Contents

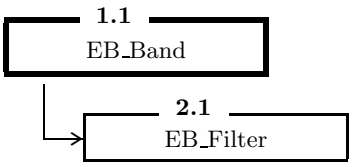
1 Grundlegendes

Names

1.1 class **EB_Band**

Diese Klasse stellt zweidimensionale Intensitätsverteilungen dar

Inheritance



Public Members

Protected Members

unsigned int	**width**	*Breite des Bildes in Pixeln.*
unsigned int	**height**	*Höhe des Bildes in Pixeln.*
float	**maxfloat**	*Obere Intensitätsgrenze.*
float	**minfloat**	*Untere Intensitätsgrenze.*
static advancementcb	**advancementcallback**	*Callback-Funktion für die Fortschrittsanzeige.*
EB_ImageRegion	**bregion**	*Region, auf die die Operationen, die das unterstützen, beschränkt bleiben.*

Diese Klasse stellt zweidimensionale Intensitätsverteilungen dar. Die Intensitäten werden als Gleitkommazahlen ausgedrückt. Die Instanzen dieser Klasse legen jeweils eine obere und untere Schranke für die Intensitäten fest. Beim Versuch, diese Grenzen zu verletzen, tritt ein Sättigungsmechanismus in Kraft. Diese Klasse kann zum Beispiel als Verkörperung der verschiedenen Bänder in Bildern genutzt werden. (Klasse EB_Image (→1.3, *page 39*) tut das zum Beispiel.) Jede der durch diese Klasse repräsentierten Verteilungen ist rechteckig mit den Kanten parallel zur x- bzw. y-Achse.Für jede Instanz läßt sich eine Region definieren, auf die die Operationen beschränkt werden sollen (bei manchen Operationen ist das nicht sinnvoll). Diese Regionen sind Instanzen der Klasse EB_ImageRegion (→1.8, *page 83*). Zur Zeit funktionieren nur Regionen, deren begrenzende Liniensegmente einander nicht überschneiden.

Author: Jürgen ËL BOSSO"Key

1.1.1

Konstruktoren und Destruktor

Names

1.1.1.1

EB_Band (void)

Default-Konstruktor

Default-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_Band (→1.1.1.1, *page 8*) der Größe 1x1.

1.1.1.2

EB_Band (unsigned int x, unsigned int y, float color = 0.0, float min = 0.0, float max = 1.0)

Konstruktor

Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_Band (→1.1.1.1, *page 8*).

Parameters:

`x` — Breite des Bildes.
`y` — Höhe des Bildes.
`color` — Intensitätswert, mit dem alle Werte dieses Bandes initialisiert werden. Liegt dieser Wert außerhalb des durch min und max bestimmten Intervalls, findet keine Initialisierung statt.
`min` — Minimaler Intensitätswert im Band.
`max` — Maximaler Intensitätswert im Band.

1.1.1.3

EB_Band (const EB_Band * band)

Konstruktor

Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_Band (→1.1.1.1, *page 8*) als Kopie von der übergebenen Instanz.

Parameters: `band` — Zeiger auf die Instanz, von der die Kopie erzeugt wird.

1.1.1.4

EB_Band (const EB_Band & band)

Copy-Konstruktor

Copy-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_Band (→1.1.1.1, *page 8*) als Kopie von der übergebenen Instanz.

Parameters: `band` — Instanz, von der die Kopie erzeugt wird.

1.1.2

Operatoren

Names

1.1.2.1

EB_Band& **operator =** (const EB_Band & Band)

Zuweisungsoperator

Zuweisungsoperator. Dieser Operator erzeugt eine Kopie der übergebenen in der aktuellen Instanz. Dies ist eine Deep Copy: Es existieren danach zwei völlig unabhängige Instanzen.

Return Value: Referenz auf die aktuelle Instanz
Parameters: `Band` — Instanz, die kopiert werden soll.

1.1.2.2

EB_Band& **operator =** (const float *content)

Zuweisungsoperator

Zuweisungsoperator. Dieser Operator überträgt mittels deep copy die Werte in einem C-Array in die aktuelle Instanz. Es muß sichergestellt sein, daß das Array minsdestens so viele Elemente enthält, wie das Band.

Return Value: Referenz auf die aktuelle Instanz
Parameters: `content` — Array mit Werten, die in das Band übertragen werden sollen.

1.1.2.3

EB_Band& **operator =** (const unsigned char *content)

Zuweisungsoperator

Zuweisungsoperator. Dieser Operator überträgt mittels deep copy die Werte in einem C-Array in die aktuelle Instanz. Es muß sichergestellt sein, daß das Array minsdestens so viele Elemente enthält, wie das Band.

Return Value: Referenz auf die aktuelle Instanz
Parameters: `content` — Array mit Werten, die in das Band übertragen werden sollen. Dabei ist es so, daß ein Wert von 0 in die untere Intensitätsgrenze umgesetzt wird, ein Wert von 255 entsprechend in die obere und alle Werte dazwischen werden entsprechend auf das Intensitätsintervall aufgeteilt.

1.1.2.4

EB_Band& **operator =** (const float content)

Zuweisungsoperator

Zuweisungsoperator. Dieser Operator überträgt den angegebenen Wert in alle Elemente des Bandes.

Return Value: Referenz auf die aktuelle Instanz
Parameters: `content` — Wert, der in alle Elemente des Bandes übertragen werden soll.

1.1.2.5

EB_Band& **operator =** (const unsigned char content)

Zuweisungsoperator

Zuweisungsoperator. Dieser Operator überträgt den angegebenen Wert in alle Elemente des Bandes.

Return Value: Referenz auf die aktuelle Instanz
Parameters: `content` — Wert, der in alle Elemente des Bandes übertragen werden soll. Dabei ist es so, daß ein Wert von 0 in die untere Intensitätsgrenze umgesetzt wird, ein Wert von 255 entsprechend in die obere und alle Werte dazwischen werden entsprechend auf das Intensitätsintervall aufgeteilt.

1.1.2.6

float& **operator**[] (unsigned int index) const

Indexoperator

Indexoperator. Diese Methode erlaubt es, auf die Intensität an einer bestimmten Stelle zuzugreifen.

Return Value: Referenz auf einen Intensitätswert.
Parameters: `index` — Laufende Nummer des Pixels. Die Pixel werden dabei von der obersten Zeile beginnend zeilenweise durchnummeriert. Es erfolgt kein Test, ob dieser Wert für die aktuelle Instanz gültig ist.

1.1.3 public Methoden

Names

1.1.3.1

unsigned int **giveWidth** (void) const

Breite

Breite. Diese Methode liefert die Breite der aktuellen Instanz.

Return Value: Breite der aktuellen Instanz.

1.1.3.2

unsigned int **giveHeight** (void) const

Höhe

Höhe. Diese Methode liefert die Höhe der aktuellen Instanz.

Return Value: Höhe der aktuellen Instanz.

1.1.3.3

float **giveMaxFloat** (void) const

Maximale Intensität

Maximale Intensität. Diese Methode liefert den maximalen Intensitätswert der aktuellen Instanz.

Return Value: Maximaler Intensitätswert der aktuellen Instanz.

1.1.3.4

float **giveMinFloat** (void) const

Minimale Intensität

Minimale Intensität. Diese Methode liefert den minimalen Intensitätswert der aktuellen Instanz.

Return Value: Minimaler Intensitätswert der aktuellen Instanz.

1.1.3.5

float **giveFloatValue** (unsigned int x, unsigned int y) const

Intensitätsbestimmung

Intensitätsbestimmung. Diese Methode erlaubt es, die Intensität an einer bestimmten Stelle zu ermitteln. Bei Koordinaten, für die keine Werte vorliegen wird 0.0 zurückgegeben.

Return Value: Intensitätswert.
Parameters: `x` — x-Koordinate des Pixels, dessen Intensitätswert ermittelt werden soll.
`y` — y-Koordinate des Pixels, dessen Intensitätswert ermittelt werden soll.

1.1.3.6

float **giveFloatValue** (unsigned int index) const

Intensitätsvektor ermitteln

Intensitätsvektor ermitteln. Diese Methode erlaubt es, die Intensität an einer bestimmten Stelle zu ermitteln. Bei Koordinaten, für die keine Werte vorliegen wird 0.0 zurückgegeben.

Return Value: Intensitätswert.
Parameters: `index` — Laufende Nummer des Pixels. Die Pixel werden dabei von der obersten Zeile beginnend zeilenweise durchnummeriert.

1.1.3.7

unsigned char **giveCharValue** (unsigned int x, unsigned int y) const

Intensitätsbestimmung

Intensitätsbestimmung. Diese Methode erlaubt es, die Intensität an einer bestimmten Stelle zu ermitteln. Der Intensitätsbereich wird dabei auf den Wertebereich des Typs unsigned char abgebildet. Bei Koordinaten, für die keine Werte vorliegen wird 0.0 zurückgegeben.

Return Value: Intensitätswert.
Parameters: `x` — x-Koordinate des Pixels, dessen Intensitätswert ermittelt werden soll.
`y` — y-Koordinate des Pixels, dessen Intensitätswert ermittelt werden soll.

1.1.3.8

unsigned char **giveCharValue** (unsigned int index) const

Intensitätsbestimmung

Intensitätsbestimmung. Diese Methode erlaubt es, die Intensität an einer bestimmten Stelle zu ermitteln. Der Intensitätsbereich wird dabei auf den Wertebereich des Typs unsigned char abgebildet. Bei Koordinaten, für die keine Werte vorliegen wird 0 zurückgegeben.

Return Value: Intensitätswert.
Parameters: `x` — x-Koordinate des Pixels, dessen Intensitätswert ermittelt werden soll.
`y` — y-Koordinate des Pixels, dessen Intensitätswert ermittelt werden soll.

1.1.3.9

void **setValue** (unsigned int x, unsigned int y, float value)

Intensitätsänderung

Intensitätsänderung. Diese Methode erlaubt es, die Intensität an einer bestimmten Stelle zu ändern. Liegt der neue Wert außerhalb des vereinbarten Intensitätsintervalls, wird an der entsprechenden Stelle der nächstliegende Grenzwert eingesetzt. Bei Koordinaten, für die keine Werte vorliegen wird die Änderung ignoriert.

Parameters:
`x` — x-Koordinate des Pixels, dessen Intensitätswert ermittelt werden soll.
`y` — y-Koordinate des Pixels, dessen Intensitätswert ermittelt werden soll.
`color` — Intensitätswert.

1.1.3.10

void **setValue** (unsigned int index, float value)

Intensitätsänderung

Intensitätsänderung. Diese Methode erlaubt es, die Intensität an einer bestimmten Stelle zu ändern. Liegt der neue Wert außerhalb des vereinbarten Intensitätsintervalls, wird an der entsprechenden Stelle der nächstliegende Grenzwert eingesetzt. Bei Koordinaten, für die keine Werte vorliegen wird die Änderung ignoriert.

Parameters:
`pindex` — Laufende Nummer des Pixels. Die Pixel werden dabei von der obersten Zeile beginnend zeilenweise durchnummeriert.
`color` — Intensitätswert.

1.1.3.11

void **setValue** (unsigned int x, unsigned int y, unsigned char value)

Intensitätsänderung

Intensitätsänderung. Diese Methode erlaubt es, die Intensität an einer bestimmten Stelle zu ändern. Dabei entspricht ein Wert von 0 dem unteren, ein Wert von 255 dem oberen Ende des für dieses Band festgelegten Intensitätsintervalls. Bei Koordinaten, für die keine Werte vorliegen wird die Änderung ignoriert.

Parameters:
`x` — x-Koordinate des Pixels, dessen Intensitätswert ermittelt werden soll.
`y` — y-Koordinate des Pixels, dessen Intensitätswert ermittelt werden soll.
`color` — Intensitätswert.

1.1.3.12

void **setValue** (unsigned int index, unsigned char value)

Intensitätsänderung

Intensitätsänderung. Diese Methode erlaubt es, die Intensität an einer bestimmten Stelle zu ändern. Dabei entspricht ein Wert von 0 dem unteren, ein Wert von 255 dem oberen Ende des für dieses Band festgelegten Intensitätsintervalls. Bei Koordinaten, für die keine Werte vorliegen wird die Änderung ignoriert.

Parameters:
`pindex` — Laufende Nummer des Pixels. Die Pixel werden dabei von der obersten Zeile beginnend zeilenweise durchnummeriert.
`color` — Intensitätswert.

1.1.3.13

EB_Band& **copyWholeTo** (EB_Band & other, int left, int top)

Komponieren von Bändern

Komponieren von Bändern. Diese Methode überträgt den Inhalt der aktuellen Instanz in eine andere. Dabei kann man noch einen Offset angeben, der die Position im Ziel angibt. Dieser Offset bezieht sich auf die linke obere Ecke. Sind im aktuellen Band keine Werte vorhanden wird eine Exception geworfen. Die aktuelle Instanz wird hierbei nicht verändert

Return Value:
Referenz auf other.
Parameters:
`other` — Instanz der Klasse EB_Band (→1.1.1.1, *page 8*) und Ziel der Kopie.
`left` — Offset vom linken Rand für die Kopie.
`top` — Offset vom oberen Rand für die Kopie.

1.1.3.14

EB_Band& **copyWholeFrom** (EB_Band & other, int left, int top)

Komponieren von Bändern

Komponieren von Bändern. Diese Methode überträgt den Inhalt einer zweiten Instanz in die aktuelle Instanz. Dabei kann man noch einen Offset angeben, der die Position im Ziel angibt. Dieser Offset bezieht sich auf die linke obere Ecke. Enthält ein Band keine Daten, wird eine Exception geworfen. Die aktuelle Instanz wird verändert.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `other` — stanz der Klasse EB_Band (→1.1.1.1, *page 8*) und Quelle der Kopie.
`left` — Offset vom linken Rand für die Kopie.
`top` — Offset vom oberen Rand für die Kopie.

1.1.3.15

EB_Band& **rescaleIntensity** (float newmin, float newmax)

Intensitätsintervall ändern

Intensitätsintervall ändern. Mit dieser Methode werden die Grenzen des Intensitätsintervalls geändert. Gleichzeitig werden alle Intensitätswerte entsprechend dem neuen Intervall skaliert. Wird das Intervall von 0 bis 10 auf 0 bis 1 geändert, wird ein Intensitätswert von beispielsweise vorher 5 auf 0.5 geändert. Die aktuelle Instanz wird verändert.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `newmin` — Minimaler Intensitätswert in jedem Band.
`newmax` — Maximaler Intensitätswert in jedem Band.

1.1.3.16

EB_Band& **aequalize** (void)

Histogrammausgleich

Histogrammausgleich. Diese Methode führt einen Histogrammausgleich auf dem Band aus. Die aktuelle Instanz wird verändert. Diese Methode berücksichtigt die eventuell definierte Region und behandelt nur Werte innerhalb einer solchen Region. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.

