ncut.kdevelop Reference Manual

1.0

Generated by Doxygen 1.4.6

Thu Jun 22 14:43:25 2006

Contents

Chapter 1

ncut.kdevelop Directory Hierarchy

1.1 ncut.kdevelop Directories

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

Chapter 2

ncut.kdevelop Hierarchical Index

2.1 ncut.kdevelop Class Hierarchy

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

Chapter 3

ncut.kdevelop Class Index

3.1 ncut.kdevelop Class List

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

Chapter 4

ncut.kdevelop Directory Documentation

4.1 /home/stefan/cpp/ncut/debug/ Directory Reference

Files

- file **config.h**

4.2 /home/stefan/cpp/ncut/ Directory Reference

Directories

- directory debug
- directory optimized
- directory segment
- directory src

Files

- file **autocompare.cpp**
- file **autocompare2.cpp**
- file **autocompare_area.cpp**
- file **compare.c**
- file **compare.cpp**
- file **compare2.c**

4.3 /home/stefan/cpp/ncut/optimized/ Directory Reference

Files

- file **config.h**

4.4 /home/stefan/cpp/ncut/segment/ Directory Reference

Files

- file **CImageSegmentationIndexFile.cpp**
- file **CImageSegmentationIndexFile.h**
- file **convolve.h**
- file **disjoint-set.h**
- file **filter.h**
- file **image.h**
- file **imconv.h**
- file **imutil.h**
- file **misc.h**
- file **pnmfile.h**
- file **segment-graph.h**
- file **segment-image.h**
- file **segment.cpp**
- file **stdafx.h**
- file **TImageSegmentation.h**

4.5 /home/stefan/cpp/ncut/src/ Directory Reference

Files

- file **CImageSegmentationIndexFile.cpp**
- file **CImageSegmentationIndexFile.h**
- file **matrix.cpp**
- file **matrix.h**
- file **motionProfile.cpp**
- file **motionProfile.h**
- file **ncut.cpp**
- file **ncutCore.cpp**
- file **ncutCore.h**
- file **ncutException.cpp**
- file **ncutException.h**
- file **ncutImage.cpp**
- file **ncutImage.h**
- file **ncutMemory.h**
- file **ncutSetting.h**
- file **ncutVis.cpp**
- file **ncutVis.h**
- file **stdafx.h**
- file **stlini.cpp**
- file **stlini.h**
- file **TImageSegmentation.h**

Chapter 5

ncut.kdevelop Class Documentation

5.1 auto_array< _Tp > Class Template Reference

A simple smart array providing strict ownership semantics.

`#include <ncutMemory.h>`

Public Types

- typedef _Tp element_type
  The pointed-to type.

Public Member Functions

- auto_array (element_type *__p=0) throw ()
  An auto_array is usually constructed from a raw array.
- auto_array (auto_array &__a) throw ()
  An auto_array can be constructed from another auto_array.
- template<typename _Tp1> auto_array (auto_array< _Tp1 > &__a) throw ()
  An auto_array can be constructed from another auto_array.
- auto_array & operator= (auto_array &__a) throw ()
  auto_array assignment operator.
- template<typename _Tp1> auto_array & operator= (auto_array< _Tp1 > &__a) throw ()
  auto_array assignment operator.
- ~auto_array ()
- element_type & operator[ ] (unsigned int i) const throw ()
  Smart array element access.
- element_type & operator * () const throw ()

Smart array dereferencing.

- element_type * operator → () const throw ()

  Smart array dereferencing.

- element_type * get () const throw ()

  Bypassing the smart array.

- element_type * release () throw ()

  Bypassing the smart array.

- void reset (element_type *__p=0) throw ()

  Forcibly deletes the managed array.

- auto_array (auto_array_ref< element_type > __ref) throw ()

  Automatic conversions.

- auto_array & **operator=** (auto_array_ref< element_type > __ref) throw ()
- template<typename _Tp1> **operator auto_array_ref** () throw ()
- template<typename _Tp1> **operator auto_array** () throw ()

5.1.1 Detailed Description

template<typename _Tp> class auto_array< _Tp >

A simple smart array providing strict ownership semantics.

An `auto_array` owns the array it holds a pointer to. Copying an `auto_array` copies the pointer and transfers ownership to the destination. If more than one `auto_array` owns the same array at the same time the behavior of the program is undefined.

The uses of `auto_array` include providing temporary exception-safety for dynamically allocated memory, passing ownership of dynamically allocated memory to a function, and returning dynamically allocated memory from a function. `auto_array` does not meet the CopyConstructible and Assignable requirements for Standard Library `container` elements and thus instantiating a Standard Library container with an `auto_array` results in undefined behavior.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 template<typename _Tp> auto_array< _Tp >::auto_array (element_type * __p = 0) throw () [inline, explicit]

An auto_array is usually constructed from a raw array.

Parameters:

__p A pointer (defaults to NULL).

This object now *owns* the array pointed to by *p*.

5.1.2.2 template<typename _Tp> auto_array< _Tp >::auto_array (auto_array< _Tp > & __a) throw () [inline]

An auto_array can be constructed from another auto_array.

Parameters:

__a Another auto_array of the same type.

This object now *owns* the object previously owned by *a*, which has given up ownsership.

5.1.2.3 template<typename _Tp> template<typename _Tp1> auto_array< _Tp >::auto_array (auto_array< _Tp1 > & __a) throw () [inline]

An auto_array can be constructed from another auto_array.

Parameters:

__a Another auto_array of a different but related type.

A pointer-to-Tp1 must be convertible to a pointer-to-Tp/element_type.

This object now *owns* the object previously owned by *a*, which has given up ownsership.

5.1.2.4 template<typename _Tp> auto_array< _Tp >::~auto_array () [inline]

When the auto_array goes out of scope, the object it owns is deleted. If it no longer owns anything (i.e., `get()` is `NULL`), then this has no effect.

The C++ standard says there is supposed to be an empty throw specification here, but omitting it is standard conforming. Its presence can be detected only if _Tp::~_Tp() throws, but this is prohibited. [17.4.3.6]/2

5.1.2.5 template<typename _Tp> auto_array< _Tp >::auto_array (auto_array_ref< element_type > __ref) throw () [inline]

Automatic conversions.

These operations convert an auto_array into and from an auto_arary_ref automatically as needed. This allows constructs such as

```
auto_array<Derived>  func_returning_auto_array(.....);
...
auto_array<Base> ptr = func_returning_auto_array(.....);
```

5.1.3 Member Function Documentation

5.1.3.1 template<typename _Tp> element_type* auto_array< _Tp >::get () const throw () [inline]

Bypassing the smart array.

Returns:

The raw pointer being managed.

You can get a copy of the pointer that this object owns, for situations such as passing to a function which only accepts a raw pointer.

Note:
This auto_array still owns the memory.

5.1.3.2 template<typename _Tp> element_type& auto_array< _Tp >::operator * () const throw () [inline]

Smart array dereferencing.

If this auto_array no longer owns anything, then this operation will crash. (For a smart pointer, "no longer owns anything" is the same as being a null pointer, and you know what happens when you dereference one of those...)

5.1.3.3 template<typename _Tp> element_type* auto_array< _Tp >::operator → () const throw () [inline]

Smart array dereferencing.

This returns the pointer itself, which the language then will automatically cause to be dereferenced.

5.1.3.4 template<typename _Tp> template<typename _Tp1> auto_array& auto_array< _Tp >::operator= (auto_array< _Tp1 > & __a) throw () [inline]

auto_array assignment operator.

Parameters:
__a Another auto_array of a different but related type.

A pointer-to-Tp1 must be convertible to a pointer-to-Tp/element_type.

This object now *owns* the object previously owned by *a*, which has given up ownsership. The object that this one *used* to own and track has been deleted.

5.1.3.5 template<typename _Tp> auto_array& auto_array< _Tp >::operator= (auto_array< _Tp > & __a) throw () [inline]

auto_array assignment operator.

Parameters:
__a Another auto_array of the same type.

This object now *owns* the object previously owned by *a*, which has given up ownsership. The object that this one *used* to own and track has been deleted.

5.1.3.6]

template<typename _Tp> element_type& auto_array< _Tp >::operator[ ] (unsigned int *i*) const throw () [inline]

Smart array element access.

If this auto_array no longer owns anything, then this operation will crash. (For a smart pointer, "no longer owns anything" is the same as being a null pointer, and you know what happens when you dereference one of those...)

5.1.3.7 template<typename _Tp> element_type* auto_array< _Tp >::release () throw () [inline]

Bypassing the smart array.

Returns:
The raw pointer being managed.

You can get a copy of the pointer that this object owns, for situations such as passing to a function which only accepts a raw pointer.

Note:
This auto_array no longer owns the memory. When this object goes out of scope, nothing will happen.

5.1.3.8 template<typename _Tp> void auto_array< _Tp >::reset (element_type * __p = 0) throw () [inline]

Forcibly deletes the managed array.

Parameters:
__p A pointer (defaults to NULL).

This object now *owns* the object pointed to by *p*. The previous object has been deleted.

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

- /home/stefan/cpp/ncut/src/ncutMemory.h

5.2 auto_array_ref< _Tp1 > Struct Template Reference

`#include <ncutMemory.h>`

Public Member Functions

- **auto_array_ref** (_Tp1 *__p)

Public Attributes

- _Tp1 * **_M_ptr**

5.2.1 Detailed Description

template<typename _Tp1> struct auto_array_ref< _Tp1 >

A wrapper class to provide auto_array with reference semantics. For example, an auto_array can be assigned (or constructed from) the result of a function which returns an auto_array by value.

All the auto_array_ref stuff should happen behind the scenes.

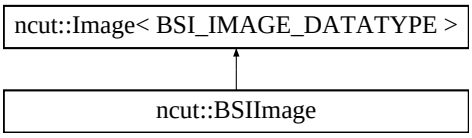
The documentation for this struct was generated from the following file:

- /home/stefan/cpp/ncut/src/ncutMemory.h

5.3 ncut::BSIImage Class Reference

`#include <ncutImage.h>`

Inheritance diagram for ncut::BSIImage::



Public Member Functions

- unsigned long nSeg () const

  Get the number of segments of the images.

- int load (const char *filename, unsigned long modNum=0)

  Load the BSI image file "filename".

- BSIImage ()

  BSIImage empty constructor.

- BSIImage (const BSIImage &clone)

  BSIImage copy constructor.

- BSIImage (const char *filename, unsigned long &modNum)

  A BSIImage can be constructed using a BSI image stored on disk.

- virtual ~BSIImage ()

  &BSIImage destructor.

- virtual BSIImage & operator= (const BSIImage &clone)

  BSIImage assignment operator.

Protected Attributes

- unsigned long nSeg_

5.3.1 Detailed Description

`BSIImage` adds BSI-specific load and functionality to its base class `Image`. It also provides information about the number of segments and the maximum label number stored.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 ncut::BSIImage::BSIImage ()

BSIImage empty constructor.

Creates an empty image with width=height=channels=0.

5.3.2.2 ncut::BSIImage::BSIImage (const BSIImage & *clone*)

BSIImage copy constructor.

Parameters:

clone The `BSIImage` to be cloned.

5.3.2.3 ncut::BSIImage::BSIImage (const char * *filename*, unsigned long & *modNum*)

A BSIImage can be constructed using a BSI image stored on disk.

Parameters:

filename A char array specifying the location of a BSI image.

modNum A constant to be added to all labels.

Creates a `BSIImage` from the BSI image located at `filename` and optionally adds the constant `modNum` to all labels in the BSI image to avoid duplicate labels in a sequence of BSI images.

5.3.3 Member Function Documentation

5.3.3.1 int ncut::BSIImage::load (const char * *filename*, unsigned long *modNum* = `0`)

Load the BSI image file "filename".

Parameters:

filename A char array specifying the location of a BSI image.

modNum A constant to be added to all labels.

Returns:

0 if successful.

Replaces this BSI image with the BSI image located at `filename` and optionally adds the constant `modNum` to all labels in the BSI image to avoid duplicate labels in a sequence of BSI images.

5.3.3.2 unsigned long ncut::BSIImage::nSeg () const `[inline]`

Get the number of segments of the images.

Returns:

The number of segemtns of the images

5.3.3.3 BSIImage & ncut::BSIImage::operator= (const BSIImage & *clone*) [virtual]

BSIImage assignment operator.

Parameters:
clone The `BSIImage` to be cloned.

Returns:
This object.

5.3.4 Member Data Documentation

5.3.4.1 unsigned long ncut::BSIImage::nSeg_ [protected]

The number of segments in this BSI image

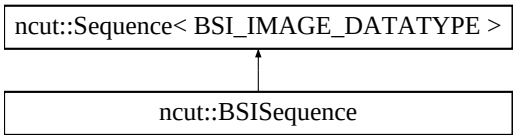
The documentation for this class was generated from the following files:

- /home/stefan/cpp/ncut/src/ncutImage.h
- /home/stefan/cpp/ncut/src/ncutImage.cpp

5.4 ncut::BSISequence Class Reference

`#include <ncutImage.h>`

Inheritance diagram for ncut::BSISequence::



Public Member Functions

- BSIImage * **operator[ ]** (unsigned long)
- const BSIImage * **operator[ ]** (unsigned long) const
- unsigned long nSeg () const

  Get the number of segments in the sequence.

- int pushThrough (const Image< BSI_IMAGE_DATATYPE > &frame)

  Push a frame onto the sequence and pop the first frame.

- int pushThrough (const BSIImage &frame)

  Push a frame onto the sequence and pop the first frame.

- int pushFrame (const Image< BSI_IMAGE_DATATYPE > &frame)

  Push a frame to the back of the sequence.

- int pushFrame (const BSIImage &frame)

  Push a frame to the back of the sequence.

- void popFrame ()

  Pop the last frame of the sequence.

- int clear ()

  Remove all frames from the sequence.

- BSISequence ()

  BSISequence empty constructor

- BSISequence (const BSISequence &clone)

  BSISequence copy constructor.

- virtual ~BSISequence ()

  BSISequence destructor.

- virtual BSISequence & operator= (const BSISequence &clone)

  BSISequence assignment operator.

Protected Attributes

- unsigned long nSeg_

5.4.1 Detailed Description

A `BSISequence` keeps track of the total number of segments in the sequence. Only images of type `BSIImage` are allowed in a `BSISequence`.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 ncut::BSISequence::BSISequence ()

BSISequence empty constructor

Creates an empty `BSISequence` with=height=channels=segments=0.

5.4.2.2 ncut::BSISequence::BSISequence (const BSISequence & *clone*)

BSISequence copy constructor.

Parameters:

clone The `BSISequence` to be cloned.

5.4.3 Member Function Documentation

5.4.3.1 int ncut::BSISequence::clear () `[virtual]`

Remove all frames from the sequence.

Returns:

0 if successful.

Deletes all frames in the sequence and sets width, height, number of channels and number of segments to zero. The length of the sequence is zero.

Reimplemented from ncut::Sequence< BSI_IMAGE_DATATYPE >.

5.4.3.2 unsigned long ncut::BSISequence::nSeg () const `[inline]`

Get the number of segments in the sequence.

Returns:

The number of segments in the seuqence

5.4.3.3 BSISequence & ncut::BSISequence::operator= (const BSISequence & *clone*) [virtual]

BSISequence assignment operator.

Parameters:
clone The `BSISequence` to be cloned.

Returns:
This object.

5.4.3.4 void ncut::BSISequence::popFrame () [virtual]

Pop the last frame of the sequence.

Deletes the last frame and removes it from the sequence. This does nothing if the sequence is empty.

Reimplemented from ncut::Sequence< BSI_IMAGE_DATATYPE >.

5.4.3.5 int ncut::BSISequence::pushFrame (const BSIImage & *frame*)

Push a frame to the back of the sequence.

Parameters:
frame The `Image` to be pushed onto the sequence.

Returns:
0 if successful.

Pushes a copy of `Image` to the back of the sequence. The length of the sequence is increased by one. Only BSI images of the same dimension as all other images in the sequence are accepted.

