WAILI — Wavelets with Integer Lifting

Geert Uytterhoeven[*] Filip Van Wulpen[*]

July 23, 1999

wavelets@cs.kuleuven.ac.be
http://www.cs.kuleuven.ac.be/~wavelets/

Abstract

This manual describes *WAILI*, a wavelet transform library. For more information about the theoretical foundations behind the library, please refer to 'Wavelet Transforms Using the Lifting Scheme' (Report ITA-Wavelets-WP1.1).

Acknowledgements

This research is supported by the Flemish Information Technology Action Program ('Vlaams Actieprogramma Informatietechnologie'), project number ITA/950244.

[*] Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001 Heverlee, Belgium

License Conditions

Contents

1 Features of WAILI

WAILI is a wavelet transform library:

- Uses integer wavelet transforms based on the lifting Scheme
- Provides various wavelet transforms of the Cohen-Daubechies-Feauveau family of biorthogonal wavelets
- Provides crop and merge operations on wavelet-transformed images
- Provides noise reduction based on wavelet thresholding using Generalized Cross Validation
- Provides scaling of images
- Provides edge enhancement of images
- Provides also some simple image operations (addition and subtraction of images)
- Allows different image representations (RGB, YUV, Lab, . . .)

2 Design and Implementation of WAILI

WAILI is meant to operate on two-dimensional images of various kinds. Applications are situated in image processing.

2.1 Design decisions

This section discusses some of the design decisions we made for this library. For more information about the theoretical foundations behind the library, please refer to 'Wavelet Transforms Using the Lifting Scheme' (Report ITA-Wavelets-WP1.1) [10, 11].

We chose to implement two-dimensional wavelet transforms using the integer version of the Lifting Scheme. The wavelets we use are a subclass of the Cohen-Daubechies-Feauveau family of biorthogonal wavelets.

2.1.1 The Lifting Scheme

The Lifting Scheme [8, 9, 7] provides a fast and simple algorithm for arbitrary wavelet transforms [4]. Furthermore the inverse transform is trivial to find.

Although the Lifting Scheme allows to transform signals with a finite length without extending the signal, we did not choose to take this approach. Instead we use the classical symmetric extension [1] because it's easier to implement and suffices for the applications we have in mind.

2.1.2 The integer wavelet transform

In many applications (e.g. image compression and processing) the input data consists of integer samples only. In addition the storage and encoding of integer numbers is easier, compared to floating point numbers.

To take advantage of this we use the integer version of the Lifting Scheme, which maps integers to integers and is reversible, retaining the perfect reconstruction property [2].

All arithmetic operations are done in 16 bit. This should suffice for applications where the input data is 8 bit wide. Of course this can easily be changed if necessary.

2.1.3 Cohen-Daubechies-Feauveau biorthogonal wavelets

The key benefits of the Cohen-Daubechies-Feauveau biorthogonal wavelets [3] are:

- They have finite support. This preserves the locality of image features.
- The scaling function $\varphi(x)$ is always symmetric, and the wavelet function $\psi(x)$ is always symmetric or antisymmetric. This is important for image processing operations.
- Its filter coefficients are of the form $\frac{z}{2^n}$, with $z \in \mathbf{Z}$ and $n \in \mathbf{N}$. This simplifies the implementation. But unfortunately this feature isn't always preserved by the decomposition in lifting steps.

We choose not to use wavelets with more than 6 vanishing moments to restrict the filter lengths. Longer filters have less locality and thus perform worse in image processing applications, in spite of their increase in smoothness.

We implemented the following wavelet transforms of this family($(n, \tilde{n})$ means that the primal wavelet has n vanishing moments, while the dual wavelet has $\tilde{n}$ vanishing moments):

(1, x): $(1, 1)$, $(1, 3)$, $(1, 5)$
(2, x): $(2, 2)$, $(2, 4)$, $(2, 6)$
(4, x): $(4, 2)$, $(4, 4)$, $(4, 6)$

We deliberately didn't implement any of the $(3, x)$ or $(5, x)$ wavelet transforms because their lifting steps require divisions by 3 or 5, which are not reversible in integer math. $(6, x)$ aren't implemented either because they require more than 16 bits (for 8 bit input data).

2.1.4 Wavelets and translation-invariance

A disadvantage of the wavelet transform is that it's not translation-invariant: if the image is translated before performing the wavelet transform, the result is not a translated version of the wavelet transform of the original image. The redundant wavelet transform is translation-invariant, but it needs much more memory and processing time, so this isn't an option in many applications.

Since we wanted to allow crop and merge operations on wavelet transformed images we came up with the following scheme.

If each transform level is considered independently, one step of a wavelet transform is translation-invariant if the translation is limited to an even number of pixels. Thus we associate with every matrix *coordinates* (a horizontal and vertical offset for the upper left pixel) which depend on the transform level. At every transform level we have two versions of the wavelet transform: an *even* and an *odd* version. Which transform is used depends on the parity of the offset.

If the parities of the coordinates match at each level, we can merge two images without retransforming one of them. If they don't match, we have to retransform one image. The main idea behind this scheme is that in many cases the coordinates of the subimage that will be pasted into another image are known in advance, so it can be transformed correctly. An example of this is the creation of one large image by concatenating several separately created subimages.

2.2 Implementation

The software library is written in C++. We extensively use features of the ISO C++ Standard, which was finalized in November 1997 (from now on called *C++ 97*), since they provide a great enrichment of the C++ language and allow for a cleaner design.

Unfortunately there aren't many compilers that adhere to C++ 97 yet. The development was done using *GNU C++ 2.7.2* and *egcs 1.0*. Fortunately these compilers are available for about any platform, and they're free[1]!

2.2.1 *Image* objects

An *Image* consists of one or more independent channels, thus allowing for different sizes and wavelet transform types per channel. No interpretation or format is imposed on the channels and its data. The actual meaning of the image data can be freely choosen by the user. Examples are grayscale, RGB, YUV or Lab color, etc. . . .

[1] Available from `ftp://prep.ai.mit.edu/pub/gnu/` and `http://egcs.cygnus.com/`.

2.2.2 *Channel* objects

The basic building block of the library is the *Channel*. A channel is a rectangular matrix containing one-valued pixels. A channel can be non-transformed (a *NTChannel*), or wavelet-transformed (a *LChannel*[2]).

Since a wavelet transform is some kind of *recursive* transform, a LChannel contains some subchannels (subbands), which can be either non-transformed or wavelet-transformed. The number of subchannels in a LChannel depends on the type of wavelet transform. You can have the following combinations:

LChannelCR Obtained by transforming both the columns and rows of a NTChannel. As a result, you have 4 subbands:

- **LL** Low pass band in both the horizontal and the vertical direction,
- **LH** Low pass band in the vertical direction, high pass in the horizontal direction,
- **HL** High pass band in the vertical direction, low pass in the horizontal direction,
- **HH** High pass band in both the horizontal and the vertical direction.

LChannelC Obtained by transforming only the columns of a NTChannel. As a result, you have 2 subbands:

- **L** Low pass band in the vertical direction,
- **H** High pass band in the vertical direction.

LChannelR Obtained by transforming only the rows of a NTChannel. As a result, you have 2 subbands:

- **L** Low pass band in the horizontal direction,
- **H** High pass band in the horizontal direction.

Fig. 1 shows an example of a channel after two transform levels.

2.2.3 *Wavelet* objects

A *Wavelet* represents the filters and lifting steps associated with a specific wavelet transform. Some wavelet transforms of the Cohen-Daubechies-Feauveau family are implemented.

You can add your own favorite wavelet transform if you have a decomposition in integer lifting steps for it.

[2] Rumors say that *NT* and *L* refer to two popular operating systems — with the goal of this project to convert as many NTChannels to LChannels as possible — but this hasn't been confirmed officially.

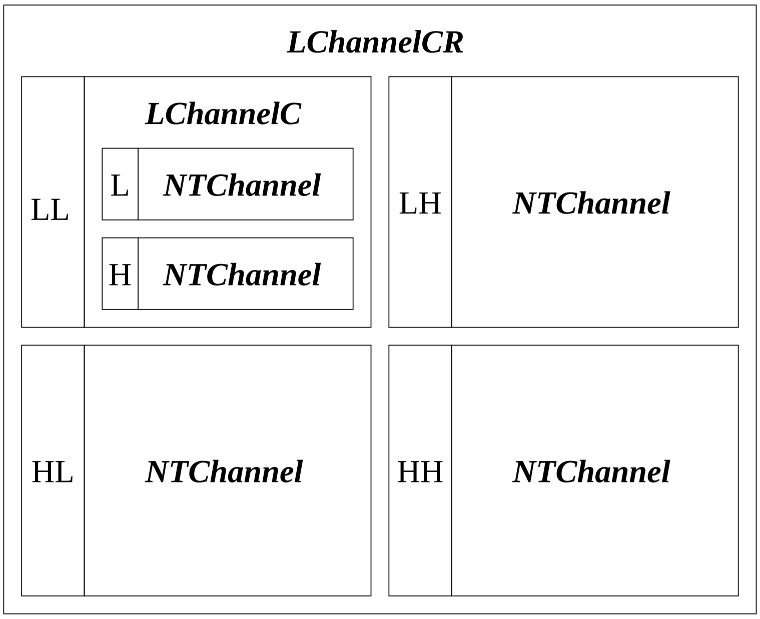

Figure 1: Example of a channel after two transform levels. In the first step both the columns and the rows are transformed, in the second step only the columns are transformed.