1.1.3.17

EB_Band& **gammaCorrect** (float factor)

Gammakorrektur

Gammakorrektur. Diese Methode führt eine Gammakorrektur auf dem Band aus. Dies ist eine über der Intensität nichtlineare Helligkeitsänderung. Dier aktuelle Instanz wird verändert. Diese Methode berücksichtigt die eventuell definierte Region und behandelt nur Werte innerhalb einer solchen Region. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `factor` — Parameter für die Gammakorrektur. Ist dieser Wert kleiner als eins, wird die durchschnittliche Intensität geringer, ansonsten höher.

1.1.3.18

EB_Band& **changeContrast** (float factor)

Kontraständerung

Kontraständerung. Diese Methode führt eine Kontraständerung auf dem Band aus. Ddie aktuelle Instanz wird verändert. Diese Methode berücksichtigt die eventuell definierte Region und behandelt nur Bildinhalte innerhalb einer solchen Region. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `factor` — Parameter für die Kontraständerung. Je höher dieser Wert ist, desto mehr werden die Intensitätswerte in Richtung der Intensitätsintervallgrenzen verschoben.

1.1.3.19

EB_Band& **transformWithLUT** (const EB_IntensityTransformation &trans)

Intensitätsänderung

Intensitätsänderung. Diese Methode führt eine Intensitätsänderung entsprechend der nichtlinearen Funktion in trans aus. Dabei dienen die Intensitäten als Argumente für diese Funktion und die Ergebnisse der Funktion als neue Intensitäten. Der Bereich zwischen der oberen und unteren Intensitätsgrenze wird bei Argumenten wie Ergebnissen auf das Intervall (0,1) gemappt. Der Intensitätsbereich wird in 256 Stufen diskretisiert um die Operation mittels einer Look-Up-Tabelle beschleunigen zu können. Ddie aktuelle Instanz wird verändert. Diese Methode berücksichtigt die eventuell definierte Region und behandelt nur Bildinhalte innerhalb einer solchen Region. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `trans` — Instanz einer von EB_IntensityTransformation (→5.1, *page 146*) abgeleiteten Klasse, die die eigentliche Funktion repräsentiert.

1.1.3.20

EB_Band& **transform** (const EB_IntensityTransformation &trans)

Intensitätsänderung

Intensitätsänderung. Diese Methode führt eine Intensitätsänderung entsprechend der nichtlinearen Funktion in trans aus. Dabei dienen die Intensitäten als Argumente für diese Funktion und die Ergebnisse der Funktion als neue Intensitäten. Der Bereich zwischen der oberen und unteren Intensitätsgrenze wird bei Argumenten wie Ergebnissen auf das Intervall (0,1) gemappt. Ddie aktuelle Instanz wird verändert. Diese Methode berücksichtigt die eventuell definierte Region und behandelt nur Bildinhalte innerhalb einer solchen Region. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `trans` — Instanz einer von EB_IntensityTransformation (→5.1, *page 146*) abgeleiteten Klasse, die die eigentliche Funktion repräsentiert.

1.1.3.21

EB_Band& **transform** (const EB_ImageTransformation &trans)

Transformation

Transformation. Das Band wird entsprechend der Vorschrift im Transformationsobjekt transformiert. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `trans` — Instanz der Klasse EB_ImageTransformation (→3.1, *page 107*) oder von ihr abgeleiteter Klassen, die die Transformation beschreibt.

1.1.3.22

EB_Band& **transform** (const EB_ImageTransformation &trans, EB_TransformationInterpolator &ip)

Transformation

Transformation. Das Band wird entsprechend der Vorschrift im Transformationsobjekt transformiert.Dabei wird das Bild entsprechend der Vorschrift im Interpolationsobjekt interpoliert. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `trans` — Instanz der Klasse EB_ImageTransformation (→3.1, *page 107*) oder von ihr abgeleiteter Klassen, die die Transformation beschreibt.
`ip` — Instanz der Klasse EB_TransformationInterpolator (→4.1, *page 134*) oder von ihr abgeleiteter Klassen, die die Interpolation beschreibt.

1.1.3.23

EB_Band& **transformWithLUT** (const EB_ImageTransformation &trans)

Transformation

Transformation. Das Band wird entsprechend der Vorschrift im Transformationsobjekt transformiert. Es wird eine Look-Up-Tabelle im Transformationobjekt erzeugt, was wiederholte Anwendungen dieses Objektes beschleunigt. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `trans` — Instanz der Klasse EB_ImageTransformation (→3.1, *page 107*) oder von ihr abgeleiteter Klassen, die die Transformation beschreibt.

1.1.3.24

EB_Band& **smoothBox** (unsigned int smoothwidth)

Glättung

Glättung. Das Band wird mit einem quadratischen Boxfilter mit beliebiger Kantenlänge geglättet. Jeder Pixel des Bandes wird mit dem Filter beaufschlagt. Die aktuelle Instanz wird verändert. Diese Methode berücksichtigt nicht die eventuell definierte Region. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `smoothwidth` — Kantenlänge des quadratischen Filters.

1.1.3.25

EB_Band& **smoothBinom** (unsigned int smoothwidth)

Glättung

Glättung. Das Band wird mit einem quadratischen Binomialfilterfilter mit beliebiger Kantenlänge geglättet. Jeder Pixel des Bandes wird mit dem Filter beaufschlagt. Die aktuelle Instanz wird verändert. Diese Methode berücksichtigt nicht die eventuell definierte Region. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `smoothwidth` — Kantenlänge des quadratischen Filters.

1.1.3.26

EB_Band& **convolute** (const EB_Filter & filter)

Filterung

Filterung. Das Bild wird mit einem von einer Instanz der Klasse EB_Filter (→2.1, *page 98*) dargestellten Filter gefiltert. Jeder Pixel des Bandes wird mit dem Filter beaufschlagt. Die aktuelle Instanz wird verändert. Diese Methode berücksichtigt die eventuell definierte Region und behandelt nur Bildinhalte innerhalb einer solchen Region. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `filter` — Instanz der Klasse EB_Filter (→2.1, *page 98*).

1.1.3.27

EB_Band& **convolute** (const EB_Filter & filter, EB_Band & alphachannel, int left = 0, int top = 0, float factor = 0.0) throw(EBIOutOfMemoryEXP)

Filterung

Filterung. Das Band wird mit einem von einer Instanz der Klasse EB_Filter (→2.1, *page 98*) dargestellten Filter gefiltert. Jeder Pixel des Bandes wird entsprechend der Intensität in alphachannel an der entsprechenden Stelle mit dem Filter beaufschlagt. Die aktuelle Instanz wird verändert. Diese Methode berücksichtigt die eventuell definierte Region und behandelt nur Bildinhalte innerhalb einer solchen Region. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.

Parameters:
`filter` — Instanz der Klasse EB_Filter (→2.1, *page 98*).
`alphaimage` — Instanz der Klasse EB_Band (→1.1.1.1, *page 8*), deren Intensitätswerte angeben, wie sehr das Ergebnis der Operation für jeden einzelnen Pixel vom gefilterten und vom Originalwert abhängt. Enthält diese Instanz keine Bänder, wird eine Exception geworfen.
`left` — Offset der linken Ecke des alphachannel.
`top` — Offset der oberen Ecke des alphachannel.
`factor` — Wenn dieser Wert größer als 1.0 oder kleiner als -1.0 ist, wird ihm der nächstliegende der beiden genannten Werte zugewiesen. Ist der Wert danach größer als 0, dient er als ganz normale Schwelle: für jeden Pixel wird bestimmt, wie sich der Intensitätswert in alphachannel zur Länge des Intensitätsintervalls verhält. Ist dieses Verhältnis größer als factor, wird der gefilterte Wert in das Ergebnis eingesetzt, ansonsten der Originalwert. Ist factor kleiner als 0, wird der Betrag als Schwelle benutzt. Im Falle der Benutzung des gefilterten Wertes wird dieser nochmals linear mit dem Original verrechnet: Mit o als Originalwert, f als gefiltertem Wert, i als Intensität in alphachannel an der entsprechenden Stelle und r als Resultat ergibt sich dann:

$$r = f * i + (1 - i) * o$$

1.1.3.28

EB_Band& **morphologicOperation** (unsigned int kernelwidth, float gamma)

Morphologische Operation

Morphologische Operation. Diese Methode stellt morphologische Operationen mit quadratischen Masken beliebiger Größe dar. Über einen Parameter kann stufenlos eine Betriebsart zwischen den beiden Extrema Erosion und Dilatation gewählt werden. Die aktuelle Instanz wird verändert. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `kernelwidth` — Kantenlänge der quadratischen Maske.
`gamma` — Parameter zur Beeinflussung des Verhaltens der Operation. Werte kleiner als 0.0 werden als 0.0 interpretiert, solche größer als 1.0 als 1.0. Dabei entspricht 0.0 der Dilatation und 1.0 der Erosion.

1.1.3.29

EB_Band& **rotate** (float angle, int rotcenterx, int rotcentery, float fillintensity=-10000.0)

Rotation

Rotation. Diese Methode rotiert das Band um den angegebenen Betrag. Die Drehung erfolgt um den angegebenen Punkt, der auch außerhalb liegen darf. Die aktuelle Instanz wird verändert.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `angle` — Dieser Wert gibt den Winkel an, um den das Band gedreht werden soll. Dabei bedeutet ein positiver Wert mathematisch positiven Drehsinn (entgegen der Uhrzeigerrichtung). Die Einheit der Winkelangabe ist Grad. Diese Methode liefert Fortschrittsberichte über advancementcallback.
`rotcenterx` — x-Koordinate des Punktes, um den das Bild rotiert wird.
`rotcentery` — y-Koordinate des Punktes,um den das Bild rotiert wird.
`fillintensity` — Bereiche im Band, die nach der Rotation undefinierten Inhalt hätten, werden mit dieser Intensität aufgefüllt.

1.1.3.30

EB_Band& **rotateToFit** (float angle, float fillintensity=-10000.0)

Rotation

Rotation. Diese Methode rotiert das Band um den angegebenen Betrag. Die Drehung erfolgt um den Bildmittelpunkt. Die Dimensionen des Resultats werden so gewählt, daß der gesamte Inhalt des Originals hineinpaßt. Die aktuelle Instanz wird verändert.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `angle` — Dieser Wert gibt den Winkel an, um den das Band gedreht werden soll. Dabei bedeutet ein positiver Wert mathematisch positiven Drehsinn (entgegen der Uhrzeigerrichtung). Die Einheit der Winkelangabe ist Grad. Diese Methode liefert Fortschrittsberichte über advancementcallback.
`fillintensity` — Bereiche im Band, die nach der Rotation undefinierten Inhalt hätten, werden mit dieser Intensität aufgefüllt.

1.1.3.31

EB_Band& **scale** (unsigned int newwidth, unsigned int newheight)

Skalierung

Skalierung. Diese Methode gestattet es, ein Band zu skalieren. Es ist möglich, die neue Breite und Höhe unabhängig voneinander anzugeben. Die aktuelle Instanz wird verändert. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `newx` — Neue Breite des Bandes.
`newy` — Neue Höhe des Bandes.

1.1.3.32

EB_Band& **overlay** (EB_Band & other, float otherfac, float ownfac, int left = 0, int top = 0)

Überblenden

Überblenden. Mit dieser Methode können zwei Bänder mit festen Faktoren überblendet werden. Dabei werden die Intensitäten des Bandes für jedes Pixel addiert. Faktoren bestimmen, mit welchem Anteil die aktuelle und die übergebene Instanz zu dieser Addition beitragen. Ist die Summe der Faktoren größer als 0, werden beide entsprechend zurückskaliert. Die aktuelle Instanz wird verändert. Diese Methode berücksichtigt die eventuell definierte Region und behandelt nur Bildinhalte innerhalb einer solchen Region. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `other` — Band, das der aktuellen Instanz überlagert werden soll.
`otherfac` — Faktor für other.
`ownfac` — Faktor für die aktuelle Instanz.
`left` — Offset von other vom linken Rand.
`top` — Offset von other vom oberen Rand.

1.1.3.33

EB_Band& **overlay** (EB_Band & other, EB_Band & alphachannel, int oleft = 0, int otop = 0, int aleft = 0, int atop = 0, float factor = 1.0f)

Überblenden

Überblenden. Mit dieser Methode können zwei Bänder mit festen Faktoren überblendet werden. Dabei werden die Intensitäten des Bandes für jedes Pixel addiert. Wie stark der Einfluß der aktuellen Instanz auf das Ergebnis ist, wird durch die Intensität an der entsprechenden Stelle im sogenannten alphachannel bestimmt. Je höher die Intensität in diesem Band ist, desto stärker bestimmt der Inhalt von other das Resultat. Die aktuelle Instanz wird verändert. Diese Methode berücksichtigt die eventuell definierte Region und behandelt nur Bildinhalte innerhalb einer solchen Region. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `other` — Band, das der aktuellen Instanz überlagert werden soll.
`alphachannel` — Band, dessen Intensitätswerte bestimmen, zu welchem Anteil die aktuelle
`left` — Offset von other vom linken Rand.
`top` — Offset von other vom oberen Rand.
`left` — Offset von alphachannel vom linken Rand.
`top` — Offset von alphachannel vom oberen Rand.
`factor` — Mit diesem Faktor ist eine Intensitätsbeeinflussung des Ergebnisses möglich. Mit diesem Wert wird das Ergebnis der beschriebenen Addition multipliziert. Bei einem Wert von 0.5 ergibt sich also ein Ergebnis, dessen Intensitätswerte halb so hoch sind, wie sie ohne diese Multiplikation wären.

1.1.3.34

EB_Band& **doubleMirror** (void)

Spiegeln

Spiegeln. Das Band wird an x- und y-Achse gespiegelt. Die aktuelle Instanz wird verändert. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.

1.1.3.35

EB_Band& **verticalMirror** (void)

Spiegeln

Spiegeln. Das Band wird an der x-Achse gespiegelt. Die aktuelle Instanz wird verändert. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.

1.1.3.36

EB_Band& **horizontalMirror** (void)

Spiegeln

Spiegeln. Das Band wird der y-Achse gespiegelt. Die aktuelle Instanz wird verändert. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.

1.1.3.37

EB_Band& **setRegion** (EB_ImageRegion & region)

Region definieren

Region definieren. Mit dieser Methode wird der aktuellen eine Instanz der Klasse EB_ImageRegion (→1.8, *page 83*) zugeordnet. Diese Klasse stellt zweidimensionale Regionen dar. Zur Zeit können nur Regionen sinnvoll benutzt werden. Werden Operationen durch diese Regionen beeinflußt, ist das in der Dokumentation angegeben.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `region` — Instanz der Klasse EB_ImageRegion (→1.8, *page 83*), die der aktuellen Instanz zugeordnet werden soll.

