copy construction, assignment and equality comparison.

14.22.2 Member Function Documentation

14.22.2.1 bool ew::Form3Embedding::operator== (const ew::Form3Embedding & *a*) const [inline]

Compares this element with another, member by member.

Parameters

a the other element

14.22.2.2 bool ew::Form3Embedding::operator!= (const ew::Form3Embedding & *a*) const [inline]

Compares this element with another, member by member.

Parameters

a the other element

14.22.3 Member Data Documentation

14.22.3.1 std::string ew::Form3Embedding::subset_id

The id of the lower dimensional element of the embedding.

14.22.3.2 std::string ew::Form3Embedding::superset_id

The id of the higher dimensional element of the embedding.

14.23 ew::Form3PointSet Class Reference

Point Set Element.

`#include <ew/Form3Pointset.h>`

Public Member Functions

- Form3PointSet ()
- bool operator== (const ew::Form3PointSet &a) const
- bool operator!= (const ew::Form3PointSet &a) const

Public Attributes

- std::string id
- int type
- int state
- int n
- std::vector< double > locations
- std::vector< int > relax_dims
- std::vector< double > relax_params
- std::vector< double > orientations
- std::vector< double > sizes

14.23.1 Detailed Description

ew::Form3PointSet contains data for a point set element of ew::Form3. This could be, for example, a single named landmark, a set of semi-landmarks or a set of landmarks which are not individually named. Also supported are features not intended to be interpreted as landmarks, such as non-landmark points or planes and frames which are represented as a point with a size and orientation.

ew::Form3PointSet is a data structure that supports default construction, copy construction, assignment and equality comparison.

It is intended that landmarks be represented one to a point set. The use of the term "point set" in this context might be confusing. The "set" part of this is only really useful for semi-landmarks. Note that all elements of the point set share the same id, type and state.

14.23.2 Constructor & Destructor Documentation

14.23.2.1 ew::Form3PointSet::Form3PointSet ( ) [inline]

This sets intrinsic member variables to their default values.

14.23.3 Member Function Documentation

14.23.3.1 bool ew::Form3PointSet::operator== (const ew::Form3PointSet & *a*) const [inline]

Compares this element with another, member by member.

Parameters

a the other element

14.23.3.2 bool ew::Form3PointSet::operator!= (const ew::Form3PointSet & *a*) const [inline]

Compares this element with another, member by member.

Parameters

a the other element

14.23.4 Member Data Documentation

14.23.4.1 std::string ew::Form3PointSet::id

The id of this element used as a key to refer to this element and as a label.

14.23.4.2 int ew::Form3PointSet::type

A code indicating the type of point set. Its value is a ew::Form3::point_t. The default is 0.

14.23.4.3 int ew::Form3PointSet::state

A code indicating the digitizing state of the point set. Its value is a ew::Form3::state_t. The default is 0.

14.23.4.4 int ew::Form3PointSet::n

The number of points in the set. It must be non-negative.

14.23.4.5 std::vector< double > ew::Form3PointSet::locations

The coordinates of the points. If all the coordinates are 0.0, this vector can have size 0. Otherwise it must have size $n * 3$.

14.23.4.6 std::vector< int > ew::Form3PointSet::relax_dims

The relax dimensions of the points in the sense of ew::Tps3. These are only relevant for points of type "landmark" or "semi-landmark". They indicate the dimension of the linear space that the landmark or semi-landmark can slide along. A value of 0 indicates that landmark or semi-landmark should not slide. A value of 3 indicates that landmark or semi-landmark slide in the whole of $\mathbb{R}^3$. In this case, the landmark or semi-landmark does contribute to the spline and its optimized image is just the image of the landmark or semi-landmark under the spline. Values of 1 and 2 are only applicable to semi-landmarks. A value of 1 indicates that the semi-landmark slides along a line. This is appropraite for a curve semi-landmark. A value of 2 indicates that the semi-landmark slides along a plane. This is appropraite for a surface emi-landmark. If all the relax dimensions are 0, this vector can have size 0. Otherwise it must have size n.

14.23.4.7 std::vector< double > ew::Form3PointSet::relax_params

The relax parameters of the points in the sense of ew::Tps3. These are only relevant for points of type "semi-landmark". They encode the direction or directions in which the semi-landmark is currently being slid. If all the relax dimensions are 0 or 3, this vector can have size 0. Otherwise it must have size n ∗ 3.

14.23.4.8 std::vector< double > ew::Form3PointSet::orientations

The orientations of the points, orthogonal matrices like ew::Transform3::orthog. These are only relevant for points of type "line", "plane" or "frame". If all the matrices are identities, this vector can have size 0. Otherwise it must have size n ∗ 9.

14.23.4.9 std::vector< double > ew::Form3PointSet::sizes

The sizes of the points, like ew::Transform3::scale. These are only relevant for points of type "line", "plane" or "frame". If all the scales are 1.0, this vector must can size 0. Otherwise it must have size n.

14.24 ew::Form3Reflection Class Reference

Reflection Relation.

`#include <ew/Form3Reflection.h>`

Public Member Functions

- bool operator== (const ew::Form3Reflection &a) const
- bool operator!= (const ew::Form3Reflection &a) const

Public Attributes

- std::string left_id
- std::string right_id

14.24.1 Detailed Description

ew::Form3Reflection contains data for a reflection relation of ew::Form3. Only symmetries where one element is on the left and one on the right, or self symmetries for elements symbolically in the center plane, are supported.

ew::Form3Reflection is a data structure that supports default construction, copy construction, assignment and equality comparison.

14.24.2 Member Function Documentation

14.24.2.1 bool ew::Form3Reflection::operator== (const ew::Form3Reflection & *a*) const [inline]

Compares this element with another, member by member.

Parameters

a the other element

14.24.2.2 bool ew::Form3Reflection::operator!= (const ew::Form3Reflection & *a*) const [inline]

Compares this element with another, member by member.

Parameters

a the other element

14.24.3 Member Data Documentation

14.24.3.1 std::string ew::Form3Reflection::left_id

The id of the left element of the symmetry.

14.24.3.2 std::string ew::Form3Reflection::right_id

The id of the right element of the symmetry.

14.25 ew::Form3Surface Class Reference

Surface Element.

`#include <ew/Form3Surface.h>`

Public Member Functions

- bool operator== (const ew::Form3Surface &a) const
- bool operator!= (const ew::Form3Surface &a) const

Public Attributes

- std::string id
- std::string file

14.25.1 Detailed Description

ew::Form3Surface contains data for a surface element of ew::Form3.

ew::Form3Surface is a data structure that supports default construction, copy construction, assignment and equality comparison.

14.25.2 Member Function Documentation

14.25.2.1 bool ew::Form3Surface::operator== (const ew::Form3Surface & *a*) const [inline]

Compares this element with another, member by member.

Parameters

a the other element

14.25.2.2 bool ew::Form3Surface::operator!= (const ew::Form3Surface & *a*) const [inline]

Compares this element with another, member by member.

Parameters

a the other element

14.25.3 Member Data Documentation

14.25.3.1 std::string ew::Form3Surface::id

The id of this element used as a key to refer to this element and as a label.

14.25.3.2 std::string ew::Form3Surface::file

The canonical name of the file containing the data for this element. An empty string indicates that there is no associated file.

14.26 ew::Form3Volume Class Reference

Volume Element.

`#include <ew/Form3Volume.h>`

Public Member Functions

- bool operator== (const ew::Form3Volume &a) const
- bool operator!= (const ew::Form3Volume &a) const

Public Attributes

- std::string id
- std::string file

14.26.1 Detailed Description

ew::Form3Volume contains data for a volume element of ew::Form3.

ew::Form3Volume is a data structure that supports default construction, copy construction, assignment and equality comparison.

14.26.2 Member Function Documentation

14.26.2.1 bool ew::Form3Volume::operator== (const ew::Form3Volume & *a*) const [inline]

Compares this element with another, member by member.

Parameters

a the other element

14.26.2.2 bool ew::Form3Volume::operator!= (const ew::Form3Volume & *a*) const [inline]

Compares this element with another, member by member.

Parameters

a the other element

14.26.3 Member Data Documentation

14.26.3.1 std::string ew::Form3Volume::id

The id of this element used as a key to refer to this element and as a label.

14.26.3.2 std::string ew::Form3Volume::file

The canonical name of the file containing the data for this element. An empty string indicates that there is no associated file.

14.27 ew::Surface3 Class Reference

Triangulated Surface.

`#include <ew/Surface3.h>`

Public Member Functions

- void reset ()
- void read_file (const char *file)
- void swap (ew::Surface3 &s)
- bool operator== (const ew::Surface3 &a) const
- bool operator!= (const ew::Surface3 &a) const

Public Attributes

- std::vector< float > points
- std::vector< int > faces

14.27.1 Detailed Description

ew::Surface3 encapsulates data for triangulated surfaces in $\mathbb{R}^3$.

ew::Surface3 is a data structure that supports default construction, copy construction, assignment and equality comparison.

The data consists of a list of points and a list of faces. If a point is shared by several faces, the surface normal will be avaraged while rendering as a smooth surface. It this is not the intent, the point should be duplicated as several entries in the list of points, all with the same coordinates.

14.27.2 Member Function Documentation

14.27.2.1 void ew::Surface3::reset ( )

This frees any current data and sets the object to the initial state.

14.27.2.2 void ew::Surface3::read_file (const char * *filename*)

This reads the surface contained in the *file*. The following surface formats are supported:

- obj format, for example tetrahedron.obj. Only these types of record are currently supported:
  - v
  - f with non-negative arguments

  Normals and texture mapping are currently ignored.

Parameters

filename The name of the file to read.

Exceptions

std::bad_alloc

ew::ErrorIO

14.27.2.3 void ew::Surface3::swap (ew::Surface3 & *that*) [inline]

Swap data between 2 surfaces without copying.

Parameters

that The surface to swap with `this`.

14.27.2.4 bool ew::Surface3::operator== (const ew::Surface3 & *a*) const

Compares this surface with another, member by member.

Parameters

a the other surface

14.27.2.5 bool ew::Surface3::operator!= (const ew::Surface3 & *a*) const [inline]

Compares this surface with another, member by member.

Parameters

a the other surface

14.27.3 Member Data Documentation

14.27.3.1 std::vector< float > ew::Surface3::points

This vector has size 3 times the number of points. Each consecutive triple defines a point.