3 Overview of WAILI

The wavelet transform library consists of the following parts:

Blit Low-level block operations
Channel Generic channel class
Color Various color representations
ColorSpace Color spaces and color space conversions
Compiler Compiler dependent definitions
Image Generic image class
Lifting Lifting steps for the Lifting Scheme
Stream Input/output with support for compression
Timer Measurement of execution times
Types Platform independent type definitions
Util Utility routines
Wavelet Wavelet transforms using the Lifting Scheme

Note: Currently only *Image, Channel* and some parts of *Wavelet* (*CreateCDF()*) are of general interest to application programmers. The other parts are only used internally or aren't completely

finished yet (*Color*, *ColorSpace*).

4 Manual pages

Name

Blit — Low-level block operations

Description

This package provides some frequently used low-level block operations. All functions are template functions, allowing for different operand types.

Declaration

#include ⟨waili/Blit.h⟩

Operations

void **Copy** (const Type∗ *src*, Type∗ *dst*, u_int *len*)

Copy *len* objects from *src* to *dst*.

void **Copy** (const Type∗ *src*, Type∗ *dst*)

Copy the object pointed to by *src* to *dst*.

void **Fill** (Type∗ *dst*, u_int *len*, Type *value*)

Fill *len* objects pointed to by *dst* with *value*.

void **Clear** (Type∗ *dst*, u_int *len*)

Clear *len* objects pointed to by *dst*.

void **Clear** (Type∗ *dst*)

Clear the object pointed to by *dst*.

void **CopyRect** (const Type∗ *src*, u_int *sw*, Type∗ *dst*, u_int *dw*, u_int *cols*,
u_int *rows*)

Copy a rectangular block of objects with *cols* columns and *rows* rows from *src* to *dst*. The source area has *sw* columns, while the destination area has *dw* columns.

void **CopyRect** (const Type∗ *src*, u_int *sw*, u_int *sx*, u_int *sy*, Type∗ *dst*,
u_int *dw*, u_int *dx*, u_int *dy*, u_int *cols*, u_int *rows*)

Copy a rectangular block of objects with *cols* columns and *rows* rows from *src* at position (*sx*, *sy*) to *dst* at position (*dx*, *dy*). The source area has *sw* columns, while the destination area has *dw* columns.

void **FillRect** (Type∗ *dst*, u_int *dw*, u_int *cols*, u_int *rows*, Type *value*)

Fill a rectangular block of objects with *cols* columns and *rows* rows, pointed to by *dst* with *value*. The destination area has *dw* columns.

void **FillRect** (Type∗ *dst*, u_int *dw*, u_int *dx*, u_int *dy*, u_int *cols*, u_int *rows*,
Type *value*)

Fill a rectangular block of objects with *cols* columns and *rows* rows, pointed to by *dst* at position (*dx*, *dy*) with *value*. The destination area has *dw* columns.

void **ClearRect** (Type∗ *dst*, u_int *dw*, u_int *cols*, u_int *rows*)

Clear a rectangular block of objects with *cols* columns and *rows* rows, pointed to by *dst*. The destination area has *dw* columns.

void **ClearRect** (Type∗ *dst*, u_int *dw*, u_int *dx*, u_int *dy*, u_int *cols*, u_int *rows*)

Clear a rectangular block of objects with *cols* columns and *rows* rows, pointed to by *dst* at position (*dx*, *dy*). The destination area has *dw* columns.

Revision `Blit.h,v 4.0 1997/05/05 09:46:21 geert Exp`

Name

Channel — Generic channel class

Description

This class provides a low-level channel abstraction. A channel is a (rectangular) matrix containing one-valued pixels (of type `PixType`).

Declaration

#include ⟨waili/Channel.h⟩

Channel is an abstract base class. No instances can be declared. Different channel types are implemented through inheritance.

Channel ( )

Create an empty channel.

Channel (u_int *cols*, u_int *rows*, int *offx* = 0, int *offy* = 0)

Create a channel with given dimensions. *cols* and *rows* are the number of columns respectively rows, *offx* and *offy* are the offsets of the upper left pixel in the universal coordinate system.

Channel (const Channel& *channel*)

Create a new channel by copying channel *channel.*

Public Operations

u_int **GetCols** (void) const

u_int **GetRows** (void) const

Get the number of columns respectively rows of the channel.

int **GetOffsetX** (void) const

int **GetOffsetY** (void) const

Get the offset of upper left pixel of the channel in the universal coordinate system.

Static Operations

Channel∗ **CreateFromDescriptor** (u_int *cols*, u_int *rows*,
const TransformDescriptor *transform[],*
u_int *depth,* int *offsetx* = 0, int *offsety* = 0)

Create a channel with given dimensions. *cols* and *rows* are the number of columns respectively rows, *offsetx* and *offsety* are the offsets of the upper left pixel in the universal coordinate system. The channel will be pretransformed using the transform descriptor *transform* with transform depth *depth.*

Virtual Operations

void **GetMask** (u_int& *maskx*, u_int& *masky*) const

Get the coordinate masks for the offsets. A set bit in a mask corresponds to a bit in the offset that can't be choosen freely without retransforming the channel.

u_int **GetDepth** (void) const

Get the transform depth of the channel.

double **Psnr** (const Channel& *channel*, PixType *maxval* = 255) const

Calculate the Peak Signal to Noise Ratio (in dB) between the current channel en channel *channel*. *maxval* is the Peak Signal value.

u64$*$ **FullHistogram** (PixType& *min*, PixType& *max*, u64& *numpixels*)const

Create a histogram for the current channel. The lower histogram limit will be put in *min*, the upper limit in *max*. The number of analyzed pixels will be put in *numpixels*. The result is an array of length $max - min + 1$ containing the occurrency counts.

double **Entropy** (void)const

Calculate the first order entropy (Shannon-Weaver) for this channel, in bits per pixel.

PixType& **operator()** (u_int *c*, u_int *r*)

PixType **operator()** (u_int *c*, u_int *r*) const

Access the ‘pixel’ at column *c* and row *r*. This may refer to a wavelet coefficient instead of a real pixel value if the channel is wavelet transformed.

void **Clear** (void)

Clear all pixel values to zero.

void **Resize** (u_int *cols*, u_int *rows*)

Change the number of columns and rows of the channel to *cols* respectively *rows*.

Channel$*$ **Clone** (void) const

Make a copy of the current channel.

int **SetOffsetX** (void) const

int **SetOffsetY** (void) const

Change the offset of the channel in the universal coordinate system. If you change the bits that are covered by the corresponding coordinate mask, the channel will be retransformed.

Channel$*$ **Crop** (int *x1*, int *y1*, int *x2*, int *y2*) const

Get a rectangular part of the current channel, of which the upper left corner is positioned at (*x1*, *y1*), and the lower right corner at (*x2*, *y2*).

void **Merge** (const Channel& *channel*)

Paste *channel* into the current channel. The paste position is determined by the offsets of *channel*.

void **Add** (const Channel& *channel*)

void **Subtract** (const Channel& *channel*)

Add respectively subtract *channel* to (from) the current channel. Both channels must have the same number of columns, number of rows, offsets and structure.

Channel∗ **Diff** (const Channel& *channel*) const

This function returns the difference channel between the current channel and *channel*. Both channels must have the same number of columns, number of rows, offsets and structure.

void **Enhance** (f32 *m*)

Enhance the channel by multiplying all pixel values with *m*. If the channel is lifted, then only its high-pass coefficients will be changed.

void **Enhance** (int *m*, u_int *shift*)

Enhance the channel by multiplying all pixel values with *m* and shifting the result *shift* binary positions to the right. If the channel is lifted, then only its high-pass coefficients will be changed.

LChannel∗ **PushFwtStepCR** (const Wavelet& *wavelet*)

LChannel∗ **PushFwtStepC** (const Wavelet& *wavelet*)

LChannel∗ **PushFwtStepR** (const Wavelet& *wavelet*)

Add one transform level, using the wavelet transform specified by *wavelet*. The transform can operate on both columns and rows (*PushFwtStepCR*), on the colums only (*PushFwtStepC*) or on the rows only (*PushFwtStepR*). Note that the current channel will be destroyed!!!

This function can return the following values:

`NULL` The operation wasn't sucessful because the maximum number of transform levels was already reached.

`this` If the result is equal to the current channel, the operation was successful.

Else The operation was successful, and the current channel should be deleted and replaced by the returned channel.

Channel∗ **PopFwtStep** (void)

Remove one transform level. This is the inverse operation of the last *PushFwtStep∗* operation.

u64 **Threshold** (double *threshold*, int *soft* = 0)

Perform hard (*soft = 0*) or soft (*soft = 1*) thresholding with the thresholding value *threshold.* The number of pixels that had a value smaller than *threshold* is returned.

int **IsLifted** (void) const

This function returns non-zero if the current channel is already lifted. *This should be removed later!! The user doesn't need to know what's the internal representation of the channel!*

Channel∗ **Scale** (f32 *s*, const Wavelet& *wavelet*)

Scale the channel with scaling factor *scale.* Note that the current channel will be destroyed!!!