1.1.3.38

EB_Band& **unsetRegion** (void)

Region löschen

Region löschen. Mit dieser Funktion wird die Zuordnung einer Region zum aktuellen Band aufgehoben. Nach Aufruf dieser Methode beeinflussen alle Methoden wieder das ganze Band.

Return Value: Referenz auf die aktuelle Instanz.

1.1.3.39

EB_Band& **doMedianFilter** (unsigned int fux, unsigned int fuy)

Medianfilterung

Medianfilterung. Das Band wird mit einem rechteckigen Medianfilter mit beliebiger Kantenlänge behandelt. Jeder Pixel des Bandes wird mit dem Filter beaufschlagt. Die aktuelle Instanz wird verändert. Diese Methode berücksichtigt die eventuell definierte Region. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `fux` — Kantenlänge des Filters in x-Richtung.
`fuy` — Kantenlänge des Filters in y-Richtung.

1.1.3.40

ostream& **writeToStream** (ostream &o, bool pack)

Schreiben in Stream

Schreiben in Stream. Diese Methode schreibt den Inhalt des Bandes in den angegebenen Stream. Dabei ist das Format wie folgt:zunächst wird der Wert des Parameters pack in den Stream geschrieben. Danach folgen die Intensitätswerte abhängig vom Wert des Parameters pack gepackt oder ungepackt. Ungepackt bedeutet hierbei jeder Intensitätswerte wird durch ein KLeerzeichen getrennt als Gleitkommazahl ausgegeben. Gepacktes Schreiben bedeutet, daß jeweils vier Intensitätswerte zu 8-Bit-Werten diskretisiert und in einen int-Wert gepreßt werden. Dieser int-Wert wird anschließend in den Stream geschrieben. Mit dieser Methode kann der Speicherufwand ungefähr um den Faktor 4 reduziert werden.

Return Value: Referenz auf den übergebenen Stream.
Parameters: `o` — Stream, in den geschrieben werden soll.
`pack` — true, wenn die Daten gepackt geschrieben werden sollen, false für ungepacktes Schreiben.

1.1.3.41

istream& **readFromStream** (istream &i) throw(EBICouldNotLoadEXP)

Lesen aus einem Stream

Lesen aus einem Stream. Mit dieser Methode kann der Inhalt eines Bandes aus einem Stream gelesen werden. Zu Beginn wird ein Wert gelesen, der bestimmt, ob die Daten gepackt oder ungepackt vorliegen. Anschließend wird eine der Breite und Höhe des Bandes entsprechende Anzahl Daten gelesenen. Tritt während des Lesens ein Fehler auf, wird eine Exception geworfen.

Return Value: Referenz auf den übergebenen Stream.
Parameters: `i` — Stream, von dem gelesen werden soll.

1.2

class **EB_ColorDistHistogram**

Diese Klasse stellt Intensitätshistogramme dar

Public Members

Protected Members

unsigned int	**samplecount**	*Anzahl der für die Erstellung des Histogramms benutzten Samples.*
unsigned int	**spointcount**	*Anzahl der Stützstellen im Histogramm.*
float*	**histogram**	*Feld zur Aufnahme der absoluten Häufigkeiten.*
float*	**scaledhistogram**	*Feld zur Aufnahme der relativen Häufigkeiten.*

Diese Klasse stellt Intensitätshistogramme dar. Intensitätshistogramme deshalb, weil hier nur die Häufigkeiten einzelner Float-Werte summiert werden und nicht die von Vektoren von Float-Werten, wie das zum Beispiel bei Farben, die ja durch eine Menge von Float-Werten dargestellt werden, wäre. Es ist möglich, die Anzahl der einzelnen Stützstellen des Histogramms frei zu bestimmen, es ist nicht möglich, mit dieser Klasse nichtäquidistante Histogramme zu modellieren.

Author: Jürgen ËL BOSSO"Key

1.2.1

Konstruktoren und Destruktor

Names

1.2.1.1

EB_ColorDistHistogram (unsigned int count)

Konstruktor

Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_ColorDistHistogram (→1.2.1.1, *page 31*) mit der angegebenen Anzahl an Stützstellen.

Parameters: `count` — Gewünschte Anzahl an Stützstellen.

1.2.1.2

EB_ColorDistHistogram (const EB_ColorDistHistogram & other)

Copy-Konstruktor

Copy-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_ColorDistHistogram (→1.2.1.1, *page 31*) als Kopie der übergebenen Instanz.

Parameters: `other` — Instanz, die zum Erstellen der Kopie benutzt wird.

1.2.2

Operatoren

Names

1.2.2.1

EB_ColorDistHistogram& **operator =** (const EB_ColorDistHistogram & other)

Zuweisungsoperator

Zuweisungsoperator. Dieser Operator erzeugt eine Kopie der übergebenen in der aktuellen Instanz. Dies ist eine Deep Copy: Es existieren danach zwei völlig unabhängige Instanzen.

Return Value: Referenz auf die aktuelle Instanz
Parameters: `other` — Instanz, die kopiert werden soll.

1.2.2.2

float& **operator[]** (unsigned int index) const

Indexoperator

Indexoperator. Diese Methode erlaubt es, auf den Wert des Histogramms an einer bestimmten Stelle zuzugreifen.

Return Value: Referenz auf einen Histogrammwert.
Parameters: `index` — Laufende Nummer.

1.2.3 public Methoden

Names

1.2.3.1

unsigned int **giveSPointCount** (void) const

Ermittlung der Stützstellenanzahl

Ermittlung der Stützstellenanzahl. Mit dieser Methode kann man die aktuelle Stützstellenanzahl der aktuellen Instanz abfragen.

Return Value: Die aktuelle Stützstellenanzahl.

1.2.3.2

unsigned int **giveSampleCount** (void) const

Ermittlung der Sampleanzahl

Ermittlung der Sampleanzahl. Mit dieser Methode kann man die Anzahl der bisher zur Berechnung des Histogramms benutzten Samples bestimmen.

Return Value: Die Zahl der bisher zur Berechnung des Histogramms benutzten Samples.

1.2.3.3

float* **giveAbsHistogramPointer** (void)

Häufigkeiten ermitteln

Häufigkeiten ermitteln. Mit dieser Methode lassen sich die absoluten Häufigkeiten der Samples über den Intervallen des Histogramms ermitteln. Jede Stützstelle des Histogramms entspricht einem Intensitätsintervall, daher ergibt die absolute Häufigkeit für eine bestimmte Stützstelle die Anzahl an Samples, deren Intensität in dem dieser Stützstelle zugeordneten Intervall liegt.

Return Value: Float-Array mit ebensovielen Elementen, wie das Histogramm in der aktuellen Instanz Stützstellen hat.

1.2.3.4

float* **giveRelHistogramPointer** (void)

Häufigkeiten ermitteln

Häufigkeiten ermitteln. Diese Methode entspricht im Prinzip giveAbsHistogramPointer (→1.2.3.3, *page 34*), mit dem Unterschied, daß nun relative Häufigkeiten zurückgegeben werden. Das heißt, daß die Häufigkeiten an jeder Stützstelle durch die Gesamtanzahl aller zur Berechnung des Histogramms benutzten Samples dividiert werden.

Return Value: Float-Array mit ebensovielen Elementen, wie das Histogramm in der aktuellen Instanz Stützstellen hat.

1.2.3.5

void **eraseHistogram** (void)

Histogramm löschen

Histogramm löschen. Mit dieser Methode werden alle Informationen in der Klasse wieder zurückgesetzt, so daß es so erscheint, als ob das Histogramm völlig leer ist.

1.2.3.6

void **eraseHistogram** (unsigned int newspointcount)

Histogramm löschen

Histogramm löschen. Mit dieser Methode werden alle Informationen in der Klasse wieder zurückgesetzt, so daß es so erscheint, als ob das Histogramm völlig leer ist. Zusätzlich werden die Stützstellen- und Intervallanzahl für das Histogramm neu spezifiziert.

Parameters: `newspointcount` — Gibt die gewünschte Anzahl an Stützstellen an.

1.2.3.7

float **giveAbsValue** (unsigned int index) const

Absolute Häufigkeit ermitteln

Absolute Häufigkeit ermitteln. Diese Methode gestattet es, für eine bestimmte Stützstelle des Histogrammes die Anzahl Samples zu ermitteln, die in das zugehörige Intervall fielen. Dies ist praktisch die absolute Häufigkeit.

Return Value: Anzahl Samples, die auf das zugehörige Intervall entfielen.
Parameters: `index` — Nummer der Stützstelle, deren absolute Häufigkeit ermittelt werden soll.

1.2.3.8

float **giveRelValue** (unsigned int index) const

Relative Häufigkeit ermitteln

Relative Häufigkeit ermitteln. Diese Methode gestattet es, für eine bestimmte Stützstelle des Histogrammes relativ zur Gesamtanzahl Samples die Anzahl der Samples zu ermitteln, die in das zugehörige Intervall fielen. Dies ist praktisch die relative Häufigkeit.

Return Value: Verhältnis zwischen der Anzahl Samples, die auf das zugehörige Intervall entfielen und der Gesamtanzahl.
Parameters: `index` — Nummer der Stützstelle, deren relative Häufigkeit ermittelt werden soll.

1.2.3.9

void **calculateHistogram** (EB_Band & band)

Histogramm berechnen

Histogramm berechnen. Diese Methode ermittelt entsprechend der eingestellten Stützstellenanzahl ein Histogramm über alle Intensitätswerte in der übergebenen Instanz der Klasse EB_Band (→1.1, *page 7*). Die Breite und Grenzen der einzelnen zu den Stützstellen gehörenden Intervalle werden durch die Grenzen des Intensitätsbereiches des Bandes definiert.

Parameters: `band` — Instanz der Klasse EB_Band (→1.1, *page 7*), über deren Intensitätswerte das Histogramm berechnet werden soll.

1.2.3.10

void **smoothBox** (unsigned int width)

Mittelwertfilterung

Mittelwertfilterung. Diese Methode erlaubt es, das Histogramm mit einem Boxfilter einstellbarer Breite zu filtern.

Parameters: `width` — Breite des angewendeten Boxfilters.

1.2.3.11

void **smoothBoxClosed** (unsigned int width)

Mittelwertfilterung

Mittelwertfilterung. Diese Methode erlaubt es, das Histogramm mit einem Boxfilter einstellbarer Breite zu filtern. Dabie wird das Histogramm als Ring aufgefasst, d.h. Zur Filterung der ersten Elemente werden die letzten mit einbezogen und umgekehrt.

Parameters: `width` — Breite des angewendeten Boxfilters.

1.2.3.12

float **giveMax** (void)

Maximumsuche

Maximumsuche. Der maximale Wert im Histogramm wird gesucht und zurückgegeben.

Return Value: Der maximale Absolutwert des Histogramms.

1.2.3.13

unsigned int **giveIndexOfMax** (void)

Maximumsuche

Maximumsuche. Der Index des maximalen Wertes im Histogramm wird gesucht und zurückgegeben.

Return Value: Der Index (die Stützstelle) des maximalen Absolutwertes des Histogramms.

1.2.3.14

float **giveMin** (void)

Minimumsuche

Minimumsuche. Der minimale Wert im Histogramm wird gesucht und zurückgegeben.

Return Value: Der minimale Absolutwert des Histogramms.

1.2.3.15

unsigned int **giveIndexOfMin** (void)

Minimumsuche

Minimumsuche. Der Index des minimalen Wertes im Histogramm wird gesucht und zurückgegeben.

Return Value: Der Index (die Stützstelle) des minimalen Absolutwertes des Histogramms.

1.2.3.16

EB_ColorDistHistogram& **EB_ColorDistHistogram::mulElementwise** (const EB_ColorDistHistogram &other)

Elementweise Multiplikation

Elementweise Multiplikation. Diese Methode wird zur Elementweisen Multiplikation zweier Instanzen der Klasse EB_ColorDistHistogram (→1.2.1.1, *page 31*) benutzt. Enthält die aktuelle Instanz mehr Elemente als other, werden die überzähligen Elemente auf 0.0 gesetzt.

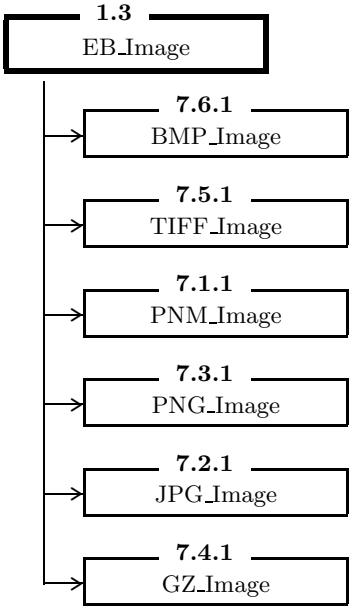
Return Value: Referenz auf die aktuelle Instanz
Parameters: `other` — Referenz auf die Instanz, deren Elemente mit denen der aktuellen multipliziert werden sollen.

1.3

class **EB_Image**

Diese Klasse stellt Bilder als abstrakte Konzepte dar

Inheritance



Public Members

Protected Members

void	**deallocatePalette** (void)	*TODO*
unsigned int	**width**	*Breite des Bildes in Pixeln.*
unsigned int	**height**	*Höhe des Bildes in Pixeln.*
float	**maxfloat**	*Obere Intensitätsgrenze.*
float	**minfloat**	*Untere Intensitätsgrenze.*
EB_PixelDescriptor	**pixel**	*Intensitätsvektor eines Pixels.*
EB_Band**	**bands**	*Vektor der Bänder.*
float**	**palette**	*Palette.*
unsigned int	**palettecolors**	*Anzahl Farben in der Palette.*
static advancementcb	**advancementcallback**	*Callback-Funktion für die Fortschrittsanzeige.*
EB_ImageRegion	**iregion**	*Region, auf die die Operationen, die das unterstützen, beschränkt bleiben.*

Diese Klasse stellt Bilder als abstrakte Konzepte dar. Die Bilder sind alle rechteckig parallel zur x- und y-Achse. Es können beliebig viele Bänder (Instanzen der Klasse EB_Band (→1.1, *page 7*)) in einem Bild vorhanden sein. Für ein Bild, dessen Pixel zum Beispiel durch je einen Rot- Grün- und Blau-Farbwert beschrieben werden, müßte das Bild drei Bänder enthalten. Für jede Instanz läßt sich eine Region definieren, auf die die Bildbearbeitungsoperationen beschränkt werden sollen (bei manchen Operationen ist das nicht sinnvoll). Diese Regionen sind Instanzen der Klasse EB_ImageRegion (→1.8, *page 83*). Zur Zeit funktionieren nur Regionen, deren begrenzende Liniensegmente einander nicht schneiden.