14.27.3.2 std::vector< int > ew::Surface3::faces

This vector has size 3 times the number of faces The elements are indices of the points list, and are in the range [0, number of points). Each consequtive triple defines a face.

14.28 ew::Tps2 Class Reference

Thin-Plate Spline in 2D.

`#include <ew/Tps2.h>`

Static Public Member Functions

- static void interaction (double *L, const double *lmks, int n_lmks)
- static void algebraic_size (int *f_size, bool *is_mixed, bool *is_reduced, const int *relax_dims_opt, int n_lmks)
- static void factorize (double *F, int *pivots, double *nonsingularity, const double *L, const int *relax_dims_opt, const double *relax_params_opt, int f_size, bool is_mixed, bool is_reduced, int n_lmks)
- static void solve (double *spline, double *energy_opt, double *relax_lmk_images_opt, const double *F, const int *pivots, const double *lmk_images, const int *relax_dims_opt, const double *relax_params_opt, const double *L, int n_lmks, int f_size, bool is_mixed, bool is_reduced)
- static void map (double *point_images, const double *points, int n_points, const double *spline, const double *lmks, int n_lmks)
- static void cache (float *cache, const float *points, int n_points, const double *lmk)

- static void cache_map (float *point_images, const float *points, float *const (*caches), int n_points, const double *spline, int n_lmks)
- static void bending_energy (double *B, const double *F, const int *pivots, const int *relax_dims_opt, const double *relax_params_opt, int n_lmks, int f_size, bool is_mixed, bool is_reduced)
- static void principal_warps (double *P, double *evals, const double *B, int n_lmks, bool is_mixed)
- static void principal_axes (double *centroid, double *sum_squares, double *axes, const double *lmks, int n_lmks)
- static void uniform_basis (double *basis, double *transforms_opt, const double *centroid, const double *sum_squares, const double *axes, const double *lmks, int n_lmks)

14.28.1 Detailed Description

ew::Tps2 is a class of static methods. It implements the thin-plate spline in $\mathbb{R}^2$. It supports semi-landmarks that are allowed to slide along affine subspaces.

The method interfaces are purely procedural. It is up to the caller to provide the space for the arguments and for the results. The only memory management performed is for internal temporary scratch space.

The numerical linear algebra is performed by the LAPACK library. The thin-plate spline system of equations has a symmetric, semi-definite matrix of coefficients. It is factorizated using the LAPACK function `DSYTRF` and solve using the LAPACK function `DSYTRS`. `DSYTRF` uses the Bunch-Kaufman diagonal pivoting algorithm. In the calculation of bending energy, the matrix inversion is performed by `DSYTRI`. In the calculation of principal warps and principal axes, the eigenanalysis is preformed by `DSYEV`. Lapack includes an error handler, `xerbla`, which will get linked into the program. However, provided this class is used correctly, `xerbla` will never be called.

On Mac OS X (as of 2010-02), the bundled lapack implementation can crash if the array arguments are not 16 byte aligned. Consequently, the array arguments of ew::Tps2 methods should be 16 byte aligned.

In this documentation, the first index of a matrix element is considered to be the row number, the second the column number. Matrix elements of all these methods are stored consecutively in memory by rows.

See Thin-Plate Spline with Semi-landmarks on Affine Subspaces and Canonical Uniform Warp Basis for details of the algorithms used by these methods.

14.28.2 Member Function Documentation

14.28.2.1 void ew::Tps2::interaction (double * *L*, const double * *lmks*, int *n_lmks*) [static]

This calculates the thin-plate spline interaction matrix of a configuration of landmarks.

Parameters

[out] ***L*** This is a matrix of size (*n_lmks* + 3) x (*n_lmks* + 3). The result is a symmetric and only the upper diagonal part is stored.

[in] ***lmks*** This is a matrix of size *n_lmks* x 2 containing the landmark coordinates.

[in] ***n_lmks*** This is the number of landmarks.

14.28.2.2 void ew::Tps2::algebraic_size (int * *f_size*, bool * *is_mixed*, bool * *is_reduced*, const int * *relax_dims_opt*, int *n_lmks*) [static]

This calculates the algebraic dimension and other characteristics of a thin-plate spline configuration with semi-landmarks. It is not necessary to call this function if there are no semi-landmarks.

Parameters

[out] ***f_size*** This is the location to store the algebraic dimension of the thin-plate spline equation. *f_size* is needed to allocate space for arguments to other functions. If *is_mixed* is `false`, it is at most *n_lmks* + 3. Otherwise it is less than 2 * (*n_lmks* + 3). If there are no semi-landmarks, it is exactly *n_lmks* + 3. In general, the difference between these upper limits and *f_size* is the sum of the elements of *relax_dims*.

[out] ***is_mixed*** This is the location to store a `boolean` value indicating whether the 2 dimensions are mixed in the thin-plate spline equation. If not, the thin-plate spline equation can be solved in each dimension separately. If there are no semi-landmarks, this will be `false`. It is `true` if there are semi-landmarks sliding in spaces of non-zero dimension and non-zero co-dimension.

[out] ***is_reduced*** This is the location to store a `boolean` value indicating whether there are any non-trivial semi-landmarks, and consequently, whether the thin-plate spline equation has been reduced. If there are no semi-landmarks, this will be `false`. It is `true` if there are semi-landmarks sliding in spaces of non-zero dimension.

[in] ***relax_dims_opt*** This is an array of size *n_lmks* containing the dimensions of the sliding spaces for each landmark. Each can be 0, 1, or 2. For landmarks

that are not semi-landmarks, the entries should be 0. If there are no semi-landmarks, this can be NULL.

[in] ***n_lmks*** See interaction.

14.28.2.3 void ew::Tps2::factorize (double * *F*, int * *pivots*, double * *nonsingularity*, const double * *L*, const int * *relax_dims_opt*, const double * *relax_params_opt*, int *f_size*, bool *is_mixed*, bool *is_reduced*, int *n_lmks*) [static]

This calculates the algebraic equation corresponding to the semi-landmark constraints, and factorizes the coefficient matrix. If *is_mixed* is true, this equation is mixed. Otherwise it separates into equations in each coordinate direction. It uses outputs from interaction and algebraic_size.

Parameters

[out] ***F*** This is a matrix of size *f_size* x *f_size*. The factorization of the coefficient matrix is stored here. It is not output if the thin-plate spline is singular.

[out] ***pivots*** This is an array of size *f_size*. The pivots used in the factorization are stored here. It is not output if the thin-plate spline is singular.

[out] ***nonsingularity*** This is the inverse of the condition number of the block diagonal matrix resulting from the matrix factorization. If this is exactly 0.0, the other outputs will not have been set and they must not used in solve. If this is close to 0.0, the other outputs are unreliable and should also not be used. A spline is exactly singular when there are 2 landmarks with the same coordinates, or when the landmarks are colinear. In this case, the *nonsingularity* might not be exactly 0.0 due to numerical error. Conversely, the spline equation might be numerically unsolvable without it being exactly singular, for example, if there are an excessive number of landmarks, if some landmarks are very close together, or if the landmarks are nearly colinear. The lower limit for the *nonsingularity* should be high enough to catch these cases, but not so high as to reject too many usable splines. A limit of 1.0e-8 is a sensible starting point.

[in] ***L*** See interaction.

[in] ***relax_dims_opt*** See algebraic_size.

[in] ***relax_params_opt*** This is a matrix of size *n_lmks* x 2. For every semi-landmark sliding along a line, the corresponding row of this matrix should be a unit vector normal to the line. The other rows need not be set. If there are no semi-landmarks of dimension 1, this can be NULL.

[in] ***f_size*** See algebraic_size.

[in] ***is_mixed*** See algebraic_size.

[in] ***is_reduced*** See algebraic_size.

[in] ***n_lmks*** See interaction.

Exceptions

std::bad_alloc

14.28.2.4 void ew::Tps2::solve (double * *spline*, double * *energy_opt*, double * *relax_lmk_images_opt*, const double * *F*, const int * *pivots*, const double * *lmk_images*, const int * *relax_dims_opt*, const double * *relax_params_opt*, const double * *L*, int *n_lmks*, int *f_size*, bool *is_mixed*, bool *is_reduced*) [static]

This calculates a thin-plate spline from a configuration of landmark images. It uses outputs from interaction, algebraic_size and factorize. It should not be called if the thin-plate spline is singular.

Parameters

[out] ***spline*** This is a matrix of size (*n_lmks* + 3) x 2. The coefficients of the solution thin-plate spline are stored here.

[out] ***energy_opt*** This is the location to store the bending energy of the thin-plate spline. It can be NULL, in which case the energy is not calculated.

[out] ***relax_lmk_images_opt*** This is a matrix of size *n_lmks* x 2. It is the location to store the optimized landmark images. These are the landmark images that would minimize bending energy consistent with the semi-landmark constraints. It can be NULL, in which case they are not calculated.

[in] ***F*** See factorize.

[in] ***pivots*** See factorize.

[in] ***lmk_images*** This is a matrix of size *n_lmks* x 2 containing the landmark image coordinates.

[in] ***relax_dims_opt*** See algebraic_size.

[in] ***relax_params_opt*** See factorize.

[in] ***L*** See interaction.

[in] ***n_lmks*** See interaction.

[in] ***f_size*** See algebraic_size.

[in] ***is_mixed*** See algebraic_size.

[in] ***is_reduced*** See algebraic_size.

Exceptions

std::bad_alloc

14.28.2.5 void ew::Tps2::map (double * *point_images*, const double * *points*, int *n_points*, const double * *spline*, const double * *lmks*, int *n_lmks*) [static]

This calculates the images of a list of points under a thin-plate spline. It uses outputs of solve. It should not be called if the thin-plate spline is singular.

Parameters

[in] ***point_images*** This is a matrix of size *n_points* x 2. It is the location to store the coordinates of the images of the points.

[in] ***points*** This is a matrix of size *n_points* x 2, containing the coordinates of the points

[in] ***n_points*** This is the number of points.

[in] ***spline*** See solve.

[in] ***lmks*** See interaction.

[in] ***n_lmks*** See interaction.