5.4.3.6 int ncut::BSISequence::pushFrame (const Image< BSI_IMAGE_DATATYPE > & *frame*) [virtual]

Push a frame to the back of the sequence.

Parameters:
frame The `Image` to be pushed onto the sequence.

Returns:
0 if successful.

Pushes a copy of `Image` to the back of the sequence. The length of the sequence is increased by one. Only BSI images of the same dimension as all other images in the sequence are accepted. `frame` is cast to a `BSIImage`.

Reimplemented from ncut::Sequence< BSI_IMAGE_DATATYPE >.

5.4.3.7 int ncut::BSISequence::pushThrough (const BSIImage & *frame*)

Push a frame onto the sequence and pop the first frame.

Parameters:
frame The `Image` to be pushed onto the sequence.

Returns:
0 if successful.

Pushes a copy of `Image` onto the sequence and removes the first frame of the sequence. The length of the sequence remains unchanged. Only BSI images of the same dimension as all other images in the sequence are accepted.

5.4.3.8 int ncut::BSISequence::pushThrough (const Image< BSI_IMAGE_DATATYPE > & *frame*) [virtual]

Push a frame onto the sequence and pop the first frame.

Parameters:
frame The `Image` to be pushed onto the sequence.

Returns:
0 if successful.

Pushes a copy of `Image` onto the sequence and removes the first frame of the sequence. The length of the sequence remains unchanged. Only BSI images of the same dimension as all other images in the sequence are accepted. `frame` is cast to a `BSIImage`.

Reimplemented from ncut::Sequence< BSI_IMAGE_DATATYPE >.

5.4.4 Member Data Documentation

5.4.4.1 unsigned long ncut::BSISequence::nSeg_ [protected]

The total number of segments in the sequence.

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

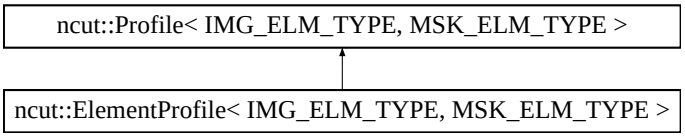
- /home/stefan/cpp/ncut/src/ncutImage.h
- /home/stefan/cpp/ncut/src/ncutImage.cpp

5.5 ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE > Class Template Reference

ElementProfile handles the similarity information of the elements in a frame window.

`#include <motionProfile.h>`

Inheritance diagram for ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::



Public Member Functions

- const Image< IMG_ELM_TYPE > * frame (unsigned int num) const
  Get the n-th frame in this frame window.
- const Image< MSK_ELM_TYPE > * mask (unsigned int num) const
  Get the n-th mask in this frame window.
- const MSK_ELM_TYPE & mask (unsigned int num, unsigned int pxl) const
  Get a pixel in the n-th mask of this frame window.
- const IMG_ELM_TYPE & frame (unsigned int num, unsigned int pxlelm) const
  Get a pixel in the n-th frame of this frame window.
- unsigned int width () const
  Get the width of the frames.
- unsigned int height () const
  Get the height of the frames.
- unsigned int nChl () const
  Get the number of channels of the frames.
- unsigned int nFrm () const
  Get the number of frames in the current frame window.
- unsigned int wSize () const
  Get the preset size of the frame window in frames.
- unsigned int nElm () const
  Get the number of elements in this profile.
- const std::vector< std::vector< unsigned int > > * pixels () const
  Get a list if pixels for each element.

- const std::vector< unsigned int > * pixels (unsigned int idx) const
  Get the pixels of an element.
- unsigned int frameOf (unsigned int idx) const
  Get the frame containing the element with index `idx`.
- const Matrix * neighbors () const
  Get the neighbourhood matrix of this profile.
- double hueVecX (unsigned int elm) const
  Get the mean hue vector x coordinate of an element.
- double hueVecY (unsigned int elm) const
  Get the mean hue vector y coordinate of an element.
- double lightness (unsigned int elm) const
  Get the mean lightness of an element.
- double saturation (unsigned int elm) const
  Get the mean saturation of an element.
- double intensity (unsigned int elm, unsigned int chl) const
  Get the mean intensity for one channel of an element.
- double locationX (unsigned int elm) const
  Get the mean location x-coordinate of an element.
- double locationY (unsigned int elm) const
  Get the mean location y-coordinate of an element.
- double locationZ (unsigned int elm) const
  Get the mean location z-coordinate of an element (frame number).
- unsigned int nPixels (unsigned int elm) const
  Get the number of pixels of an element.
- unsigned int minIdx (unsigned int frm) const
  Get the minimum element index in the given frame.
- unsigned int maxIdx (unsigned int frm) const
  Get the maximum element index in the given frame.
- int push (const Image< IMG_ELM_TYPE > *frame, const Image< MSK_ELM_TYPE > *mask)
  Calculate element info of a new frame.
- ElementProfile (unsigned int wSize, const Setting *setting)
  ElementProfile is constucted passing a frame window size.
- ElementProfile (const ElementProfile &clone)
  ElementProfile copy constructor.

- virtual ~ElementProfile ()

  ElementProfile destructor.

- virtual ElementProfile & operator= (const ElementProfile &clone)

  ElementProfile assignment operator.

Public Attributes

- const double normFac

Protected Attributes

- Sequence< IMG_ELM_TYPE > frameSeq_
- Sequence< MSK_ELM_TYPE > maskSeq_
- unsigned int wSize_
- unsigned int width_
- unsigned int height_
- unsigned int nChl_
- unsigned int nElm_
- std::vector< std::vector< unsigned int > > pxls_
- std::vector< unsigned int > maxElmIdx_
- Matrix neighbors_
- double * hue_
- double * hueVariation_
- double * hueVecX_
- double * hueVecY_
- double * lightness_
- double * litVariation_
- double * saturation_
- double * satVariation_
- double ** intensity_
- double * locationX_
- double * locationY_
- double * locationZ_
- unsigned int * nPixels_

5.5.1 Detailed Description

template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> class ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >

ElementProfile handles the similarity information of the elements in a frame window.

Segmentation is only done on a few frames (= frame window) of the sequence. The frame window is moved over the whole sequence to get a complete segmentation. An `ElementProfile` is used to store information about the elements in one frame window (= the frames that are used for one segmentation step). It is also used to calculate the similarity between those elements, which is then stored in the `similarity` matrix. `ElementProfile` administers the frame and mask sequences.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::ElementProfile (unsigned int *wSize*, const Setting * *setting*)

ElementProfile is constucted passing a frame window size.

Parameters:

wSize The size of the frame window.

setting The parameters of the current segmentation.

A new empty ElementProfile is constructed. Use `push` until the frame window is filled to start info and similarity calculation.

5.5.2.2 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::ElementProfile (const ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE > & *clone*)

ElementProfile copy constructor.

Parameters:

clone The `ElementProfile` to be cloned.

5.5.3 Member Function Documentation

5.5.3.1 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> const IMG_ELM_TYPE& ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::frame (unsigned int *num*, unsigned int *pxlelm*) const `[inline]`

Get a pixel in the n-th frame of this frame window.

Parameters:

num the number of the frame

pxlelm e.g. to get channel c of pixel p of an n-channel image use p*n+c

Returns:

the value of one channel of a pixel in the specified frame

5.5.3.2 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> const Image<IMG_ELM_TYPE>* ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::frame (unsigned int *num*) const `[inline]`

Get the n-th frame in this frame window.

Parameters:

num the number of the frame

Returns:

a pointer to the specified frame

5.5.3.3 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> unsigned int ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::frameOf (unsigned int *idx*) const

Get the frame containing the element with index `idx`.

Parameters:
idx The index of the element.

Returns:
The number of the frame containing the element.

5.5.3.4 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> unsigned int ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::height () const [inline]

Get the height of the frames.

Returns:
The height of the frames.

5.5.3.5 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::hueVecX (unsigned int *elm*) const [inline]

Get the mean hue vector x coordinate of an element.

Parameters:
elm The index of the element.

Returns:
The hue vector x coordinate of an element.

5.5.3.6 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::hueVecY (unsigned int *elm*) const [inline]

Get the mean hue vector y coordinate of an element.

Parameters:
elm The index of the element.

Returns:
The hue vector xycoordinate of an element.

5.5.3.7 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::intensity (unsigned int *elm*, unsigned int *chl*) const [inline]

Get the mean intensity for one channel of an element.

Parameters:

elm The index of the element.

chl The colour channel.

Returns:

The mean intensity of the element.

5.5.3.8 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::lightness (unsigned int *elm*) const [inline]

Get the mean lightness of an element.

Parameters:

elm The index of the element.

Returns:

The lightness of an element.

5.5.3.9 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::locationX (unsigned int *elm*) const [inline]

Get the mean location x-coordinate of an element.

Parameters:

elm The index of the element.

Returns:

The mean location x-coordinate of an element (in pixels).

5.5.3.10 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::locationY (unsigned int *elm*) const [inline]

Get the mean location y-coordinate of an element.

Parameters:

elm The index of the element.

Returns:

The mean location y-coordinate of an element (in pixels).

5.5.3.11 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::locationZ (unsigned int *elm*) const [inline]

Get the mean location z-coordinate of an element (frame number).

Parameters:

elm The index of the element.

Returns:

The mean location z-coordinate of an element (in frames). This function returns the "mean" frame number in this frame window. This number does not have to be an integer if the element spans multiple frames.

5.5.3.12 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> const MSK_ELM_TYPE& ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::mask (unsigned int *num*, unsigned int *pxl*) const [inline]

Get a pixel in the n-th mask of this frame window.

Parameters:

num the number of the mask

pxl the number of the pixel

Returns:

the element number of the element the pixel belongs to

5.5.3.13 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> const Image<MSK_ELM_TYPE>* ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::mask (unsigned int *num*) const [inline]

Get the n-th mask in this frame window.

Parameters:

num the number of the mask

Returns:

a pointer the specified mask

5.5.3.14 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> unsigned int ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::maxIdx (unsigned int *frm*) const [inline]

Get the maximum element index in the given frame.

Parameters:

frm The number of the frame in tghe current frame window.

Returns:

The maximum element index in the given frame.

5.5.3.15 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> unsigned int ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::minIdx (unsigned int *frm*) const [inline]

Get the minimum element index in the given frame.

Parameters:

frm The number of the frame in tghe current frame window.

Returns:

The minimum element index in the given frame.

5.5.3.16 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> unsigned int ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::nChl () const [inline]

Get the number of channels of the frames.

Returns:

The number of channels of the frames.

5.5.3.17 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> const Matrix* ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::neighbors () const [inline]

Get the neighbourhood matrix of this profile.

Returns:

The neighbourhood matrix of this profile.

5.5.3.18 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> unsigned int ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::nElm () const [inline]

Get the number of elements in this profile.

Returns:

The number of elements in this profile.

5.5.3.19 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> unsigned int ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::nFrm () const [inline]

Get the number of frames in the current frame window.

Returns:

The number of frames in the current frame window. note: this may be smaller than wSize if the frame window is not completely filled yet.

5.5.3.20 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> unsigned int ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::nPixels (unsigned int *elm*) const [inline]

Get the number of pixels of an element.

Parameters:
elm The index of the element.

Returns:
The number of pixels of an element.

5.5.3.21 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE > & ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::operator= (const ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE > & *clone*) [virtual]

ElementProfile assignment operator.

Parameters:
clone The `ElementProfile` to be cloned.

5.5.3.22 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> const std::vector<unsigned int>* ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::pixels (unsigned int *idx*) const [inline]

Get the pixels of an element.

Parameters:
idx The index of the element.

Returns:
A list of pixel numbers for an element.

5.5.3.23 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> const std::vector<std::vector<unsigned int> >* ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::pixels () const [inline]

Get a list if pixels for each element.

Returns:
A list of pixels for each element. This function returns a vector containing vectors of pixel numbers. Vector number i holds the pixel numbers of element i.

5.5.3.24 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> int ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::push (const Image< IMG_ELM_TYPE > * *frame*, const Image< MSK_ELM_TYPE > * *mask*)

Calculate element info of a new frame.

Parameters:

frame An new frame for segmentation.

mask Segment mask from a prior segmentation step.

Returns:

1 if sequence is not long enough to start segmentation yet (smaller then `wSize`), 0 otherwise

Gathers all needed element info and calculates the similarity matrix.

5.5.3.25 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::saturation (unsigned int *elm*) const [inline]

Get the mean saturation of an element.

Parameters:

elm The index of the element.

Returns:

The saturation of an element.

5.5.3.26 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> unsigned int ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::width () const [inline]

Get the width of the frames.

Returns:

The width of the frames.

5.5.3.27 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> unsigned int ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::wSize () const [inline]

Get the preset size of the frame window in frames.

Returns:

The preset size of the frame window in frames.

5.5.4 Member Data Documentation

5.5.4.1 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> Sequence<IMG_ELM_TYPE> ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::frameSeq_ [protected]

contains all frames in the current frame window

5.5.4.2 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> unsigned int ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::height_ [protected]

the height of each image in the sequence in pixels

5.5.4.3 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double* ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::hue_ [protected]

the mean hue of each element

5.5.4.4 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double* ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::hueVariation_ [protected]

the hue variation of each element

5.5.4.5 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double* ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::hueVecX_ [protected]

the hue vector X component of each element

5.5.4.6 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double* ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::hueVecY_ [protected]

the hue vector Y component of each element

5.5.4.7 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double** ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::intensity_ [protected]

the mean intensity of each image channel

5.5.4.8 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double* ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::lightness_ [protected]

the mean lightness of each element

5.5.4.9 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double* ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::litVariation_ [protected]

the lightness variation of each element

5.5.4.10 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double* ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::locationX_ [protected]

the center-of-gravity X coordinate

5.5.4.11 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double* ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::locationY_ [protected]

the center-of-gravity Y coordinate

5.5.4.12 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double* ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::locationZ_ [protected]

the center-of-gravity Z coordinate

5.5.4.13 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> Sequence<MSK_ELM_TYPE> ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::maskSeq_ [protected]

contains the segment masks of all frames in the current frame window

5.5.4.14 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> std::vector<unsigned int> ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::maxElmIdx_ [protected]

the highest used element index in each frame

5.5.4.15 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> unsigned int ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::nChl_ [protected]

the number of channels of each image in the sequence

5.5.4.16 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> Matrix ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::neighbors_ [protected]

stores the number of neighboring pixels of every element to every other element

5.5.4.17 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> unsigned int ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::nElm_ [protected]

the number of elements in the current frame window

5.5.4.18 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> const double ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::normFac

a constant factor to normalize a datatype of the image sequence to the interval [0,1]

5.5.4.19 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> unsigned int* ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::nPixels_ [protected]

the number of pixels in each element

5.5.4.20 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> std::vector<std::vector<unsigned int> > ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::pxls_ [protected]

a list of pixels in each element

5.5.4.21 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double* ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::saturation_ [protected]

the mean saturation of each element

5.5.4.22 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double* ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::satVariation_ [protected]

the saturation variation of each element

5.5.4.23 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> unsigned int ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::width_ [protected]

the width of each image in the sequence in pixels

5.5.4.24 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> unsigned int ncut::ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::wSize_ [protected]

the size of the frame window (in frames)

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

- /home/stefan/cpp/ncut/src/motionProfile.h

5.6 greedy_ptr< _Tp > Class Template Reference

A simple smart pointer providing strict ownership semantics.

`#include <ncutMemory.h>`

Public Types

- typedef _Tp element_type
  The pointed-to type.