Author: Jürgen ËL BOSSO"Key

1.3.1

Konstruktoren und Destruktor

Names

1.3.1.1

EB_Image (void)

Parameterloser Konstruktor

Parameterloser Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_Image (→1.3.1.1, *page 41*),die keine B ä nder enth ä lt.Damit hat er eine sehr geringe Zeit - und Platzkomplexität.

1.3.1.2

EB_Image (unsigned int x, unsigned int y, unsigned int usedbands, float color = -1.0, float min = 0.0, float max = 1.0)

Konstruktor

Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_Image (→1.3.1.1, *page 41*).

Parameters:
`x` — Breite des Bildes.
`y` — Höhe des Bildes.
`usedbands` — Anzahl an Bändern im Bild.
`color` — Intensitätswert, mit dem alle Pixel aller Bänder initialisiert werden. Liegt dieser Wert außerhalb des durch min und max bestimmten Intervalls, findet keine Initialisierung statt. Voreinstellung ist -1.0.
`min` — Minimaler Intensitätswert in jedem Band. Voreinstellung ist 0.0.
`max` — Maximaler Intensitätswert in jedem Band. Voreinstellung ist 1.0.

1.3.1.3

EB_Image (const EB_Image & Image)

Copy-Konstruktor

Copy-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_Image (→1.3.1.1, *page 41*) als Kopie von der übergebenen Instanz.

Parameters: `Image` — Instanz, von der die Kopie erzeugt wird.

1.3.1.4

EB_Image (const EB_Band & Band)

Konstruktor

Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_Image (→1.3.1.1, *page 41*) als Kopie von der übergebenen Instanz der Klasse EB_Band (→1.1, *page 7*). Als Resultat entsteht ein Bild mit nur einem Band, welches eine Kopie der übergebenen Instanz darstellt.

Parameters: `Band` — Instanz, die zum Erstellen des Images benutzt wird.

1.3.1.5

EB_Image (const EB_Band * Band)

Konstruktor

Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_Image (→1.3.1.1, *page 41*) als Kopie Instanz der Klasse EB_Band (→1.1, *page 7*), auf die der Zeiger verweist. Als Resultat entsteht ein Bild mit nur einem Band, welches eine Kopie der übergebenen Instanz darstellt.

Parameters: `Band` — Zeiger auf eine Instanz, die zum Erstellen des Images benutzt wird.

1.3.1.6

EB_Image (const char *name)

File-Konstruktor

File-Konstruktor. Dieser Konstruktor entspricht dem parameterlosen Konstruktor.

1.3.2

Operatoren

Names

1.3.2.1

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

Zuweisungsoperator

Zuweisungsoperator. Dieser Operator erzeugt eine Kopie der übergebenen in der aktuellen Instanz. Dies ist eine Deep Copy: Es existieren danach zwei völlig unabhängige Instanzen.

Return Value: Referenz auf die aktuelle Instanz
Parameters: `Image` — Instanz, die kopiert werden soll.

1.3.3 public Methoden

Names

1.3.3.1

unsigned int **giveWidth** (void) const

Breite

Breite. Diese Methode liefert die Breite des Bildes in der aktuellen Instanz.

Return Value: Breite der aktuellen Instanz.

1.3.3.2

unsigned int **giveHeight** (void) const

Höhe

Höhe. Diese Methode liefert die Höhe des Bildes in der aktuellen Instanz.

Return Value: Höhe der aktuellen Instanz.

1.3.3.3

float **giveMaxFloat** (void) const

Maximale Intensität

Maximale Intensität. Diese Methode liefert den maximalen Intensitätswert der aktuellen Instanz.

Return Value: Maximaler Intensitätswert der aktuellen Instanz.

1.3.3.4

float **giveMinFloat** (void) const

Minimale Intensität

Minimale Intensität. Diese Methode liefert den minimalen Intensitätswert der aktuellen Instanz.

Return Value: Minimaler Intensitätswert der aktuellen Instanz.

1.3.3.5

unsigned int **giveBandCount** (void) const

Bandanzahl

Bandanzahl. Diese Methode liefert die Anzahl an Bändern im Bild.

Return Value: Anzahl Bänder im Bild.

1.3.3.6

EB_Image& **addBands** (const EB_Image & source) throw(EBIImageCorruptedEXP)

Bänder hinzufügen

Bänder hinzufügen. Mit dieser Methode ist es möglich, die Bänder eines anderen Bildes hinter denen der aktuellen Instanz dieser hinzuzufügen. Dabei werden die Bänder hinter den bereits vorhandenen angehängt. Hatte die aktuelle Instanz also beispielsweise bereits 3 Bänder (Indizes 0-2), so ist nach dieser Operation das zweite Band der Instanz source jetzt das mit Index 4 in der aktuellen Instanz. Die Kopieen werden mittels deep copy angelegt, die beiden Instanzen sind nach der Operation vollständig unabhängig voneinander.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `source` — Instanz, von der die Bänder übernommen werden sollen.

1.3.3.7

EB_Image& **addBand** (const EB_Band & source) throw(EBIImageCorruptedEXP)

Band hinzufügen

Band hinzufügen. Mit dieser Methode ist es möglich, ein Band hinter denen der aktuellen Instanz dieser hinzuzufügen. Dabei wird das Band hinter den bereits vorhandenen angehängt. Hatte die aktuelle Instanz also beispielsweise bereits 3 Bänder (Indizes 0-2), so ist nach dieser Operation der Index für das neue Band 3. Das Band wird mittels deep copy kopiert, die beiden Instanzen sind nach der Operation vollständig unabhängig voneinander.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `source` — Instanz, die zur aktuellen Instanz hinzugefügt werden soll.

1.3.3.8

EB_Image& **overwriteBand** (const EB_Band & source, unsigned int index) throw(EBIIndexOutOfRangeEXP, EBIImageCorruptedEXP)

Band ersetzen

Band ersetzen. Mit dieser Methode ist es möglich, ein Band in der aktuellen Instanz zu ersetzen. Dazu wird das zu ersetzende Band zerstört und an seiner Stelle mittels Copy- Konstruktor von der übergebenen Instanz ein neues erstellt.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `source` — Instanz, die zur aktuellen Instanz hinzugefügt werden soll.
`index` — Nummer des Bandes, das ersetzt werden soll. Existiert kein Band mit diesem Index, wird eine Exception geworfen.

1.3.3.9

EB_Band* **giveBand** (unsigned int index) const throw(EBIIndexOutOfRangeEXP)

Handle auf Band

Handle auf Band. Diese Methode liefert einen Pointer auf das Band mit dem entsprechenden Index in der aktuellen Instanz

Return Value: Zeiger auf eine Instanz der Klasse EB_Band (→1.1, *page 7*)
Parameters: `index` — Nummer des Bandes, das ersetzt werden soll. Existiert kein Band mit diesem Index, wird eine Exception geworfen.

1.3.3.10

EB_Image& **copyWholeTo** (EB_Image & other, int left, int top) throw(EBINoValuesInBandEXP, EBINoEqualNumberOfBandsEXP, EBIImageCorruptedEXP)

Komponieren von Bildern

Komponieren von Bildern. Diese Methode überträgt den Inhalt des von der aktuellen Instanz verkörperten Bildes in ein anderes. Dabei kann man noch einen Offset angeben, der die Position des Bildes im Zielbild angibt. Dieser Offset bezieht sich auf die linke obere Ecke. Sind im aktuellen Bild keine Bänder vorhanden oder stimmen die Anzahlen der Bänder beider Bilder nicht überein, werden Exceptions geworfen. Das Bild in der aktuellen Instanz wird hierbei nicht verändert

Return Value: Referenz auf other.
Parameters: `other` — Instanz der Klasse EB_Image (→1.3.1.1, *page 41*) und Ziel der Kopie.
`left` — Offset vom linken Rand für die Kopie.
`top` — Offset vom oberen Rand für die Kopie.

1.3.3.11

EB_Image& **copyWholeFrom** (const EB_Image & other, int left, int top) throw(EBINoEqualNumberOfBandsEXP, EBIImageCorruptedEXP)

Komponieren von Bildern

Komponieren von Bildern. Diese Methode überträgt den Inhalt des von der Zielinstanz verkörperten Bildes in die aktuelle Instanz. Dabei kann man noch einen Offset angeben, der die Position des Bildes im Zielbild angibt. Dieser Offset bezieht sich auf die linke obere Ecke. Stimmen die Anzahlen der Bänder beider Bilder nicht überein oder enthält ein Band eines der beiden Bilder keine Daten, werden Exceptions geworfen. Das Bild in der aktuellen Instanz wird verändert.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `other` — stanz der Klasse EB_Image (→1.3.1.1, *page 41*) und Quelle der Kopie.
`left` — Offset vom linken Rand für die Kopie.
`top` — Offset vom oberen Rand für die Kopie.

1.3.3.12

EB_Image& **rescaleBands** (float newmin, float newmax)

Dynamikbereich ändern

Dynamikbereich ändern. Mit dieser Methode werden nachträglich die Grenzen der Intensitätswerte in den einzelnen Bändern geändert. Dabei wird nicht abgeschnitten, sondern skaliert. Wird der Dynamikbereich von 0 bis 10 auf 0 bis 1 geändert, wird der Intensitätswert eines Pixels von beispielsweise vorher 5 auf 0.5 geändert. Das Bild in der aktuellen Instanz wird nicht verändert.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `newmin` — Minimaler Intensitätswert in jedem Band.
`newmax` — Maximaler Intensitätswert in jedem Band.

1.3.3.13

EB_Image& **convertRGBToPOMI** (void) throw(EBIWrongNumberOfBandsEXP)

Farbraumkonversion

Farbraumkonversion. Diese Methode erlaubt es, ein Bild in den physiologischen Farbraum zu konvertieren. Dabei wird angenommen, daß das Ausgangsbild im RGB- Farbraum lag. Das Bild in der aktuellen Instanz wird verändert. Hat die aktuelle Instanz nicht genau drei Bänder, wird eine Exception geworfen. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.

1.3.3.14

EB_Image& **convertPOMIToRGB** (void) throw(EBIWrongNumberOfBandsEXP)

Farbraumkonversion

Farbraumkonversion. Diese Methode erlaubt es, ein Bild in den RGB- Farbraum zu konvertieren. Dabei wird angenommen, daß das Ausgangsbild im physiologischen Farbraum lag. Das Bild in der aktuellen Instanz wird verändert. Hat die aktuelle Instanz nicht genau drei Bänder, wird eine Exception geworfen. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.

1.3.3.15

EB_Image& **convertRGBToLAB** (void) throw(EBIWrongNumberOfBandsEXP)

Farbraumkonversion

Farbraumkonversion. Diese Methode erlaubt es, ein Bild in den L*a*b- Farbraum zu konvertieren. Dabei wird angenommen, daß das Ausgangsbild im RGB-Farbraum lag. Das Bild in der aktuellen Instanz wird verändert. Hat die aktuelle Instanz nicht genau drei Bänder, wird eine Exception geworfen. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.

1.3.3.16

EB_Image& **convertLABToRGB** (void) throw(EBIWrongNumberOfBandsEXP)

Farbraumkonversion

Farbraumkonversion. Diese Methode erlaubt es, ein Bild in den RGB- Farbraum zu konvertieren. Dabei wird angenommen, daß das Ausgangsbild im L*a*b-Farbraum lag. Das Bild in der aktuellen Instanz wird verändert. Hat die aktuelle Instanz nicht genau drei Bänder, wird eine Exception geworfen. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.

1.3.3.17

EB_Image **toGray** (const EB_PixelDescriptor &p=EB_PixelDescriptor(1, 1.0))

Grauwertwandlung

Grauwertwandlung. Diese Methode wandelt die Intensitäten aller Bänder des Bildes in einen Intensitätswert um. Bei einem normalen RGB Bild entspricht das der Umwandlung in das entsprechende Grauwertbild. Es ist möglich, für jedes Band Faktoren einzugeben, mit denen das jeweilige Band gewichtet werden soll.

Return Value: Instanz der Klasse EB_Image (→1.3.1.1, *page 41*) mit genau einem Band.

Parameters: `p` — Instanz der Klasse EB_PixelDescriptor (→1.10, *page 94*), die die Faktoren für die entsprechenden Bänder enthält. Enthält sie weniger Werte als Bänder vorhanden sind, wird für die überzähligen der letzte wert in p benutzt.

1.3.3.18

EB_Image& **extractSingleBand** (unsigned int index) throw(EBINoValuesInBandEXP, EBIIndexOutOfRangeEXP)

Band extrahieren

Band extrahieren. Diese Methode erlaubt es, ein Band der aktuellen Instanz zu extrahieren. Daraus wird dann eine neue Instanz der Klasse EB_Image (→1.3.1.1, *page 41*) konstruiert. Das entsprechende Band wird mittels deep copy kopiert, so daß beide Instanzen vollkommen unabhängig voneinander sind. Das Bild in der aktuellen Instanz wird nicht verändert. Ist in der aktuellen Instanz kein Band definiert oder ist für diesen Index kein Band definiert, wird eine Exception geworfen.

Return Value: Instanz der Klasse EB_Image (→1.3.1.1, *page 41*) mit nur einem Band.

Parameters: `index` — Nummer des Bandes, das extrahiert werden soll. Liegt dieser Wert über der Anzahl an Bändern im Bild, wird das erste Band benutzt.

1.3.3.19

EB_Image **extractSingleBandAsRGB** (unsigned int index, unsigned int mode = TOGRAY) throw(EBINoValuesInBandEXP, EBIIndexOutOfRangeEXP)

Band extrahieren

Band extrahieren. Diese Methode erlaubt es, ein Band der aktuellen Instanz zu extrahieren. Daraus wird dann eine Falschfarbendarstellung der Intensitäten des Ursprungsbandes in einer neuen Instanz der Klasse EB_Image (→1.3.1.1, *page 41*) im RGB-Farbraum. Die neue und

die aktuelle Instanz sind anschließend vollkommen unabhängig voneinander. Das Bild in der aktuellen Instanz wird nicht verändert. Ist in der aktuellen Instanz kein Band definiert oder ist für diesen Index kein Band definiert, wird eine Exception geworfen.