14.28.2.6 void ew::Tps2::cache (float * *cache*, const float * *points*, int *n_points*, const double * *lmk*) [static]

This evaluates a single thin-plate spline kernel function at a list of points. It is intended for speed critical situations where incremental changes are made to a thin-plate spline that is being applied to a fixed set of points. It requires a *cache* for each landmark, which could be a require a significant amount of memory. The *points* and *cache* are of type `float` rather than `double` to save space. As a consequence, the accuracy of the mapped points is reduced.

Parameters

[out] ***cache*** This is an array of size n_points. The cached evaluations are stored here.

[in] ***points*** See map, except that this is a `float` array rather than `double`.

[in] ***n_points*** See map.

[in] ***lmk*** This points to an array of size 2 containing the coordinates of the landmark.

14.28.2.7 void ew::Tps2::cache_map (float * *point_images*, const float * *points*, float *const * *caches*, int *n_points*, const double * *spline*, int *n_lmks*) [static]

This uses previously calculated cached kernel evaluations to calculate the images of a list of points under a thin-plate spline. It uses outputs from solve and cache. It should not be called if the thin-plate spline is singular.

Parameters

[in] ***point_images*** See map.

[in] ***points*** See map.

[in] ***caches*** This is an array of size *n_lmks*. It is an array of pointers, each pointing to the corresponding cache.

[in] ***n_points*** See map.

[in] ***spline*** See solve.

[in] ***n_lmks*** See interaction.

14.28.2.8 void ew::Tps2::bending_energy (double * *B*, const double * *F*, const int * *pivots*, const int * *relax_dims_opt*, const double * *relax_params_opt*, int *n_lmks*, int *f_size*, bool *is_mixed*, bool *is_reduced*) [static]

This calculates the bending energy matrix. If *is_mixed*, it is a quadratic form on the mixed coordinates. Otherwise, it is a quadratic form that is applied separately to the x coordinates and to the y coordinates. If there are any non-trivial semi-landmarks, this is the form for the minimized bending energy. It uses outputs from interaction, algebraic_size and factorize. It should not be called if the thin-plate spline is singular.

Parameters

[out] ***B*** This is the bending energy matrix. It is a symmetric matrix, with both upper and lower diagonal entries stored. If *is_mixed*, it has size (2 * *n_lmks*) x (2 * *n_lmks*). Otherwise, it has size *n_lmks* x *n_lmks*.

[in] ***F*** See factorize.

[in] ***pivots*** See factorize.

[in] ***relax_dims_opt*** See algebraic_size.

[in] ***relax_params_opt*** See factorize.

[in] ***n_lmks*** See interaction.

[in] ***f_size*** See algebraic_size.

[in] ***is_mixed*** See algebraic_size.

[in] ***is_reduced*** See algebraic_size.

Exceptions

std::bad_alloc

14.28.2.9 void ew::Tps2::principal_warps (double * *P*, double * *evals*, const double * *B*, int *n_lmks*, bool *is_mixed*) [static]

This calculates the principal warps. If *is_mixed*, they are a basis of the full space of landmark displacements. Otherwise, they are a basis of the space of landmarks in a single coordinate directions, either x or y. It uses outputs from bending_energy. It should not be called if the thin-plate spline is singular.

Parameters

[out] ***P*** If *is_mixed* is `false`, this is a matrix of size *n_lmks* x *n_lmks*. Otherwise it is a matrix of size (2 * *n_lmks*) x (2 * *n_lmks*). The eigenvectors are stored here. The form an orthogonal matrix.

[out] ***evals*** If *is_mixed* is `false`, this is an array of size *n_lmks*. Otherwise it is an array of size 2 * *n_lmks*. The eigenvalues are stored here. The eigenvectors are ordered so that the eigenvalues are in non-decreasing order.

[in] ***B*** See bending_energy.

[in] ***n_lmks*** See interaction.

[in] ***is_mixed*** See algebraic_size.

Exceptions

std::runtime_error The eigenanalysis failed to converge.

std::bad_alloc

14.28.2.10 void ew::Tps2::principal_axes (double * *centroid*, double * *sum_squares*, double * *axes*, const double * *lmks*, int *n_lmks*) [static]

This calculates a set of principal axes for the landmark configuration. The axes are not canonical, which is why their calculation and their use in uniform_basis are separated into different functions. They are canonical up to sign if the sum_squares are different. It should not be called with an empty configuration of landmarks.

Parameters

[out] ***centroid*** This points to an array of size 2 to store the centroid of the configuration.

[out] ***sum_squares*** This points to an array of size 2 to store the sum of squared coordinates in the principal axes.

[out] ***axes*** This is a matrix of size *n_lmks* x *n_lmks*. It is the location to store the principal axes. They form an orthogonal matrix.

[in] ***lmks*** See interaction.

[in] ***n_lmks*** See interaction.

Exceptions

std::runtime_error The eigenanalysis failed to converge.

std::bad_alloc

14.28.2.11 void ew::Tps2::uniform_basis (double * *basis*, double * *transforms_opt*, const double * *centroid*, const double * *sum_squares*, const double * *axes*, const double * *lmks*, int *n_lmks*) [static]

This calculates a canonical basis of the uniform subspace of the space of shape variations, given a set of principal axes for the configuration. The result is relative to the original axes, not the principal axes. If the configuration is degenerate (contained within a line), this should not be called.

Parameters

[out] ***basis*** This is a matrix of size 2 x *n_lmks* x 2. The shape variables of the 2 basis elements are stored here. The first index is the number of the basis element.

[out] ***transforms_opt*** This points to an array of size 2 * 2 * 2. Uniform shape variations are linear maps, so can be represented as 2 x 2 matrices. The matrices of the 2 basis elements are stored here. The first index is the number of the basis element. This can be NULL, in which case the transformations are not calculated.

[in] ***centroid*** See principal_axes.

[in] ***sum_squares*** See principal_axes.

[in] ***axes*** See principal_axes.

[in] ***lmks*** See interaction.

[in] ***n_lmks*** See interaction.

14.29 ew::Tps3 Class Reference

Thin-Plate Spline in 3D.

`#include <ew/Tps3.h>`

Static Public Member Functions

- static void interaction (double *L, const double *lmks, int n_lmks)

- static void algebraic_size (int *f_size, bool *is_mixed, bool *is_reduced, const int *relax_dims_opt, int n_lmks)
- static void factorize (double *F, int *pivots, double *nonsingularity, const double *L, const int *relax_dims_opt, const double *relax_params_opt, int f_size, bool is_mixed, bool is_reduced, int n_lmks)
- static void solve (double *spline, double *energy_opt, double *relax_lmk_images_opt, const double *F, const int *pivots, const double *lmk_images, const int *relax_dims_opt, const double *relax_params_opt, const double *L, int n_lmks, int f_size, bool is_mixed, bool is_reduced)
- static void map (double *point_images, const double *points, int n_points, const double *spline, const double *lmks, int n_lmks)
- static void cache (float *cache, const float *points, int n_points, const double *lmk)
- static void cache_map (float *point_images, const float *points, float *const (*caches), int n_points, const double *spline, int n_lmks)
- static void bending_energy (double *B, const double *F, const int *pivots, const int *relax_dims_opt, const double *relax_params_opt, int n_lmks, int f_size, bool is_mixed, bool is_reduced)
- static void principal_warps (double *P, double *evals, const double *B, int n_lmks, bool is_mixed)
- static void principal_axes (double *centroid, double *sum_squares, double *axes, const double *lmks, int n_lmks)
- static void uniform_basis (double *basis, double *transforms_opt, const double *centroid, const double *sum_squares, const double *axes, const double *lmks, int n_lmks)

14.29.1 Detailed Description

ew::Tps3 is a class of static methods. It implements the thin-plate spline in $\mathbb{R}^3$. It supports semi-landmarks that are allowed to slide along affine subspaces.

The method interfaces are purely procedural. It is up to the caller to provide the space for the arguments and for the results. The only memory management performed is for internal temporary scratch space.

The numerical linear algebra is performed by the LAPACK library. The thin-plate spline system of equations has a symmetric, semi-definite matrix of coefficients. It is factorizated using the LAPACK function `DSYTRF` and solve using the LAPACK function `DSYTRS`. `DSYTRF` uses the Bunch-Kaufman diagonal pivoting algorithm. In the calculation of bending energy, the matrix inversion is performed by `DSYTRI`. In the calculation of principal warps and principal axes, the eigenanalysis is preformed by `DSYEV`. Lapack includes an error handler, `xerbla`, which will get linked into the program. However, provided this class is used correctly, `xerbla` will never be called.

On Mac OS X (as of 2010-02), the bundled lapack implementation can crash if the array arguments are not 16 byte aligned. Consequently, the array arguments of ew::Tps3

methods should be 16 byte aligned.

In this documentation, the first index of a matrix element is considered to be the row number, the second the column number. Matrix elements of all these methods are stored consecutively in memory by rows.

See Thin-Plate Spline with Semi-landmarks on Affine Subspaces and Canonical Uniform Warp Basis for details of the algorithms used by these methods.

14.29.2 Member Function Documentation

14.29.2.1 void ew::Tps3::interaction (double * *L*, const double * *lmks*, int *n_lmks*) [static]

This calculates the thin-plate spline interaction matrix of a configuration of landmarks.

Parameters

[out] ***L*** This is a matrix of size (*n_lmks* + 4) x (*n_lmks* + 4). The result is a symmetric and only the upper diagonal part is stored.

[in] ***lmks*** This is a matrix of size *n_lmks* x 3 containing the landmark coordinates.

[in] ***n_lmks*** This is the number of landmarks.

14.29.2.2 void ew::Tps3::algebraic_size (int * *f_size*, bool * *is_mixed*, bool * *is_reduced*, const int * *relax_dims_opt*, int *n_lmks*) [static]

This calculates the algebraic dimension and other characteristics of a thin-plate spline configuration with semi-landmarks. It is not necessary to call this function if there are no semi-landmarks.

Parameters

[out] ***f_size*** This is the location to store the algebraic dimension of the thin-plate spline equation. *f_size* is needed to allocate space for arguments to other functions. If *is_mixed* is `false`, it is at most *n_lmks* + 4. Otherwise it is less than 3 * (*n_lmks* + 4). If there are no semi-landmarks, it is exactly *n_lmks* + 4. In general, the difference between these upper limits and *f_size* is the sum of the elements of *relax_dims*.