Public Member Functions

- greedy_ptr (element_type *__p=0) throw ()
  A greedy_ptr is usually constructed from a raw pointer.
- greedy_ptr (element_type &__p) throw ()
  A greedy_ptr can be constructed from a reference.
- greedy_ptr (const greedy_ptr &__a) throw ()
  A greedy_ptr can be constructed from another greedy_ptr.
- template<typename _Tp1> greedy_ptr (std::auto_ptr< _Tp1 > &__a) throw ()
  A greedy_ptr can be constructed from an auto_ptr.
- greedy_ptr (std::auto_ptr< element_type > &__a) throw ()
  A greedy_ptr can be constructed from an auto_ptr.
- template<typename _Tp1> greedy_ptr (const greedy_ptr< _Tp1 > &__a) throw ()
  A greedy_ptr can be constructed from another greedy_ptr.
- greedy_ptr & operator= (const greedy_ptr &__a) throw ()
  greedy_ptr assignment operator.
- template<typename _Tp1> greedy_ptr & operator= (const greedy_ptr< _Tp1 > &__a) throw ()
  greedy_ptr assignment operator.
- greedy_ptr & operator= (element_type *__p) throw ()
  greedy_ptr assignment operator.
- greedy_ptr & operator= (element_type &__p) throw ()
  greedy_ptr assignment operator.
- ~greedy_ptr ()
- element_type & operator * () const throw ()
  Smart pointer dereferencing.
- element_type * operator -> () const throw ()

Smart pointer dereferencing.

- element_type * get () const throw ()

  Bypassing the smart pointer.

- bool owns () const throw ()

  Getting ownership status.

- element_type * release () throw ()

  Bypassing the smart pointer.

- void reset (element_type *__p=0) throw ()

  Forcibly deletes the managed object.

- void reset (element_type &__p=0) throw ()

  Forcibly deletes the managed object.

- greedy_ptr (greedy_ptr_ref< element_type > __ref) throw ()

  Automatic conversions.

- greedy_ptr & **operator=** (greedy_ptr_ref< element_type > __ref) throw ()
- template<typename _Tp1> **operator greedy_ptr_ref** () throw ()
- template<typename _Tp1> **operator greedy_ptr** () throw ()

5.6.1 Detailed Description

template<typename _Tp> class greedy_ptr< _Tp >

A simple smart pointer providing strict ownership semantics.

A `greedy_ptr` only owns the object it holds a pointer to, if it was constructed from a raw pointer (i.e. if it was the first greedy_ptr to manage the pointed-to object). Copying a `greedy_ptr` copies the pointer but does not transfer ownership to the destination, the source `greedy_ptr` retains ownership. If more than one `greedy_ptr` owns the same object at the same time the behavior of the program is undefined.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 template<typename _Tp> greedy_ptr< _Tp >::greedy_ptr (element_type * __p = 0) throw () [inline, explicit]

A greedy_ptr is usually constructed from a raw pointer.

Parameters:

__p A pointer (defaults to NULL).

This object now *owns* the object pointed to by *p*.

5.6.2.2 template<typename _Tp> greedy_ptr< _Tp >::greedy_ptr (element_type & __p) throw () [inline, explicit]

A greedy_ptr can be constructed from a reference.

Parameters:

__p A reference (defaults to NULL).

This object does *not* own the object *p*.

5.6.2.3 template<typename _Tp> greedy_ptr< _Tp >::greedy_ptr (const greedy_ptr< _Tp > & __a) throw () [inline]

A greedy_ptr can be constructed from another greedy_ptr.

Parameters:

__a Another greedy_ptr of the same type.

This object does *not* own the object owned by *a*, which still holds ownership.

5.6.2.4 template<typename _Tp> template<typename _Tp1> greedy_ptr< _Tp >::greedy_ptr (std::auto_ptr< _Tp1 > & __a) throw () [inline]

A greedy_ptr can be constructed from an auto_ptr.

Parameters:

__a An auto_ptr of a different but related type.

A pointer-to-Tp1 must be convertible to a pointer-to-Tp/element_type.

This object now *owns* the object previously owned by *a*, which has given up ownership.

5.6.2.5 template<typename _Tp> greedy_ptr< _Tp >::greedy_ptr (std::auto_ptr< element_type > & __a) throw () [inline]

A greedy_ptr can be constructed from an auto_ptr.

Parameters:

__a An auto_ptr of the same type.

This object now *owns* the object previously owned by *a*, which has given up ownership.

5.6.2.6 template<typename _Tp> template<typename _Tp1> greedy_ptr< _Tp >::greedy_ptr (const greedy_ptr< _Tp1 > & __a) throw () [inline]

A greedy_ptr can be constructed from another greedy_ptr.

Parameters:

__a Another greedy_ptr of a different but related type.

A pointer-to-Tp1 must be convertible to a pointer-to-Tp/element_type.

This object does *not* own the object owned by *a*, which still holds ownership.

5.6.2.7 template<typename _Tp> greedy_ptr< _Tp >::~greedy_ptr () [inline]

When the greedy_ptr goes out of scope, the object it owns is deleted. If it no longer owns anything then this has no effect.

The C++ standard says there is supposed to be an empty throw specification here, but omitting it is standard conforming. Its presence can be detected only if _Tp::~_Tp() throws, but this is prohibited. [17.4.3.6]/2

5.6.2.8 template<typename _Tp> greedy_ptr< _Tp >::greedy_ptr (greedy_ptr_ref< element_type > __ref) throw () [inline]

Automatic conversions.

These operations convert an greedy_ptr into and from an greedy_ptr_ref automatically as needed. This allows constructs such as

```
greedy_ptr<Derived>  func_returning_greedy_ptr(.....);
...
greedy_ptr<Base> ptr = func_returning_greedy_ptr(.....);
```

5.6.3 Member Function Documentation

5.6.3.1 template<typename _Tp> element_type* greedy_ptr< _Tp >::get () const throw () [inline]

Bypassing the smart pointer.

Returns:
The raw pointer being managed.

You can get a copy of the pointer that this object owns, for situations such as passing to a function which only accepts a raw pointer.

Note:
If this greedy_ptr was owning the memory, it still does.

5.6.3.2 template<typename _Tp> element_type& greedy_ptr< _Tp >::operator * () const throw () [inline]

Smart pointer dereferencing.

If this greedy_ptr is a null pointer, then this operation will crash.

5.6.3.3 template<typename _Tp> element_type* greedy_ptr< _Tp >::operator → () const throw () [inline]

Smart pointer dereferencing.

This returns the pointer itself, which the language then will automatically cause to be dereferenced.

5.6.3.4 template<typename _Tp> greedy_ptr& greedy_ptr< _Tp >::operator= (element_type & __p) throw () [inline]

greedy_ptr assignment operator.

Parameters:
__p A reference.

A convenience function, does exactly the same thing as `reset(__p)`.

5.6.3.5 template<typename _Tp> greedy_ptr& greedy_ptr< _Tp >::operator= (element_type * __p) throw () [inline]

greedy_ptr assignment operator.

Parameters:
__p A pointer.

A convenience function, does exactly the same thing as `reset(__p)`.

5.6.3.6 template<typename _Tp> template<typename _Tp1> greedy_ptr& greedy_ptr< _Tp >::operator= (const greedy_ptr< _Tp1 > & __a) throw () [inline]

greedy_ptr assignment operator.

Parameters:
__a Another greedy_ptr of a different but related type.

A pointer-to-Tp1 must be convertible to a pointer-to-Tp/element_type.

This object does *not* own the object owned by *a*, which still holds ownership. If this greedy_ptr *used* to own an object, that object has been deleted.

5.6.3.7 template<typename _Tp> greedy_ptr& greedy_ptr< _Tp >::operator= (const greedy_ptr< _Tp > & __a) throw () [inline]

greedy_ptr assignment operator.

Parameters:
__a Another greedy_ptr of the same type.

This object does *not* own the object owned by *a*, which still holds ownership. If this greedy_ptr *used* to own an object, that object has been deleted.

5.6.3.8 template<typename _Tp> bool greedy_ptr< _Tp >::owns () const throw () [inline]

Getting ownership status.

Returns:
`true` if this greedy_ptr owns the object being managed, `false` otherwise.

You can query the greedy_ptr for ownership of the object being managed. `NULL` pointers are never owned.

5.6.3.9 template<typename _Tp> element_type* greedy_ptr< _Tp >::release () throw () [inline]

Bypassing the smart pointer.

Returns:
The raw pointer being managed.

You can get a copy of the pointer that this object owns, for situations such as passing to a function which only accepts a raw pointer.

Note:
This greedy_ptr no longer owns the memory. When this object goes out of scope, nothing will happen.

5.6.3.10 template<typename _Tp> void greedy_ptr< _Tp >::reset (element_type & *__p* = 0) throw () [inline]

Forcibly deletes the managed object.

Parameters:
__p A reference.

This object now holds a pointer to *p* , but does *not* own it. The previously held object has been deleted.

5.6.3.11 template<typename _Tp> void greedy_ptr< _Tp >::reset (element_type * *__p* = 0) throw () [inline]

Forcibly deletes the managed object.

Parameters:
__p A pointer (defaults to NULL).

This object now *owns* the object pointed to by *p*. The previous object has been deleted.

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

- /home/stefan/cpp/ncut/src/ncutMemory.h

5.7 greedy_ptr_ref< _Tp1 > Struct Template Reference

`#include <ncutMemory.h>`

Public Member Functions

- **greedy_ptr_ref** (_Tp1 *__p)

Public Attributes

- _Tp1 * **_M_ptr**

5.7.1 Detailed Description

template<typename _Tp1> struct greedy_ptr_ref< _Tp1 >

A wrapper class to provide greedy_ptr with reference semantics. For example, an greedy_ptr can be assigned (or constructed from) the result of a function which returns an greedy_ptr by value.

All the greedy_ptr_ref stuff should happen behind the scenes.

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

- /home/stefan/cpp/ncut/src/ncutMemory.h

5.8 ncut::Image< ELEMENT_TYPE > Class Template Reference

`#include <ncutImage.h>`

Public Member Functions

- ELEMENT_TYPE & val (unsigned int elm)
  Get the value of the colour channel of the specified element.
- ELEMENT_TYPE & val (unsigned int pxl, unsigned int chl)
  Get the value of the colour channel of the specified pixel and channel.
- const ELEMENT_TYPE & val (unsigned int elm) const
  Get the constant value of the colour channel of the specified element.
- const ELEMENT_TYPE & val (unsigned int pxl, unsigned int chl) const
  Get the value of the colour channel of the specified pixel and channel.
- unsigned int width () const
  Get the with of the image.
- unsigned int height () const
  Get the height of the image.
- unsigned int nChl () const
  Get the number of channels of the image.
- Image ()
  Image empty constructor.
- Image (int width, int height, int nChl)
  Image custom constructor.
- Image (const Image &clone)
  Image copy constructor.
- virtual ~Image ()
  Image destructor.
- virtual Image & operator= (const Image &clone)
  Image assignment operator.

Protected Attributes

- std::vector< ELEMENT_TYPE > imageData_
- unsigned int width_
- unsigned int height_
- unsigned int nChl_

5.8.1 Detailed Description

template<class ELEMENT_TYPE = unsigned char> class ncut::Image< ELEMENT_TYPE >

`Image` is the base class for `PNGImage` and `BSIImage`. `Image` is a small wrapper around a STL vector containing the raw image values, providing additional information, such as width and height, and more convenient image element access functions.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 template<class ELEMENT_TYPE> ncut::Image< ELEMENT_TYPE >::Image ()

Image empty constructor.

Creates an empty image with width=height=channels=0.

5.8.2.2 template<class ELEMENT_TYPE> ncut::Image< ELEMENT_TYPE >::Image (int *width*, int *height*, int *nChl*)

Image custom constructor.

Parameters:

width The width of the image.

height The height of the image.

nChl The number of channels of the image.

Creates an empty image with the specified dimensions and initializes the data to zero.

5.8.2.3 template<class ELEMENT_TYPE> ncut::Image< ELEMENT_TYPE >::Image (const Image< ELEMENT_TYPE > & *clone*)

Image copy constructor.

Parameters:

clone The `Image` to be cloned.

5.8.3 Member Function Documentation

5.8.3.1 template<class ELEMENT_TYPE = unsigned char> unsigned int ncut::Image< ELEMENT_TYPE >::height () const [inline]

Get the height of the image.

Returns:

The height of the image

5.8.3.2 template<class ELEMENT_TYPE = unsigned char> unsigned int ncut::Image< ELEMENT_TYPE >::nChl () const [inline]

Get the number of channels of the image.

Returns:
The number of channles

5.8.3.3 template<class ELEMENT_TYPE> Image< ELEMENT_TYPE > & ncut::Image< ELEMENT_TYPE >::operator= (const Image< ELEMENT_TYPE > & *clone*) [virtual]

Image assignment operator.

Parameters:
clone The `Image` to be cloned.

Returns:
This object.

5.8.3.4 template<class ELEMENT_TYPE = unsigned char> const ELEMENT_TYPE& ncut::Image< ELEMENT_TYPE >::val (unsigned int *pxl*, unsigned int *chl*) const [inline]

Get the value of the colour channel of the specified pixel and channel.

Parameters:
pxl the specified pixel

chl the colour channel

Returns:
The value of the colour channel

5.8.3.5 template<class ELEMENT_TYPE = unsigned char> const ELEMENT_TYPE& ncut::Image< ELEMENT_TYPE >::val (unsigned int *elm*) const [inline]

Get the constant value of the colour channel of the specified element.

Parameters:
elm the element

Returns:
The constant value of the colour channel

5.8.3.6 template<class ELEMENT_TYPE = unsigned char> ELEMENT_TYPE& ncut::Image< ELEMENT_TYPE >::val (unsigned int *pxl*, unsigned int *chl*) [inline]

Get the value of the colour channel of the specified pixel and channel.

Parameters:

pxl the specified pixel

chl the colour channel

Returns:

the value of the colour channel

5.8.3.7 template<class ELEMENT_TYPE = unsigned char> ELEMENT_TYPE& ncut::Image< ELEMENT_TYPE >::val (unsigned int *elm*) [inline]

Get the value of the colour channel of the specified element.

Parameters:

elm the element

Returns:

the value of the colour channel

5.8.3.8 template<class ELEMENT_TYPE = unsigned char> unsigned int ncut::Image< ELEMENT_TYPE >::width () const [inline]

Get the with of the image.

Returns:

The width of the image

5.8.4 Member Data Documentation

5.8.4.1 template<class ELEMENT_TYPE = unsigned char> unsigned int ncut::Image< ELEMENT_TYPE >::height_ [protected]

The height of this image.

5.8.4.2 template<class ELEMENT_TYPE = unsigned char> std::vector<ELEMENT_TYPE> ncut::Image< ELEMENT_TYPE >::imageData_ [protected]

The raw image values stored sequentially in the form pxl0chl0, pxl0chl1, ... , pxl1chl0, ...

5.8.4.3 template<class ELEMENT_TYPE = unsigned char> unsigned int ncut::Image< ELEMENT_TYPE >::nChl_ [protected]

The number of channels of this image.

5.8.4.4 template<class ELEMENT_TYPE = unsigned char> unsigned int ncut::Image< ELEMENT_TYPE >::width_ [protected]

The width of this image.

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

- /home/stefan/cpp/ncut/src/ncutImage.h

5.9 Matrix Class Reference

Simple matrix class.

`#include <matrix.h>`

Public Member Functions

- double & val (unsigned int row, unsigned int col)
  the value of a cell of the matrix
- const double & val (unsigned int row, unsigned int col) const
  read-only access to the values of a matrix cell
- unsigned int dim () const
  get the dimension of the matrix
- void changeDimension (const unsigned int dimension)
  Reset matrix to new dimension.
- void add (const Matrix *m)
  Adds the matrix to the current matrix.
- void subtract (const Matrix *m)
  Subtracts a matrix from the current matrix.
- auto_array< double > band (int &n, int &nsdiag) const
  Returns the matrix in bandfrom which is needed for arpack++.
- int split (Matrix &a, Matrix &b, std::vector< unsigned int > &indexMap, std::list< unsigned int > &indices) const
  Splits the matrix into two new matrices.
- Matrix & inject (const Matrix &m, unsigned int index)
  Injects a small matrix into a larger one at an arbitrary position.
- void transform (double *vec) const
  Transforms a vector.
- void diag (const double *diag, unsigned int dim)
  Resets the matrix to have diag with dimension dim on its diagonal.
- Matrix & operator= (double val)
  Sets all elements in the matrix to the given value `val`.
- Matrix & operator+= (const Matrix &operand)
  Convience function for add().
- Matrix & operator-= (const Matrix &operand)
  Convience function for subtract().

- Matrix (unsigned int dimension=0)

  Creates an empty matrix with the given dimension.