Return Value: Instanz der Klasse EB_Image (→1.3.1.1, *page 41*) mit drei Bändern.
Parameters: `index` — Nummer des Bandes, das extrahiert werden soll. Liegt dieser Wert über der Anzahl an Bändern im Bild, wird das erste Band benutzt.
`mode` — Falschfarbenmodus. Hier sind bisher folgende Modi definiert:

- TORED
- TOGREEN
- TOBLUE
- TOGRAY
- TOYELLOW
- TOPURPLE
- TOTURQUOIS
- BLACKORANGEGREEN
- YELLOWGREENRED
- BLACKGREENRED
- YELLOWBROWNGREEN

1.3.3.20

EB_Image **cutoutAPiece** (int xstart, int ystart, unsigned int xdim, unsigned int ydim) throw(EBINoValuesInBandEXP, EBINoEqualNumberOfBandsEXP, EBIImageCorruptedEXP)

Bildausschnitt extrahieren

Bildausschnitt extrahieren. Diese Methode gestattet es. einen definierten, rechteckigen Bereich des durch die aktuelle Instanz verkörperten Bildes auszuschneiden und in einer neuen Instanz zu speichern. Die Parameter beschereiben dabei die linke obere Ecke sowie die Breite und Höhe des Ausschnittes. Ragt der Ausschnitt dabei über das Bild hinaus, wird den entsprechenden

Pixeln der niedrigstmögliche Intensitätswert zugewiesen. Stimmen die Anzahlen der Bänder beider Bilder nicht überein oder enthält ein Band eines der beiden Bilder keine Daten, werden Exceptions geworfen.Das Bild in der aktuellen Instanz wird nicht verändert.

Return Value: Instanz der Klasse EB_Image (→1.3.1.1, *page 41*).
Parameters: `xstart` — x-Koordinate der linken oberen Ecke des Ausschnittes.
`ystart` — y-Koordinate der linken oberen Ecke des Ausschnittes.
`xdim` — Breite des Ausschnittes.
`ydim` — Höhe des Ausschnittes.

1.3.3.21

EB_Image& **aequalize** (void)

Histogrammausgleich

Histogrammausgleich. Diese Methode führt einen Histogrammausgleich auf dem Bild aus. Dabei wird jedes Band separat behandelt. Das Bild in der aktuellen Instanz wird verändert. Diese Methode berücksichtigt die eventuell definierte Region und behandelt nur Bildinhalte innerhalb einer solchen Region. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.

1.3.3.22

EB_Image& **gammaCorrect** (float factor)

Gammakorrektur

Gammakorrektur. Diese Methode führt eine Gammakorrektur auf dem Bild aus. Dies ist eine über der Intensität nichtlineare Helligkeitsänderung. Dabei wird jedes Band separat behandelt. Das Bild in der aktuellen Instanz wird verändert. Diese Methode berücksichtigt die eventuell definierte Region und behandelt nur Bildinhalte innerhalb einer solchen Region. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `factor` — Parameter für die Gammakorrektur. Ist dieser Wert kleiner als eins, wird das Bild dunkler, ansonsten heller.

1.3.3.23

EB_Image& **changeContrast** (float factor)

Kontraständerung

Kontraständerung. Diese Methode führt eine Kontraständerung auf dem Bild aus. Dabei wird jedes Band separat behandelt. Das Bild in der aktuellen Instanz wird verändert. Diese Methode berücksichtigt die eventuell definierte Region und behandelt nur Bildinhalte innerhalb einer solchen Region. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `factor` — Parameter für die Kontraständerung. Je höher dieser Wert ist, desto mehr wird der Kontrast verstärkt.

1.3.3.24

EB_Image& **transformWithLUT** (const EB_IntensityTransformation &trans)

Intensitätsänderung

Intensitätsänderung. Diese Methode führt in den einzelnen Bändern eine Intensitätsänderung entsprechend der nichtlinearen Funktion in trans aus. Dabei dienen die Intensitäten als Argumente für diese Funktion und die Ergebnisse der Funktion als neue Intensitäten. Jedes Band wird separat behandelt. Der Intensitätsbereich wird in 256 Stufen diskretisiert um die Operation mittels einer Look-Up-Tabelle beschleunigen zu können. Die aktuelle Instanz wird verändert. Diese Methode berücksichtigt die eventuell definierte Region und behandelt nur Bildinhalte innerhalb einer solchen Region. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `trans` — Instanz einer von EB_IntensityTransformation (→5.1, *page 146*) abgeleiteten Klasse, die die eigentliche Funktion repräsentiert.

1.3.3.25

EB_Image& **transform** (const EB_IntensityTransformation &trans)

Intensitätsänderung

Intensitätsänderung. Diese Methode führt in den einzelnen Bändern eine Intensitätsänderung entsprechend der nichtlinearen Funktion in trans aus. Dabei dienen die Intensitäten als Argumente für diese Funktion und die Ergebnisse der Funktion als neue Intensitäten. Jedes Band wird separat behandelt. Die aktuelle Instanz wird verändert. Diese Methode berücksichtigt die eventuell definierte Region und behandelt nur Bildinhalte innerhalb einer solchen Region. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `trans` — Instanz einer von EB_IntensityTransformation (→5.1, *page 146*) abgeleiteten Klasse, die die eigentliche Funktion repräsentiert.

1.3.3.26

EB_Image& **transform** (const EB_ImageTransformation &trans)

Koordinatentransformation

Koordinatentransformation. Das Bild wird entsprechend der Vorschrift im Transformationsobjekt transformiert. Jedes Band wird separat behandelt. Diese Methode liefert Fortschrittsberichte über advancementcallback. Die aktuelle Instanz wird verändert.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `trans` — Instanz der Klasse EB_ImageTransformation (→3.1, *page 107*) oder von ihr abgeleiteter Klassen, die die Transformation beschreibt.

1.3.3.27

EB_Image& **transform** (const EB_ImageTransformation &trans, EB_TransformationInterpolator &ip)

Koordinatentransformation

Koordinatentransformation. Das Bild wird entsprechend der Vorschrift im Transformationsobjekt transformiert.Dabei wird das Bild entsprechend der Vorschrift im Interpolationsobjekt interpoliert. Jedes Band wird separat behandelt. Diese Methode liefert Fortschrittsberichte über advancementcallback. Die aktuelle Instanz wird verändert.

Return Value: Referenz auf die aktuelle Instanz.

Parameters: `trans` — Instanz der Klasse EB_ImageTransformation (→3.1, *page 107*) oder von ihr abgeleiteter Klassen, die die Transformation beschreibt.
`ip` — Instanz der Klasse EB_TransformationInterpolator (→4.1, *page 134*) oder von ihr abgeleiteter Klassen, die die Interpolation beschreibt.

1.3.3.28

EB_Image& **transformWithLUT** (const EB_ImageTransformation &trans)

Koordinatentransformation

Koordinatentransformation. Das Band wird entsprechend der Vorschrift im Transformationsobjekt transformiert. Jedes Band wird separat behandelt. Es wird eine Look-Up-Tabelle im Transformationobjekt erzeugt, was wiederholte Anwendungen dieses Objektes beschleunigt. Diese Methode liefert Fortschrittsberichte über advancementcallback. Die aktuelle Instanz wird verändert.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `trans` — Instanz der Klasse EB_ImageTransformation (→3.1, *page 107*) oder von ihr abgeleiteter Klassen, die die Transformation beschreibt.

1.3.3.29

EB_Image **computeColorDistances** (const EB_ImageSegmentation &segment)

Farbähnlichkeit

Farbähnlichkeit. Betrachtet man die Gesamtheit aller Intensitätswerte der Bänder an einer bestimmten Stelle im Bild, kann man das als die Farbe eines Pixels betrachten. Diese Methode berechnet die Ähnlichkeit der Farbe jedes Pixels zu der in der Instanz EB_ImageSegmentation (→6.1, *page 154*) vorliegenden Farbdefinition mittels des dort definierten Distanzmaßes. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Instanz der Klasse EB_Image (→1.3.1.1, *page 41*), die nur ein Band enthält, in dem die Ähnlichkeit des entsprechenden Pixels der aktuellen Instanz zu der in trans hinterlegten Farbdefinition gespeichert ist.
Parameters: `trans` — Instanz der Klasse EB_ImageSegmentation (→6.1, *page 154*) oder von ihr abgeleiteter Klassen.

1.3.3.30

EB_Image **segment** (const EB_ImageSegmentation &segment, float maxdistance)

Farbsegmentation

Farbsegmentation. Betrachtet man die Gesamtheit aller Intensitätswerte der Bänder an einer bestimmten Stelle im Bild, kann man das als die Farbe eines Pixels betrachten. Diese Methode segmentiert das Bild in der aktuellen Instanz abhängig von der Ähnlichkeit der Farbe jedes Pixels zu der in der Instanz EB_ImageSegmentation (→6.1, *page 154*) vorliegenden Farbdefinition. Mittels des dort definierten Distanzmaßes wird ein Abstand der beiden Farbwerte berechnet. Dieser Abstand wird in das Ergebnisbild eingetragen. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Instanz der Klasse EB_Image (→1.3.1.1, *page 41*), die nur ein Band enthält. Dieses Band stellt in seinen Intensitäten den Abstand der Farbe des jeweiligen Pixels von der in trans definierten dar. Dabei bedeutet eine hohe Intensität einen hohen Abstand.

Parameters: `trans` — Instanz der Klasse EB_ImageSegmentation (→6.1, *page 154*) oder von ihr abgeleiteter Klassen.
`maxdistance` — Ist der berechnete Abstand größer als dieser Wert, ist das Ergebnisbild an dieser Stelle schwarz, ansonsten weiß.

1.3.3.31

EB_Image **mask** (const EB_ImageSegmentation &segment, float maxdistance)

Farbsegmentation

Farbsegmentation. Betrachtet man die Gesamtheit aller Intensitätswerte der Bänder an einer bestimmten Stelle im Bild, kann man das als die Farbe eines Pixels betrachten. Diese Methode segmentiert das Bild in der aktuellen Instanz abhängig von der Ähnlichkeit der Farbe jedes Pixels zu der in der Instanz EB_ImageSegmentation (→6.1, *page 154*) vorliegenden Farbdefinition. Mittels des dort definierten Distanzmaßes wird ein Abstand der beiden Farbwerte berechnet. Abhängig von diesem Abstand wird entweder die Originalfarbe oder Schwarz in das Ergebnisbild eingetragen. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Instanz der Klasse EB_Image (→1.3.1.1, *page 41*), die nur ein Band enthält. Dieses Band stellt in seinen Intensitäten den Abstand der Farbe des jeweiligen Pixels von der in trans definierten dar. Dabei bedeutet eine hohe Intensität einen hohen Abstand.

Parameters: `trans` — Instanz der Klasse EB_ImageSegmentation (→6.1, *page 154*) oder von ihr abgeleiteter Klassen.
`maxdistance` — Ist der berechnete Abstand kleiner als dieser Wert, wird in das Ergebnisbild an dieser Stelle die Originalfarbe eingesetzt, ansonsten schwarz.

1.3.3.32

EB_Image **invmask** (const EB_ImageSegmentation &segment, float mindistance)

Farbsegmentation

Farbsegmentation. Betrachtet man die Gesamtheit aller Intensitätswerte der Bänder an einer bestimmten Stelle im Bild, kann man das als die Farbe eines Pixels betrachten. Diese Methode segmentiert das Bild in der aktuellen Instanz abhängig von der Ähnlichkeit der Farbe jedes Pixels zu der in der Instanz EB_ImageSegmentation (→6.1, *page 154*) vorliegenden Farbdefinition. Mittels des dort definierten Distanzmaßes wird ein Abstand der beiden Farbwerte berechnet. Abhängig von diesem Abstand wird entweder die Originalfarbe oder Schwarz in das Ergebnisbild eingetragen. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Instanz der Klasse EB_Image (→1.3.1.1, *page 41*), die nur ein Band enthält. Dieses Band stellt in seinen Intensitäten den Abstand der Farbe des jeweiligen Pixels von der in trans definierten dar. Dabei bedeutet eine hohe Intensität einen hohen Abstand.

Parameters: `trans` — Instanz der Klasse EB_ImageSegmentation (→6.1, *page 154*) oder von ihr abgeleiteter Klassen.
`mindistance` — Ist der berechnete Abstand größer als dieser Wert, wird in das Ergebnisbild an dieser Stelle die Originalfarbe eingesetzt, ansonsten schwarz.

1.3.3.33

float **giveFloatValue** (unsigned int x, unsigned int y, unsigned int index)

Intensitätsbestimmung

Intensitätsbestimmung. Diese Methode erlaubt es, die Intensität in einem bestimmten Band an einer bestimmten Stelle zu ermitteln. Bei Koordinaten, für die keine Werte vorliegen wird 0.0 zurückgegeben.

Return Value: Intensitätswert.
Parameters: `x` — x-Koordinate des Pixels, dessen Intensitätswert ermittelt werden soll.
`y` — y-Koordinate des Pixels, dessen Intensitätswert ermittelt werden soll.
`index` — Nummer des Bandes, aus dem der Intensitätswert ermittelt werden soll. Liegt dieser Wert über der Anzahl an Bändern im Bild, wird 0.0 zurückgegeben.

1.3.3.34

EB_PixelDescriptor& **givePixel** (unsigned int x, unsigned int y)

Intensitätsvektor ermitteln

Intensitätsvektor ermitteln. Diese Methode ermittelt auf einmal alle Intensitätswerte aus allen Bändern an der spezifizierten Stelle. Bei Koordinaten, für die keine Werte vorliegen wird 0.0 zurückgegeben.

Return Value: Referenz auf eine Instanz der Klasse EB_PixelDescriptor (→1.10, *page 94*).
Parameters: `x` — x-Koordinate des Pixels, dessen Intensitätswert ermittelt werden soll.
`y` — y-Koordinate des Pixels, dessen Intensitätswert ermittelt werden soll.

1.3.3.35

EB_PixelDescriptor& **givePixel** (unsigned int index)

Intensitätsvektor ermitteln

Intensitätsvektor ermitteln. Diese Methode ermittelt auf einmal alle Intensitätswerte aus allen Bändern an der spezifizierten Stelle. Bei Koordinaten, für die keine Werte vorliegen wird 0.0 zurückgegeben.

Return Value: Referenz auf eine Instanz der Klasse EB_PixelDescriptor (→1.10, *page 94*).
Parameters: `index` — Laufende Nummer des Pixels. Die Pixel werden dabei von der obersten Zeile beginnend zeilenweise durchnummeriert.

1.3.3.36

unsigned char **giveCharValue** (unsigned int x, unsigned int y, unsigned int index)

Intensitätsbestimmung

Intensitätsbestimmung. Diese Methode erlaubt es, die Intensität in einem bestimmten Band an einer bestimmten Stelle zu ermitteln. Der Intensitätsbereich wird dabei auf den Wertebereich des Typs unsigned char abgebildet. Bei Koordinaten, für die keine Werte vorliegen wird 0 zurückgegeben.