[out] ***is_mixed*** This is the location to store a `boolean` value indicating whether the 3 dimensions are mixed in the thin-plate spline equation. If not, the thin-plate spline equation can be solved in each dimension separately. If there are no semi-landmarks, this will be `false`. It is `true` if there are semi-landmarks sliding in spaces of non-zero dimension and non-zero co-dimension.

[out] ***is_reduced*** This is the location to store a `boolean` value indicating whether there are any non-trivial semi-landmarks, and consequently, whether the thin-plate spline equation has been reduced. If there are no semi-landmarks, this will be `false`. It is `true` if there are semi-landmarks sliding in spaces of non-zero dimension.

[in] ***relax_dims_opt*** This is an array of size *n_lmks* containing the dimensions of the sliding spaces for each landmark. Each can be 0, 1, 2 or 3. For landmarks that are not semi-landmarks, the entries should be 0. If there are no semi-landmarks, this can be `NULL`.

[in] ***n_lmks*** See interaction.

14.29.2.3 void ew::Tps3::factorize (double * *F*, int * *pivots*, double * *nonsingularity*, const double * *L*, const int * *relax_dims_opt*, const double * *relax_params_opt*, int *f_size*, bool *is_mixed*, bool *is_reduced*, int *n_lmks*) [static]

This calculates the algebraic equation corresponding to the semi-landmark constraints, and factorizes the coefficient matrix. If *is_mixed* is `true`, this equation is mixed. Otherwise it separates into equations in each coordinate direction. It uses outputs from interaction and algebraic_size.

Parameters

[out] ***F*** This is a matrix of size *f_size* x *f_size*. The factorization of the coefficient matrix is stored here. It is not output if the thin-plate spline is singular.

[out] ***pivots*** This is an array of size *f_size*. The pivots used in the factorization are stored here. It is not output if the thin-plate spline is singular.

[out] ***nonsingularity*** This is the inverse of the condition number of the block diagonal matrix resulting from the matrix factorization. If this is exactly 0.0, the other outputs will not have been set and they must not used in solve. If this is close to 0.0, the other outputs are unreliable and should also not be used. A spline is exactly singular when there are 2 landmarks with the same coordinates, or when the landmarks are coplanar. In this case, the *nonsingularity* might not be exactly 0.0 due to numerical error. Conversely, the spline equation might be numerically unsolvable without it being exactly singular, for example, if there are an excessive number of landmarks, if some landmarks are very close together, or if the landmarks are nearly coplanar. The lower limit for the *nonsingularity* should be high enough to catch these cases, but not so high as to reject too many usable splines. A limit of `1.0e-8` is a sensible starting point.

[in] ***L*** See interaction.

[in] ***relax_dims_opt*** See algebraic_size.

[in] ***relax_params_opt*** This is a matrix of size *n_lmks* x 3. For every semi-landmark sliding along a line, the corresponding row of this matrix should be a unit vector in the direction of the line. For every semi-landmark sliding along a plane, the corresponding row of this matrix should be a unit vector normal to the plane. The other rows need not be set. If there are no semi-landmarks of dimension 1 or 2, this can be NULL.

[in] ***f_size*** See algebraic_size.

[in] ***is_mixed*** See algebraic_size.

[in] ***is_reduced*** See algebraic_size.

[in] ***n_lmks*** See interaction.

Exceptions

std::bad_alloc

14.29.2.4 void ew::Tps3::solve (double * *spline*, double * *energy_opt*, double * *relax_lmk_images_opt*, const double * *F*, const int * *pivots*, const double * *lmk_images*, const int * *relax_dims_opt*, const double * *relax_params_opt*, const double * *L*, int *n_lmks*, int *f_size*, bool *is_mixed*, bool *is_reduced*) [static]

This calculates a thin-plate spline from a configuration of landmark images. It uses outputs from interaction, algebraic_size and factorize. It should not be called if the thin-plate spline is singular.

Parameters

[out] ***spline*** This is a matrix of size (*n_lmks* + 4) x 3. The coefficients of the solution thin-plate spline are stored here.

[out] ***energy_opt*** This is the location to store the bending energy of the thin-plate spline. It can be NULL, in which case the energy is not calculated.

[out] ***relax_lmk_images_opt*** This is a matrix of size *n_lmks* x 3. It is the location to store the optimized landmark images. These are the landmark images that would minimize bending energy consistent with the semi-landmark constraints. It can be NULL, in which case they are not calculated.

[in] ***F*** See factorize.

[in] ***pivots*** See factorize.

[in] ***lmk_images*** This is a matrix of size *n_lmks* x 3 containing the landmark image coordinates.

[in] ***relax_dims_opt*** See algebraic_size.

[in] ***relax_params_opt*** See factorize.

[in] ***L*** See interaction.

[in] ***n_lmks*** See interaction.

[in] ***f_size*** See algebraic_size.

[in] ***is_mixed*** See algebraic_size.

[in] ***is_reduced*** See algebraic_size.

Exceptions

std::bad_alloc

14.29.2.5 void ew::Tps3::map (double * *point_images*, const double * *points*, int *n_points*, const double * *spline*, const double * *lmks*, int *n_lmks*) [static]

This calculates the images of a list of points under a thin-plate spline. It uses outputs of solve. It should not be called if the thin-plate spline is singular.

Parameters

[in] ***point_images*** This is a matrix of size *n_points* x 3. It is the location to store the coordinates of the images of the points.

[in] ***points*** This is a matrix of size *n_points* x 3, containing the coordinates of the points

[in] ***n_points*** This is the number of points.

[in] ***spline*** See solve.

[in] ***lmks*** See interaction.

[in] ***n_lmks*** See interaction.

14.29.2.6 void ew::Tps3::cache (float * *cache*, const float * *points*, int *n_points*, const double * *lmk*) [static]

This evaluates a single thin-plate spline kernel function at a list of points. It is intended for speed critical situations where incremental changes are made to a thin-plate spline that is being applied to a fixed set of points. It requires a *cache* for each landmark, which could be a require a significant amount of memory. The *points* and *cache* are of type `float` rather than `double` to save space. As a consequence, the accuracy of the mapped points is reduced.

Parameters

[out] ***cache*** This is an array of size n_points. The cached evaluations are stored here.

[in] ***points*** See map, except that this is a `float` array rather than `double`.

[in] ***n_points*** See map.

[in] ***lmk*** This points to an array of size 3 containing the coordinates of the landmark.

14.29.2.7 void ew::Tps3::cache_map (float * *point_images*, const float * *points*, float *const * *caches*, int *n_points*, const double * *spline*, int *n_lmks*) [static]

This uses previously calculated cached kernel evaluations to calculate the images of a list of points under a thin-plate spline. It uses outputs from solve and cache. It should not be called if the thin-plate spline is singular.

Parameters

[in] ***point_images*** See map.

[in] ***points*** See map.

[in] ***caches*** This is an array of size *n_lmks*. It is an array of pointers, each pointing to the corresponding cache.

[in] ***n_points*** See map.

[in] ***spline*** See solve.

[in] ***n_lmks*** See interaction.

14.29.2.8 void ew::Tps3::bending_energy (double * *B*, const double * *F*, const int * *pivots*, const int * *relax_dims_opt*, const double * *relax_params_opt*, int *n_lmks*, int *f_size*, bool *is_mixed*, bool *is_reduced*) [static]

This calculates the bending energy matrix. If *is_mixed*, it is a quadratic form on the mixed coordinates. Otherwise, it is a quadratic form that is applied separately to the x coordinates and to the y coordinates. If there are any non-trivial semi-landmarks, this is the form for the minimized bending energy. It uses outputs from interaction, algebraic_size and factorize. It should not be called if the thin-plate spline is singular.

Parameters

[out] ***B*** This is the bending energy matrix. It is a symmetric matrix, with both upper and lower diagonal entries stored. If *is_mixed*, it has size (3 * *n_lmks*) x (3 * *n_lmks*). Otherwise, it has size *n_lmks* x *n_lmks*.

[in] ***F*** See factorize.

[in] ***pivots*** See factorize.

[in] ***relax_dims_opt*** See algebraic_size.

[in] ***relax_params_opt*** See factorize.

[in] ***n_lmks*** See interaction.

[in] ***f_size*** See algebraic_size.

[in] ***is_mixed*** See algebraic_size.

[in] ***is_reduced*** See algebraic_size.

Exceptions

std::bad_alloc

14.29.2.9 void ew::Tps3::principal_warps (double * *P*, double * *evals*, const double * *B*, int *n_lmks*, bool *is_mixed*) [static]

This calculates the principal warps. If *is_mixed*, they are a basis of the full space of landmark displacements. Otherwise, they are a basis of the space of landmarks in a single coordinate directions, either x or y. It uses outputs from bending_energy. It should not be called if the thin-plate spline is singular.

Parameters

[out] ***P*** If *is_mixed* is `false`, this is a matrix of size *n_lmks* x *n_lmks*. Otherwise it is a matrix of size (3 * *n_lmks*) x (3 * *n_lmks*). The eigenvectors are stored here. The form an orthogonal matrix.

[out] ***evals*** If *is_mixed* is `false`, this is an array of size *n_lmks*. Otherwise it is an array of size 3 * *n_lmks*. The eigenvalues are stored here. The eigenvectors are ordered so that the eigenvalues are in non-decreasing order.

[in] ***B*** See bending_energy.

[in] ***n_lmks*** See interaction.

[in] ***is_mixed*** See algebraic_size.

Exceptions

std::runtime_error The eigenanalysis failed to converge.

std::bad_alloc

14.29.2.10 void ew::Tps3::principal_axes (double * *centroid*, double * *sum_squares*, double * *axes*, const double * *lmks*, int *n_lmks*) [static]

This calculates a set of principal axes for the landmark configuration. The axes are not canonical, which is why their calculation and their use in uniform_basis are separated

into different functions. They are canonical up to sign if the sum_squares are different. It should not be called with an empty configuration of landmarks.

Parameters

[out] ***centroid*** This points to an array of size 3 to store the centroid of the configuration.

[out] ***sum_squares*** This points to an array of size 3 to store the sum of squared coordinates in the principal axes.

[out] ***axes*** This is a matrix of size *n_lmks* x *n_lmks*. It is the location to store the principal axes. They form an orthogonal matrix.