- Matrix (const double *diag, unsigned int dim)

  Create a new diagonal matrix.

- Matrix (const Matrix &clone)

  Matrix copy constructor.

- virtual ~Matrix ()

  Matrix destructor.

- virtual Matrix & operator= (const Matrix &clone)

  Matrix assignment operator.

Protected Attributes

- double ** value_
- unsigned int dimension_

5.9.1 Detailed Description

Simple matrix class.

`Matrix` is a simple quadratic matrix class containing all operations needed by the ncut library.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 Matrix::Matrix (unsigned int *dimension* = `0`)

Creates an empty matrix with the given dimension.

Parameters:

dimension The dimension for the new matrix (default=0).

5.9.2.2 Matrix::Matrix (const double * *diag*, unsigned int *dim*)

Create a new diagonal matrix.

Parameters:

diag The vector that will be on the matrix's diagonal.

dim The dimension for the new matrix (default=0).

Creates a matrix of dimension `dim` with `diag` on its diagonal

5.9.2.3 Matrix::Matrix (const Matrix & *clone*)

Matrix copy constructor.

Parameters:
clone The matrix to be cloned.

5.9.3 Member Function Documentation

5.9.3.1 void Matrix::add (const Matrix * *m*)

Adds the matrix to the current matrix.

Parameters:
m The matrix which should be added.

5.9.3.2 auto_array< double > Matrix::band (int & *n*, int & *nsdiag*) const

Returns the matrix in bandfrom which is needed for arpack++.

Parameters:
n The dimension of the matrix will be written into this variable.

nsdiag The number of the non-zero diagonals will be written into this variable.

Returns:
An array of doubles (which is needed for aprack++).

5.9.3.3 void Matrix::changeDimension (const unsigned int *dimension*)

Reset matrix to new dimension.

Parameters:
dimension The new dimension of the matrix.

Changes the dimension, deletes the old matrix and creates a new one with the given dimension

5.9.3.4 void Matrix::diag (const double * *diag*, unsigned int *dim*)

Resets the matrix to have diag with dimension dim on its diagonal.

Parameters:
diag The new diagonal vector of this matrix.

dim The new dimension of this matrix.

5.9.3.5 unsigned int Matrix::dim () const [inline]

get the dimension of the matrix

Returns:
the dimension of the matrix

5.9.3.6 Matrix & Matrix::inject (const Matrix & *m*, unsigned int *index*)

Injects a small matrix into a larger one at an arbitrary position.

Parameters:
m The `Matrix` to be injected.
index The position where `m` should be injected.

Returns:
This object.

`Matrix` m is copied into this `Matrix` starting at position (`index` , `index`). `m` does not have to fit inside this matrix, but if it doesn't, only the values of `m` that do fit are copied. All values in this matrix remain unchanged except the ones overwritten by `m`.

5.9.3.7 Matrix & Matrix::operator+= (const Matrix & *operand*)

Convience function for add().

Parameters:
operand he matrix to be added.

Returns:
This object.

5.9.3.8 Matrix & Matrix::operator-= (const Matrix & *operand*)

Convience function for subtract().

Parameters:
operand The matrix to be subtracted.

Returns:
This object.

5.9.3.9 Matrix & Matrix::operator= (const Matrix & *clone*) [virtual]

Matrix assignment operator.

Parameters:
clone The matrix to be cloned.

Returns:
This object.

5.9.3.10 Matrix & Matrix::operator= (double *val*)

Sets all elements in the matrix to the given value `val`.

Parameters:
val All elements of the matrix will be set to this value.

Returns:
This matrix.

5.9.3.11 int Matrix::split (Matrix & *a*, Matrix & *b*, std::vector< unsigned int > & *indexMap*, std::list< unsigned int > & *indices*) const

Splits the matrix into two new matrices.

Parameters:
a One part will be written into this matrix.

b The second part will be written into this matrix.

indexMap A mapping of index numbers in the old matrix to the associated index in one of the new matrices.

indices The list of indices in matrix `a`.

Returns:
0 if there was no error

5.9.3.12 void Matrix::subtract (const Matrix * *m*)

Subtracts a matrix from the current matrix.

Parameters:
m The matrix which should be subtracted.

5.9.3.13 void Matrix::transform (double * *vec*) const

Transforms a vector.

Parameters:
vec The vector to be transformed.

The vector vec is multiplied from the left with this matrix. The size of the vec array has to match the dimension of this matrix, otherwise the result is undefined.

5.9.3.14 const double& Matrix::val (unsigned int *row*, unsigned int *col*) const [inline]

read-only access to the values of a matrix cell

Parameters:
row the row

col the column

Returns:
constant value of the specified cell

5.9.3.15 double& Matrix::val (unsigned int *row*, unsigned int *col*) [inline]

the value of a cell of the matrix

Parameters:
row the row
col the column

Returns:
the value of the specified cell

5.9.4 Member Data Documentation

5.9.4.1 unsigned int Matrix::dimension_ [protected]

The dimension of the matrix

5.9.4.2 double** Matrix::value_ [protected]

All elements in the matrix

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

- /home/stefan/cpp/ncut/src/matrix.h
- /home/stefan/cpp/ncut/src/matrix.cpp

5.10 ncut::Ncut Class Reference

Ncut represents one ncut of a similarity matrix.

`#include <ncutCore.h>`

Public Member Functions

- const Matrix * similarity () const
  Get a pointer to the similarity matrix.
- double assoc (unsigned int i) const
  Get the total association of element with index i.
- const double * assoc () const
  Get a pointer to the total association array.
- unsigned int dim () const
  Get the dimension of this ncut.
- double eigVal (unsigned int i) const
  Get the eigenvalue of the eigenvector with index i.
- const double * eigVal () const
  Get a pointer to the eigenvalue array.
- double eigVec (unsigned int i) const
  Get the eigenvector component with index i.
- const double * eigVec () const
  Get a pointer to the eigenvectors array.
- unsigned int nConv () const
  Get the number of converged eigenvectors.
- int calculate (unsigned int nEigVecs, double *initVec=NULL)
  Calculate one minimal ncut.
- Ncut (const Matrix *similarity)
  Ncut can be constructed from a similarity matrix.
- Ncut (const Ncut &clone)
  Ncut copy constructor.
- virtual ~Ncut ()
  Ncut destructor.
- virtual Ncut & operator= (const Ncut &clone)
  Ncut assignment operator.

Protected Attributes

- const Matrix * similarity_
- unsigned int dim_
- double * assoc_
- double * eigVal_
- double * eigVec_
- unsigned int nConv_

5.10.1 Detailed Description

Ncut represents one ncut of a similarity matrix.

This class is used by NcutNode to calculate the ncuts and construct the tree accordingly. Do not use `Ncut` directly, use the functions provided by the class `Segmentation`.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 ncut::Ncut::Ncut (const Matrix * *similarity*)

Ncut can be constructed from a similarity matrix.

Parameters:

similarity A pointer to the similarity matrix on which the ncut will be calculated.

5.10.2.2 ncut::Ncut::Ncut (const Ncut & *clone*)

Ncut copy constructor.

Parameters:

clone The ncut to be cloned.

Creates a completely independent copy of `clone`.

5.10.3 Member Function Documentation

5.10.3.1 const double* ncut::Ncut::assoc () const [inline]

Get a pointer to the total association array.

Returns:

Pointer to the total association array.

5.10.3.2 double ncut::Ncut::assoc (unsigned int *i*) const [inline]

Get the total association of element with index i.

Parameters:
i element index

Returns:
Total association of element with index i.

5.10.3.3 int ncut::Ncut::calculate (unsigned int *nEigVecs*, double * *initVec* = NULL)

Calculate one minimal ncut.

Parameters:
nEigVecs The desired number of eigenvectors.

initVec An initialization vector for arpack, defaults to NULL.

Returns:
0 if successful.

Calculates the minimal ncut of the similarity matrix pointed to by the member similarity and tries to produce nEigVecs eigenvectors and eigenvalues, but nEigVecs eigenvectors are *not* guaranteed. Use nConv to obtain the actual number of calculated eigenvectors.

5.10.3.4 unsigned int ncut::Ncut::dim () const [inline]

Get the dimension of this ncut.

Returns:
Dimension of this ncut.

5.10.3.5 const double* ncut::Ncut::eigVal () const [inline]

Get a pointer to the eigenvalue array.

Returns:
Pointer to the eigenvalue array.

5.10.3.6 double ncut::Ncut::eigVal (unsigned int *i*) const [inline]

Get the eigenvalue of the eigenvector with index i.

Parameters:
i eigenvector index

Returns:
Eigenvalue of the eigenvector with index i.

5.10.3.7 const double* ncut::Ncut::eigVec () const [inline]

Get a pointer to the eigenvectors array.

Returns:
Pointer to the eigenvectors array.

5.10.3.8 double ncut::Ncut::eigVec (unsigned int *i*) const [inline]

Get the eigenvector component with index i.

Parameters:
i component index, the components of all eigenvectors are in this array.

Returns:
Eigenvector component with index i.

5.10.3.9 unsigned int ncut::Ncut::nConv () const [inline]

Get the number of converged eigenvectors.

Returns:
Number of converged eigenvectors.

5.10.3.10 Ncut & ncut::Ncut::operator= (const Ncut & *clone*) [virtual]

Ncut assignment operator.

Parameters:
clone An Ncut to be cloned.

Returns:
This object.

5.10.3.11 const Matrix* ncut::Ncut::similarity () const [inline]

Get a pointer to the similarity matrix.

Returns:
Pointer to the similarity matrix.

5.10.4 Member Data Documentation

5.10.4.1 double* ncut::Ncut::assoc_ [protected]

an array containing the total association of each node

5.10.4.2 unsigned int ncut::Ncut::dim_ [protected]

the dimension of the ncut (= similarity matrix dimension)

5.10.4.3 double* ncut::Ncut::eigVal_ [protected]

an array of the calculated eigenvalues (its length is `nConv_`)

5.10.4.4 double* ncut::Ncut::eigVec_ [protected]

an array of the calculated eigenvectors (its length is `nConv_ * dim_`)

5.10.4.5 unsigned int ncut::Ncut::nConv_ [protected]

the actual number of converged eigenvalues and -vectors

5.10.4.6 const Matrix* ncut::Ncut::similarity_ [protected]

the similarity matrix on which the ncut is calculated

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

- /home/stefan/cpp/ncut/src/ncutCore.h

5.11 ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE > Class Template Reference

NcutNode is one Node in the ncut dendogram.

`#include <ncutCore.h>`

Public Member Functions

- const greedy_ptr< const Profile< IMG_ELM_TYPE, MSK_ELM_TYPE > > profile () const

  Get the Profile this Node uses.

- const std::map< MSK_ELM_TYPE, unsigned int > * indices () const

  Get the index map of the elements in this NcutNode.

- const greedy_ptr< const Ncut > ncut () const

  Get the Ncut calculated by this NcutNode.

- double eigVal () const

  Get the eigenvalue of the eigenvector this NcutNode uses for partitioning.

- const double * eigVec () const

  Get a pointer to the eigenvector this NcutNode uses for partitioning.

- double splitPoint () const

  Get the splitting point this NcutNode uses for partitioning.

- double eigVecNum () const

  Get the number of the eigenvector that was used for partitioning.

- const greedy_ptr< const NcutNode > leftNode () const

  Get the left child NcutNode of this NcutNode.

- const greedy_ptr< const NcutNode > rightNode () const

  Get the right child NcutNode of this NcutNode.

- int calculate (unsigned int maxDepth, unsigned int depth, unsigned int nEigVecs, unsigned int eigvn, double *initVec=NULL)

  Constructs the ncut dendogram.

- bool isLeft (unsigned int idx) const

  Checks to which half an element is assigned in this node.

- NcutNode (std::auto_ptr< Profile< IMG_ELM_TYPE, MSK_ELM_TYPE > > &profile, const std::map< MSK_ELM_TYPE, unsigned int > &indices, greedy_ptr< Ncut > &ncut, const Setting *setting)

  NcutNode owning constructor, takes ownership of `profile`.

- NcutNode (greedy_ptr< Profile< IMG_ELM_TYPE, MSK_ELM_TYPE > > &profile, const std::map< MSK_ELM_TYPE, unsigned int > &indices, greedy_ptr< Ncut > &ncut, const Setting *setting)

  NcutNode non-owning constructor, does not own `profile`.

- NcutNode (const NcutNode &clone)

  NcutNode copy constructor.

- NcutNode (const NcutNode &clone, greedy_ptr< Profile< IMG_ELM_TYPE, MSK_ELM_TYPE > > &pclone, greedy_ptr< Ncut > &nclone)

  NcutNode custom copy constructor.

- virtual ~NcutNode ()

  NcutNode destructor.

- virtual NcutNode & operator= (const NcutNode &clone)

  NcutNode assignment operator.

Protected Attributes

- greedy_ptr< Profile< IMG_ELM_TYPE, MSK_ELM_TYPE > > profile_
- std::map< MSK_ELM_TYPE, unsigned int > indices_
- greedy_ptr< Ncut > ncut_
- double eigVal_
- const double * eigVec_
- double splitPoint_
- unsigned int eigVecNum_
- greedy_ptr< NcutNode > leftNode_
- greedy_ptr< NcutNode > rightNode_
- const Setting * setting_

5.11.1 Detailed Description

template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> class ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >

NcutNode is one Node in the ncut dendogram.

A node represents one bipartition of a segment into two halves. The root `NcutNode` bipartitions (splits) the whole image. `leftNode` and `rightNode` point to the `NcutNode` s (=partitions) of the two resulting segments or to `NULL` if the resulting segments aren't partitioned any further. An eigenvector, calculated by the class `Ncut`, is used as an indicator vector to bipartition a segment. The node does not neccessarily contain a new ncut, an eigenvector of an ncut calculated in a level closer to the root may also be used. If this is the case, `ncut_` points to the old ncut, but `eigVec_` points to the eigenvector that is used to partition this segment. Segmentation is not performed on pixels, but on elements, for example the segments of a pre-segmentation. Elements have global identifiers and local indices. `indices_` is a map from element identifiers to element indices in each `NcutNode`. This map is also a list of elements that are part of the current segment. Do not use `NcutNode` directly, construct the ncut dendogram using the functions provided by the class `Segmentation`.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::NcutNode (std::auto_ptr< Profile< IMG_ELM_TYPE, MSK_ELM_TYPE > > & *profile*, const std::map< MSK_ELM_TYPE, unsigned int > & *indices*, greedy_ptr< Ncut > & *ncut*, const Setting * *setting*)

NcutNode owning constructor, takes ownership of `profile`.

Parameters:

profile the profile used for segmentation

indices a list of elements still active in this node and their indices

ncut the ncut from the parent node

setting The parameters of the current segmentation.

Creates a new NcutNode with the Profile `profile`, element indices list `indices` and optionally with Ncut `ncut`. `profile` *is* owned by this NcutNode.

5.11.2.2 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::NcutNode (greedy_ptr< Profile< IMG_ELM_TYPE, MSK_ELM_TYPE > > & *profile*, const std::map< MSK_ELM_TYPE, unsigned int > & *indices*, greedy_ptr< Ncut > & *ncut*, const Setting * *setting*)

NcutNode non-owning constructor, does not own `profile`.

Parameters:

profile the profile used for segmentation

indices a list of elements still active in this node and their indices

ncut the ncut this node should use

setting The parameters of the current segmentation.

Creates a new NcutNode with the Profile `profile`, element indices list `indices` and optionally with Ncut `ncut`. `profile` is *not* owned by this NcutNode.

5.11.2.3 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::NcutNode (const NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE > & *clone*)

NcutNode copy constructor.

Parameters:

clone The root of the subtree to be cloned.

Clones the NcutNode `clone` and all its child nodes (e.g. clones the subtree `clone` is root of).

5.11.2.4 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::NcutNode (const NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE > & *clone*, greedy_ptr< Profile< IMG_ELM_TYPE, MSK_ELM_TYPE > > & *pclone*, greedy_ptr< Ncut > & *nclone*)

NcutNode custom copy constructor.