Protected Operations

Channel∗ **UpScale** (u_int *s*, const Wavelet& *wavelet*)

Scale the current channel up. The applied scaling factor is the next power of 2 of *s* if *s* is not a power of 2 by itself. Note that the current channel will be destroyed!!!

Virtual Protected Operations

void **Destroy** (void)

Delete the contents of the channel and zero the number of columns and rows.

Channel∗ **DownScale** (u_int *s*, const Wavelet& *wavelet*)

Scale the current channel down. The applied scaling factor is the previous power of 2 of *s* if *s* is not a power of 2 by itself. Note that the current channel will be destroyed!!!

Static Protected Operations

int **GetEven** (int *len*)

int **GetOdd** (int *len*)

Calculate the number of even respectively odd coefficients for a signal with length *len.*

Derived Classes

These are derived classes of the generic *Channel* class that add support for non-transformed and wavelet transformed channels.

Name

NTChannel — Class for a non-transformed channel.

Declaration

NTChannel ( )

Create an empty non-transformed channel.

NTChannel (u_int *cols*, u_int *rows*, int *offx* = 0, int *offy* = 0)

Create a non-transformed channel with given dimensions. *cols* and *rows* are the number of columns respectively rows, *offx* and *offy* are the offsets of the upper left pixel in the universal coordinate system.

NTChannel (const NTChannel& *channel*)

Create a new non-transformed channel by copying the non-transformed channel *channel*.

Public Operations

void **GetMinMax** (PixType& *min*, PixType& *max*, u_int *smoothing* = 0)const

Return the minimum and maximum pixel values in *min* respectively *max*. *smoothing* defines the degree of neglection of extraordinary pixel values. *Smoothing is not yet implemented*

s32∗ **Correlate** (const NTChannel& *channel*, u_int *diff*) const

Calculate the correlation matrix between the current channel and *channel*. *diff* indicates the difference of resolution level between the current channel and the more coarse *channel*. Note that the returned correlation matrix has the same dimensions as the current channel.

u64∗ **Histogram** (PixType *min*, PixType *max*)

Create a histogram for the current channel. The lower histogram limit will be *min*, the upper limit will be *max*. The result is an array of length $max - min + 1$ containing the occurrency counts.

u64 **ThresholdHard** (u_int *threshold*)

u64 **ThresholdSoft** (u_int *threshold*)

Perform hard or soft thresholding with the thresholding value *threshold*. The number of pixels that had a value smaller than *threshold* is returned.

u_int **OptimalGCVThreshold** (void) const

Calculate the optimal soft thresholding value using a technique called Generalized Cross Validation. Note that the channel should contain at least 1000 pixels to give a meaningful result.

NTChannel∗ **DupliScale** (u_int *s*) const

Create a scaled version of the current channel by duplicating the original pixel values. *s* is the scaling factor.

Virtual Operations

NTChannel∗ **Clone** (void) const

Make a copy of the current channel.

NTChannel∗ **Crop** (int *x1*, int *y1*, int *x2*, int *y2*) const

Get a rectangular part of the current channel, of which the upper left corner is positioned at (*x1*, *y1*), and the lower right corner at (*x2*, *y2*).

NTChannel∗ **Diff** (const Channel& *channel*) const

This function returns the difference channel between the current channel and *channel*. Both channels must have the same number of columns, number of rows, offsets and structure.

LChannel∗ **Fwt** (const TransformDescriptor *transform[]*, u_int *depth*)

Transform the channel using the Fast Wavelet Transform. The two-dimensional wavelet transform will be applied to all channels independently. The type of wavelet transform is determined by the *transform* array and its length *depth*. Note that the current channel will be destroyed!!!

Protected Operations

void **Interpolate** (f32 *s*)

Scale the channel using a linear interpolation scheme.

double **GCV** (u_int *threshold*) const

Calculate the GCV value of the channel for Generalized Cross Validation.

Virtual Protected Operations

NTChannel∗ **DownScale** (u_int *s*, const Wavelet& *wavelet*)

Scale the current channel down. The applied scaling factor is the previous power of 2 of *s* if *s* is not a power of 2 by itself.

Name

LChannel — Class for a wavelet transformed channel.

Declaration

LChannel is an abstract base class. No instances can be declared. Different wavelet transformed channel types are implemented through inheritance.

LChannel (const LChannel& *channel*)

Create a new channel by copying channel *channel*.

Public Operations

Channel$*$& **operator[]** (SubBand *band*)

const Channel$*$& **operator[]** (SubBand *band*) const

Get the subband of type *band*. *Make sure the subband does exist!*

TransformDescriptor$*$ **GetTransform** (void)

Get a transform descriptor array for all transform levels.

int **GetShift** (SubBand *band*)

Get the number of steps (in base-$\sqrt{2}$!) the coefficients of subband *band* have to be shifted to the left to obtain their real values.

NTChannel$*$ **IFwt** (void)

Transform the channel using the inverse fast wavelet transform. Note that the current channel will be destroyed!!!

Virtual Operations

TransformType **GetTransformType** (void) const

Get the transform type for this transform level.

LChannel$*$ **Clone** (void) const

Make a copy of the current channel.

NTChannel$*$ **IFwt** (int *x1*, int *y1*, int *x2*, int *y2*) const

Perform recursively the inverse fast wavelet transform on the rows and columns of the channel determined by the upper left and lower right corner respectively (*x1*, *y1*) and (*x2*, *y2*).

LChannel$*$ **Crop** (int *x1*, int *y1*, int *x2*, int *y2*) const

Get a rectangular part of the current channel, of which the upper left corner is positioned at (*x1*, *y1*), and the lower right corner at (*x2*, *y2*).

Protected Operations

LChannel (const Wavelet& *filter*, u_int *numsubbands*, u_int *cols* $= 0$,
u_int *rows* $= 0$)

Create a channel with given dimensions. *numsubbands* is the number of subbands, *cols* and *rows* are the number of columns respectively rows.

Virtual Protected Operations

NTChannel$*$ **IFwtStep** (void)

Perform one step of the inverse fast wavelet transform. Note that the current channel will be destroyed!!!

void **Lazy** (const NTChannel& *source*)

Calculate the Lazy Wavelet Transform of *source* and store the result in the current channel.

void **ILazy** (NTChannel& *dest*) const

Calculate the inverse Lazy Wavelet Transform of the current channel and put the result in *dest*.

void **CakeWalk** (void)

void **ICakeWalk** (void)

Perform the (inverse) ‘Cake Walk’ operation on the current channel.

LChannel∗ **Crop_rec** (int *x1*, int *y1*, int *x2*, int *y2*, NTChannel∗ *top*,
NTChannel∗ *bottom*, NTChannel∗ *left*,
NTChannel∗ *right*)const

Get a rectangular part of the current channel, of which the upper left corner is positioned at (*x1*, *y1*), and the lower right corner at (*x2*, *y2*). *top*, *bottom*, *left* and *right* are the resulting borders in the LP-band of the higher resolution level which affect lower resolutions.

void **Merge_rec** (const *Channel∗ channel*, NTChannel∗ *top*,
NTChannel∗ *bottom*, NTChannel∗ *left*, NTChannel∗ *right*)

Paste *channel* into the current channel. The paste position is determined by the offsets of *channel*. *top*, *bottom*, *left* and *right* are the resulting borders in the LP-band of the higher resolution level which affect lower resolutions.

Subband Types

Valid subband types are

LL	lowpass in both the vertical and the horizontal direction
LH	lowpass in the vertical, highpass in the horizontal direction
HL	highpass in the vertical, lowpass in the horizontal direction
HH	highpass in both the vertical and the horizontal direction

Name

LChannelCR — Class for a wavelet transformed channel (both columns and rows).

Declaration

LChannelCR (const Wavelet& *filter*)

Create an empty channel.

LChannelCR (const Wavelet& *filter*, u_int *cols*, u_int *rows*, int *offx*, int *offy*)

Create a channel with given dimensions. *cols* and *rows* are the number of columns respectively rows, *offx* and *offy* are the coordinates in the universal coordinate system.

LChannelCR (const LChannelCR& *channel*)

Create a new channel by copying channel *channel*.

Public Operations

u_int **GetClow** (void) const

u_int **GetChigh** (void) const

u_int **GetRlow** (void) const

u_int **GetRhigh** (void) const

Get the number of columns and rows in the low and high frequency subbands.

Virtual Operations

LChannelCR∗ **Clone** (void) const

Make a copy of the current channel.

LChannelCR∗ **Diff** (const Channel& *channel*) const

This function returns the difference channel between the current channel and *channel*. Both channels must have the same number of columns, number of rows, offsets and structure.