[in] ***lmks*** See interaction.

[in] ***n_lmks*** See interaction.

Exceptions

std::runtime_error The eigenanalysis failed to converge.

std::bad_alloc

14.29.2.11 void ew::Tps3::uniform_basis (double * *basis*, double * *transforms_opt*, const double * *centroid*, const double * *sum_squares*, const double * *axes*, const double * *lmks*, int *n_lmks*) [static]

This calculates a canonical basis of the uniform subspace of the space of shape variations, given a set of principal axes for the configuration. The result is relative to the original axes, not the principal axes. If the configuration is degenerate (contained within a plane), this should not be called.

Parameters

[out] ***basis*** This is a matrix of size 5 x *n_lmks* x 3. The shape variables of the 5 basis elements are stored here. The first index is the number of the basis element.

[out] ***transforms_opt*** This points to an array of size $5 * 3 * 3$. Uniform shape variations are linear maps, so can be represented as 3 x 3 matrices. The matrices of the 5 basis elements are stored here. The first index is the number of the basis element. This can be NULL, in which case the transformations are not calculated.

[in] ***centroid*** See principal_axes.

[in] ***sum_squares*** See principal_axes.

[in] ***axes*** See principal_axes.

[in] ***lmks*** See interaction.

[in] ***n_lmks*** See interaction.

14.30 ew::Transform2 Class Reference

2D Similarity Transformation

`#include <ew/Transform2.h>`

Public Member Functions

- void set_to_identity ()
- void set_to_inverse (const ew::Transform2 *a)
- void set_to_composition (const ew::Transform2 *a, const ew::Transform2 *b)
- bool set_to_interpolation (const ew::Transform2 *a, const ew::Transform2 *b, double e)
- void get_matrix_gl (double *buffer) const
- void apply (float *dst, const float *src) const
- void apply (double *dst, const double *src) const
- double get_denormalization () const
- void set_to_normalization (const ew::Transform2 *a)
- bool get_valid () const
- bool operator== (const ew::Transform2 &a) const
- bool operator!= (const ew::Transform2 &a) const

Public Attributes

- double orthog [2][2]
- double scale
- double translate [2]
- unsigned int comps_cnt

Static Public Attributes

- static const ew::Transform2 identity_transform

14.30.1 Detailed Description

ew::Transform2 represents similarity transformations of $\mathbb{R}^2$, each as the composition of an orthogonal linear map, a scaling and a translation. The translation is applied after the orthogonal linear map and the scaling.

ew::Transform2 is a POD type class.

14.30.2 Member Function Documentation

14.30.2.1 void ew::Transform2::set_to_identity ( )

This sets this transformation to the identity transformation.

14.30.2.2 void ew::Transform2::set_to_inverse (const ew::Transform2 * *a*)

This sets this transformation to the inverse of another transformation.

Parameters

a This points to the transformation to invert. *a* can equal `this`.

14.30.2.3 void ew::Transform2::set_to_composition (const ew::Transform2 * *a*, const ew::Transform2 * *b*)

This sets this transformation to the composition of two transformations.

Parameters

a This points to the transformation that is applied last in the composition. *a* can equal `this`.

b This points to the transformation that is applied first in the composition. *b* can equal `this`.

14.30.2.4 bool ew::Transform2::set_to_interpolation (const ew::Transform2 * *a*, const ew::Transform2 * *b*, double *e*)

This sets this transformation to an interpolation between two transformations.

Parameters

a This points to the first transformation. *a* can equal `this`.

b This points to the second transformation. *b* can equal `this`.

e This is the interpolation paramater. When 0.0, the transformation is set to `*a`, when 1.0 to `*b`. *e* is not restricted to the range [0.0, 1.0].

Returns

`true` if a smooth canonical interpolation was possible. If `false`, the transformation is set to either the first or second transformation, depending on whether *e* is closer to 0.0 or 1.0. This can occur if one transformation is a reflection and the other transformation is a rotation, or if the orthogonal parts differ by a half turn.

14.30.2.5 void ew::Transform2::get_matrix_gl (double * *buffer*) const

This outputs this transformation as a 4 x 4 matrix in the format used by **OpenGL**.

Parameters

`[out]` ***buffer*** The result is stored in this array of size 16.

14.30.2.6 void ew::Transform2::apply (float * *dst*, const float * *src*) const [inline]

This applies the transformation to a point.

Parameters

`[out]` ***dst*** The result is stored in this array of size 2.

`[in]` ***src*** The original point is contained in this array of size 2.

14.30.2.7 void ew::Transform2::apply (double * *dst*, const double * *src*) const [inline]

This applies the transformation to a point.

Parameters

`[out]` ***dst*** The result is stored in this array of size 2.

`[in]` ***src*** The original point is contained in this array of size 2.

14.30.2.8 double ew::Transform2::get_denormalization ( ) const

This calculates how far orthog is from being orthogonal.

Returns

The return value is a non-negative number representing this distance.

14.30.2.9 void ew::Transform2::set_to_normalization (const ew::Transform2 * *a*)

This sets this transformation to another transformation and then coerces orthog to be orthogonal.

Parameters

a This points to the transformation to normalize. *a* can equal `this`.

14.30.2.10 bool ew::Transform2::get_valid ( ) const

This checks that all components of orthog, scale and translate are finite floating point numbers and that scale is positive and not sub-normal.

Returns

`true` if all these conditions are satisfied.

14.30.2.11 bool ew::Transform2::operator== (const ew::Transform2 & *a*) const

Compares this transform with another, member by member, except for comps_cnt, which is ignored.

Parameters

a the other transform

14.30.2.12 bool ew::Transform2::operator!= (const ew::Transform2 & *a*) const [inline]

Compares this transform with another, member by member, except for comps_cnt, which is ignored.

Parameters

a the other transform

14.30.3 Member Data Documentation

14.30.3.1 double ew::Transform2::orthog[2][2]

This is the matrix representing the orthogonal part of the transformation. The matrix elements are ordered by row, and the matrix is applied on the left. The coordinates should be finite.

14.30.3.2 double ew::Transform2::scale

This is the scale part of the transformation. This should be positive.

14.30.3.3 double ew::Transform2::translate[2]

This is the translation part of the transformation. The coordinates should be finite.

14.30.3.4 unsigned int ew::Transform2::comps_cnt

This is a count of the accumulated number of compositions performed in the calculation of the current transformation. It is reset by set_to_identity and set_to_-normalization and set by set_to_inverse, set_to_composition and set_to_interpolation. If orthog is set directly, comps_cnt should be set appropriately.

set_to_composition will automatically normalize its result when comps_cnt exceeds an internal threshold.

This is to prevent exponential deviation of the orthogonal part from orthogonality under iterated compositions.

14.30.3.5 const ew::Transform2 ew::Transform2::identity_transform [static]

Initial value:

```
{
 {{1.0, 0.0}, {0.0, 1.0}},
 1.0,
 {0.0, 0.0},
 0
}
```

This is the identity transform.

14.31 ew::Transform3 Class Reference

3D Similarity Transformation

`#include <ew/Transform3.h>`

Public Member Functions

- void set_to_identity ()
- void set_to_inverse (const ew::Transform3 *a)
- void set_to_composition (const ew::Transform3 *a, const ew::Transform3 *b)
- bool set_to_interpolation (const ew::Transform3 *a, const ew::Transform3 *b, double e)
- void get_matrix_gl (double *buffer) const
- void apply (float *dst, const float *src) const
- void apply (double *dst, const double *src) const
- double get_denormalization () const
- void set_to_normalization (const ew::Transform3 *a)
- bool get_valid () const

- bool scan (const char **se, const char *s)
- bool format (char *buf, int bufl, int *outl) const
- bool operator== (const ew::Transform3 &a) const
- bool operator!= (const ew::Transform3 &a) const

Public Attributes

- double orthog [3][3]
- double scale
- double translate [3]
- unsigned int comps_cnt

Static Public Attributes

- static const int FORMAT_LEN = 480
- static const ew::Transform3 identity_transform

14.31.1 Detailed Description

ew::Transform3 represents similarity transformations of $\mathbb{R}^3$, each as the composition of an orthogonal linear map, a scaling and a translation. The translation is applied after the orthogonal linear map and the scaling.

ew::Transform3 is a POD type class.

14.31.2 Member Function Documentation

14.31.2.1 void ew::Transform3::set_to_identity ( )

This sets this transformation to the identity transformation.

14.31.2.2 void ew::Transform3::set_to_inverse (const ew::Transform3 * *a*)

This sets this transformation to the inverse of another transformation.

Parameters

a This points to the transformation to invert. *a* can equal `this`.

14.31.2.3 void ew::Transform3::set_to_composition (const ew::Transform3 * *a,* const ew::Transform3 * *b*)

This sets this transformation to the composition of two transformations.

Parameters

a This points to the transformation that is applied last in the composition. *a* can equal `this`.

b This points to the transformation that is applied first in the composition. *b* can equal `this`.

14.31.2.4 bool ew::Transform3::set_to_interpolation (const ew::Transform3 * *a,* const ew::Transform3 * *b,* double *e*)

This sets this transformation to an interpolation between two transformations.

Parameters

a This points to the first transformation. *a* can equal `this`.

b This points to the second transformation. *b* can equal `this`.

e This is the interpolation paramater. When 0.0, the transformation is set to `*a`, when 1.0 to `*b`. *e* is not restricted to the range [0.0, 1.0].

Returns

`true` if a smooth canonical interpolation was possible. If `false`, the transformation is set to either the first or second transformation, depending on whether *e* is closer to 0.0 or 1.0. This can occur if one transformation is a reflection and the other transformation is a rotation, or if the orthogonal parts differ by a half turn.

14.31.2.5 void ew::Transform3::get_matrix_gl (double * *buffer*) const

This outputs this transformation as a 4 x 4 matrix in the format used by **OpenGL**.

Parameters

`[out]` ***buffer*** The result is stored in this array of size 16.

14.31.2.6 void ew::Transform3::apply (float * *dst,* const float * *src*) const [inline]

This applies the transformation to a point.

Parameters

`[out]` ***dst*** The result is stored in this array of size 3.

`[in]` ***src*** The original point is contained in this array of size 3.

14.31.2.7 void ew::Transform3::apply (double * *dst*, const double * *src*) const [inline]

This applies the transformation to a point.