Parameters:

clone The root of the subtree to be cloned.

pclone The Profile this subtree should point to.

nclone The Ncut this subtree should point to.

This copy constructor uses a pointer to a different `Profile` and a pointer to a different `Ncut`. These pointers are handed down the tree to make sure every child node of this node will be independent of `clone`.

5.11.2.5 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::~NcutNode () [virtual]

NcutNode destructor.

Deletes the subtree this `NcutNode` is root of.

5.11.3 Member Function Documentation

5.11.3.1 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> int ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::calculate (unsigned int *maxDepth*, unsigned int *depth*, unsigned int *nEigVecs*, unsigned int *eigvn*, double * *initVec* = NULL)

Constructs the ncut dendogram.

Parameters:

maxDepth The maximum depth of the ncut dendogram.

depth The current depth this node is in.

nEigVecs The number of eigenvectors that should be used of each ncut.

eigvn The number of the current eigenvector used to bipartition this segment.

initVec The starting vector used for calculating the ncut.

Returns:

0 if a valid Node was constructed, 1 if the node is invalid and has to be deleted.

Bipartitions the elements contained in `indices_` (either using an existing `Ncut` or calculating a new one). Child nodes are recursively constructed until the maximum depth is reached. If one of the child nodes needs a new `Ncut`, a new element profile with reduced similarity matrix (only containing the elements in the respective segment) is constructed and passed to the child node.

5.11.3.2 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> double ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::eigVal () const [inline]

Get the eigenvalue of the eigenvector this NcutNode uses for partitioning.

Returns:
Eigenvalue of the eigenvector this NcutNode uses for partitioning.

5.11.3.3 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const double* ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::eigVec () const [inline]

Get a pointer to the eigenvector this NcutNode uses for partitioning.

Returns:
Pointer to the eigenvector this NcutNode uses for partitioning.

5.11.3.4 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> double ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::eigVecNum () const [inline]

Get the number of the eigenvector that was used for partitioning.

Returns:
Number of the eigenvector that was used for partitioning.

5.11.3.5 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const std::map<MSK_ELM_TYPE, unsigned int>* ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::indices () const [inline]

Get the index map of the elements in this NcutNode.

Returns:
Index map of the elements in this NcutNode.

5.11.3.6 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> bool ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::isLeft (unsigned int *idx*) const

Checks to which half an element is assigned in this node.

Parameters:
idx The index of the element to be checked.

Returns:
`true` if the element is on the left half, `false` otherwise.

Checks if the element with index `idx` belongs to the left half of this node's bipartition.

5.11.3.7 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const greedy_ptr<const NcutNode> ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::leftNode () const [inline]

Get the left child NcutNode of this NcutNode.

Returns:
Left child NcutNode of this NcutNode.

5.11.3.8 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const greedy_ptr<const Ncut> ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::ncut () const [inline]

Get the Ncut calculated by this NcutNode.

Returns:
Ncut calculated by this NcutNode.

5.11.3.9 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE > & ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::operator= (const NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE > & *clone*) [virtual]

NcutNode assignment operator.

Parameters:
clone The root of the subtree to be cloned.

Returns:
This object.

Clones the NcutNode `clone` and all its child nodes (e.g. clones the subtree `clone` is root of).

5.11.3.10 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const greedy_ptr<const Profile<IMG_ELM_TYPE, MSK_ELM_TYPE> > ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::profile () const [inline]

Get the Profile this Node uses.

Returns:
Profile this Node uses.

5.11.3.11 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const greedy_ptr<const NcutNode> ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::rightNode () const [inline]

Get the right child NcutNode of this NcutNode.

Returns:
Right child NcutNode of this NcutNode.

5.11.3.12 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> double ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::splitPoint () const [inline]

Get the splitting point this NcutNode uses for partitioning.

Returns:
Splitting point this NcutNode uses for partitioning.

5.11.4 Member Data Documentation

5.11.4.1 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> double ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::eigVal_ [protected]

the eigenvalue corresponding to the eigenvector used to bipartition this segment

5.11.4.2 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const double* ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::eigVec_ [protected]

the eigenvector used to bipartition this segment

5.11.4.3 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> unsigned int ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::eigVecNum_ [protected]

the number of the eigenvector that was used for bipartitioning

5.11.4.4 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> std::map<MSK_ELM_TYPE, unsigned int> ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::indices_ [protected]

the list of elements bipartitioned in this node and their indices

5.11.4.5 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> greedy_ptr<NcutNode> ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::leftNode_ [protected]

the left child node represents the further bipartition of one of the two segments resulting from this bipartition

5.11.4.6 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> greedy_ptr<Ncut> ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::ncut_ [protected]

the Ncut that calculated the current eigenvector

5.11.4.7 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> greedy_ptr<Profile<IMG_ELM_TYPE, MSK_ELM_TYPE> > ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::profile_ [protected]

the profile used in this Node

5.11.4.8 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> greedy_ptr<NcutNode> ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::rightNode_ [protected]

the left child node represents the further bipartition of the other of the two segments resulting from this bipartition

5.11.4.9 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const Setting* ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::setting_ [protected]

Contains all needed parameters

5.11.4.10 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> double ncut::NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE >::splitPoint_ [protected]

the threshold for bipartitioning the graph

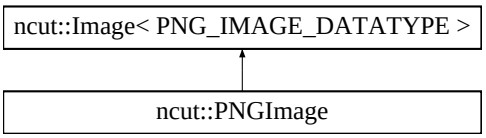
The documentation for this class was generated from the following file:

- /home/stefan/cpp/ncut/src/ncutCore.h

5.12 ncut::PNGImage Class Reference

`#include <ncutImage.h>`

Inheritance diagram for ncut::PNGImage::



Public Member Functions

- int load (const char *filename)
  Load the PNG image file "filename".
- int save (const char *filename) const
  Save this PNGImage into the file "filename".
- PNGImage ()
  PNGImage empty constructor.
- PNGImage (const PNGImage &clone)
  PNGImage copy constructor.
- PNGImage (const char *filename)
  A PNGImage can be contructed using a PNG image stored on disk.
- PNGImage (int width, int height, int nChl)
  PNGImage custom constructor.
- virtual ~PNGImage ()
  PNGImage destructor.
- virtual PNGImage & operator= (const PNGImage &clone)
  PNGImage assignment operator.

5.12.1 Detailed Description

`PNGImage` adds PNG-specific load and save functionality to its base class `Image`.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 ncut::PNGImage::PNGImage ()

PNGImage empty constructor.

Creates an empty image with width=height=channels=0.

5.12.2.2 ncut::PNGImage::PNGImage (const PNGImage & *clone*)

PNGImage copy constructor.

Parameters:
clone The `PNGImage` to be cloned.

5.12.2.3 ncut::PNGImage::PNGImage (const char * *filename*)

A PNGImage can be contructed using a PNG image stored on disk.

Parameters:
filename A char array specifying the location of a PNG image.

Creates a `PNGImage` from the PNG image located at `filename`.

5.12.2.4 ncut::PNGImage::PNGImage (int *width*, int *height*, int *nChl*)

PNGImage custom constructor.

Parameters:
width The width of the image.
height The height of the image.
nChl The number of channels of the image.

Creates an empty PNG image with the specified dimensions and initializes the data to zero.

5.12.3 Member Function Documentation

5.12.3.1 int ncut::PNGImage::load (const char * *filename*)

Load the PNG image file "filename".

Parameters:
filename A char array specifying the location of a PNG image.

Returns:
0 if successful.

Replaces this PNG image with the PNG image located at `filename`.

5.12.3.2 PNGImage & ncut::PNGImage::operator= (const PNGImage & *clone*) [virtual]

PNGImage assignment operator.

Parameters:
clone The `PNGImage` to be cloned.

Returns:
This object.

5.12.3.3 int ncut::PNGImage::save (const char * *filename*) const

Save this PNGImage into the file "filename".

Parameters:

filename A char array specifying the destination of this `PNGimage`.

Returns:

0 if successful.

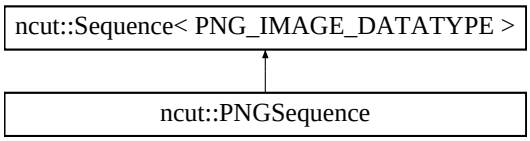
The documentation for this class was generated from the following files:

- /home/stefan/cpp/ncut/src/ncutImage.h
- /home/stefan/cpp/ncut/src/ncutImage.cpp

5.13 ncut::PNGSequence Class Reference

`#include <ncutImage.h>`

Inheritance diagram for ncut::PNGSequence::



Public Member Functions

- PNGImage * **operator[ ]** (unsigned long)
- const PNGImage * **operator[ ]** (unsigned long) const
- int pushThrough (const Image< PNG_IMAGE_DATATYPE > &frame)

  Push a frame onto the sequence and pop the first frame.
- int pushThrough (const PNGImage &frame)

  Push a frame onto the sequence and pop the first frame.
- int pushFrame (const Image< PNG_IMAGE_DATATYPE > &frame)

  Push a frame to the back of the sequence.
- int pushFrame (const PNGImage &frame)

  Push a frame to the back of the sequence.
- void popFrame ()

  Pop the last frame of the sequence.
- int save (const char *filename) const

  Save all PNG images in this sequence.
- PNGSequence ()

  PNGSequence empty constructor
- PNGSequence (const PNGSequence &clone)

  PNGSequence copy constructor.
- virtual ~PNGSequence ()

  PNGSequence destructor.
- virtual PNGSequence & operator= (const PNGSequence &clone)

  PNGSequence assignment operator.

5.13.1 Detailed Description

`PNGSequence` adds PNG-specific save functionality to its base class `Sequence`. Only images of type `PNGImage` are allows in a `PNGSequence`.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 ncut::PNGSequence::PNGSequence ()

PNGSequence empty constructor

Creates an empty `PNGSequence` with=height=channels=0.

5.13.2.2 ncut::PNGSequence::PNGSequence (const PNGSequence & *clone*)

PNGSequence copy constructor.

Parameters:
clone The `PNGSequence` to be cloned.

5.13.3 Member Function Documentation

5.13.3.1 PNGSequence & ncut::PNGSequence::operator= (const PNGSequence & *clone*) [virtual]

PNGSequence assignment operator.

Parameters:
clone The `PNGSequence` to be cloned.

Returns:
This object.

5.13.3.2 void ncut::PNGSequence::popFrame () [virtual]

Pop the last frame of the sequence.

Deletes the last frame and removes it from the sequence. This does nothing if the sequence is empty.

Reimplemented from ncut::Sequence< PNG_IMAGE_DATATYPE >.

5.13.3.3 int ncut::PNGSequence::pushFrame (const PNGImage & *frame*)

Push a frame to the back of the sequence.

Parameters:
frame The `Image` to be pushed onto the sequence.

Returns:
0 if successful.

Pushes a copy of `Image` to the back of the sequence. The length of the sequence is increased by one. Only PNG images of the same dimension as all other images in the sequence are accepted.

5.13.3.4 int ncut::PNGSequence::pushFrame (const Image< PNG_IMAGE_DATATYPE > & *frame*) [virtual]

Push a frame to the back of the sequence.

Parameters:
frame The Image to be pushed onto the sequence.

Returns:
0 if successful.

Pushes a copy of Image to the back of the sequence. The length of the sequence is increased by one. Only PNG images of the same dimension as all other images in the sequence are accepted. frame is cast to a PNGImage.

Reimplemented from ncut::Sequence< PNG_IMAGE_DATATYPE >.

5.13.3.5 int ncut::PNGSequence::pushThrough (const PNGImage & *frame*)

Push a frame onto the sequence and pop the first frame.

Parameters:
frame The Image to be pushed onto the sequence.

Returns:
0 if successful.

Pushes a copy of Image onto the sequence and removes the first frame of the sequence. The length of the sequence remains unchanged. Only PNG images of the same dimension as all other images in the sequence are accepted.

5.13.3.6 int ncut::PNGSequence::pushThrough (const Image< PNG_IMAGE_DATATYPE > & *frame*) [virtual]

Push a frame onto the sequence and pop the first frame.

Parameters:
frame The Image to be pushed onto the sequence.

Returns:
0 if successful.

Pushes a copy of Image onto the sequence and removes the first frame of the sequence. The length of the sequence remains unchanged. Only PNG images of the same dimension as all other images in the sequence are accepted. frame is cast to a PNGImage.

Reimplemented from ncut::Sequence< PNG_IMAGE_DATATYPE >.

5.13.3.7 int ncut::PNGSequence::save (const char * *filename*) const

Save all PNG images in this sequence.

Parameters:

filename The base filename for all images.

Returns:

0 if successful.

The sequence number of each frame is appended to the filename and the frame is saved.

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

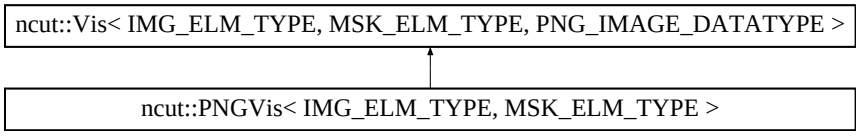
- /home/stefan/cpp/ncut/src/ncutImage.h
- /home/stefan/cpp/ncut/src/ncutImage.cpp

5.14 ncut::PNGVis< IMG_ELM_TYPE, MSK_ELM_TYPE > Class Template Reference

A visualization of a segmentation using png images Creates a visualization of a given segmentation using png images.

`#include <ncutVis.h>`

Inheritance diagram for ncut::PNGVis< IMG_ELM_TYPE, MSK_ELM_TYPE >::



Public Member Functions

- const std::map< MSK_ELM_TYPE, std::vector< int > > & **rememberedPosX** () const
- const std::map< MSK_ELM_TYPE, std::vector< int > > & **rememberedPosY** () const
- const std::map< MSK_ELM_TYPE, double > & **segmentSumArea** () const
- int **firstFrame** (const MSK_ELM_TYPE &segId) const
- virtual void create (PNGSequence &resultSequence, int topFrame=-1)
- virtual int drawLine (int x1, int y1, int x2, int y2, PNGImage *image)
- PNGVis (const Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE > *segmentation)
- virtual ~PNGVis ()

Protected Attributes

- int baseFrame_
- std::vector< PNG_color_t > colors_
- std::map< MSK_ELM_TYPE, unsigned int > colorMap_
- std::map< MSK_ELM_TYPE, std::vector< int > > rememberedPosX_
- std::map< MSK_ELM_TYPE, std::vector< int > > rememberedPosY_
- std::map< MSK_ELM_TYPE, double > segmentSumArea_
- std::map< MSK_ELM_TYPE, int > firstFrame_

5.14.1 Detailed Description

template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> class ncut::PNGVis< IMG_ELM_TYPE, MSK_ELM_TYPE >

A visualization of a segmentation using png images Creates a visualization of a given segmentation using png images.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> ncut::PNGVis< IMG_ELM_TYPE, MSK_ELM_TYPE >::PNGVis (const Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE > * *segmentation*)

PNGVis is constructed by passing a reference to a Segmentation

Parameters:

segmentation reference to a segmenation which should be visualized

5.14.2.2 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> ncut::PNGVis< IMG_ELM_TYPE, MSK_ELM_TYPE >::~PNGVis () [virtual]

PNGVis Destructor

5.14.3 Member Function Documentation

5.14.3.1 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> void ncut::PNGVis< IMG_ELM_TYPE, MSK_ELM_TYPE >::create (PNGSequence & *resultSequence*, int *topFrame* = -1) [virtual]

Creates a visualization for the current frame window of the sequence

Parameters:

resultSequence the sequence where the visualization should be saved

topFrame current Framenumber

5.14.3.2 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> int ncut::PNGVis< IMG_ELM_TYPE, MSK_ELM_TYPE >::drawLine (int *x1*, int *y1*, int *x2*, int *y2*, PNGImage * *image*) [virtual]

Draws a line from (x1,y1) to (x2,y2) in the given image

Parameters:

x1 x-coordinate of the first point

y1 y-coordinate of the first point

x2 x-coordinate of the second point

y2 y-coordinate of the second point

image image where line should be drawn

Returns:

0 if successful

5.14.4 Member Data Documentation

5.14.4.1 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> int ncut::PNGVis< IMG_ELM_TYPE, MSK_ELM_TYPE >::baseFrame_ [protected]

framenumber of first segmented frame

5.14.4.2 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> std::map<MSK_ELM_TYPE, unsigned int> ncut::PNGVis< IMG_ELM_TYPE, MSK_ELM_TYPE >::colorMap_ [protected]

list of possible colors

5.14.4.3 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> std::vector<PNG_color_t> ncut::PNGVis< IMG_ELM_TYPE, MSK_ELM_TYPE >::colors_ [protected]

color of final segment

5.14.4.4 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> std::map<MSK_ELM_TYPE, int> ncut::PNGVis< IMG_ELM_TYPE, MSK_ELM_TYPE >::firstFrame_ [protected]

framenumber of first occurence of a segment

5.14.4.5 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> std::map<MSK_ELM_TYPE,std::vector<int> > ncut::PNGVis< IMG_ELM_TYPE, MSK_ELM_TYPE >::rememberedPosX_ [protected]

history of midpoint positions of a final segment (x-coordinate)

5.14.4.6 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> std::map<MSK_ELM_TYPE,std::vector<int> > ncut::PNGVis< IMG_ELM_TYPE, MSK_ELM_TYPE >::rememberedPosY_ [protected]

history of midpoint positions of a final segment (y-coordinate)

5.14.4.7 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> std::map<MSK_ELM_TYPE,double > ncut::PNGVis< IMG_ELM_TYPE, MSK_ELM_TYPE >::segmentSumArea_ [protected]

total sum of an final segment in all frames

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

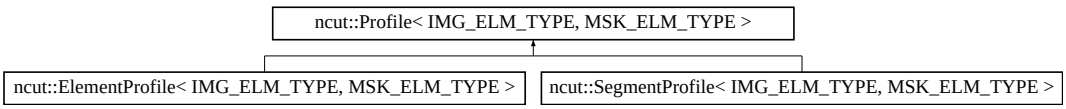
- /home/stefan/cpp/ncut/src/ncutVis.h

5.15 ncut::Profile< IMG_ELM_TYPE, MSK_ELM_TYPE > Class Template Reference

Profile handles similarity information of one frame window.