Return Value: Intensitätswert.
Parameters:
`x` — x-Koordinate des Pixels, dessen Intensitätswert ermittelt werden soll.
`y` — y-Koordinate des Pixels, dessen Intensitätswert ermittelt werden soll.
`index` — Nummer des Bandes, aus dem der Intensitätswert ermittelt werden soll. Liegt dieser Wert über der Anzahl an Bändern im Bild, wird 0.0 zurückgegeben.

1.3.3.37

void **setValue** (unsigned int x, unsigned int y, float color, unsigned int index)

Intensitätsänderung

Intensitätsänderung. Diese Methode erlaubt es, die Intensität in einem bestimmten Band an einer bestimmten Stelle zu ändern. Liegt der neue Wert außerhalb des vereinbarten Intensitätsintervalls, wird an der entsprechenden Stelle der nächstliegende Grenzwert eingesetzt. Bei Koordinaten, für die keine Werte vorliegen wird die Änderung ignoriert.

Parameters:
`x` — x-Koordinate des Pixels, dessen Intensitätswert ermittelt werden soll.
`y` — y-Koordinate des Pixels, dessen Intensitätswert ermittelt werden soll.
`color` — Intensitätswert.
`index` — Nummer des Bandes, aus dem der Intensitätswert ermittelt werden soll. Liegt dieser Wert über der Anzahl an Bändern im Bild, wird 0.0 zurückgegeben.

1.3.3.38

void **setPixel** (unsigned int x, unsigned int y, EB_PixelDescriptor color)

Intensitätsänderung

Intensitätsänderung. Diese Methode erlaubt es, die Intensität in allen Bändern an einer bestimmten Stelle zu ändern. Liegt der neue Wert außerhalb des vereinbarten Intensitätsintervalls, wird an der entsprechenden Stelle der nächstliegende Grenzwert eingesetzt. Bei Koordinaten, für die keine Werte vorliegen wird die Änderung ignoriert.

Parameters:

`x` — x-Koordinate des Pixels, dessen Intensitätswert ermittelt werden soll.
`y` — y-Koordinate des Pixels, dessen Intensitätswert ermittelt werden soll.
`color` — Instanz der Klasse EB_PixelDescriptor (→1.10, *page 94*), der einen Vektor mit Intensitäten enthält. Enthält sie weniger Komponenten als das Bild Bänder, werden nur die Bänder geändert, für die Intensitäten vorhanden sind; sind mehr enthalten, werden die überzähligen ignoriert.

1.3.3.39

void **setValue** (unsigned int pindex, float color, unsigned int index)

Intensitätsänderung

Intensitätsänderung. Diese Methode erlaubt es, die Intensität in einem bestimmten Band an einer bestimmten Stelle zu ändern. Liegt der neue Wert außerhalb des vereinbarten Intensitätsintervalls, wird an der entsprechenden Stelle der nächstliegende Grenzwert eingesetzt. Bei Koordinaten, für die keine Werte vorliegen wird die Änderung ignoriert.

Parameters:

`pindex` — Laufende Nummer des Pixels. Die Pixel werden dabei von der obersten Zeile beginnend zeilenweise durchnummeriert.
`color` — Intensitätswert.
`index` — Nummer des Bandes, aus dem der Intensitätswert ermittelt werden soll. Liegt dieser Wert über der Anzahl an Bändern im Bild, wird 0.0 zurückgegeben.

1.3.3.40

void **setPixel** (unsigned int pindex, EB_PixelDescriptor color)

Intensitätsänderung

Intensitätsänderung. Diese Methode erlaubt es, die Intensität in allen Bändern an einer bestimmten Stelle zu ändern. Liegt der neue Wert außerhalb des vereinbarten Intensitätsintervalls, wird an der entsprechenden Stelle der nächstliegende Grenzwert eingesetzt. Bei Koordinaten, für die keine Werte vorliegen wird die Änderung ignoriert.

Parameters:

`pindex` — Laufende Nummer des Pixels. Die Pixel werden dabei von der obersten Zeile beginnend zeilenweise durchnummeriert.
`color` — Instanz der Klasse EB_PixelDescriptor (→1.10, *page 94*), der einen Vektor mit Intensitäten enthält. Enthält sie weniger Komponenten als das Bild Bänder, werden nur die Bänder geändert, für die Intensitäten vorhanden sind; sind mehr enthalten, werden die überzähligen ignoriert.

1.3.3.41

void **setValue** (unsigned int x, unsigned int y, unsigned char color, unsigned int index)

Intensitätsänderung

Intensitätsänderung. Diese Methode erlaubt es, die Intensität in einem bestimmten Band an einer bestimmten Stelle zu ändern. Dabei entspricht ein Wert von 0 dem unteren, ein Wert von 255 dem oberen Ende des für dieses Band festgelegten Intensitätsintervalls. Bei Koordinaten, für die keine Werte vorliegen wird die Änderung ignoriert.

Parameters:

`x` — x-Koordinate des Pixels, dessen Intensitätswert ermittelt werden soll.
`y` — y-Koordinate des Pixels, dessen Intensitätswert ermittelt werden soll.
`color` — Intensitätswert.
`index` — Nummer des Bandes, aus dem der Intensitätswert ermittelt werden soll. Liegt dieser Wert über der Anzahl an Bändern im Bild, wird 0.0 zurückgegeben.

1.3.3.42

void **setValue** (unsigned int pindex, unsigned char color, unsigned int index)

Intensitätsänderung

Intensitätsänderung. Diese Methode erlaubt es, die Intensität in einem bestimmten Band an einer bestimmten Stelle zu ändern. Dabei entspricht ein Wert von 0 dem unteren, ein Wert von 255 dem oberen Ende des für dieses Band festgelegten Intensitätsintervalls. Bei Koordinaten, für die keine Werte vorliegen wird die Änderung ignoriert.

Parameters:
`pindex` — Laufende Nummer des Pixels. Die Pixel werden dabei von der obersten Zeile beginnend zeilenweise durchnummeriert.
`color` — Intensitätswert.
`index` — Nummer des Bandes, aus dem der Intensitätswert ermittelt werden soll. Liegt dieser Wert über der Anzahl an Bändern im Bild, wird 0.0 zurückgegeben.

1.3.3.43

EB_Image& **rotate** (float angle, const EB_PixelDescriptor &p=EB_PixelDescriptor(1, -10000.0)) throw(EBIImageCorruptedEXP)

Rotation

Rotation. Diese Methode rotiert das Bild um den angegebenen Betrag. Die Drehung erfolgt um den Mittelpunkt des Bildes. Das Bild in der aktuellen Instanz wird verändert. Es ist möglich, für jedes Band einen Intensitätswert anzugeben, mit dem Bildbereiche des Resultats ausgefüllt werden, die keine Entsprechung im Original haben.

Return Value:
Referenz auf die aktuelle Instanz.
Parameters:
`angle` — Dieser Wert gibt den Winkel an, um den das Bild gedreht werden soll. Dabei bedeutet ein positiver Wert mathematisch positiven Drehsinn (entgegen der Uhrzeigerrichtung). Die Einheit der Winkelangabe ist Grad. Diese Methode liefert Fortschrittsberichte über advancementcallback.
`p` — Bereiche im Bild, die nach der Rotation undefinierten Inhalt hätten, werden mit dieser Farbe aufgefüllt. Instanz der Klasse EB_PixelDescriptor (→1.10, *page 94*), die die Intensitäten für die entsprechenden Bänder enthält. Enthält sie weniger Werte als Bänder vorhanden sind, wird für die überzähligen der letzte Wert in p benutzt.

1.3.3.44

EB_Image& **rotate** (float angle, int rotcenterx, int rotcentery, const EB_PixelDescriptor &p=EB_PixelDescriptor(1, -10000.0)) throw(EBIImageCorruptedEXP)

Rotation

Rotation. Diese Methode rotiert das Bild um den angegebenen Betrag. Die Drehung erfolgt um den angegebenen Punkt des Bildes, der auch außerhalb liegen darf. Das Bild in der aktuellen Instanz wird verändert.

Return Value: Referenz auf die aktuelle Instanz.

Parameters:
`angle` — Dieser Wert gibt den Winkel an, um den das Bild gedreht werden soll. Dabei bedeutet ein positiver Wert mathematisch positiven Drehsinn (entgegen der Uhrzeigerrichtung). Die Einheit der Winkelangabe ist Grad. Diese Methode liefert Fortschrittsberichte über advancementcallback.
`rotcenterx` — x-Koordinate des Punktes, um den das Bild rotiert wird.
`rotcentery` — y-Koordinate des Punktes,um den das Bild rotiert wird.
`p` — Bereiche im Bild, die nach der Rotation undefinierten Inhalt hätten, werden mit dieser Farbe aufgefüllt. Instanz der Klasse EB_PixelDescriptor (→1.10, *page 94*), die die Intensitäten für die entsprechenden Bänder enthält. Enthält sie weniger Werte als Bänder vorhanden sind, wird für die überzähligen der letzte Wert in p benutzt.

1.3.3.45

EB_Image& **rotateToFit** (float angle, const EB_PixelDescriptor &p=EB_PixelDescriptor(1, -10000.0)) throw(EBIImageCorruptedEXP)

Rotation

Rotation. Diese Methode rotiert das Bild um den angegebenen Betrag. Die Drehung erfolgt um den Mittelpunkt des Bildes. Das Bild in der aktuellen Instanz wird verändert und in seinen Dimensionen so geändert, daß der gesamte Bildinhalt des Originals hineinpaßt. Es ist möglich, für jedes Band einen Intensitätswert anzugeben, mit dem Bildbereiche des Resultats ausgefüllt werden, die keine Entsprechung im Original haben.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `angle` — Dieser Wert gibt den Winkel an, um den das Bild gedreht werden soll. Dabei bedeutet ein positiver Wert mathematisch positiven Drehsinn (entgegen der Uhrzeigerrichtung). Die Einheit der Winkelangabe ist Grad. Diese Methode liefert Fortschrittsberichte über advancementcallback.
`p` — Bereiche im Bild, die nach der Rotation undefinierten Inhalt hätten, werden mit dieser Farbe aufgefüllt. Instanz der Klasse EB_PixelDescriptor (→1.10, *page 94*), die die Intensitäten für die entsprechenden Bänder enthält. Enthält sie weniger Werte als Bänder vorhanden sind, wird für die überzähligen der letzte Wert in p benutzt.

1.3.3.46

EB_Image& **scale** (unsigned int newx, unsigned int newy) throw(EBIImageCorruptedEXP)

Skalierung

Skalierung. Diese Methode gestattet es, ein Bild zu skalieren. Es ist möglich, die neue Breite und Höhe unabhängig voneinander anzugeben. Das Bild in der aktuellen Instanz wird verändert. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `newx` — Neue Breite des Bildes.
`newy` — Neue Höhe des Bildes.

1.3.3.47

EB_Image& **scale** (unsigned int sqrpixels) throw(EBIImageCorruptedEXP)

Skalierung

Skalierung. Diese Methode gestattet es, ein Bild zu skalieren. Die neuen Werte für Breite und Höhe werden aus der übergebenen Pixelmaximalanzahl berechnet. Das Bild in der aktuellen Instanz wird verändert.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `sqrpixels` — Maximalanzahl an Pixeln im Bild nach der Skalierung.

1.3.3.48

EB_Image& **scaleToWidth** (unsigned int newwidth) throw(EBIImageCorruptedEXP)

Skalierung

Skalierung. Diese Methode gestattet es, ein Bild zu skalieren. Die neue Höhe wird unter Beachtung des Höhen-Breiten-Verhältnisses aus der angegebenen neuen Breite berechnet. Das Bild in der aktuellen Instanz wird verändert. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `newwidth` — Neue Breite des Bildes.

1.3.3.49

EB_Image& **scaleToHeight** (unsigned int newheight) throw(EBIImageCorruptedEXP)

Skalierung

Skalierung. Diese Methode gestattet es, ein Bild zu skalieren. Die neue Breite wird unter Beachtung des Höhen-Breiten-Verhältnisses aus der angegebenen neuen Höhe berechnet. Das Bild in der aktuellen Instanz wird verändert. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `newheight` — Neue Höhe des Bildes.

1.3.3.50

EB_Image& **overlay** (EB_Image & other, float otherfac = 0.5, float ownfac = 0.5, int left = 0, int top = 0) throw(EBINoEqualNumberOfBandsEXP, EBIImageCorruptedEXP)

Überblenden

Überblenden. Mit dieser Methode können zwei Bilder mit festen Faktoren überblendet werden. Dabei werden die Intensitäten jedes Bildes für jedes Pixel bandweise addiert. Faktoren bestimmen, mit welchem Anteil die aktuelle und die übergebene Instanz zu dieser Addition beitragen. Ist die Summe der Faktoren größer als 0, werden beide entsprechend zurückskaliert. Stimmen die Bänderanzahlen beider Bilder nicht überein, wird eine Exception geworfen. Das Bild in der aktuellen Instanz wird verändert. Diese Methode berücksichtigt die eventuell definierte Region und behandelt nur Bildinhalte innerhalb einer solchen Region. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `other` — Bild, das demjenigen in der aktuellen Instanz überlagert werden soll.
`otherfac` — Faktor für other.
`ownfac` — Faktor für die aktuelle Instanz.
`left` — Offset von other vom linken Rand.
`top` — Offset von other vom oberen Rand.

1.3.3.51

EB_Image& **overlay** (EB_Image & other, EB_Image & alphaimage, unsigned int bandnumber, int left = 0, int top = 0, int aleft = 0, int atop = 0, float factor = 1.0f) throw(EBINoEqualNumberOfBandsEXP, EBIImageCorruptedEXP)

Überblenden

Überblenden. Mit dieser Methode können zwei Bilder mit festen Faktoren überblendet werden. Dabei werden die Intensitäten jedes Bildes für jedes Pixel bandweise addiert. Wie stark der Einfluß des Bildes in der aktuellen Instanz auf das Ergebnis ist, wird durch die Intensität an der entsprechenden Stelle im festgelegten Band im sogenannten alphaimage bestimmt. Je höher die Intensität in diesem Band ist, desto stärker bestimmt der Inhalt von other das Resultat. Stimmen die Bänderanzahlen der aktuellen Instanz und von other nicht überein, wird eine Exception geworfen. Das Bild in der aktuellen Instanz wird verändert. Diese Methode berücksichtigt die eventuell definierte Region und behandelt nur Bildinhalte innerhalb einer solchen Region. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.

Parameters:
`other` — Bild, das demjenigen in der aktuellen Instanz überlagert werden soll.
`alphaimage` — Bild, dessen Intensitätswerte bestimmen, zu welchem Anteil die aktuelle
`bandnumber` — Nummer des Bandes aus alphaimage, das als Maske benutzt werden soll.
`left` — Offset von other vom linken Rand.
`top` — Offset von other vom oberen Rand.
`left` — Offset von alphaimage vom linken Rand.
`top` — Offset von alphaimage vom oberen Rand.
`factor` — Mit diesem Faktor ist eine Intensitätsbeeinflussung des Ergebnisses möglich. Mit diesem Wert wird das Ergebnis der beschriebenen Addition multipliziert. Bei einem Wert von 0.5 ergibt sich also ein Ergebnis, welches halb so hell als ohne diese Multiplikation wäre.