Parameters

`[out]` ***dst*** The result is stored in this array of size 3.

`[in]` ***src*** The original point is contained in this array of size 3.

14.31.2.8 double ew::Transform3::get_denormalization ( ) const

This calculates how far orthog is from being orthogonal.

Returns

The return value is a non-negative number representing this distance.

14.31.2.9 void ew::Transform3::set_to_normalization (const ew::Transform3 * *a*)

This sets this transformation to another transformation and then coerces orthog to be orthogonal.

Parameters

a This points to the transformation to normalize. *a* can equal `this`.

14.31.2.10 bool ew::Transform3::get_valid ( ) const

This checks that all components of orthog, scale and translate are finite floating point numbers and that scale is positive and not sub-normal.

Returns

`true` if all these conditions are satisfied.

14.31.2.11 bool ew::Transform3::scan (const char ** *se*, const char * *s*)

This scans a text string for a representation of a transform. Initial whitespace is not skipped, so would result in failure. The string must be null terminated. If successfully scanned, comps_cnt is set to zero. The transform is not automatically normalized, but can be manually.

Parameters

`[out]` ***se*** If non-zero, this is set to the address of the first unprocessed character.

`[in]` ***s*** This is a pointer to the text to scan.

Returns

`true` if the initial segment of the text is a correctly formatted transform.

14.31.2.12 bool ew::Transform3::format (char * *buf*, int *bufl*, int * *outl*) const

This formats this transform as a text string.

The format used to represent transforms is as follows:

[tr1 tr2 tr3] [or11 or12 or13 or21 or22 or23 or31 or32 or33] sc

For example, this represents a transformation consisting of an enlargement by a factor 1.5, followed by a rotation about the z axis, and then a translation along the y axis:

[0 4.5 0] [0.6 0.8 0 -0.8 0.6 0 0 0 1] 1.5

The floating point numbers are represented exactly, but efficiently.

Parameters

`[out]` ***buf*** The output is written here.

bufl The length of *buf*.

`[out]` ***outl*** If non-null, the length of the output (even if it was not written), not including the terminating null.

Returns

`true` if the buffer is long enough for a successful format.

14.31.2.13 bool ew::Transform3::operator== (const ew::Transform3 & *a*) const

Compares this transform with another, member by member, except for comps_cnt, which is ignored.

Parameters

a the other transform

14.31.2.14 bool ew::Transform3::operator!= (const ew::Transform3 & *a*) const [inline]

Compares this transform with another, member by member, except for comps_cnt, which is ignored.

Parameters

a the other transform

14.31.3 Member Data Documentation

14.31.3.1 double ew::Transform3::orthog[3][3]

This is the matrix representing the orthogonal part of the transformation. The matrix elements are ordered by row, and the matrix is applied on the left. The coordinates should be finite.

14.31.3.2 double ew::Transform3::scale

This is the scale part of the transformation. This should be positive.

14.31.3.3 double ew::Transform3::translate[3]

This is the translation part of the transformation. The coordinates should be finite.

14.31.3.4 unsigned int ew::Transform3::comps_cnt

This is a count of the accumulated number of compositions performed in the calculation of the current transformation. It is reset by set_to_identity and set_to_normalization and set by set_to_inverse, set_to_composition, set_to_interpolation and scan. If orthog is set directly, comps_cnt should be set appropriately.

set_to_composition will automatically normalize its result when comps_cnt exceeds an internal threshold.

This is to prevent exponential deviation of the orthogonal part from orthogonality under iterated compositions.

14.31.3.5 const int ew::Transform3::FORMAT_LEN = 480 [static]

This is the length of buffer that is guaranteed to be long enough for the output of format.

14.31.3.6 const ew::Transform3 ew::Transform3::identity_transform [static]

Initial value:

```
{
 {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}},
 1.0,
 {0.0, 0.0, 0.0},
 0
}
```

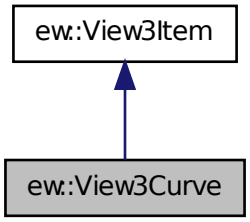
This is the identity transform.

14.32 ew::View3Curve Class Reference

Curve Item.

`#include <ew/View3Curve.h>`

Inheritance diagram for ew::View3Curve:



Public Member Functions

- View3Curve (ew::View3Widget *view, int i_type=0)
- void set_curve (const ew::DataflowCurve3 *cur)
- const ew::DataflowCurve3 * get_curve () const
- void set_color (const unsigned char *rgb)
- const unsigned char * get_color () const

14.32.1 Detailed Description

ew::View3Curve is an ew::View3Widget item to display a ew::DataflowCurve3, a piecewise-linear curve.

ew::View3Curve is a class without assignment or comparison. There are private member variables.

14.32.2 Constructor & Destructor Documentation

14.32.2.1 ew::View3Curve::View3Curve (ew::View3Widget * *view,* int *i_type* = *0*) [explicit]

This creates an item.

Parameters

view The view widget the item should belong to.

i_type The initializer for ew::View3Item::type.

14.32.3 Member Function Documentation

14.32.3.1 void ew::View3Curve::set_curve (const ew::DataflowCurve3 * *cur*)

Parameters

cur A pointer to the new curve node to display, or 0.

14.32.3.2 const ew::DataflowCurve3 * ew::View3Curve::get_curve ( ) const [inline]

Returns

A pointer to the current curve node being displayed, or 0.

14.32.3.3 void ew::View3Curve::set_color (const unsigned char * *rgb*)

Parameters

[in] ***rgb*** This points to an array of size 3 containing the new colour for the curve.

14.32.3.4 const unsigned char * ew::View3Curve::get_color ( ) const [inline]

Returns

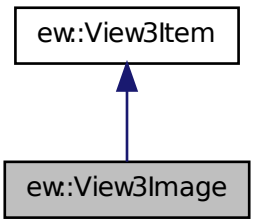
A pointer to the current colour for the curve.

14.33 ew::View3Image Class Reference

Image Item.

`#include <ew/View3Image.h>`

Inheritance diagram for ew::View3Image:



Public Member Functions

- View3Image (ew::View3Widget *view, int i_type=0)
- void set_image (const ew::DataflowImage3 *im)
- const ew::DataflowImage3 * get_image () const
- void set_color (const unsigned char *rgb)
- const unsigned char * get_color () const

14.33.1 Detailed Description

ew::View3Image is an ew::View3Widget item to display a ew::DataflowImage3, a raster image projected onto an array of points.

ew::View3Image is a class without assignment or comparison. There are private member variables.

This is barely implemented, and currently just displays a solid rectangle, on a 2x2x1 rectangular array of points.

The default colour is [128, 128, 128].

14.33.2 Constructor & Destructor Documentation

14.33.2.1 ew::View3Image::View3Image (ew::View3Widget * *view,* int *i_type* = *0*) [explicit]

This creates an item.

Parameters

view The view widget the item should belong to.

i_type The initializer for ew::View3Item::type.

14.33.3 Member Function Documentation

14.33.3.1 void ew::View3Image::set_image (const ew::DataflowImage3 * *im*)

Parameters

im A pointer to the new image node to display, or 0.

14.33.3.2 const ew::DataflowImage3 * ew::View3Image::get_image ( ) const [inline]

Returns

A pointer to the current image node being displayed, or 0.

14.33.3.3 void ew::View3Image::set_color (const unsigned char * *rgb*)

Parameters

[in] ***rgb*** This points to an array of size 3 containing the new colour for the image.

14.33.3.4 const unsigned char * ew::View3Image::get_color ( ) const [inline]

Returns

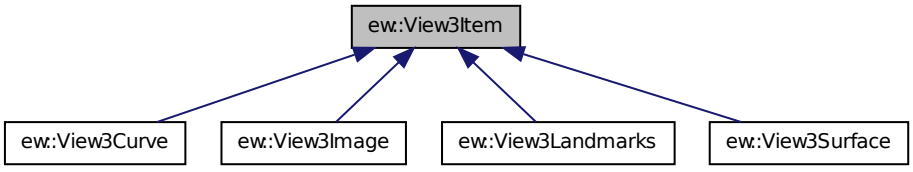
A pointer to the current colour for the image.

14.34 ew::View3Item Class Reference

Item Base Class.

`#include <ew/View3Item.h>`

Inheritance diagram for ew::View3Item:



Public Member Functions

- bool get_state () const
- void set_state (bool st)
- virtual const ew::Bbox3 * get_bbox ()=0
- int get_index () const
- void destroy ()

Public Attributes

- ew::DataflowNetwork *const network
- ew::View3Widget *const view
- const int type

14.34.1 Detailed Description

ew::View3Item is the base class for items displayed in ew::View_widget.

ew::View3Item is a class without assignment or comparison. There are private member variables.

Items must be created with `new` and then belong to the ew::View3Widget. They must only be deleted indirectly with the destroy method. Remaining items will be automatically deleted when the ew::View3Widget is destroyed.

14.34.2 Member Function Documentation

14.34.2.1 bool ew::View3Item::get_state ( ) const [inline]

An item is displayed iff the state is `true`.

Returns

The current state of the item.

14.34.2.2 void ew::View3Item::set_state (bool *i*)

Parameters

i The new state of the item.

14.34.2.3 const ew::Bbox3 * ew::View3Item::get_bbox ( ) [pure virtual]

Returns

A pointer to a bounding box for the item as currently being displayed.

14.34.2.4 int ew::View3Item::get_index ( ) const [inline]

Returns

The current index of the item in the widget's item list.

14.34.2.5 void ew::View3Item::destroy ( ) [inline]

This removes the item from the widget and destroys it.

14.34.3 Member Data Documentation

14.34.3.1 ew::DataflowNetwork *const ew::View3Item::network

This points to the ew::DataflowNetwork that the view was created with.

14.34.3.2 ew::View3Widget *const ew::View3Item::view

This points to the ew::View3Widget the this ew::View3Item was created with.

14.34.3.3 const int ew::View3Item::type

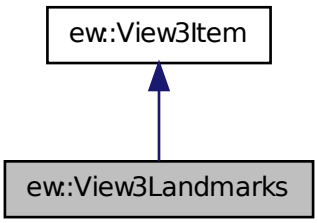
This is an arbitrary value supplied in the constructor. It is client data, not touched by ew::View3Item or ew::View3Widget.