`#include <motionProfile.h>`

Inheritance diagram for ncut::Profile< IMG_ELM_TYPE, MSK_ELM_TYPE >::



Public Member Functions

- const Matrix * similarity () const

  Similarity matrix of this profile.

- const std::map< MSK_ELM_TYPE, unsigned int > * indices () const

  Get the indices of the regions in this profile.

- const std::vector< MSK_ELM_TYPE > * identifiers () const

  Get the identifiers of the regions in this profile.

- unsigned int idx (const MSK_ELM_TYPE &id) const

  Get the index of a region identified by an id.

- const MSK_ELM_TYPE & id (unsigned int idx) const

  Get the identifier of the region with the given index.

- Profile (const Setting *setting)

  Profile empty constructor.

- Profile (const Profile &clone)

  Profile copy constructor

- Profile (const Matrix &similarity, const std::map< MSK_ELM_TYPE, unsigned int > &indices, const std::vector< MSK_ELM_TYPE > &identifiers, const Setting *setting)

  Profile can be constructed by manually setting all members.

- virtual ~Profile ()

  Profile destructor.

- virtual Profile & operator= (const Profile &clone)

  Profile assignment operator.

Protected Attributes

- Matrix similarity_
- std::map< MSK_ELM_TYPE, unsigned int > indices_
- std::vector< MSK_ELM_TYPE > identifiers_
- const Setting * setting_

5.15.1 Detailed Description

template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> class ncut::Profile< IMG_ELM_TYPE, MSK_ELM_TYPE >

Profile handles similarity information of one frame window.

The base class for `ElementProfile` and `SegmentProfile`. A Profile contains Information about its associated type. Segmentation is done on Elements (e.g. from a prioir segmentation step) using information stored in an `ElementProfile` and the information of the resulting segments is stored in a `SegmentProfile`. Elements and segments are identified by numbers unique in all of the sequence. To get the index used for an element or a segment in a `Profile` (e.g. the index of this element in the similarity matrix) use the function `idx(elementID)`.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> ncut::Profile< IMG_ELM_TYPE, MSK_ELM_TYPE >::Profile (const Setting * *setting*)

Profile empty constructor.

The resulting `Profile` will contain empty indices and identifier maps and an empty similarity Matrix.

5.15.2.2 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> ncut::Profile< IMG_ELM_TYPE, MSK_ELM_TYPE >::Profile (const Profile< IMG_ELM_TYPE, MSK_ELM_TYPE > & *clone*)

Profile copy constructor

Parameters:

clone The `Profile` to be cloned.

5.15.2.3 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> ncut::Profile< IMG_ELM_TYPE, MSK_ELM_TYPE >::Profile (const Matrix & *similarity*, const std::map< MSK_ELM_TYPE, unsigned int > & *indices*, const std::vector< MSK_ELM_TYPE > & *identifiers*, const Setting * *setting*)

Profile can be constructed by manually setting all members.

Parameters:

similarity A similarity matrix.

indices An indices map.

identifiers An identifiers map.

setting The parameters of the current segmentation.

`similarity` , `indices` and `identifiers` must have the same dimension and `indices` and `identifiers` must be valid.

5.15.3 Member Function Documentation

5.15.3.1 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> const MSK_ELM_TYPE& ncut::Profile< IMG_ELM_TYPE, MSK_ELM_TYPE >::id (unsigned int *idx*) const [inline]

Get the identifier of the region with the given index.

Parameters:
idx the idex of the region

Returns:
the id of the specified region

5.15.3.2 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> const std::vector<MSK_ELM_TYPE>* ncut::Profile< IMG_ELM_TYPE, MSK_ELM_TYPE >::identifiers () const [inline]

Get the identifiers of the regions in this profile.

Returns:
the identifiers of the regions

5.15.3.3 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> unsigned int ncut::Profile< IMG_ELM_TYPE, MSK_ELM_TYPE >::idx (const MSK_ELM_TYPE & *id*) const

Get the index of a region identified by an id.

Parameters:
idx the id of the region

Returns:
the index of the specified region

5.15.3.4 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> const std::map<MSK_ELM_TYPE, unsigned int>* ncut::Profile< IMG_ELM_TYPE, MSK_ELM_TYPE >::indices () const [inline]

Get the indices of the regions in this profile.

Returns:
the indices of the regions

5.15.3.5 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> Profile< IMG_ELM_TYPE, MSK_ELM_TYPE > & ncut::Profile< IMG_ELM_TYPE, MSK_ELM_TYPE >::operator= (const Profile< IMG_ELM_TYPE, MSK_ELM_TYPE > & *clone*) [virtual]

Profile assignment operator.

Parameters:
clone The `Profile` to be cloned.

Returns:
This object.

5.15.3.6 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> const Matrix* ncut::Profile< IMG_ELM_TYPE, MSK_ELM_TYPE >::similarity () const [inline]

Similarity matrix of this profile.

Returns:
the similarity matrix of this profile

5.15.4 Member Data Documentation

5.15.4.1 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> std::vector<MSK_ELM_TYPE> ncut::Profile< IMG_ELM_TYPE, MSK_ELM_TYPE >::identifiers_ [protected]

A map from local element indices to global element identifiers (inverse of `indices_`)

5.15.4.2 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> std::map<MSK_ELM_TYPE, unsigned int> ncut::Profile< IMG_ELM_TYPE, MSK_ELM_TYPE >::indices_ [protected]

A map from global element identifiers that are consistently used in every iteration to the local indices these elements have in this profile (e.g. the index in the similarity matrix)

5.15.4.3 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> const Setting* ncut::Profile< IMG_ELM_TYPE, MSK_ELM_TYPE >::setting_ [protected]

Contains all needed parameters

5.15.4.4 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> Matrix ncut::Profile< IMG_ELM_TYPE, MSK_ELM_TYPE >::similarity_ [protected]

The similarity matrix stores the weight (=similarity) from every element to every other element

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

- /home/stefan/cpp/ncut/src/motionProfile.h

5.16 ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE > Class Template Reference

Segmentation represents the complete segmentation of the elements in an image sequence.

`#include <ncutCore.h>`

Public Member Functions

- Setting * setting ()
  Get a pointer to the settings (parameters) used by this Segmentation.
- const Image< IMG_ELM_TYPE > * frame (unsigned int num) const
  Get a pointer to one frame of the frame sequence.
- const Image< MSK_ELM_TYPE > * elmMask (unsigned int num) const
  Get a pointer to one frame of the mask sequence.
- const MSK_ELM_TYPE & elmMask (unsigned int num, unsigned int pxl) const
  Get the value of one pixel of a frame in the mask sequence.
- const IMG_ELM_TYPE & frameMask (unsigned int num, unsigned int pxl) const
  Get the value of one pixel of a frame in the frame sequence.
- unsigned int width () const
  Get the width of the video sequences.
- unsigned int height () const
  Get the height of the video sequences.
- unsigned int nChl () const
  Get the number of channels of the video sequence.
- unsigned int nElm () const
  Get the number of elements in the current frame window.
- double locationX (unsigned int seg) const
  Get the x-midpoint position of a segment in the current frame window.
- double locationY (unsigned int seg) const
  Get the y-midpoint position of a segment in the current frame window.
- unsigned int nSeg () const
  Get the number of segments in the current frame window.
- unsigned int nFrm () const
  Get the number of segmented frames since the creation of this Segmentation.
- unsigned int wSize () const

Get the size of the frame window in frames.

- const std::map< MSK_ELM_TYPE, unsigned int > * indices () const
  Get the index map of the segments in the current frame window.
- unsigned int idx (const MSK_ELM_TYPE &segId) const
  Get the index of a segment in the current frame window.
- const MSK_ELM_TYPE & id (unsigned int segIdx) const
  Get the unique identifier of a segment in the current frame window.
- const MSK_ELM_TYPE & segMask (const MSK_ELM_TYPE &elmId) const
  Get the segment an element is part of.
- const std::vector< unsigned int > * pixels (unsigned int elmIdx) const
  Get a list of pixels in an element.
- double nPixels (int seg) const
  Get the number of pixels in a segment.
- const std::vector< unsigned int > * elements (unsigned int segIdx) const
  Get a list of elements in a segment.
- unsigned int frameOf (unsigned int elmIdx) const
  Get the frame number in the current frame window an element is in.
- unsigned int age (const MSK_ELM_TYPE &segId) const
  Get the age of a segment.
- const std::map< MSK_ELM_TYPE, unsigned int > * age () const
  Get a pointer to the age map.
- const bool inactive (const MSK_ELM_TYPE &segId) const
  Check if a segment is active (present) in the current frame window.
- const std::map< MSK_ELM_TYPE, bool > * inactive () const
  Get a pointer to the inactive map.
- const MSK_ELM_TYPE & simSeg (const MSK_ELM_TYPE &segId) const
  Get the most similar segment to a given segment.
- const NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE > * dendogram () const
  Get the root NcutNode of the dendogram of the current frame window.
- const ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE > * elmProfile () const
  Get the ElementProfile of the current frame window.
- const SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE > * segProfile () const
  Get the SegmentProfile of the current frame window.

- const Setting * setting () const

  Get a pointer to the settings (parameters) used by this Segmentation.

- int push (const Image< IMG_ELM_TYPE > *frame, const Image< MSK_ELM_TYPE > *mask)

  Push frame and mask onto the sequence for segmentation.

- Segmentation (unsigned int wSize=3, Setting setting=Setting())

  Segmentation constructor.

- Segmentation (const Segmentation &clone)

  Segmentation copy constructor.

- virtual ~Segmentation ()

  Segmentation destructor.

- virtual Segmentation & operator= (const Segmentation &clone)

  Segmentation assignment operator.

Protected Attributes

- Setting setting_
- NcutNode< IMG_ELM_TYPE, MSK_ELM_TYPE > * dendogram_
- ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE > elmProfile_
- SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE > segProfile_

5.16.1 Detailed Description

template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> class ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >

Segmentation represents the complete segmentation of the elements in an image sequence.

The datastructure used for segmentation is a dendogram consisting of `NcutNodes`. `Segmentation` contains a pointer to the root `NcutNode`. Each `NcutNode` represents a bipartition of a part of the image into two halves. Each `NcutNode` may have two child NcutNodes representing the further bipartition of each of the two halves. The dendogram has a maximum depth and a desired number of eigenvectors per ncut. Use `push` to push furher images and correspondings masks for segmentation.

5.16.2 Constructor & Destructor Documentation

5.16.2.1 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::Segmentation (unsigned int *wSize* = 3, Setting *setting* = `Setting()`)

Segmentation constructor.

Parameters:

wSize The size of the time window in frames.

setting The parameters of the current segmentation.

Creates a new Segmentation that moves a frame window of `wSize` frames over the whole sequence and calculates ncuts for the window in every step.

5.16.2.2 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::Segmentation (const Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE > & *clone*)

Segmentation copy constructor.

Parameters:

clone The Segmentation to be cloned.

5.16.3 Member Function Documentation

5.16.3.1 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const std::map<MSK_ELM_TYPE, unsigned int>* ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::age () const [inline]

Get a pointer to the age map.

Returns:

Pointer to the age map.

5.16.3.2 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> unsigned int ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::age (const MSK_ELM_TYPE & *segId*) const [inline]

Get the age of a segment.

Parameters:

segId Identifier of the segment.

Returns:

Age of a segment.

5.16.3.3 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const NcutNode<IMG_ELM_TYPE, MSK_ELM_TYPE>* ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::dendogram () const [inline]

Get the root NcutNode of the dendogram of the current frame window.

Returns:

Root NcutNode of the dendogram of the current frame window.

5.16.3.4 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const std::vector<unsigned int>* ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::elements (unsigned int *segIdx*) const [inline]

Get a list of elements in a segment.

Parameters:

segIdx Index of the segment.

Returns:

List of the indices of the elements in the segment.

5.16.3.5 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const MSK_ELM_TYPE& ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::elmMask (unsigned int *num*, unsigned int *pxl*) const [inline]

Get the value of one pixel of a frame in the mask sequence.

Parameters:

num The sequence number of the frame.

pxl The number of the pixel (starting from top left).

Returns:

Value of one pixel of a frame in the mask sequence.

5.16.3.6 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const Image<MSK_ELM_TYPE>* ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::elmMask (unsigned int *num*) const [inline]

Get a pointer to one frame of the mask sequence.

Parameters:

num The sequence number of the frame.

Returns:

Pointer to one frame of the mask sequence.

5.16.3.7 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const ElementProfile<IMG_ELM_TYPE, MSK_ELM_TYPE>* ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::elmProfile () const [inline]

Get the ElementProfile of the current frame window.

Returns:

ElementProfile of the current frame window.

5.16.3.8 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const Image<IMG_ELM_TYPE>* ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::frame (unsigned int *num*) const [inline]

Get a pointer to one frame of the frame sequence.

Parameters:

num The sequence number of the frame.

Returns:

Pointer to one frame of the frame sequence.

5.16.3.9 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const IMG_ELM_TYPE& ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::frameMask (unsigned int *num*, unsigned int *pxl*) const [inline]

Get the value of one pixel of a frame in the frame sequence.

Parameters:

num The sequence number of the frame.

pxl The number of the pixel (starting from top left).

Returns:

Value of one pixel of a frame in the frame sequence.

5.16.3.10 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> unsigned int ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::frameOf (unsigned int *elmIdx*) const [inline]

Get the frame number in the current frame window an element is in.

Parameters:

elmIdx Index of the element.

Returns:

Number of the frame in the current frame window (not the sequence number of the frame).

5.16.3.11 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> unsigned int ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::height () const [inline]

Get the height of the video sequences.

Returns:

Height of the video sequences.

5.16.3.12 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const MSK_ELM_TYPE& ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::id (unsigned int *segIdx*) const [inline]

Get the unique identifier of a segment in the current frame window.

Parameters:
segIdx Index of the segment in the current frame window.

Returns:
Unique identifier of a segment in the current frame window.