LChannelCR∗ **Crop_rec** (int *x1*, int *y1*, int *x2*, int *y2*, NTChannel∗ *top*,
NTChannel∗ *bottom*, NTChannel∗ *left*,
NTChannel∗ *right*)const

Get a rectangular part of the current channel, of which the upper left corner is positioned at (*x1*, *y1*), and the lower right corner at (*x2*, *y2*). *top*, *bottom*, *left* and *right* are the resulting borders in the LP-band of the higher resolution level which affect lower resolutions.

Name

LChannelC — Class for a wavelet transformed channel (columns only).

Declaration

LChannelC (const Wavelet& *filter*)

Create an empty channel.

LChannelC (const Wavelet& *filter*, u_int *cols*, u_int *rows*, int *offx*, int *offy*)

Create a channel with given dimensions. *cols* and *rows* are the number of columns respectively rows, *offx* and *offy* are the coordinates in the universal coordinate system.

LChannelC (const LChannelC& *channel*)

Create a new channel by copying channel *channel.*

Public Operations

u_int **GetRlow** (void) const

u_int **GetRhigh** (void) const

Get the number of rows in the low and high frequency subbands.

Virtual Operations

LChannelC∗ **Clone** (void) const

Make a copy of the current channel.

LChannelC∗ **Diff** (const Channel& *channel*) const

This function returns the difference channel between the current channel and *channel.* Both channels must have the same number of columns, number of rows, offsets and structure.

LChannelC∗ **Crop_rec** (int *x1*, int *y1*, int *x2*, int *y2*, NTChannel∗ *top*,
NTChannel∗ *bottom*, NTChannel∗ *left*,
NTChannel∗ *right*)const

Get a rectangular part of the current channel, of which the upper left corner is positioned at (*x1*, *y1*), and the lower right corner at (*x2*, *y2*). *top*, *bottom*, *left* and *right* are the resulting borders in the LP-band of the higher resolution level which affect lower resolutions.

Name

LChannelR — Class for a wavelet transformed channel (rows only).

Declaration

LChannelR (const Wavelet& *filter*)

Create an empty channel.

LChannelR (const Wavelet& *filter*, u_int *cols*, u_int *rows*, int *offx*, int *offy*)

Create a channel with given dimensions. *cols* and *rows* are the number of columns respectively rows, *offx* and *offy* are the coordinates in the universal coordinate system.

LChannelR (const LChannelR& *channel*)

Create a new channel by copying channel *channel.*

Public Operations

u_int **GetClow** (void) const

u_int **GetChigh** (void) const

Get the number of columns in the low and high frequency subbands.

Virtual Operations

LChannelR∗ **Clone** (void) const

Make a copy of the current channel.

LChannelR∗ **Diff** (const Channel& *channel*) const

This function returns the difference channel between the current channel and *channel*. Both channels must have the same number of columns, number of rows, offsets and structure.

LChannelR∗ **Crop_rec** (int *x1*, int *y1*, int *x2*, int *y2*, NTChannel∗ *top*,
NTChannel∗ *bottom*, NTChannel∗ *left*,
NTChannel∗ *right*)const

Get a rectangular part of the current channel, of which the upper left corner is positioned at (*x1*, *y1*), and the lower right corner at (*x2*, *y2*). *top*, *bottom*, *left* and *right* are the resulting borders in the LP-band of the higher resolution level which affect lower resolutions.

TransformDescriptor

The TransformDescriptor determines the kind of wavelet transform for one transform level. It contains 2 parts:

TransformType **type**

type is the transform type and can be one of:

`TT_ColsRows`	Transform both columns and rows
`TT_Cols`	Transform columns only
`TT_Rows`	Transform rows only

const Wavelet∗ **filter**

filter is a pointer to a wavelet filter.

Dependency Graphs

Fig. 2 Inheritance dependency graph for the channel class hierarchy (*Channel*).

See Also

The *Wavelet* and *Lifting* classes.

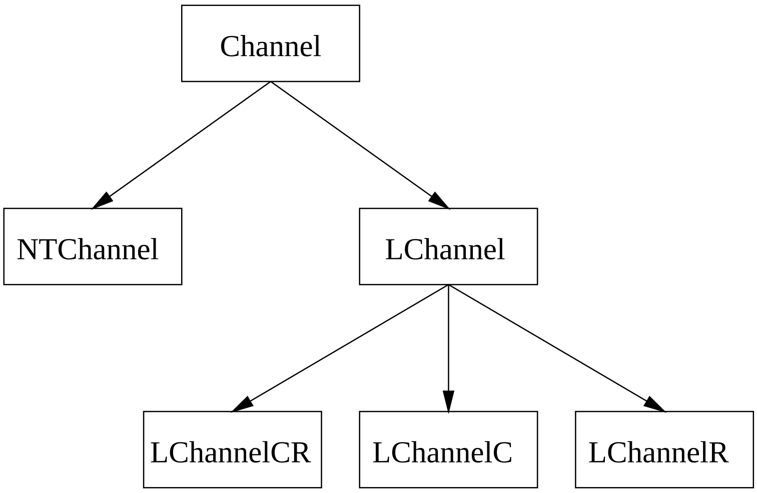

Figure 2: Inheritance dependency graph for the channel class hierarchy (*Channel*).

Revision

```
Channel.C,v 4.4.2.2 1999/04/15 09:35:09 geert Exp
Channel.h,v 4.5.2.3 1999/07/20 13:18:57 geert Exp
LChannel.C,v 4.5.2.2 1999/07/20 13:19:02 geert Exp
LChannel.h,v 4.3.2.2 1999/07/20 13:18:57 geert Exp
LChannelC.C,v 4.6.2.1 1999/07/20 13:19:02 geert Exp
LChannelC.h,v 4.3.2.1 1999/07/20 13:18:57 geert Exp
LChannelCR.C,v 4.6.2.1 1999/07/20 13:19:03 geert Exp
LChannelCR.h,v 4.3.2.1 1999/07/20 13:18:58 geert Exp
LChannelR.C,v 4.6.2.1 1999/07/20 13:19:04 geert Exp
LChannelR.h,v 4.3.2.1 1999/07/20 13:18:58 geert Exp
NTChannel.C,v 4.12.2.3 1999/07/20 13:19:04 geert Exp
NTChannel.h,v 4.8.2.1 1999/07/20 13:18:58 geert Exp
```

Name Color — Various color representations

Description Various color representations.

Declaration #include ⟨waili/Color.h⟩

Color_RGB ( )

Color_XYZ ( )

Color_LAB ( )

Color_RGB8 ( )

Color_XYZ8 ( )

Color_LAB8 ( )

Color_CIEY ( )

Color_CIEL ( )

Color_RGB16 ( )

Color_YUVr16 ( )

Plans Use `PixType` to represent the color components.

See Also The *ColorSpace* class.

Revision

```
Color.C,v 4.0.2.1 1999/04/15 10:06:45 geert Exp
Color.h,v 4.0.2.1 1999/04/15 10:06:42 geert Exp
```

Name ColorSpace — Color spaces and color space conversions

Description Color space specification and conversion.

Declaration #include ⟨waili/Color.h⟩

See Also The various *Color* classes.

Revision

```
Color.C,v 4.0.2.1 1999/04/15 10:06:45 geert Exp
Color.h,v 4.0.2.1 1999/04/15 10:06:42 geert Exp
```

Name Compiler — Compiler dependent definitions

Description This file contains the compiler dependent definitions. Currently these are definitions for *GNU C++* only.

Declaration #include ⟨waili/Compiler.h⟩

Revision `Compiler.h,v 4.1 1997/05/05 09:46:35 geert Exp`

Name

Image — Generic image class

Description

This class provides a low-level image abstraction. An image consists of a number of channels containing pixels (of type `PixType`). Each channel can have a different size. No interpretation or format is imposed on the channels and it's data.

Declaration

#include ⟨waili/Image.h⟩

Image ( )

Create an empty image.

Image (u_int *channels*)

Create an empty image containing*channels* channels.

Image (u_int *cols*, u_int *rows*, u_int *channels* = 1)

Create an image with given dimensions. *cols* and *rows* are the number of columns respectively rows, *channels* is the number of channels. All channels will have the same size.

Image (const u_int *cols[]*, const u_int *rows[]*, u_int *channels*)

Create an image with given dimensions. *cols* and *rows* are arrays containing the number of columns respectively rows for every channel, *channels* is the number of channels.

Image (const Channel& *channel*, u_int *channels* = 1)

Create an image containing*channels* channels. Every channel will be a copy of *channel*.

Image (const Channel∗ *channel[]*, u_int *channels*)

Create an image containing*channels* channels. The channels will be initialized using the array of channels *channel*.

Image (const Image& *im*)

Create a new image by copying image *im*.

Public Operations

ImageType **Import** (const char∗ *filename*, ImageFormat *format* = IF_AUTO)

Import an image from a file named *filename*, stored in a specific format *format*. If *format* is IF_AUTO, the routine will try to guess the file format by examining the file name. The type of the image is returned. Images are assumed to contain 8-bit data, which is converted to the range $-128\ldots127$ for internal use.

void **Export** (const char$*$ *filename*, ImageFormat *format* = IF_AUTO) const

Export the to a file named *filename* using the specific file format *format*. If *format* is IF_AUTO, the routine will try to guess the file format by examining the file name. The pixel values are considered to lay within the range $-128\ldots127$. If a pixel value doesn't fit, it will be clipped.