14.35 ew::View3Landmarks Class Reference

Landmarks Item.

`#include <ew/View3Landmarks.h>`

Inheritance diagram for ew::View3Landmarks:



Public Types

- enum symbol_t { SYMBOL_CROSS, SYMBOL_CIRCLE, SYMBOL_DOT }

Public Member Functions

- View3Landmarks (ew::View3Widget *view, int i_type=0)
- void set_form (const ew::DataflowForm3 *frm)
- const ew::DataflowForm3 * get_form () const
- void set_color (const unsigned char *rgb)
- const unsigned char * get_color () const
- void set_highlight (int ps, int i)
- int get_highlight_ps () const
- int get_highlight_i () const
- void set_symbol (int sym)
- int get_symbol () const

Static Public Attributes

- static const int N_SYMBOLS = 3
- static const char *const symbol_names [ ]

14.35.1 Detailed Description

ew::View3Landmarks is an ew::View3Widget item to display the landmarks in a ew::DataflowForm3.

ew::View3Landmarks is a class without assignment or comparison. There are private member variables.

14.35.2 Member Enumeration Documentation

14.35.2.1 enum ew::View3Landmarks::symbol_t

These are the possible values for set_symbol and get_symbol. The first, with value zero, is SYMBOL_CROSS.

Enumerator:

SYMBOL_CROSS A cross symbol.

SYMBOL_CIRCLE A circle symbol.

SYMBOL_DOT A dot symbol.

14.35.3 Constructor & Destructor Documentation

14.35.3.1 ew::View3Landmarks::View3Landmarks (ew::View3Widget * *i_view,* int *i_type* = *0*) [explicit]

This creates an item.

Parameters

view_i The view widget the item should belong to.

type_i The initializer for ew::View3Item::type.

14.35.4 Member Function Documentation

14.35.4.1 void ew::View3Landmarks::set_form (const ew::DataflowForm3 * *frm*)

Parameters

frm A pointer to the new form node to display, or 0.

14.35.4.2 const ew::DataflowForm3 * ew::View3Landmarks::get_form ( ) const [inline]

Returns

A pointer to the current form node being displayed, or 0.

14.35.4.3 void ew::View3Landmarks::set_color (const unsigned char * *rgb*)

Parameters

[in] ***rgb*** This points to an array of size 3 containing the new colour for the landmark symbols.

14.35.4.4 const unsigned char * ew::View3Landmarks::get_color ( ) const [inline]

Returns

A pointer to the current colour for the landmark symbols.

14.35.4.5 void ew::View3Landmarks::set_highlight (int *ps*, int *i*)

Parameters

ps The pointset of the landmark to highlight.

i The index of the landmark within the pointset of the landmark to highlight.

14.35.4.6 int ew::View3Landmarks::get_highlight_ps ( ) const [inline]

Returns

The currently highlighted pointset index, or -1.

14.35.4.7 int ew::View3Landmarks::get_highlight_i ( ) const [inline]

Returns

The currently highlighted landmark index within the pointset, or -1.

14.35.4.8 void ew::View3Landmarks::set_symbol (int *sym*)

Parameters

sym The new landmark symbol.

14.35.4.9 int ew::View3Landmarks::get_symbol ( ) const [inline]

Returns

The current landmark symbol.

14.35.5 Member Data Documentation

14.35.5.1 const int ew::View3Landmarks::N_SYMBOLS = 3 [static]

They number of possible symbol_t values.

14.35.5.2 const char *const ew::View3Landmarks::symbol_names [static]

Initial value:

```
{
 "cross",
 "circle",
 "dot",
 0
}
```

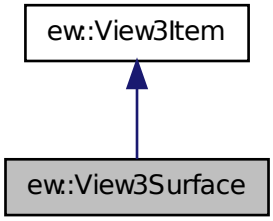
The names of the symbol_t choices.

14.36 ew::View3Surface Class Reference

Surface Item.

`#include <ew/View3Surface.h>`

Inheritance diagram for ew::View3Surface:



Public Member Functions

- View3Surface (ew::View3Widget *view, int i_type=0)
- void set_surface (const ew::DataflowSurface3 *sur)
- const ew::DataflowSurface3 * get_surface () const
- void set_front_color (const unsigned char *rgb)
- const unsigned char * get_front_color () const
- void set_back_color (const unsigned char *rgb)
- const unsigned char * get_back_color () const

14.36.1 Detailed Description

ew::View3Surface is an ew::View3Widget item to display a ew::DataflowSurface3, a triangulated surface.

ew::View3Surface is a class without assignment or comparison. There are private member variables.

14.36.2 Constructor & Destructor Documentation

14.36.2.1 ew::View3Surface::View3Surface (ew::View3Widget * *v*, int *i_type* = *0*) [explicit]

This creates an item.

Parameters

v The view widget the item should belong to.

type_i The initializer for ew::View3Item::type.

14.36.3 Member Function Documentation

14.36.3.1 void ew::View3Surface::set_surface (const ew::DataflowSurface3 * *s*)

Parameters

s A pointer to the new surface node to display, or 0.

14.36.3.2 const ew::DataflowSurface3 * ew::View3Surface::get_surface ( ) const [inline]

Returns

A pointer to the current surface node being displayed, or 0.

14.36.3.3 void ew::View3Surface::set_front_color (const unsigned char * *rgb*)

Parameters

[in] ***rgb*** This points to an array of size 3 containing the new colour for the front of the surface.

14.36.3.4 const unsigned char * ew::View3Surface::get_front_color ( ) const [inline]

Returns

A pointer to the current colour for the front of the surface.

14.36.3.5 void ew::View3Surface::set_back_color (const unsigned char * *rgb*)

Parameters

[in] ***rgb*** This points to an array of size 3 containing the new colour for the back of the surface.

14.36.3.6 const unsigned char * ew::View3Surface::get_back_color ( ) const [inline]

Returns

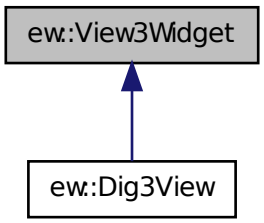
A pointer to the current colour for the back of the surface.

14.37 ew::View3Widget Class Reference

3D viewing widget

`#include <ew/View3Widget.h>`

Inheritance diagram for ew::View3Widget:



Public Member Functions

- View3Widget (ew::DataflowNetwork *i_network)
- virtual ~View3Widget ()
- void reset ()
- void set_clip_ratio (double cr)
- double get_clip_ratio () const
- void set_view_mapping (const ew::Transform3 *tr)
- void view_post_compose (const ew::Transform3 *tr)
- const ew::Transform3 * get_view_mapping () const
- void get_bbox (ew::Bbox3 *b) const
- void get_pointer_location (double *loc, double x, double y, double z=0.0) const
- void get_center_location (double *loc) const
- void get_pointer_translate_tr (ew::Transform3 *tr, double x, double y, double z) const
- void get_pointer_roll_tr (ew::Transform3 *tr, double x, double y) const
- void get_pointer_twist_tr (ew::Transform3 *tr, double x0, double y0, double x1, double y1, double acc) const
- void get_pointer_scale_tr (ew::Transform3 *tr, double x0, double y0, double x1, double y1, double acc) const
- void get_jump_tr (ew::Transform3 *tr, const double *loc, bool do_z) const
- bool get_bbox_tr (ew::Transform3 *tr, const ew::Bbox3 *bbox, bool do_z, bool do_scale) const
- void set_background_color (const unsigned char *rgb)
- const unsigned char * get_background_color () const
- void set_highlight_color (const unsigned char *rgb)

- const unsigned char * get_highlight_color () const
- ew::View3Item * get_highlight_item () const
- void clear_highlight ()
- void set_use_depth (bool d)
- bool get_use_depth () const
- void move_item (ew::View3Item *it, int i)
- ew::View3Item *const * get_items () const
- int get_n_items () const
- bool pick (double x, double y, double sz, double burrow, ew::View3Item *constrain_it, int constrain_cmpt, int constrain_dim, ew::View3Item **pick_it, int *pick_cmpt, int *pick_dim, double *pick_z)
- void pickv (std::vector< ew::View3Pick > &outp, double x, double y, double sz)

Public Attributes

- ew::DataflowNetwork *const network

Protected Member Functions

- void init_gl_context ()
- void set_window_size (int w, int h)
- void render ()
- void idle_handler ()
- void set_currently_unrendered ()
- void destruction_in_progress ()

Private Member Functions

- virtual void schedule_idle_handler_cb ()=0
- virtual void redraw_cb ()=0

14.37.1 Detailed Description

This class implements a widget that manages a 3D scene consisting of a list of items.

Items, derived from ew::View3Item, may be created in the widget.

This class is independent of any particular toolkit and it is intended that the class will be inherited by a wrapper that implements the necessary interface with the toolkit.

All view widgets associated with a particular ew::DataflowNetwork must share OpenGL contexts. The context must have a depth buffer. Many class methods require this context to be active when they are called.

Items in a widget are kept in a list in rendering order. If the depth buffer is not being used, later items will be rendered on top of earlier items. It the depth buffer is being used, the rendering order only effects items with exactly the same depth. Items are put at the end of the order initially on creation.

14.37.2 Constructor & Destructor Documentation

14.37.2.1 ew::View3Widget::View3Widget (ew::DataflowNetwork * *i_network*) [explicit]

Create an empty view widget.

Parameters

[in] ***i_network*** a pointer to a network to associate this view with

14.37.2.2 ew::View3Widget::~View3Widget ( ) [virtual]

Destroy the widget and all its items. The OpenGL context should be active when this is called.

14.37.3 Member Function Documentation

14.37.3.1 void ew::View3Widget::reset ( )

Delete all items in the widget and reset it to its initial state.

14.37.3.2 void ew::View3Widget::set_clip_ratio (double *cr*)

Set the clip ratio. The viewing volume width and height are determined by the window dimensions as specified in set_window_size. The clipping ratio is the ratio of the viewing volume depth to the smaller of its width and height. The viewing volume coordinate system is centered in the viewing volume, with axes, in order, pointing right, up and out of the screen, and with homogeneous scale such that the range of coordinates in the direction of the smaller of the width and height is [-1.0, 1.0]. The clip ratio is initially 1.0. It must be positive, but also not unreasonably small or large values.