1.3.3.52

EB_Image& **overlay** (EB_Image & other, EB_Image & alphaimage, int left = 0, int top = 0, int aleft = 0, int atop = 0) throw(EBINoEqualNumberOfBandsEXP, EBIImageCorruptedEXP)

Überblenden

Überblenden. Mit dieser Methode können zwei Bilder mit festen Faktoren überblendet werden. Dabei werden die Intensitäten jedes Bildes für jedes Pixel bandweise addiert. Wie stark der Einfluß des Bildes in der aktuellen Instanz auf das Ergebnis ist, wird durch die Intensität an der entsprechenden Stelle im entsprechenden Band im sogenannten alphaimage bestimmt. Je höher die Intensität in diesem Band ist, desto stärker bestimmt der Inhalt von other das Resultat. Stimmen die Bänderanzahlen der drei Bilder nicht überein, wird eine Exception geworfen. Das Bild in der aktuellen Instanz wird verändert. Diese Methode berücksichtigt die eventuell definierte Region und behandelt nur Bildinhalte innerhalb einer solchen Region. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters:
`other` — Bild, das demjenigen in der aktuellen Instanz überlagert werden soll.
`alphaimage` — Bild, dessen Intensitätswerte bestimmen, zu welchem Anteil die aktuelle Instanz und otherfac in des Ergebnis eingehen sollen.
`left` — Offset von other vom linken Rand.
`top` — Offset von other vom oberen Rand.
`left` — Offset von alphaimage vom linken Rand.
`top` — Offset von alphaimage vom oberen Rand.

1.3.3.53

EB_Image& **quantColors** (unsigned int colorcount, float quality = 0.3) throw(EBIOutOfMemoryEXP, EBINoValuesInBandEXP)

Farbreduktion

Farbreduktion.Werden die Zusammenfassungen der Intensitätswerte aller Bänder an einer bestimmten Stelle als Farbe dieses Pixels bezeichnet, ist diese Funktion in der Lage, die Anzahl unterschiedlicher Farben im Bild zu reduzieren. Diese Reduktion geschieht in der vorliegenden Implementation durch eine Clusterung der Farben bei vorher festgelegter Anzahl der Cluster. Ist dies geschehen, wird jedem Pixel als Farbe das Zentrum des Clusters zugeordnet, zu dem seine Farbe gehört. Das Bild in der aktuellen Instanz wird verändert.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `colorcount` — Anzahl der Farben, die übrigbleiben sollen.
`quality` — Ein zwischen 0.3 und 1.0 beschränkter Faktor, der bestimmt, wie viele Pixel zur Bestimmung der Clusterzentren einbezogen werden sollen. Je höher dieser Wert, desto besser ist das Ergebnis, desto mehr Rechenzeit wird aber auch verbraucht.

1.3.3.54

EB_Image& **correctColors** (unsigned int neuroncount = 10, float colorizer = 1.0) throw(EBIWrongNumberOfBandsEXP, EBIOutOfMemoryEXP)

Farbkorrektur

Farbkorrektur. Diese Methode erlaubt eine Farbkorrektur nach einem patentrechtlich geschützten Verfahren.

1.3.3.55

EB_Image& **doubleMirror** (void) throw(EBIImageCorruptedEXP)

Spiegeln

Spiegeln. Das Bild wird an x- und y-Achse gespiegelt. Das Bild in der aktuellen Instanz wird verändert. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.

1.3.3.56

EB_Image& **verticalMirror** (void) throw(EBIImageCorruptedEXP)

Spiegeln

Spiegeln. Das Bild wird an x-Achse gespiegelt. Das Bild in der aktuellen Instanz wird verändert. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.

1.3.3.57

EB_Image& **horizontalMirror** (void) throw(EBIImageCorruptedEXP)

Spiegeln

Spiegeln. Das Bild wird an y-Achse gespiegelt. Das Bild in der aktuellen Instanz wird verändert. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.

1.3.3.58

EB_Image& **smoothBox** (unsigned int smoothwidth)

Glättung

Glättung. Das Bild wird mit einem quadratischen Boxfilter mit beliebiger Kantenlänge geglättet. Jedes Band wird separat behandelt. Jeder Pixel des Bandes wird mit dem Filter beaufschlagt. Die aktuelle Instanz wird verändert. Diese Methode berücksichtigt nicht die eventuell definierte Region. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `smoothwidth` — Kantenlänge des quadratischen Filters.

1.3.3.59

EB_Image& **smoothBinom** (unsigned int smoothwidth)

Glättung

Glättung. Das Bild wird mit einem quadratischen Binomialfilter mit beliebiger Kantenlänge geglättet. Jedes Band wird separat behandelt. Jeder Pixel des Bandes wird mit dem Filter beaufschlagt. Die aktuelle Instanz wird verändert. Diese Methode berücksichtigt nicht die eventuell definierte Region. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `smoothwidth` — Kantenlänge des quadratischen Filters.

1.3.3.60

EB_Image& **convolute** (const EB_Filter & filter) throw(EBIImageCorruptedEXP)

Filterung

Filterung. Das Bild wird mit einem von einer Instanz der Klasse EB_Filter (→2.1, *page 98*) dargestellten Filter gefiltert. Jeder Pixel jedes Bandes des Bildes wird mit dem Filter beaufschlagt. Es werden zur Zeit nur zweidimensionale Filter zugelassen. Das bedeutet, daß jedes Band mit der angegebenen Filtermaske beaufschlagt wird und nur Werte des jeweiligen Bandes zur Berechnung des neuen Intensitätswertes benutzt werden können. Das Bild in der aktuellen Instanz wird verändert. Diese Methode berücksichtigt die eventuell definierte Region und behandelt nur Bildinhalte innerhalb einer solchen Region. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `filter` — Instanz der Klasse EB_Filter (→2.1, *page 98*).

1.3.3.61

EB_Image& **convolute** (const EB_Filter & filter, EB_Image & alphaimage, int left = 0, int top = 0, float factor = 0.0) throw(EBINoValuesInBandEXP, EBINoEqualNumberOfBandsEXP, EBIImageCorruptedEXP)

Filterung

Filterung. Das Bild wird mit einem von einer Instanz der Klasse EB_Filter (→2.1, *page 98*) dargestellten Filter gefiltert. Jeder Pixel jedes Bandes des Bildes wird entsprechend der Intensität in alphaimage an der entsprechenden Stelle mit dem Filter beaufschlagt. Es werden zur Zeit nur zweidimensionale Filter zugelassen. Das bedeutet, daß jedes Band mit der angegebenen Filtermaske beaufschlagt wird und nur Werte des jeweiligen Bandes zur Berechnung des neuen Intensitätswertes benutzt werden können. Das Bild in der aktuellen Instanz wird verändert. Diese Methode berücksichtigt die eventuell definierte Region und behandelt nur Bildinhalte innerhalb einer solchen Region. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.

Parameters:
`filter` — Instanz der Klasse EB_Filter (→2.1, *page 98*).
`alphaimage` — Instanz der Klasse EB_Image (→1.3.1.1, *page 41*), deren Intensitätswerte angeben, wie sehr das Ergebnis der Operation für jeden einzelnen Pixel vom gefilterten und vom Originalwert abhängt. Dabei wird jeweils das entsprechende Band benutzt. Enthält diese Instanz keine Bänder, wird eine Exception geworfen. Enthält sie nur ein Band, wird dieses für alle Bänder benutzt. Enthält sie mehr als ein Band, jedoch weniger als die aktuelle Instanz, wird eine Exception geworfen.
`left` — Offset der linken Ecke des alphaimage.
`top` — Offset der oberen Ecke des alphaimage.
`factor` — Wenn dieser Wert größer als 1.0 oder kleiner als -1.0 ist, wird ihm der nächstliegende der beiden genannten Werte zugewiesen. Ist der Wert danach größer als 0, dient er als ganz normale Schwelle: für jeden Pixel wird bestimmt, wie sich der Intensitätswert des jeweiligen Bands in alphaimage zur Länge des Intensitätsintervalls verhält. Ist dieses Verhältnis größer als factor, wird der gefilterte Wert in das Ergebnis eingesetzt, ansonsten der Originalwert. Ist factor kleiner als 0, wird der Betrag als Schwelle benutzt. Im Falle der Benutzung des gefilterten Wertes wird dieser nochmals linear mit dem Original verrechnet: Mit o als Originalwert, f als gefiltertem Wert, i als Intensität in alphaimage an der entsprechenden Stelle und r als Resultat ergibt sich dann:

$$r = f * i + (1 - i) * o$$

1.3.3.62

EB_Image& **morphologicOperation** (unsigned int kernelwidth, float gamma)

Morphologische Operation

Morphologische Operation. Diese Methode stellt morphologische Operationen mit quadratischen Masken beliebiger Größe dar. Über einen Parameter kann stufenlos eine Betriebsart zwischen den beiden Extrema Erosion und Dilatation gewählt werden. Jedes Band wird gesondert behandelt. Die aktuelle Instanz wird verändert. Diese Methode liefert Fortschrittsberichte über advancementcallback.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `kernelwidth` — Kantenlänge der quadratischen Maske.
`gamma` — Parameter zur Beeinflussung des Verhaltens der Operation. Werte kleiner als 0.0 werden als 0.0 interpretiert, solche größer als 1.0 als 1.0. Dabei entspricht 0.0 der Dilatation und 1.0 der Erosion.

1.3.3.63

virtual EB_Image& **load** (const char *)

Laden

Laden. Diese Methode stellt zum einen die Schnittstelle für alle Operationen zum Laden von Bildinhalten aus Dateien verschiedener Formate zur Verfügung, zum anderen implementiert sie die Freigabe aller Resourcen, die sowieso neu allokiert werden müssen. Das Bild in der aktuellen Instanz wird verändert.

Return Value: Referenz auf die aktuelle Instanz

1.3.3.64

virtual void **save** (const char *)

Speichern

Speichern. Diese Methode stellt die Schnittstelle für alle Operationen zum Speichern von Bildinhalten in Dateien verschiedener Formate zur Verfügung. Das Bild in der aktuellen Instanz wird nicht verändert.

1.3.3.65

EB_Image& **setRegion** (EB_ImageRegion & region)

Region definieren

Region definieren. Mit dieser Methode wird der aktuellen eine Instanz der Klasse EB_ImageRegion (→1.8, *page 83*) zugeordnet. Diese Klasse stellt zweidimensionale Regionen dar. Zur Zeit können nur Regionen sinnvoll benutzt werden. Werden Operationen durch diese Regionen beeinflußt, ist das in der Dokumentation angegeben.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `region` — Instanz der Klasse EB_ImageRegion (→1.8, *page 83*), die der aktuellen Instanz zugeordnet werden soll.

1.3.3.66

EB_Image& **unsetRegion** (void)

Region löschen

Region löschen. Mit dieser Funktion wird die Zuordnung einer Region zum aktuellen Bild aufgehoben. Nach Aufruf dieser Methode beeinflussen alle Methoden wieder das ganze Bild.

Return Value: Referenz auf die aktuelle Instanz.

1.3.3.67

istream& **readFromStream** (istream &i) throw(EBICouldNotLoadEXP)

Lesen aus einem Stream

Lesen aus einem Stream. Mit dieser Methode kann der Inhalt einer Instanz der Klasse EB_Image (→1.3.1.1, *page 41*) aus einem Stream gelesen werden. Zu Beginn werden der minimale, der maximale Intensitätswert, die Breite, die Höhe und die Anzahl der Bänder gelesen. Anschließend werden die Bänder im Bild den entsprechenden Werten angepaßt und schließlich die Inhalte der Bänder mittels EB_Band::readFromStream (→1.1.3.41, *page 30*) eingelesen. Tritt während des Lesens ein Fehler auf, wird eine Exception geworfen.

Return Value: Referenz auf den übergebenen Stream.
Parameters: `i` — Stream, von dem gelesen werden soll.

1.3.3.68

static bool **giveStreamMode** (void)

Abfrage des Streammodus

Abfrage des Streammodus. Siehe EB_Band::writeToStream (→1.1.3.40, *page 29*).

Return Value: bool true, wenn die Bilder gepackt geschrieben werden sollen.

1.3.3.69

static void **setPackedStreamMode** ()

Setzen des Streammodus

Setzen des Streammodus. Diese Methode bestimmt, daß Bilder fortan gepackt in den Stream geschrieben werden sollen.

1.3.3.70

static void **setHumanReadableStreamMode** ()

Setzen des Streammodus

Setzen des Streammodus. Diese Methode bestimmt, daß Bilder fortan ungepackt in den Stream geschrieben werden sollen.

1.3.4

void **allocateBands** (float color, float min, float max)

Bänder konstruieren

Bänder konstruieren. Diese Methode dient als Unterstützung an Stellen, wo alle Bänder einer Instanz der Klasse EB_Image (→1.3.1.1, *page 41*) neu konstruiert werden müssen.

Parameters:
`color` — Intensitätswert, mit dem alle Pixel aller Bänder initialisiert werden. Liegt dieser Wert außerhalb des durch min und max bestimmten Intervalls, findet keine Initialisierung statt.
`min` — Minimaler Intensitätswert in jedem Band.
`max` — Maximaler Intensitätswert in jedem Band.

1.3.5

void **deallocateBands** (void)

Bänder zerstören

Bänder zerstören. Diese Methode erlaubt es, auf einen Schlag alle Bänder in der aktuellen Instanz zu zerstören.

1.4

ostream& **operator<<** (ostream &o, const EB_Image &pic)

Streamausgabeoperator für EB_Image (→1.3, page 39*)*

Streamausgabeoperator für EB_Image (→1.3, *page 39*). Mit diesem Operator kann man Instanzen der Klasse EB_Image (→1.3, *page 39*) auf C++ Streams ausgeben. Dabei werden zunächst der minimale, der maximale Intensitätswert, die Breite, die Höhe und die Anzahl der Bänder ausgegeben. Anschließend erfolgt die Ausgabe der einzalnen Bänder mittels EB_Band::writeToStream (→1.1.3.40, *page 29*).

Return Value: Eine Referenz auf den übergebenen Stream.
Parameters: `o` — Eine Referenz auf einen Stream, auf den die Ausgabe erfolgen soll.
`pic` — Eine Referenz auf die Instanz der Klasse EB_Image (→1.3, *page 39*), die ausgegeben werden soll.