Note: export in *IF_TIFF* is not yet supported.

void **Convert** (ImageType *from*, ImageType *to*)

Convert the image from type *from* to type *to*. *Not all conversions are implemented yet.*

u_int **GetChannels** (void) const

u_int **GetCols** (void) const

u_int **GetRows** (void) const

Get the number of channels, columns or rows of the image.

int **GetOffsetX** (void) const

int **GetOffsetY** (void) const

Get the offset of the first channel of the image in the universal coordinate system.

Channel$*$& **operator[]** (u_int *channel*)

const$*$ Channel& **operator[]** (u_int *channel*) const

Access channel *channel*.

PixType& **operator()** (u_int *c*, u_int *r*, u_int *ch* = 0)

const PixType **operator()** (u_int *c*, u_int *r*, u_int *ch* = 0) const

Access the ‘pixel’ at column *c* and row *r* in channel *ch*. This may refer to a wavelet coefficient instead of a real pixel value if the channel is wavelet transformed.

void **Clear** (void)

Clear all pixel values to zero.

void **Resize** (u_int *cols*, u_int *rows*)

Change the number of columns and rows of the image to *cols* respectively *rows*. The number of channels is unchanged.

void **Resize** (u_int *cols*, u_int *rows*, u_int *channels*)

Change the number of columns, rows and channels of the image to *cols*, *rows* and *channels*. All channels will have the same size.

Image& **operator=** (const Image& *im*)

Make a copy of image *im*.

Image∗ **Clone** (void) const

Make a copy of the current image.

void **SetOffsetX** (int *offx*) const

void **SetOffsetY** (int *offy*) const

Set the offset of the first channel of the image in the universal coordinate system.

Image∗ **Crop** (u_int *x1*, u_int *y1*, u_int *x2*, u_int *y2*) const

Cut the image so the upper left corner is positioned at (*x1*, *y1*), and the lower right corner at (*x2*, *y2*).

void **Merge** (const Image& *im*)

Paste image *im* into the current image. The paste position is determined by the offsets of *im*.

void **Add** (const Image& *im*)

void **Subtract** (const Image& *im*)

Add respectively subtract image *im* to (from) the current image. Both images must have the same number of columns, rows and channels and their corresponding channels must have the same structure.

Image∗ **Diff** (const Image& *im*) const

This function returns the difference image between the current image and *im*. Both images must have the same number of columns, rows and channels and their corresponding channels must have the same structure.

void **InsertChannel** (Channel& *data*, u_int *ch*)

Replace channel number *ch* of the image by the contents of channel *channel*.

void **DeleteChannel** (u_int *channel*)

Delete channel number *ch* from the image.

void **Fwt** (const TransformDescriptor *transform[]*, u_int *depth*)

Transform the image using the Fast Wavelet Transform. The two-dimensional wavelet transform will be applied to all channels independently. The type of wavelet transform is determined by the *transform* array and its length *depth*.

void **IFwt** (void)

Transform the image using the inverse Fast Wavelet Transform. The two-dimensional inverse wavelet transform will be applied to all channels independently. This is the inverse operation of *Fwt*.

void **Scale** (f32 *scale*)

Scale the image with scaling factor *scale*.

Image Types and Formats

The following image types are defined:

`IT_Unknown`	Unknown
`IT_Mono`	Monochrome (black/white)
`IT_CIEY`	Greyscale
`IT_CIEL`	CIE luminance
`IT_RGB`	RGB color
`IT_CIEXYZ`	CIE XYZ color
`IT_CIELab`	CIE L*a*b* color
`IT_YUV`	YUV
`IT_YUVr`	Reversible YUV

The following image formats are defined:

`IF_AUTO`	Automatic
`IF_PNMASCII`	Portable AnyMap ASCII
`IF_PNMRAW`	Portable AnyMap Binary
`IF_TIFF`	Tag(ged) Image File Format

See Also

The *Channel*, *Wavelet*, *Color* and *ColorSpace* classes.

Example

```
//
//      Simple image compression example
//

#ifndef NULL
#define NULL    0
#endif

#include <waili/Image.h>

int main(void)
{
    const char infile[] = "image.pgm";
    const char outfile[] = "result.pgm";
    double threshold = 20.0;
    Image image;

    // Read the image
    image.Import(infile);

    // Transform the image using the Cohen-Daubechies-Feauveau
    // (2, 2) biorthogonal wavelets
    Wavelet *wavelet = Wavelet::CreateCDF(2, 2);
    TransformDescriptor transform[] = {
{ TT_ColsRows, wavelet },
{ TT_ColsRows, wavelet },
{ TT_ColsRows, wavelet },
{ TT_ColsRows, wavelet },
{ TT_ColsRows, wavelet },
{ TT_ColsRows, wavelet },
{ TT_ColsRows, wavelet },
{ TT_ColsRows, wavelet }
    };
    image.Fwt(transform, sizeof(transform)/sizeof(*transform));

    // Zero all entries smaller than the threshold
    for (u_int ch = 0; ch < image.GetChannels(); ch++)
image[ch]->Threshold(threshold);

    // Inverse wavelet transform
    image.IFwt();

    // Write the reconstructed image to a file
    image.Export(outfile);
    return(0);
}
```

Revision

```
Image.C,v 4.4.2.4 1999/07/20 13:19:02 geert Exp
Image.h,v 4.6.2.3 1999/07/20 13:18:57 geert Exp
Example.C,v 4.0.2.1 1998/06/22 13:49:10 geert Exp
```

Name

Lifting — Generic class for integer *Lifting Scheme* steps

Description

This class provides a generic lifting step interface, to be used for wavelet transforms using the *Lifting Scheme*.

Declaration

#include ⟨waili/Lifting.h⟩

Lifting is an abstract base class. No instances can be declared. Lifting steps on different types of data are implemented through inheritance.

Virtual Operations

void **Lift_L1R1_FR** (int *primal*, const s16 *b[2]*, const u16 *a*) const

void **ILift_L1R1_FR** (int *primal*, const s16 *b[2]*, const u16 *a*) const

void **Lift_L2R2_FR** (int *primal*, const s16 *b[4]*, const u16 *a*) const

void **ILift_L2R2_FR** (int *primal*, const s16 *b[4]*, const u16 *a*) const

void **Lift_L3R3_FR** (int *primal*, const s16 *b[6]*, const u16 *a*) const

void **ILift_L3R3_FR** (int *primal*, const s16 *b[6]*, const u16 *a*) const

Primal (*primal* $= 1$) and dual (*primal* $= 0$) integer lifting steps with full rounding. *Lift_LmRn_FR* implements a lifting operation of the form

$$x_i \leftarrow x_i + \left\{ \frac{\sum_{j=-m}^{n-1} b_{j+m} y_j}{a} \right\},$$

with x_i and y_i the low pass and high pass samples (or vice versa, depending on the value of *primal*), and $\{\}$ a rounding operation. *ILift_LmRn_FR* is the corresponding inverse operation.

void **Lift_L1R1_NR** (int *primal*, const s16 *b[2]*, const u16 *a*) const

void **ILift_L1R1_NR** (int *primal*, const s16 *b[2]*, const u16 *a*) const

void **Lift_L2R2_NR** (int *primal*, const s16 *b[4]*, const u16 *a*) const

void **ILift_L2R2_NR** (int *primal*, const s16 *b[4]*, const u16 *a*) const

void **Lift_L3R3_NR** (int *primal*, const s16 *b[6]*, const u16 *a*) const

void **ILift_L3R3_NR** (int *primal*, const s16 *b[6]*, const u16 *a*) const

Primal (*primal* $= 1$) and dual (*primal* $= 0$) integer lifting steps without rounding. *Lift_LmRn_FR* implements a lifting operation of the form

$$x_i \leftarrow a x_i + \sum_{j=-m}^{n-1} b_{j+m} y_j,$$

with x_i and y_i the low pass and high pass samples (or vice versa, depending on the value of *primal*). *ILift_LmRn_NR* is the corresponding inverse operation.

void **Lift_L1R1_MX** (int *primal*, const s16 *b[2]*, const u16 *a1*, const u16 *a2*) const

void **ILift_L1R1_MX** (int *primal*, const s16 *b[2]*, const u16 *a1*, const u16 *a2*) const

void **Lift_L2R2_MX** (int *primal*, const s16 *b[4]*, const u16 *a1*, const u16 *a2*) const

void **ILift_L2R2_MX** (int *primal*, const s16 *b[4]*, const u16 *a1*, const u16 *a2*) const

void **Lift_L3R3_MX** (int *primal*, const s16 *b[6]*, const u16 *a1*, const u16 *a2*) const

void **ILift_L3R3_MX** (int *primal*, const s16 *b[6]*, const u16 *a1*, const u16 *a2*) const

Primal (*primal* $= 1$) and dual (*primal* $= 0$) integer lifting steps with mixed rounding. *Lift_LmRn_FR* implements a lifting operation of the form

$$x_i \leftarrow a_1 x_i + \left\{ \frac{\sum_{j=-m}^{n-1} b_{j+m} y_j}{a_2} \right\},$$

with x_i and y_i the low pass and high pass samples (or vice versa, depending on the value of *primal*), and $\{\}$ a rounding operation. *ILift_LmRn_MX* is the corresponding inverse operation.