5.16.3.13 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> unsigned int ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::idx (const MSK_ELM_TYPE & *segId*) const [inline]

Get the index of a segment in the current frame window.

Parameters:
segId Unique identifier of the segment.

Returns:
Index of a segment in the current frame window.

5.16.3.14 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const std::map<MSK_ELM_TYPE, bool>* ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::inactive () const [inline]

Get a pointer to the inactive map.

Returns:
Pointer to the inactive map.

5.16.3.15 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const bool ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::inactive (const MSK_ELM_TYPE & *segId*) const [inline]

Check if a segment is active (present) in the current frame window.

Parameters:
segId Identifier of the segment.

Returns:
true if inactive, false otherwise.

5.16.3.16 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const std::map<MSK_ELM_TYPE, unsigned int>* ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::indices () const [inline]

Get the index map of the segments in the current frame window.

Returns:
Index map of the segments in the current frame window.

5.16.3.17 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> double ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::locationX (unsigned int *seg*) const [inline]

Get the x-midpoint position of a segment in the current frame window.

Parameters:
seg The index of the segment in the current frame window.

Returns:
X-midpoint position of a segment in the current frame window.

5.16.3.18 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> double ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::locationY (unsigned int *seg*) const [inline]

Get the y-midpoint position of a segment in the current frame window.

Parameters:
seg The index of the segment in the current frame window.

Returns:
Y-midpoint position of a segment in the current frame window.

5.16.3.19 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> unsigned int ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::nChl () const [inline]

Get the number of channels of the video sequence.

Returns:
Number of channels of the video sequence.

5.16.3.20 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> unsigned int ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::nElm () const [inline]

Get the number of elements in the current frame window.

Returns:
Number of elements in the current frame window.

5.16.3.21 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> unsigned int ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::nFrm () const [inline]

Get the number of segmented frames since the creation of this Segmentation.

Returns:
Number of segmented frames since the creation of this Segmentation.

5.16.3.22 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> double ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::nPixels (int *seg*) const [inline]

Get the number of pixels in a segment.

Parameters:
seg Index of the segment.

Returns:
Number of pixels in a segment.

5.16.3.23 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> unsigned int ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::nSeg () const [inline]

Get the number of segments in the current frame window.

Returns:
Number of segments in the current frame window.

5.16.3.24 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE > & ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::operator= (const Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE > & *clone*) [virtual]

Segmentation assignment operator.

Parameters:
clone The Segmentation to be cloned.

Returns:
This object.

5.16.3.25 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const std::vector<unsigned int>* ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::pixels (unsigned int *elmIdx*) const [inline]

Get a list of pixels in an element.

Parameters:
elmIdx Index of the element.

Returns:
List of pixels in an element.

5.16.3.26 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> int ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::push (const Image< IMG_ELM_TYPE > * *frame*, const Image< MSK_ELM_TYPE > * *mask*)

Push frame and mask onto the sequence for segmentation.

Parameters:
frame The frame image that should be pushed to the frame sequence for segmentation.
mask The pre-segmentation mask for the frame image.

Returns:
1 if there aren't enough frames in the sequence to start segmentation yet, 0 otherwise.

Pushes frame and mask to the back of the frame and mask sequences, calculates new profiles, constructs a new ncut dendogram and updates segment information.

5.16.3.27 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const MSK_ELM_TYPE& ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::segMask (const MSK_ELM_TYPE & *elmId*) const [inline]

Get the segment an element is part of.

Parameters:
elmId Identifier of the element.

Returns:
Identifier of the segment the element is part of.

5.16.3.28 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const SegmentProfile<IMG_ELM_TYPE, MSK_ELM_TYPE>* ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::segProfile () const [inline]

Get the SegmentProfile of the current frame window.

Returns:
SegmentProfile of the current frame window.

5.16.3.29 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const Setting* ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::setting () const [inline]

Get a pointer to the settings (parameters) used by this Segmentation.

Returns:
Pointer to the settings (parameters) used by this Segmentation. This function is for read access only.

5.16.3.30 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> Setting* ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::setting () [inline]

Get a pointer to the settings (parameters) used by this Segmentation.

Returns:
Pointer to the settings (parameters) used by this Segmentation. This function is for read&write access.

5.16.3.31 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const MSK_ELM_TYPE& ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::simSeg (const MSK_ELM_TYPE & *segId*) const [inline]

Get the most similar segment to a given segment.

Parameters:
segId Identifier of the segment.

Returns:
The most similar segment to the given segment.

5.16.3.32 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> unsigned int ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::width () const [inline]

Get the width of the video sequences.

Returns:
Width of the video sequences.

5.16.3.33 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> unsigned int ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::wSize () const [inline]

Get the size of the frame window in frames.

Returns:
Size of the frame window in frames.

5.16.4 Member Data Documentation

5.16.4.1 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> NcutNode<IMG_ELM_TYPE, MSK_ELM_TYPE>* ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::dendogram_ [protected]

the root node of the ncut dendogram

5.16.4.2 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> ElementProfile<IMG_ELM_TYPE, MSK_ELM_TYPE> ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::elmProfile_ [protected]

holds the element information and the similarity matrix

5.16.4.3 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> SegmentProfile<IMG_ELM_TYPE, MSK_ELM_TYPE> ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::segProfile_ [protected]

holds the segment information

5.16.4.4 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> Setting ncut::Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE >::setting_ [protected]

Contains all needed parameters

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

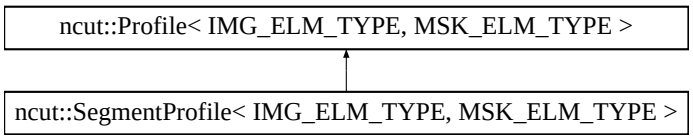
- /home/stefan/cpp/ncut/src/ncutCore.h

5.17 ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE > Class Template Reference

SegmentProfile handles the similarity information of the segments in a frame window.

`#include <motionProfile.h>`

Inheritance diagram for ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::



Public Member Functions

- const std::vector< std::vector< MSK_ELM_TYPE > > * elements () const
  Get a list of elements for each segment in this profile.
- const std::vector< MSK_ELM_TYPE > * elements (unsigned int idx) const
  Get a list of elements for a segment.
- const ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE > * elmProfile () const
  Get the element profile for the elements contained in the segments.
- const MSK_ELM_TYPE & mask (const MSK_ELM_TYPE &elmId) const
  Get the segment id for the element identified by `elmId`.
- unsigned int idxMask (unsigned int elmIdx) const
  Get the segment index for the element with the given index.
- unsigned int age (const MSK_ELM_TYPE &segId) const
  Get the age of a segment.
- bool inactive (const MSK_ELM_TYPE &segId) const
  Get the active/inactive status of a segment.
- const std::map< MSK_ELM_TYPE, bool > * inactive () const
  Get the active/inactive list.
- const std::map< MSK_ELM_TYPE, unsigned int > * age () const
  Get the age list.
- const MSK_ELM_TYPE & simSeg (const MSK_ELM_TYPE &segId) const
  Get the most similar segment to the given segment.
- unsigned int nElm () const
  Get the number of elements in this segment profile.

- unsigned int nSeg () const

  Get the number of segments in this segment profile.

- unsigned int nChl () const

  Get the number of channels of the frames in this frame window.

- double locationX (unsigned int seg) const

  Get the mean location x-coordinate of a segment.

- double locationY (unsigned int seg) const

  Get the mean location y-coordinate of a segment.

- unsigned int nPixels (unsigned int seg) const

  Get the number of pixels in a segment.

- double dif (unsigned int idx1, unsigned int idx2, const std::vector< std::vector< MSK_ELM_TYPE > > *elements=NULL) const

  Calculate the difference between two segments.

- int push (const ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE > *elmProfile, const std::vector< std::vector< MSK_ELM_TYPE > > &segments)

  Calculate segment info of a new frame window and segmentation.

- int calcSim ()

  Calculate segment similarity.

- SegmentProfile (const Setting *setting)

  SegmentProfile empty constructor.

- SegmentProfile (const SegmentProfile &clone)

  SegmentProfile copy constructor.

- virtual ~SegmentProfile ()

  SegmentProfile destructor.

- virtual SegmentProfile & operator= (const SegmentProfile &clone)

  SegmentProfile assignment operator.

Protected Attributes

- std::vector< std::vector< MSK_ELM_TYPE > > elements_
- const ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE > * elmProfile_
- std::map< MSK_ELM_TYPE, MSK_ELM_TYPE > mask_
- std::vector< unsigned int > idxMask_
- std::map< MSK_ELM_TYPE, unsigned int > age_
- std::map< MSK_ELM_TYPE, bool > inactive_
- std::map< MSK_ELM_TYPE, MSK_ELM_TYPE > simSeg_
- unsigned int nSeg_
- unsigned int nChl_

- MSK_ELM_TYPE **nextSegId_**
- Matrix neighbors_
- double ** intensity_
- double * locationX_
- double * locationY_
- unsigned int * nElements_
- unsigned int * nPixels_

5.17.1 Detailed Description

template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> class ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >

SegmentProfile handles the similarity information of the segments in a frame window.

A `SegmentProfile` is used to store information about the segments resulting of the segmentation of one frame window. `SegmentProfile` is also used to determine the connections of segments from this frame window to segments from the last frame window. Throghout the sequence, segments are connected through their identifiers. segments that appear in consecutive frames and have the same identifiers are to be considered the same segment.

5.17.2 Constructor & Destructor Documentation

5.17.2.1 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::SegmentProfile (const Setting * *setting*)

SegmentProfile empty constructor.

Parameters:

setting The parameters of the current segmentation.

Construct an empty `SegmentProfile`.

5.17.2.2 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::SegmentProfile (const SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE > & *clone*)

SegmentProfile copy constructor.

Parameters:

clone The `SegmentProfile` to be cloned.

5.17.3 Member Function Documentation

5.17.3.1 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> const std::map<MSK_ELM_TYPE, unsigned int>* ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::age () const [inline]

Get the age list.

Returns:
A map from segment identifiers to integers indicating the age of a segment with that identifier.

5.17.3.2 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> unsigned int ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::age (const MSK_ELM_TYPE & *segId*) const

Get the age of a segment.

Parameters:
segId The identifier of the segment.

Returns:
The age of a segment with the given identifier.

5.17.3.3 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> int ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::calcSim ()

Calculate segment similarity.

Returns:
0 if successful

Calculate the similarity of every segment to every other segment.

5.17.3.4 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> double ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::dif (unsigned int *idx1*, unsigned int *idx2*, const std::vector< std::vector< MSK_ELM_TYPE > > * *elements* = NULL) const

Calculate the difference between two segments.

Parameters:
idx1 Index of segment 1.

idx2 Index of segment 2.

elements A segmentation, this paramter is optional.

Returns:
The difference between segment 1 and segment 2.

If the segments don't touch, a maximal value of 500 is returned. Optionally, a segmentation different from the segmentation in this SegmentProfile can be specified.

5.17.3.5 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> const std::vector<MSK_ELM_TYPE>* ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::elements (unsigned int *idx*) const [inline]

Get a list of elements for a segment.

Parameters:
idx The index of the segment.

Returns:
A list of elements for the given segment.

5.17.3.6 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> const std::vector<std::vector<MSK_ELM_TYPE> >* ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::elements () const [inline]

Get a list of elements for each segment in this profile.

Returns:
A list of elements for each segment in this profile. This function returns a vector containing vectors of element indices. Vector i contains the element indices of the elements contained in the segment with index i.

5.17.3.7 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> const ElementProfile<IMG_ELM_TYPE, MSK_ELM_TYPE>* ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::elmProfile () const [inline]

Get the element profile for the elements contained in the segments.

Returns:
The element profile for the elements contained in the segments.

5.17.3.8 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> unsigned int ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::idxMask (unsigned int *elmIdx*) const [inline]

Get the segment index for the element with the given index.

Parameters:
elmIdx The index of the element.

Returns:
The segment id for an element identified by `elmId`.

5.17.3.9 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> const std::map<MSK_ELM_TYPE, bool>* ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::inactive () const [inline]

Get the active/inactive list.

Returns:
A list of booleans indicating wether a segment is active or inactive (true for inactive, false for active).

5.17.3.10 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> bool ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::inactive (const MSK_ELM_TYPE & *segId*) const

Get the active/inactive status of a segment.

Parameters:
segId The identifier of the segment.

Returns:
true if the segment is inactive, false otherwise.

5.17.3.11 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::locationX (unsigned int *seg*) const [inline]

Get the mean location x-coordinate of a segment.

Parameters:
seg The index of the segment.

Returns:
The mean location x-coordinate of a segment (in pixels).

5.17.3.12 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::locationY (unsigned int *seg*) const [inline]

Get the mean location y-coordinate of a segment.

Parameters:
seg The index of the segment.

Returns:
The mean location y-coordinate of a segment (in pixels).

5.17.3.13 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const MSK_ELM_TYPE & ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::mask (const MSK_ELM_TYPE & *elmId*) const

Get the segment id for the element identified by `elmId`.

Parameters:
elmId The identifier of the element.

Returns:
The segment id for an element identified by `elmId`.

5.17.3.14 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> unsigned int ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::nChl () const [inline]

Get the number of channels of the frames in this frame window.

Returns:
The number of channels of the frames in this frame window.

5.17.3.15 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> unsigned int ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::nElm () const [inline]

Get the number of elements in this segment profile.

Returns:
The number of elements in this segment profile.

5.17.3.16 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> unsigned int ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::nPixels (unsigned int *seg*) const [inline]

Get the number of pixels in a segment.

Parameters:
seg The index of the segment.

Returns:
The number of pixels of the given segment.

5.17.3.17 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> unsigned int ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::nSeg () const [inline]

Get the number of segments in this segment profile.

Returns:
The number of segments in this segment profile.

5.17.3.18 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE > & ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::operator= (const SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE > & *clone*) [virtual]

SegmentProfile assignment operator.

Parameters:
clone The `SegmentProfile` to be cloned.

Returns:
This object.

5.17.3.19 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> int ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::push (const ElementProfile< IMG_ELM_TYPE, MSK_ELM_TYPE > * *elmProfile*, const std::vector< std::vector< MSK_ELM_TYPE > > & *segments*)

Calculate segment info of a new frame window and segmentation.

Parameters:

elmProfile An `ElementProfile`.

segments A segmentation on the elements of `elmProfile`.

Returns:

0 if successful

Calculate all needed segment info using information from `elmProfile` and connect the new segments to the segments from the last frame window.

5.17.3.20 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE> const MSK_ELM_TYPE & ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::simSeg (const MSK_ELM_TYPE & *segId*) const

Get the most similar segment to the given segment.

Parameters:

segId The identifier of the segment.

Returns:

The segment most similar to the given segment.

5.17.4 Member Data Documentation

5.17.4.1 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> std::map<MSK_ELM_TYPE, unsigned int> ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::age_ [protected]

the number of frames each segment has been active (= in the sequence) so far

5.17.4.2 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> std::vector<std::vector<MSK_ELM_TYPE> > ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::elements_ [protected]

a list of elements for each segment

5.17.4.3 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> const ElementProfile<IMG_ELM_TYPE, MSK_ELM_TYPE>* ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::elmProfile_ [protected]

the `ElementProfile` containing information about the elements the segments consist of

5.17.4.4 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> std::vector<unsigned int> ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::idxMask_ [protected]

a map from element indices to the segment indices of the segments they are part of

5.17.4.5 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> std::map<MSK_ELM_TYPE, bool> ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::inactive_ [protected]

true if a segment from the last frame window wasn't assigned to any segment in this frame window

5.17.4.6 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double** ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::intensity_ [protected]

the mean intensity of each image channel

5.17.4.7 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double* ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::locationX_ [protected]

the center-of-gravity X coordinate

5.17.4.8 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> double* ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::locationY_ [protected]

the center-of-gravity Y coordinate

5.17.4.9 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> std::map<MSK_ELM_TYPE, MSK_ELM_TYPE> ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::mask_ [protected]

a map from element identifiers to the segment identifers of the segments they are part of

5.17.4.10 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> unsigned int ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::nChl_ [protected]

the number of channels of the images

5.17.4.11 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> Matrix ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::neighbors_ [protected]

the number of neigboring pixels of every segment to every other segment

5.17.4.12 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> unsigned int* ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::nElements_ [protected]

the number of elements in each segment

5.17.4.13 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> unsigned int* ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::nPixels_ [protected]

the number of pixels in each segment

5.17.4.14 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> unsigned int ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::nSeg_ [protected]

the number of segments in this frame window

5.17.4.15 template<class IMG_ELM_TYPE = unsigned char, class MSK_ELM_TYPE = signed long> std::map<MSK_ELM_TYPE, MSK_ELM_TYPE> ncut::SegmentProfile< IMG_ELM_TYPE, MSK_ELM_TYPE >::simSeg_ [protected]

the most similar segment of each segment

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

- /home/stefan/cpp/ncut/src/motionProfile.h

5.18 ncut::Sequence< ELEMENT_TYPE > Class Template Reference

`#include <ncutImage.h>`

Public Member Functions

- virtual Image< ELEMENT_TYPE > * **operator[ ]** (unsigned long)
- virtual const Image< ELEMENT_TYPE > * **operator[ ]** (unsigned long) const
- unsigned int width () const

  Get the with of the images.