Parameters

cr the new clipping ratio

14.37.3.3 double ew::View3Widget::get_clip_ratio ( ) const [inline]

Returns

the current clipping ratio

14.37.3.4 void ew::View3Widget::set_view_mapping (const ew::Transform3 * *tr*)

Set the view mapping. The view mapping is the transformation from model coordinates to viewing volume coordinates.

Parameters

[in] ***tr*** the address of a transform containing the new view mapping

Reimplemented in ew::Dig3View.

14.37.3.5 void ew::View3Widget::view_post_compose (const ew::Transform3 * *tr*)

Composes the view mapping with a transform on the left. This is for incremental changes to the view mapping relative to the viewing volume coordinate system.

Parameters

[in] ***tr*** the address of a transform containing the view mapping increment

Reimplemented in ew::Dig3View.

14.37.3.6 const ew::Transform3 * ew::View3Widget::get_view_mapping ( ) const [inline]

Returns

the address of a transform containing the current view mapping

14.37.3.7 void ew::View3Widget::get_bbox (ew::Bbox3 * *b*) const

Calculate the bounding box of all items displayed in the widget.

Parameters

[out] ***b*** where to store the result

14.37.3.8 void ew::View3Widget::get_pointer_location (double * *loc*, double *x*, double *y*, double *z* = *0.0*) const

Calculate the location of the point with given pixel window coordinates.

Parameters

[out] ***loc*** The address of an array of size 3 to store the location in

x,y,z the pixel window coordinates (right from left side, down from top and into window from the midplane of the viewing volume)

14.37.3.9 void ew::View3Widget::get_center_location (double * *loc*) const

Calculate the location of center of the space currently in view.

Parameters

[out] ***loc*** the address of an array of size 3 to store the location in

14.37.3.10 void ew::View3Widget::get_pointer_translate_tr (ew::Transform3 * *tr*, double *x*, double *y*, double *z*) const

Construct a translation suitable for view_post_compose from window pixel coordinate changes.

Parameters

[out] ***tr*** where to store the view mapping increment

x,y,z the coordinate changes

14.37.3.11 void ew::View3Widget::get_pointer_roll_tr (ew::Transform3 * *tr*, double *x*, double *y*) const

Constructs a roll-ball type rotation suitable for view_post_compose from window pixel coordinate changes.

Parameters

[out] ***tr*** where to store the view mapping increment

x,y the coordinate changes

14.37.3.12 void ew::View3Widget::get_pointer_twist_tr (ew::Transform3 * *tr*, double *x0*, double *y0*, double *x1*, double *y1*, double *acc*) const

Constructs a twist rotation, about the center of the window, suitable for view_post_compose from window pixel coordinate changes.

Parameters

[out] ***tr*** where to store the view mapping increment

x0,y0 the coordinate initial values

x1,y1 the coordinate final values

acc an acceleration factor that scales the rotation relative to the rotation indicated by the coordinate changes

14.37.3.13 void ew::View3Widget::get_pointer_scale_tr (ew::Transform3 * *tr*, double *x0*, double *y0*, double *x1*, double *y1*, double *acc*) const

Constructs a scaling about the center of the window suitable for view_post_compose from window pixel coordinate changes.

Parameters

[out] ***tr*** where to store the view mapping increment

x0,y0 the coordinate initial values

x1,y1 the coordinate final values

acc an acceleration factor that scales the scale relative to the scale indicated by the coordinate changes

14.37.3.14 void ew::View3Widget::get_jump_tr (ew::Transform3 * *tr*, const double * *loc*, bool *do_z*) const

Construct a translation suitable for view_post_compose that would put the specified point (in model space, not viewing space) at the center of the view.

Parameters

[out] ***tr*** where to store the view mapping increment

[in] ***loc*** the address of an array of size 3 containing the point coordinates

do_z whether to translate in the z direction so that the point is at the center of the view in all 3 directions

14.37.3.15 bool ew::View3Widget::get_bbox_tr (ew::Transform3 * *tr*, const ew::Bbox3 * *bbox*, bool *do_z*, bool *do_scale*) const

Constructs a translation suitable for view_post_compose that would put the specified bounding box (in model space, not viewing space) at the center of the view. The orientation of the view is not effected. This will return `false` and not otherwise do anything if the bounding box is empty, or if is a single point and *do_scale* is `true`.

Parameters

`[out]` ***tr*** where to store the view mapping increment, if successful

`[in]` ***bbox*** the address of the bounding box

do_z whether to translate in the z direction

do_scale whether to scale so that the box fits comfortably in the view

Returns

`true` if successful

14.37.3.16 void ew::View3Widget::set_background_color (const unsigned char * *rgb*)

Set the background color. The background is by default [0, 0, 0].

Parameters

`[in]` ***rgb*** the address of an array of size 3 containing the new background colour components

14.37.3.17 const unsigned char * ew::View3Widget::get_background_color ( ) const `[inline]`

Returns

the address of an array containing the current background colour components

14.37.3.18 void ew::View3Widget::set_highlight_color (const unsigned char * *rgb*)

Set the highlight color. The highlight colour is by default [255, 0, 0].

Parameters

[in] ***rgb*** the address of an array of size 3 containing the new highlight colour components

14.37.3.19 const unsigned char * ew::View3Widget::get_highlight_color ( ) const [inline]

Returns

the address of an array containing the current highlight colour components

14.37.3.20 ew::View3Item * ew::View3Widget::get_highlight_item ( ) const [inline]

Returns

the address of the current highlighted item, or 0

14.37.3.21 void ew::View3Widget::clear_highlight ( )

Clear the current highlighted item, if any.

14.37.3.22 void ew::View3Widget::set_use_depth (bool *d*)

Turn on or off the use of the depth buffer. It is initially `true`.

Parameters

d whether to use the depth buffer

14.37.3.23 bool ew::View3Widget::get_use_depth ( ) const [inline]

Returns

the current setting

14.37.3.24 void ew::View3Widget::move_item (ew::View3Item * *it*, int *n*)

Move an item in the rendering order.

Parameters

[in] ***it*** a pointer to the item to move

n the position in the order to move the item to

14.37.3.25 ew::View3Item *const * ew::View3Widget::get_items ( ) const [inline]

Returns

a pointer to the list of items

14.37.3.26 int ew::View3Widget::get_n_items ( ) const [inline]

Returns

the size of the list of items

14.37.3.27 bool ew::View3Widget::pick (double *x*, double *y*, double *sz*, double *burrow*, ew::View3Item * *constrain_it*, int *constrain_cmpt*, int *constrain_dim*, ew::View3Item ** *pick_it*, int * *pick_cmpt*, int * *pick_dim*, double * *pick_z*)

Pick an item fragment rendered in a sub-window of the current view. Fragments of the lowest dimension visible are picked. With a depth buffer, the nearest eligible item fragment of this dimensions is picked, provided it is not behind, with a tolerance, any surface fragment. This is to allow picking of, for example, points in an inclined surface where otherwise, some part of the surface would always be closer than the point. Without a depth buffer, the latest eligible item fragment of this dimension in the rendering order is picked, provided it is not before any surface fragment. The meaning of an item component depends on the item. The *pick_z* result is in pixels, with 0 at the center of the viewing volume, and positive further away. It is suitable for use in get_pointer_location.

Parameters

x,y the pixel window pointer coordinates

sz the side of the square, in pixels, centered on *x* and *y* within which rendered items are eligible for picking.

burrow the tolerance in the depth buffer, for an eligible item to be picked behind a surface.

`[in]` ***constrain_it*** if non-zero, only fragments rendered by this item will be eligible to be picked

constrain_cmpt if non-negative, only fragments rendered as part of this component in the item will be eligible for picking

`[in]` ***constrain_dim*** if non-negative, only fragments of this dimension (0, 1 or 2) will be eligible for picking

`[out]` ***pick_it*** the location where a pointer to the picked item should be stored, if non-zero and if the pick was successful

`[out]` ***pick_cmpt*** the location where the component of the picked item should be stored, if non-zero and if the pick was successful

`[out]` ***pick_dim*** the location where the dimension of the picked item should be stored, if non-zero and if the pick was successful

`[out]` ***pick_z*** the location where the z coordinate of the nearest part of the picked item should be stored, if non-zero and if the pick was successful

14.37.3.28 void ew::View3Widget::pickv (std::vector< ew::View3Pick > & *outp,* double *x,* double *y,* double *sz*)

Find all item fragment picks, in order of most prominent first.

Parameters

`[out]` ***outp*** a reference to the vector to store the output list in

x,y the pixel window coordinates

sz the side of the square, in pixels, centered on *x* and *y* within which rendered items are eligible for picking

14.37.3.29 void ew::View3Widget::init_gl_context ( ) [protected]

Perform initialization related to the OpenGL context. This should be called after construction, once the context is active.

14.37.3.30 void ew::View3Widget::set_window_size (int *w,* int *h*) [protected]

Performs initialization related to the window size. This should be called after construction and whenever the window is resized.

Parameters

w,h the new window width and height

14.37.3.31 void ew::View3Widget::render ( ) [protected]

Render the scene in the current OpenGL context and associated window. If the context has a double buffer, this will render to the back buffer. The buffers must then be swapped.

14.37.3.32 void ew::View3Widget::idle_handler ( ) [protected]

schedule_idle_handler_cb should arrange for this function to be called at an appropriate time.

14.37.3.33 void ew::View3Widget::set_currently_unrendered ( ) [protected]

This should be called when the window associated with this widget is unmapped, to avoid unnecessary activity. A call to render undoes the effect of this.

14.37.3.34 void ew::View3Widget::destruction_in_progress ( ) [protected]

This should be called when a derived object is being destroyed, to prevent any callbacks being invoked.

14.37.3.35 void ew::View3Widget::schedule_idle_handler_cb ( ) [private, pure virtual]

This is a callback that must be implemented by the inheriting class. It must arrange for idle_handler to be called later when the GUI is idle. Outstanding handlers must be cancelled by the inheriting class on destruction.

14.37.3.36 void ew::View3Widget::redraw_cb ( ) [private, pure virtual]

This is a callback than must be implemented in the inheriting class. It must arrange for the widget to be rendered immediately.

14.37.4 Member Data Documentation

14.37.4.1 ew::DataflowNetwork *const ew::View3Widget::network

This points to the ew::DataflowNetwork that this ew::View3Widget was created with.

Index