Derived Classes

Lifting operations on various objects are available through classes derived from the *Lifting* class:

Name

LiftChannelR — Lifting operations on the rows of 2 *NTChannel*s

Declaration

LiftChannelR (NTChannel∗ *lowpass*, NTChannel∗ *highpass*)

Create a Lifting object for lifting operations on the rows of 2 *NTChannels*. *lowpass* contains the low pass samples, while *highpass* contains the high pass samples. Both *lowpass* and *highpass* must have the same number of rows, and the number of columns of *lowpass* and *highpass* must differ maximum 1.

Name

LiftChannelC — Lifting operations on the columns of 2 *NTChannel*s

Declaration

LiftChannelC (NTChannel∗ *lowpass*, NTChannel∗ *highpass*)

Create a Lifting object for lifting operations on the columns of 2 *NTChannel*s. *lowpass* contains the low pass samples, while *highpass* contains the high pass samples. Both *lowpass* and *highpass* must have the same number of columns, and the number of rows of *lowpass* and *highpass* must differ maximum 1.

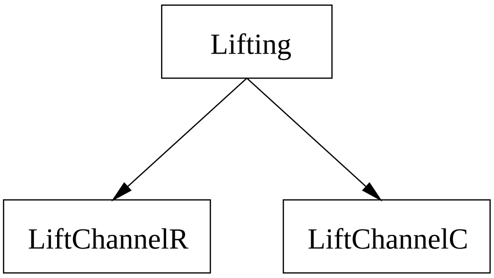

Figure 3: Inheritance dependency graph for the lifting class hierarchy (*Lifting*).

Plans Add support for transforms of a rectangular subarea of a channel. emph???

Dependency Graphs **Fig. 3** Inheritance dependency graph for the lifting class hierarchy (*Lifting*).

See Also The *Wavelet* and *Channel* classes.

Revision

```
Lifting.C,v 4.5.2.1 1999/07/15 10:18:15 geert Exp
Lifting.h,v 4.3 1997/05/05 09:46:35 geert Exp
Lifting.inline.h,v 4.0.2.1 1999/07/20 13:18:58 geert Exp
```

Name

Stream — Input/output with support for compression

Description

Flexible file storage with support for simple data compression. If a filename ends with `.gz` it will be compressed/decompressed automatically using *gzip*.

Stream ( )

Create a file handler for a stream.

Stream (const char∗ *name*, const char∗ *type* = `"r"`)

Create a file handler for a stream and open it.

Declaration

#include ⟨waili/Storage.h⟩

Public Operations

void **Read** (u8∗ *x*, u_int *cnt* = 1)

void **Read** (u16∗ *x*, u_int *cnt* = 1)

void **Read** (u32∗ *x*, u_int *cnt* = 1)

void **Read** (u64∗ *x*, u_int *cnt* = 1)

void **Read** (s8∗ *x*, u_int *cnt* = 1)

void **Read** (s16∗ *x*, u_int *cnt* = 1)

void **Read** (s32∗ *x*, u_int *cnt* = 1)

void **Read** (s64∗ *x*, u_int *cnt* = 1)

void **Read** (f32∗ *x*, u_int *cnt* = 1)

void **Read** (f64∗ *x*, u_int *cnt* = 1)

Read *cnt* elements from the stream.

void **Write** (const u8∗ *x*, u_int *cnt* = 1)

void **Write** (const u16∗ *x*, u_int *cnt* = 1)

void **Write** (const u32∗ *x*, u_int *cnt* = 1)

void **Write** (const u64∗ *x*, u_int *cnt* = 1)

void **Write** (const s8∗ *x*, u_int *cnt* = 1)

void **Write** (const s16∗ *x*, u_int *cnt* = 1)

void **Write** (const s32∗ *x*, u_int *cnt* = 1)

void **Write** (const s64∗ *x*, u_int *cnt* = 1)

void **Write** (const f32∗ *x*, u_int *cnt* = 1) void **Write** (const f64$*$ *x*, u_int *cnt* = 1)

Write *cnt* elements to the stream.

void **Read** (u8& *x*)

void **Read** (u16& *x*)

void **Read** (u32& *x*)

void **Read** (u64& *x*)

void **Read** (s8& *x*)

void **Read** (s16& *x*)

void **Read** (s32& *x*)

void **Read** (s64& *x*)

void **Read** (f32& *x*)

void **Read** (f64& *x*)

Read the element *x* from the stream.

void **Write** (const u8& *x*)

void **Write** (const u16& *x*)

void **Write** (const u32& *x*)

void **Write** (const u64& *x*)

void **Write** (const s8& *x*)

void **Write** (const s16& *x*)

void **Write** (const s32& *x*)

void **Write** (const s64& *x*)

void **Write** (const f32& *x*)

void **Write** (const f64& *x*)

Write the element *x* to the stream.

void **Puts** (const char$*$ *s*)

Write the string *s* to the stream.

void **Printf** (const char$*$ *fmt*, ...)

Format and write a string to the stream.

Virtual Operations

virtual void **Open** (const char∗ *name*, const char∗ *mode* = `"r"`)

Open the stream using filename *name* and mode *mode*.

virtual void **Close** (void)

Close the stream.

virtual void **RawRead** (void∗ *data*, int *size*)

virtual void **RawWrite** (const void∗ *data*, int *size*)

Read or write a raw block of memory from or to the stream. Note that no endianness conversion will be done!

Endiannes

All I/O operations are done using network byte order, i.e. most significant byte first or big endian.

See Also

The *gzip* command.

Revision

```
Storage.C,v 4.0.2.2 1999/07/20 13:14:18 geert Exp
Storage.h,v 4.0.2.2 1999/07/20 13:14:16 geert Exp
```

Name

Timer — Measurement of execution times

Description

This is a simple class for the measurement of execution times.

Declaration

#include ⟨waili/Timer.h⟩

Timer ( )

Create a timer.

Timer (const Timer& *t*)

Create a timer by copying timer *t*.

Public Operations

void **Start** (void)

Start the timer.

void **Stop** (void)

Stop the timer.

void **Reset** (void)

Reset the timer to zero.

f32 **GetReal** (void) const

Get the *Real* part of the run time.

f32 **GetUser** (void) const

Get the *User* part of the run time.

f32 **GetSystem** (void) const

Get the *System* part of the run time.

Timer **GetStamp** (void) const

Get a time stamp copy of the timer.

int **IsRunning** (void) const

Check whether the timer is running.

void **Tic** (void)

Reset and start the timer.

void **Toc** (void)

Dump the current *Real*, *User* and *System* run time to *stderr*.

Timer **operator**$+$ (const Timer& *t*)

Timer **operator**$-$ (const Timer& *t*)

Add or subtract two timers and return a sum or difference timer.

void **operator**$+=$ (const Timer& *t*)

void **operator**$-=$ (const Timer& *t*)

Add or subtract a timer to or from the current timer.

See Also The *times* function.

Revision

```
Timer.C,v 4.0 1997/05/05 09:42:23 geert Exp
Timer.h,v 4.0 1997/05/05 09:47:07 geert Exp
```

Name Types — Platform independent type definitions

Description This package provides some platform independent type definitions for very common types of specific sizes.

Declaration #include ⟨waili/Types.h⟩

Generic Types Available types are:

- Unsigned integer

`u8`	8 bit unsigned integer
`u16`	16 bit unsigned integer
`u32`	32 bit unsigned integer
`u64`	64 bit unsigned integer

- Signed integer

`s8`	8 bit signed integer
`s16`	16 bit signed integer
`s32`	32 bit signed integer
`s64`	64 bit signed integer

- *IEEE* Floating point

`f32`	32 bit floating point
`f64`	64 bit floating point

Pixel type All pixels are of type `PixType`:

`PixType`	16 bit signed integer

Revision `Types.h,v 4.0 1997/05/05 09:47:15 geert Exp`

Name Utility — Utility routines

Description This file contains some miscellaneous utility routines and definitions.

Declaration #include ⟨waili/Util.h⟩

Operations

void **Die** (const char$*$ *fmt*, . . .)

Exit the program with a formatted error message.

NotYetImplemented

Exit the program with a verbose ‘Not yet implemented’ message.

Type **Min** (Type *x*, Type *y*)

Calculate the minimum of two objects.

Type **Max** (Type *x*, Type *y*)

Calculate the maximum of two objects.

int **Odd** (int *x*)

Check whether a number is odd.

int **Even** (int *x*)

Check whether a number is even.

Type **Abs** (Type *x*)

Calculate the absolute value of a number.

Definitions

EPS

ε-value.

Revision

```
Util.C,v 4.0.2.2 1999/07/20 12:34:51 geert Exp
Util.h,v 4.0 1997/05/05 09:47:22 geert Exp
```

Name

Wavelet — Integer wavelet transforms using the *Lifting Scheme*

Description

The basic operational step of a wavelet transform is a filter bank with 2 kinds of filters: a low pass and a high pass filter. These 2 filters depend on the type of wavelet. In a wavelet transform the filter operations are performed iteratively on the low pass part of a signal.