- unsigned int height () const

  Get the height of the images.

- unsigned int nChl () const

  Get the number of channels of the images.

- unsigned int length () const

  Get the length of the sequence.

- virtual int pushThrough (const Image< ELEMENT_TYPE > &frame)

  Push a frame onto the sequence and pop the first frame.

- virtual int pushFrame (const Image< ELEMENT_TYPE > &frame)

  Push a frame to the back of the sequence.

- virtual void popFrame ()

  Pop the last frame of the sequence.

- virtual int clear ()

  Remove all frames from the sequence.

- Sequence ()

  Sequence empty constructor

- Sequence (const Sequence &clone)

  Sequence copy constructor.

- virtual ~Sequence ()

  Sequence destructor.

- virtual Sequence & operator= (const Sequence &clone)

  Sequence assignment operator.

Protected Member Functions

- bool validateFrame (const Image< ELEMENT_TYPE > &frame)

  Checks if a frame can be added to this sequence.

Protected Attributes

- std::deque< Image< ELEMENT_TYPE > * > frames_
- unsigned int width_
- unsigned int height_
- unsigned int nChl_

5.18.1 Detailed Description

template<class ELEMENT_TYPE = unsigned char> class ncut::Sequence< ELEMENT_TYPE >

`Sequence` is the base class for `PNGSequence` and `BSISequence`. It stores pointers to a sequence images. `Sequence` is a small wrapper around a STL deque containing pointers to all `Sequence` s in the sequence, providing additional information, such as width, height and length, and special sequence manipulation functions.

5.18.2 Constructor & Destructor Documentation

5.18.2.1 template<class ELEMENT_TYPE> ncut::Sequence< ELEMENT_TYPE >::Sequence ()

Sequence empty constructor

Creates an empty `Sequence` with=height=channels=0.

5.18.2.2 template<class ELEMENT_TYPE> ncut::Sequence< ELEMENT_TYPE >::Sequence (const Sequence< ELEMENT_TYPE > & *clone*)

Sequence copy constructor.

Parameters:

clone The `Sequence` to be cloned.

5.18.3 Member Function Documentation

5.18.3.1 template<class ELEMENT_TYPE> int ncut::Sequence< ELEMENT_TYPE >::clear () [virtual]

Remove all frames from the sequence.

Returns:

0 if successful.

Deletes all frames in the sequence and sets width, height and number of channels to zero. The length of the sequence is zero.

Reimplemented in ncut::BSISequence.

5.18.3.2 template<class ELEMENT_TYPE = unsigned char> unsigned int ncut::Sequence< ELEMENT_TYPE >::height () const [inline]

Get the height of the images.

Returns:
The height of the images

5.18.3.3 template<class ELEMENT_TYPE = unsigned char> unsigned int ncut::Sequence< ELEMENT_TYPE >::length () const [inline]

Get the length of the sequence.

Returns:
The length of the sequence

5.18.3.4 template<class ELEMENT_TYPE = unsigned char> unsigned int ncut::Sequence< ELEMENT_TYPE >::nChl () const [inline]

Get the number of channels of the images.

Returns:
The number of channles of the images

5.18.3.5 template<class ELEMENT_TYPE> Sequence< ELEMENT_TYPE > & ncut::Sequence< ELEMENT_TYPE >::operator= (const Sequence< ELEMENT_TYPE > & *clone*) [virtual]

Sequence assignment operator.

Parameters:
clone The Sequence to be cloned.

Returns:
This object.

5.18.3.6 template<class ELEMENT_TYPE> void ncut::Sequence< ELEMENT_TYPE >::popFrame () [virtual]

Pop the last frame of the sequence.

Deletes the last frame and removes it from the sequence. This does nothing if the sequence is empty.

Reimplemented in ncut::PNGSequence, and ncut::BSISequence.

5.18.3.7 template<class ELEMENT_TYPE> int ncut::Sequence< ELEMENT_TYPE >::pushFrame (const Image< ELEMENT_TYPE > & *frame*) [virtual]

Push a frame to the back of the sequence.

Parameters:
frame The `Image` to be pushed onto the sequence.

Returns:
0 if successful.

Pushes a copy of `Image` to the back of the sequence. The length of the sequence is increased by one. Only images of the same dimension as all other images in the sequence are accepted.

Reimplemented in ncut::PNGSequence, and ncut::BSISequence.

5.18.3.8 template<class ELEMENT_TYPE> int ncut::Sequence< ELEMENT_TYPE >::pushThrough (const Image< ELEMENT_TYPE > & *frame*) [virtual]

Push a frame onto the sequence and pop the first frame.

Parameters:
frame The `Image` to be pushed onto the sequence.

Returns:
0 if successful.

Pushes a copy of `Image` onto the sequence and removes the first frame of the sequence. The length of the sequence remains unchanged. Only images of the same dimension as all other images in the sequence are accepted.

Reimplemented in ncut::PNGSequence, and ncut::BSISequence.

5.18.3.9 template<class ELEMENT_TYPE> bool ncut::Sequence< ELEMENT_TYPE >::validateFrame (const Image< ELEMENT_TYPE > & *frame*) [protected]

Checks if a frame can be added to this sequence.

Parameters:
frame The frame to be checked.

Returns:
`true` if the frame has the right dimension, `false` otherwise.

Every frame in a sequence has to be of the same dimension. This function checks if width, height and number of channels match.

5.18.3.10 template<class ELEMENT_TYPE = unsigned char> unsigned int ncut::Sequence< ELEMENT_TYPE >::width () const [inline]

Get the with of the images.

Returns:
The width of the images

5.18.4 Member Data Documentation

5.18.4.1 template<class ELEMENT_TYPE = unsigned char> std::deque<Image<ELEMENT_TYPE>* > ncut::Sequence< ELEMENT_TYPE >::frames_ [protected]

Contains pointers to all images in this sequence. All images are owned by Sequence

5.18.4.2 template<class ELEMENT_TYPE = unsigned char> unsigned int ncut::Sequence< ELEMENT_TYPE >::height_ [protected]

The height of all images in this sequence.

5.18.4.3 template<class ELEMENT_TYPE = unsigned char> unsigned int ncut::Sequence< ELEMENT_TYPE >::nChl_ [protected]

The number of channels of all images in this sequence.

5.18.4.4 template<class ELEMENT_TYPE = unsigned char> unsigned int ncut::Sequence< ELEMENT_TYPE >::width_ [protected]

The width of all images in this sequence.

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

- /home/stefan/cpp/ncut/src/ncutImage.h

5.19 ncut::Setting Class Reference

all settings needed by the algorithm This class stores all parameters. It is referenced by every class. If parameters are not given in the ini file default values are used.

`#include <ncutSetting.h>`

Public Member Functions

- Setting ()

  Initialize all values to defaults.

- Setting (const Setting &clone)

  Setting copy constructor.

- virtual Setting & operator= (const Setting &clone)

  Setting assignment operator.

- virtual ~Setting ()

  Setting destructor.

Public Attributes

- unsigned int ncut_tree_depth
- unsigned int number_of_eigenvectors
- unsigned int number_of_eigenvector_splittest_steps
- unsigned int variation_bins
- double saturation_multiplier
- double intensity_weight
- double position_weight
- double intensity_falloff
- double distance_falloff
- int stretch_saturation
- int merge_segments
- double merge_threshold
- unsigned int minimum_segment_count
- int stretch_saturation_merge
- int connected_segments
- unsigned int segment_adult_age
- double segment_age_weight
- double min_segment_attach_similarity
- unsigned int segment_delay
- double stretch_point0_x
- double stretch_point0_y
- double stretch_point1_x
- double stretch_point1_y
- double stretch_point2_x
- double stretch_point2_y
- double stretch_point3_x

- double stretch_point3_y
- int visualisation_mode
- int drawTrace
- int drawCross
- int drawLine
- int tracelength

5.19.1 Detailed Description

all settings needed by the algorithm This class stores all parameters. It is referenced by every class. If parameters are not given in the ini file default values are used.

5.19.2 Constructor & Destructor Documentation

5.19.2.1 ncut::Setting::Setting (const Setting & *clone*)

Setting copy constructor.

Parameters:
clone The `Setting` to be cloned.

5.19.3 Member Function Documentation

5.19.3.1 Setting & ncut::Setting::operator= (const Setting & *clone*) `[virtual]`

Setting assignment operator.

Parameters:
clone The `Setting` to be cloned.

Returns:
This object.

5.19.4 Member Data Documentation

5.19.4.1 int ncut::Setting::connected_segments

wheter segmentes consisting of multiple unconnected components should be split (1=yes)

5.19.4.2 double ncut::Setting::distance_falloff

falloff of the similarity along the position difference of two elements

5.19.4.3 int ncut::Setting::drawCross

draw a cross at the actual midpoint position of final segment

5.19.4.4 int ncut::Setting::drawLine

connect old midpoint positions of final segments

5.19.4.5 int ncut::Setting::drawTrace

draw old midpoint positions of final segments

5.19.4.6 double ncut::Setting::intensity_falloff

falloff of the similarity along the intensity difference of two elements

5.19.4.7 double ncut::Setting::intensity_weight

weight for the intensity used for the similarity calculation

5.19.4.8 int ncut::Setting::merge_segments

wheter the segments should be merged (1=on)

5.19.4.9 double ncut::Setting::merge_threshold

the minimum similarty where segments are still merged

5.19.4.10 double ncut::Setting::min_segment_attach_similarity

minimum similarity two segments of different frame windows must have to be considered the same segment

5.19.4.11 unsigned int ncut::Setting::minimum_segment_count

the minimum number of segments, segments won't be merged any further if this number is reached

5.19.4.12 unsigned int ncut::Setting::ncut_tree_depth

the maximum depth of the dendogram

5.19.4.13 unsigned int ncut::Setting::number_of_eigenvector_splittest_steps

number of splitting points in the interval of possible eigenvector values that are tested for their ncut quality

5.19.4.14 unsigned int ncut::Setting::number_of_eigenvectors

number of eigenvectors in each Ncut used for bi-partitioning the image

5.19.4.15 double ncut::Setting::position_weight

weight for the position used for the similarity calculation

5.19.4.16 double ncut::Setting::saturation_multiplier

the saturation of each pixel in a sequence is multiplied by this value

5.19.4.17 unsigned int ncut::Setting::segment_adult_age

the age from which on an element should not get lost if possible

5.19.4.18 double ncut::Setting::segment_age_weight

the influence of the age on the overall frame window similarity

5.19.4.19 unsigned int ncut::Setting::segment_delay

delay in frames after which segments are shown in the result (to increase stability)

5.19.4.20 double ncut::Setting::stretch_point0_x

leftmost point 0 in similarity streching function (x-coordinate)

5.19.4.21 double ncut::Setting::stretch_point0_y

leftmost point 0 in similarity streching function (y-coordinate)

5.19.4.22 double ncut::Setting::stretch_point1_x

point 1 in similarity streching function (x-coordinate)

5.19.4.23 double ncut::Setting::stretch_point1_y

point 1 in similarity streching function (y-coordinate)

5.19.4.24 double ncut::Setting::stretch_point2_x

point 2 in similarity streching function (x-coordinate)

5.19.4.25 double ncut::Setting::stretch_point2_y

point 2 in similarity streching function (y-coordinate)

5.19.4.26 double ncut::Setting::stretch_point3_x

rightmost point 3 in similarity streching function (x-coordinate)

5.19.4.27 double ncut::Setting::stretch_point3_y

rightmost point 3 in similarity streching function (y-coordinate)

5.19.4.28 int ncut::Setting::stretch_saturation

wheter strech saturation is used or not (1=on)

5.19.4.29 int ncut::Setting::stretch_saturation_merge

wheter the saturation should be streched while calculating similarty for merging (1=on)

5.19.4.30 int ncut::Setting::tracelength

length of the trace in frames

5.19.4.31 unsigned int ncut::Setting::variation_bins

number of accumulator bins used for variation measurement

5.19.4.32 int ncut::Setting::visualisation_mode

visualistation mode: 0=segmented Objects 1=original image

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

- /home/stefan/cpp/ncut/src/ncutSetting.h

5.20 ncut::Vis< IMG_ELM_TYPE, MSK_ELM_TYPE, VIS_ELM_TYPE > Class Template Reference

Visualization of a segmenation This class is the abstract class for visualization.

`#include <ncutVis.h>`

Public Member Functions

- const Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE > * seg () const

  Get the segmentation used by this visualization.

- virtual void create (Sequence< VIS_ELM_TYPE > &resultSequence, int topFrame=-1)
- Vis (const Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE > *segmentation)
- virtual ~Vis ()

Protected Attributes

- const Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE > * seg_

5.20.1 Detailed Description

template<class IMG_ELM_TYPE, class MSK_ELM_TYPE, class VIS_ELM_TYPE> class ncut::Vis< IMG_ELM_TYPE, MSK_ELM_TYPE, VIS_ELM_TYPE >

Visualization of a segmenation This class is the abstract class for visualization.

5.20.2 Constructor & Destructor Documentation

5.20.2.1 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE, class VIS_ELM_TYPE> ncut::Vis< IMG_ELM_TYPE, MSK_ELM_TYPE, VIS_ELM_TYPE >::Vis (const Segmentation< IMG_ELM_TYPE, MSK_ELM_TYPE > * *segmentation*)

Vis is constructed by passing a reference to a Segmentation

Parameters:

segmentation reference to a segmenation which should be visualized

5.20.2.2 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE, class VIS_ELM_TYPE> ncut::Vis< IMG_ELM_TYPE, MSK_ELM_TYPE, VIS_ELM_TYPE >::~Vis () [virtual]

Vis destructor

5.20.3 Member Function Documentation

5.20.3.1 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE, class VIS_ELM_TYPE> void ncut::Vis< IMG_ELM_TYPE, MSK_ELM_TYPE, VIS_ELM_TYPE >::create (Sequence< VIS_ELM_TYPE > & *resultSequence*, int *topFrame* = -1) [virtual]

Creates a visualization for the current frame window of the sequence

Parameters:

resultSequence the sequence where the visualization should be saved

topFrame current Framenumber

5.20.3.2 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE, class VIS_ELM_TYPE> const Segmentation<IMG_ELM_TYPE, MSK_ELM_TYPE>* ncut::Vis< IMG_ELM_TYPE, MSK_ELM_TYPE, VIS_ELM_TYPE >::seg () const [inline]

Get the segmentation used by this visualization.

Returns:

The segmentation used by this visualization.

5.20.4 Member Data Documentation

5.20.4.1 template<class IMG_ELM_TYPE, class MSK_ELM_TYPE, class VIS_ELM_TYPE> const Segmentation<IMG_ELM_TYPE, MSK_ELM_TYPE>* ncut::Vis< IMG_ELM_TYPE, MSK_ELM_TYPE, VIS_ELM_TYPE >::seg_ [protected]

segmentation which should be visualized

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

- /home/stefan/cpp/ncut/src/ncutVis.h

Index