The two-dimensional wavelet transform uses the same algorithm, applied to both the rows and the columns of a matrix. One can consider the wavelet transform as a 'black box' operation: a matrix is transformed into another matrix, its wavelet representation.

Here the filter operations are performed in integer math using techniques based on the *Lifting Scheme*. The sequence of Lifting steps is called a 'Cake Walk' and strongly depends on the wavelet type.

Declaration

#include ⟨waili/Wavelet.h⟩

Wavelet is an abstract base class. No instances can be declared. Different wavelet filters are implemented through inheritance.

Public Operations

int **GetGStart** ( ) const

int **GetGEnd** ( ) const

int **GetHStart** ( ) const

int **GetHEnd** ( ) const

Get the start respectively end position of the high pass ('G') respectively low pass('H') filter.

int **GetShiftL** (void) const

int **GetShiftH** (void) const

Get the number of steps (in base-$\sqrt{2}$!) the coefficients of the low pass respectively high pass subband have to be shifted to the left to obtain their real values.

u8 **GetID** (void) const

Get the unique private ID for this type of wavelet.

Virtual Operations

Wavelet∗ **Clone** (void) const

Make a copy of the current wavelet filter.

void **CakeWalk** (Lifting& *lifting*) const

Perform a 'Cake Walk' operation on the lifting object *lifting*.

void **ICakeWalk** (Lifting& *lifting*) const

Perform an inverse ‘Cake Walk’ operation on the lifting object *lifting*.

Static Operations

Wavelet$*$ **CreateCDF** (u_int *np*, u_int *nd*)

Create a *Wavelet* object for some wavelet filters of the biorthogonal Cohen-Daubechies-Feauveau family. *np* and *nd* are the numbers of vanishing moments for the primal respectively dual wavelet function. The following wavelet bases are available. Table entries are in the form (*np, nd*):

$$\begin{array}{cccccc} (1,1) & (1,3) & (1,5) & (2,2) & (2,4) & (2,6) \\ (4,2) & (4,4) & (4,6) & & & \end{array}$$

Note that $(1,1)$ is the Haar basis, and $(1,3)$ is the wavelet basis used by Ricoh’s CREW.

$(0,0)$ is used for the lazy wavelet filter.

Wavelet$*$ **CreateFromID** (u8 *id*)

Create a *Wavelet* object that corresponds to the unique private ID *id*.

Name Wavelet_Lazy — *Lazy* integer wavelet transform using the *Lifting Scheme*

Declaration

Wavelet_Lazy ( )

Create a *Wavelet* object for the lazy integer wavelet transform.

Name Wavelet_CDF_x_y — *Cohen-Daubechies-Feauveau* (x, y) integer wavelet transforms using the *Lifting Scheme*

Declaration

Wavelet_CDF_1_1 ( )

Wavelet_CDF_1_3 ( )

Wavelet_CDF_1_5 ( )

Wavelet_CDF_2_2 ( )

Wavelet_CDF_2_4 ( )

Wavelet_CDF_2_6 ( )

Wavelet_CDF_4_2 ( )

Wavelet_CDF_4_4 ( )

Wavelet_CDF_4_6 ( )

Create a *Wavelet* object for the Cohen-Daubechies-Feauveau (x, y) integer wavelet transform.

Note

Internally there also exist the classes *LiftCoefI_CDF_?_?*.

Name

Wavelet_CRF_13_7, Wavelet_SWE_13_7 — Integer wavelet transforms using the *Lifting Scheme* for some more wavelets used by JPEG2000.

Declaration

Wavelet_CRF_13_7 ( )

Wavelet_SWE_13_7 ( )

Create a *Wavelet* object for the CRF (13, 7) and SWE (13, 7) integer wavelet transforms.

Dependency Graphs

Fig. 4 Inheritance dependency graph for the Wavelet class hierarchy (*Wavelet*).

See Also

The *Lifting* and *Channel* classes.

Revision

```
Wavelet.C,v 4.1.2.3 1999/04/15 12:26:44 geert Exp
Wavelet.h,v 4.1.2.4 1999/04/15 12:26:48 geert Exp
Wavelet_CDF_1_x.C,v 4.1 1997/05/05 09:42:33 geert Exp
Wavelet_CDF_2_x.C,v 4.2.2.1 1999/03/16 15:05:38 geert Exp
Wavelet_CDF_4_x.C,v 4.2.2.1 1999/03/16 15:05:39 geert Exp
Wavelet_JPEG2000.C,v 5.1.2.1 1999/04/15 10:06:05 geert Exp
```

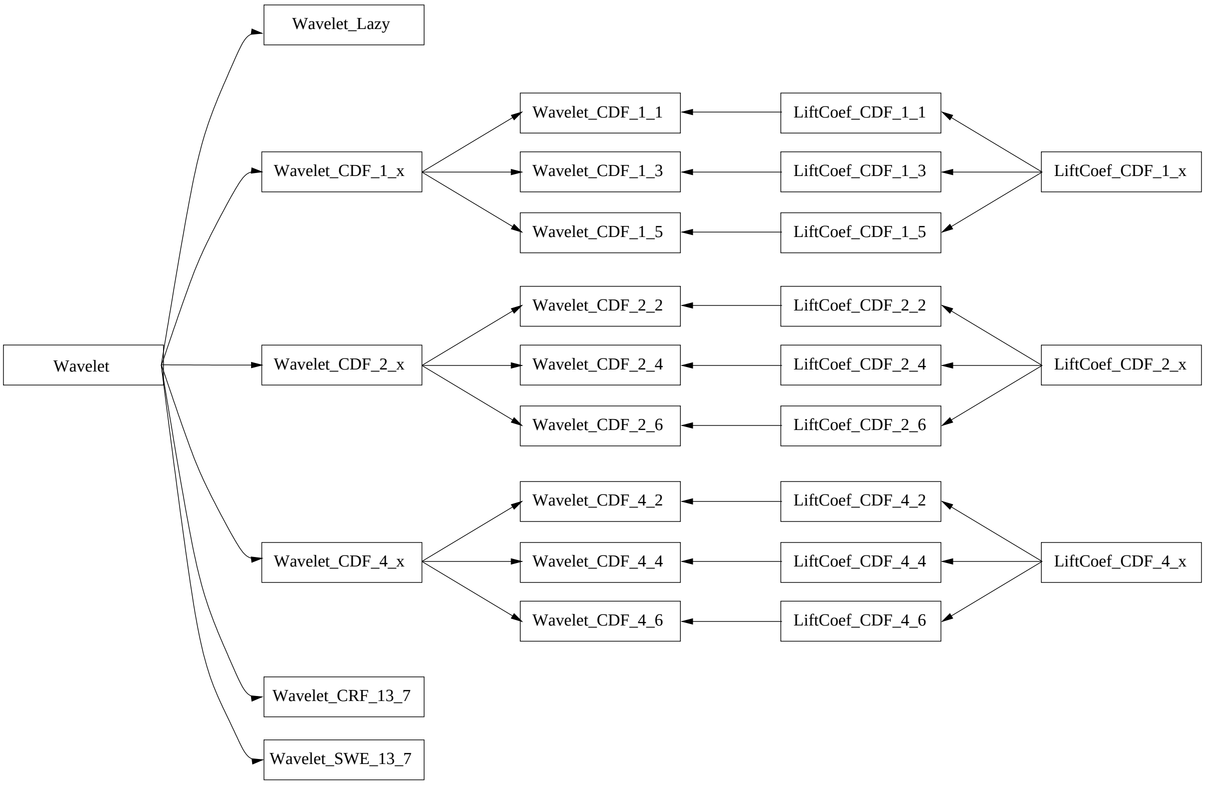

Figure 4: Inheritance dependency graph for the Wavelet class hierarchy (*Wavelet*).

5 Include dependencies

Fig. 5 Dependency graph for the various include files.

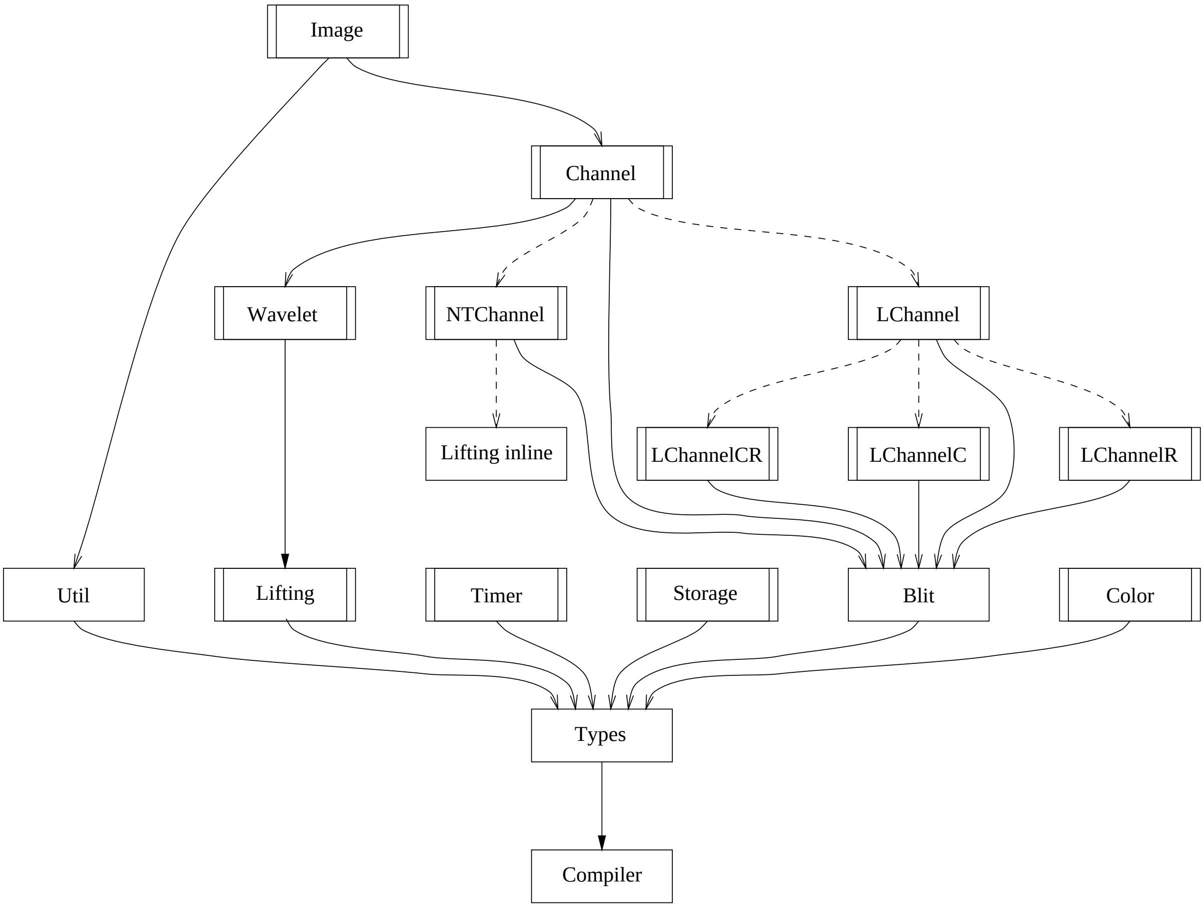

Figure 5: Dependency graph for the various include files.

6 Installation

6.1 Requirements

Before you start building the library, make sure you have the following items at hand:

- a decent UNIX system,
- *GNU Make*,
- a C++ compiler that adheres to the *ISO C++ Standard*, e.g. e.g. *GNU C++ 2.7.2*,

6.2 Building the package

- Extract the archive.
- Change the current directory to `Lifting`.
- Enter
  ```
  make
  ```
  to build the package.
- For the experts, the following make targets are available:
  - Configure the package:
    ```
    make config
    ```
  - Create the dependencies among the various source files:
    ```
    make depend
    ```
  - Clean up all object files and executables:
    ```
    make clean
    ```
  - Clean up all object files, executables and configuration files:
    ```
    make distclean
    ```

6.3 Additional notes

- By default it is assumed that the *TIFF* library is available on the Linux platform, and that it's not available on the other platforms. If you want to change this, you'll have to edit the *TIFF*∗ definitions in the file `Rules.config` (created by `make config`).
- Currently the `Makefile` assumes you're using *GNU Make* and *GNU C++ 2.7.2*.
- The development was mainly done under *Linux/ia32 2.0.x* and *2.2.x*, with some testing under *Solaris/SPARC 2.5.x* and *2.6.x*, and *Linux 2.0.x* through *2.3.x* on non-Intel platforms (*m68k*, *PPC*, *AXP*). Your mileage may vary on other systems.
- The linker might complain about undefined symbols on systems where the native linker doesn't support constructors and *collect2* is used.
- If you want support for *TIFF*, then you need the *TIFF* library.

A A simple demo program

`Lifting/test/Demo` is a simple interactive demo program that allows you to play with wavelet transforms. It understands the following commands:

`Help`

`?`

Display some help information.

`Quit`

`Exit`

Terminate the program.

`Load` ***image***

Load an image from file *image*. Make sure this file does exist!

`Save` ***image***

Save the current image to file *image*.

`View`

View the current image using *xv*. Make sure the `xv` executable is in your path!

`Wavelet` $n\ \tilde{n}$

Use the biorthogonal Cohen-Daubechies-Feauveau wavelet with $(n, \tilde{n})$ vanishing moments. Make sure you select a supported wavelet!

`Wavelet` n

Use a biorthogonal wavelet from the JPEG2000 draft. Values of n:

1 CRF (13, 7)

2 SWE (13, 7)

`Fstep cr`

`Fstep c`

`Fstep r`

Add one transform level. The transform can operate on both colums and rows (default), or on the columns or rows only.

`Bstep`

Remove one transform level.

`Ifwt`

Perform the full inverse transform.

`Noise` ***var***

Add white Gaussian noise with variance *var*.

`Denoise`

Denoise the wavelet transformed image by using soft thresholding with a GCV (Generalized Cross Validation) estimated threshold. Only subbands that count at least 1000 pixels will be thresholded.

`Backup`

Create a backup of the current image for later comparison.

`Psnr`

Calculate the PSNR (Peak Signal to Noise Ratio) of the current image, compared to the backup image.

`Threshold` ***value***

Perform hard thresholding with threshold value *value*.

`Scale` ***value***

Scale the image with factor *value*.

`Histogram` ***level subband channel***

View the histogram of subband *subband* at level *level* of the decomposition of channel *channel*.

`Entropy`

Calculate the first order entropy (Shannon-Weaver) for this channel, in bits per pixel.

`Yuv`

Convert from RGB to YUVr (or vice versa).

All commands can be abbreviated.

Revision

```
Demo.C,v 4.6.2.2 1999/04/15 10:10:14
```

B Credits

WAILI was developed by

Geert Uytterhoeven
Department of Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200A
B-3001 Heverlee
Belgium
`Geert.Uytterhoeven@cs.kuleuven.ac.be`
`http://www.cs.kuleuven.ac.be/~geert/`

Filip Van Wulpen
Department of Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200A
B-3001 Heverlee
Belgium
`Filip.VanWulpen@cs.kuleuven.ac.be`
`http://www.cs.kuleuven.ac.be/~filip/`

The denoising algorithm [6, 5] was developed by

Maarten Jansen
Department of Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200A
B-3001 Heverlee
Belgium
`Maarten.Jansen@cs.kuleuven.ac.be`
`http://www.cs.kuleuven.ac.be/~maarten/`

This library has been developed within the project ‘Wavelet Based Interactive Video Communication and Image Database Consulting’, funded by IWT — Vlaams Actieprogramma Informatietechnologie (project ITA/950244).

References

[1] C. M. Brislawn. Classification of nonexpansive symmetric extension transforms for multirate filter banks. *Appl. Comput. Harmon. Anal.*, 3:337–357, 1996.

[2] R. Calderbank, I. Daubechies, W. Sweldens, and B.-L. Yeo. Wavelet transforms that map integers to integers. Technical Report 3, 1998.

[3] A. Cohen, I. Daubechies, and J. Feauveau. Bi-orthogonal bases of compactly supported wavelets. *Comm. Pure Appl. Math.*, 45:485–560, 1992.

[4] I. Daubechies and W. Sweldens. Factoring wavelet transforms into lifting steps. Technical Report 3, 1998.

[5] M. Jansen and A. Bultheel. Multiple wavelet threshold estimation by generalised cross validation for images with correlated noise. *IEEE Transactions on Image Processing*, 1998. Accepted.

[6] M. Jansen, M. Malfait, and A. Bultheel. Generalized cross validation for wavelet thresholding. *Signal Processing*, 56(1):33–44, January 1997.

[7] W. Sweldens. The lifting scheme: A new philosophy in biorthogonal wavelet constructions. In A. F. Laine and M. Unser, editors, *Wavelet Applications in Signal and Image Processing III*, pages 68–79. Proc. SPIE 2569, 1995.

[8] W. Sweldens. The lifting scheme: A custom-design construction of biorthogonal wavelets. *Appl. Comput. Harmon. Anal.*, 3(2):186–200, 1996.

[9] W. Sweldens. The lifting scheme: A construction of second generation wavelets. *SIAM J. Math. Anal.*, 29(2):511–546, 1997.

[10] G. Uytterhoeven, D. Roose, and A. Bultheel. Wavelet transforms using the Lifting Scheme. ITA-Wavelets Report WP 1.1, Department of Computer Science, Katholieke Universiteit Leuven, Belgium, November 1996.

[11] G. Uytterhoeven, F. Van Wulpen, M. Jansen, D. Roose, and A. Bultheel. WAILI: A software library for image processing using integer wavelet transforms. In K.M. Hanson, editor, *Medical Imaging 1998: Image Processing*, volume 3338 of *SPIE Proceedings*, pages 1490–1501. The International Society for Optical Engineering, February 1998.