1.5

istream& **operator>>** (istream &i, EB_Image &pic) throw(EBICouldNotLoadEXP)

Streameingabeoperator für EB_Image (→1.3, page 39*)*

Streameingabeoperator für EB_Image (→1.3, *page 39*). Mit diesem Operator kann man Daten aus einem Stream in Instanzen der Klasse EB_Image (→1.3, *page 39*) laden. Dabei wir ddavon ausgegangen, daß die Daten in einem Format vorliegen, welches dem entspricht, das bei der Beschreibung des Streamausgabeoperators beschrieben wird. Tritt während des Lesens ein Fehler auf, wird eine Exception geworfen.

Return Value: Eine Referenz auf den Stream.
Parameters: `i` — Eine Referenz auf einen Stream, von dem gelesen werden soll.
`s` — Eine Referenz auf die Instanz der Klasse EB_Image (→1.3, *page 39*), die die gelesenen Daten aufnehmen soll.

1.6 class **EB_ImageAdvancement**

Diese Klasse ist als Hilfe für die Klasse EB_Image (→1.3, page 39*) gedacht*

Public Members

Diese Klasse ist als Hilfe für die Klasse EB_Image (→1.3, *page 39*) gedacht. Damit kann man eine komfortable Callback-Funktionalität speziell für Fortschrittsmeldungen aufbauen. (Im Prinzip ist dies nur ein besserer struct.) Funktionen des Typs advancementcb benutzen Instanzen dieser Klasse als Botschaften, die Informationen über den Bearbeitungssatand einer bestimmten Operation beinhalten.

Author: Jürgen ËL BOSSO"Key

1.6.1 Konstruktoren und Destruktor

Names

1.6.1.1

EB_ImageAdvancement (void)

Parameterloser Konstruktor

Parameterloser Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_ImageAdvancement (→1.6.1.1, *page 82*).

1.6.2

unsigned int **a_type**

Typ der Botschaft

Typ der Botschaft. Es gibt derzeit vier Typen, die eine Botschaft haben kann:

- STARTPROCESS zur Anzeige, daß eine Operation begonnen hat,
- CURRENTPROCESS zur Information über den momentanen Status einer laufenden Operation,
- ENDPROCESS zur Anzeige des Endes einer laufenden Operation und schließlich
- MESSAGEONLY, die keine besonder Bedeutung hat.

1.7

class **EB_ImageCoordinatePair**

Diese Klasse stellt zweidimensionale Koordinaten dar

Public Members

Diese Klasse stellt zweidimensionale Koordinaten dar. Sie wird zum Beispiel für die Implementierung des Features der Beschränkung von Bildverarbeitungsoperationen auf konvexe Regionen in der Klasse EB_Image (→1.3, *page 39*) benötigt.

Author: Jürgen ËL BOSSO"Key

1.7.1 Konstruktoren und Destruktor

Names

1.7.1.1 EB_ImageCoordinatePair (void)

Parameterloser Konstruktor

Parameterloser Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_ImageCoordinatePair (→1.7.1.1, *page 83*), die die Koordinate (0,0) verkörpert.

1.7.1.2 EB_ImageCoordinatePair (int x, int y)

Konstruktor

Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_ImageCoordinatePair (→1.7.1.1, *page 83*), die die angegebene Koordinate verkörpert.

Parameters:
`x` — x-Komponente der Koordinate.
`y` — y-Komponente der Koordinate.

1.8 (x,y) class EB_ImageRegion

Diese Klasse stellt eine abstrakte Implementation zweidimensionaler geschlossener Regionen dar

Public Members

Protected Members

EB_UnsortedList < EB_ImageCoordinatePair >

coordlist *Liste der Koordinaten der Eckpunkte der Region.*

bool **preprocessed** *Flags, die anzeigen, ob bereits die Vorverarbeitung stattgefunden hat und ob diese noch gültig ist und ob eine Region definiert ist (ob Punkte in der Liste enthalten sind)*

int **xmin** *Die Eckpunkte des die Region umschließenden Rechtecks.*

Diese Klasse stellt eine abstrakte Implementation zweidimensionaler geschlossener Regionen dar. Die Definition der Umgrenzung von Regionen erfolgt mittels einer Punktliste. Die Punkte werden in der Reihenfolge ihrer Festlegung mit Liniensegmenten verbunden und zum Abschluß wird der letzte noch mit dem ersten verbunden, damit die Region geschlossen ist.

Author: Jürgen ËL BOSSO"Key

1.8.1

Konstruktoren und Destruktor

Names

virtual ~ **EB_ImageRegion** (void)
Destruktor.

1.8.1.1

EB_ImageRegion (void)

Parameterloser Konstruktor

Parameterloser Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_ImageRegion (→1.8.1.1, *page 85*), die noch keine Punkte definiert hat

1.8.2

Operatoren

Names

1.8.2.1

EB_ImageRegion& **operator =** (EB_UnsortedList < EB_ImageCoordinatePair > &list)

Zuweisungsoperator

Zuweisungsoperator. Dieser Operator erzeugt eine Kopie der übergebenen in der aktuellen Instanz. Dies ist eine Deep Copy: Es existieren danach zwei völlig unabhängige Instanzen.

Return Value: Referenz auf die aktuelle Instanz
Parameters: `other` — Instanz, die kopiert werden soll.

1.8.3

public Methoden

Names

1.8.3.1

EB_ImageRegion& **addPoint** (int x, int y)

Punkt hinzufügen

Punkt hinzufügen. Diese Methode fügt einen Punkt durch Angabe der expliziten Koordinaten zur Liste der Eckpunkte der Region hinzu.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `x` — x-Komponente des Punktes.
`y` — y-Komponente des Punktes.

1.8.3.2

EB_ImageRegion& **addPoint** (EB_ImageCoordinatePair &p)

Punkt hinzufügen

Punkt hinzufügen. Diese Methode fügt einen Punkt als Instanz der Klasse EB_ImageCoordinatePair (→1.7, *page 82*) zur Liste der Eckpunkte der Region hinzu.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `p` — Instanz der Klasse EB_ImageCoordinatePair (→1.7, *page 82*), die einen Punkt in zweidimensionalen Daten repräsentiert.

1.8.3.3

EB_ImageRegion& **undefine** (void)

Region löschen

Region löschen. Diese Methode löscht sämtliche Eckpunkte der Region und löst sie damit praktisch auf.

Return Value: Referenz auf die aktuelle Instanz.

1.8.3.4

bool **isInside** (int x, int y)

Check, ob Punkt in der Region liegt

Check, ob Punkt in der Region liegt. Zur Zeit kann dieser Check nur richtig erfolgen, wenn sich die Begrenzungslinien der Region nicht überschneiden.

Return Value: true, wenn der Punkt innerhalb der Region liegt, ansonsten false.
Parameters: `x` — x-Komponente des zu prüfenden Punktes.
`y` — y-Komponente des zu prüfenden Punktes.

1.8.3.5

bool **isRegion** (void)

Test, ob Region definiert

Test, ob Region definiert. Diese Methode zeigt an, ob die Punkte in der Liste bereits eine Region definieren oder nicht, Zur Zeit erfolgt der Test nur anhand der Anzahl an Punkten in der Liste.

Return Value: true, wenn die Punkte in der Liste eine Region definieren, sonst false.

1.8.4

void **doPreprocess** (void)

Preprocessing

Preprocessing. Viele Operationen brauchen, wenn sich die Anzahl und Positionen der Regioneneckpunkte nicht ändern, beim Test, ob Punkte inner- oder außerhalb der Region liegen, nur einmal und nicht für jeden zu testenden Punkt ausgeführt werden. Diese Operationen sind in dieser Methode vereint.

1.8.5

void **findExtrema** (void)

Extremwerte finden

Extremwerte finden. Diese Methode sucht die Werte der größten Ausdehnung der Region in x- und y-Richtung. Mit dem resultierenden umschließenden Rechteck wird dann in isInside (→1.8.3.4, *page 87*) ein schneller Ablehnungstest durchgeführt.

1.9

class **EB_LookUpTable**

Diese Klasse stellt eine abstrakte Implementation einer Lookuptable dar

Public Members

Diese Klasse stellt eine abstrakte Implementation einer Lookuptable dar. Die Tabelle wird mit Integerzahlen indiziert und enthält Gleitkommawerte.

Author: Jürgen ËL BOSSO"Key

1.9.1 Konstruktoren und Destruktor

Names

1.9.1.1 EB_LookUpTable (unsigned int number)

Konstruktor

Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_LookUpTable (→1.9.1.1, *page 89*)mit der festgelegten Anzahl an Einträgen.

Parameters: `number` — Gewünschte Anzahl der Einträge.

1.9.1.2 EB_LookUpTable (unsigned int number, float initvalue)

Konstruktor

Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_LookUpTable (→1.9.1.1, *page 89*)mit der festgelegten Anzahl an Einträgen. Alle Einträge in der Tabelle werden mit dem angegebenen Wert initialisiert.

Parameters: `number` — Gewünschte Anzahl der Einträge.
`initvalue` — Wert, mit dem die Einträge in der Tabelle initialisiert werden sollen.

1.9.1.3

EB_LookUpTable (const EB_LookUpTable &other)

Copy-Konstruktor

Copy-Konstruktor. Dieser Konstruktor erzeugt eine identische Kopie der übergebenen Instanz. Die Kopie wird mittels deep copy erzeugt. Beide Instanzen sind vollständig unabhängig.

Parameters: `other` — Instanz, von der die Kopie erzeugt werden soll.

1.9.2

Operatoren

Names

1.9.2.1

EB_LookUpTable& **operator=** (const EB_LookUpTable &other)

Zuweisungsoperator

Zuweisungsoperator. Dieser Operator erstellt eine Kopie der übergebenen Instanz in der aktuellen Instanz. Dabei wird eine deep copy durchgeführt, danach sind beide Instanzen vollständig unabhängig voneinander.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `other` — Instanz, von der eine Kopie erzeugt werden soll.

1.9.3

public Methoden

Names

1.9.3.1

void **redimension** (unsigned int newsize)

Eintragszahl ändern

Eintragszahl ändern. Mit dieser Methode läßt sich die Anzahl der Einträge in der Tabelle nachträglich ändern. Die bereits in der Tabelle enthaltenen Einträge gehen dann verloren.

Parameters: `newsize` — Neue Anzahl an Einträgen für die Tabelle.

1.9.3.2

void **redimension** (unsigned int newsize, float initvalue)

Eintragszahl ändern

Eintragszahl ändern. Mit dieser Methode läßt sich die Anzahl der Einträge in der Tabelle

nachträglich ändern. Die bereits in der Tabelle enthaltenen Einträge gehen dann verloren. Alle Einträge in der Tabelle werden mit dem angegebenen Wert initialisiert.

Parameters: `newsize` — Neue Anzahl an Einträgen für die Tabelle.
`initvalue` — Wert, mit dem die Einträge in der Tabelle initialisiert werden sollen.

1.9.3.3

float **giveValue** (unsigned int index) const throw(EBIIndexOutOfRangeEXP)

Tabellenwert holen

Tabellenwert holen. Diese Methode gibt den in der Tabelle am spezifizierten Platz stehenden Wert zurück.

Return Value: Gleitkommawert aus der Tabelle.
Parameters: `index` — Position in der Tabelle, deren Wert abgefragt werden soll. Zeigt dieser Wert auf eine Position außerhalb der Tabelle, wird eine Exception geworfen.

1.9.3.4

unsigned int **giveEntryCount** (void) const

Abfrage Eitragsanzahl

Abfrage Eitragsanzahl. Diese Methode liefert die Anzahl der Einträge in der Tabelle.

Return Value: Die Anzahl der Einträge in der Tabelle.

1.9.3.5

void **setValue** (unsigned int index, float value)

Tabellenwert setzen

Tabellenwert setzen. Diese Methode erlaubt es, den Tabellenwert am spezifizierten Platz neu zu setzen.

Parameters: `index` — Position in der Tabelle, deren Wert geändert werden soll.
`value` — Wert, der an der angegebenen Position in die Tabelle geschrieben werden soll.

1.9.3.6

void **init** (float value)

Initialisierung

Initialisierung. Diese Methode erlaubt es, alle Einträge der Tabelle auf einmal mit einem Wert zu beschreiben.

Parameters: `value` — Wert, mit dem die Tabelleneinträge initialisiert werden sollen.

1.9.3.7

void **calculateGammas** (float factor)

Gammakorrektur

Gammakorrektur. Diese Methode berechnet die Einträge für die Tabelle so, daß sie dem Verlauf der Gammakorrekturfunktion entsprechen.

Parameters: `factor` — Argument der Gammafunktion.

1.9.3.8

void **calculateContrast** (float factor)

Kontrastkorrektur

Kontrastkorrektur. Diese Methode berechnet die Einträge in der Tabelle so, daß sie auf ein Bild angewendet eine Änderung des Kontrastes bewirken würden.

1.9.3.9

void **calculateIntensityTransformation** (const EB_IntensityTransformation &trans)

Nichtlineare Funktion

Nichtlineare Funktion. Mit dieser Methode wird die Look-Up-Tabelle mit Einträgen entsprechend der Transformationsvorschrift in trans gefüllt. Dabei wird der Bereich der Funktion von 0 bis 1 benutzt.

Parameters: `trans` — Instanz einer von EB_IntensityTransformation (→5.1, *page 146*) abgeleiteten Klasse, die die eigentliche Funktion repräsentiert.

1.10

class **EB_PixelDescriptor**

Diese Klasse implementiert das abstrakte Konzept eines Bildelements (PICture ELement)

Public Members

Protected Members

unsigned int	**componentcount**	*Anzahl der Komponenten*
float*	**components**	*Zeiger auf das Feld mit den Komponenten*

Diese Klasse implementiert das abstrakte Konzept eines Bildelements (PICture ELement).Sie hängt eng mit dem mit der Klasse EB_Image (→1.3, *page 39*) eingeführten Konzept zusammen. Ein Bild besteht aus mehreren Bändern, die zum Beispiel die einzelnen Dimensionen eines Farbraumes darstellen können. Ein Pixel definiert die Farbe des Bildes an einer ganz bestimmten Stelle durch Zusammenfassen der Intensitätswerte aller Bänder an dieser Stelle. In der Klasse EB_Image (→1.3, *page 39*) werden die Intensitäten aller Pixel durch Float-Werte dargestellt. Daher enthält ein Pixel so viele Float-Werte, wie das korrespondierende Bild Bänder enthält.

Author: Jürgen ËL BOSSO"Key

1.10.1 Konstruktoren und Destruktor

Names

1.10.1.1 EB_PixelDescriptor (unsigned int bands=1, float initvalue = 0.0)

Konstruktor

Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_PixelDescriptor (→1.10.1.1, *page 95*). @param bands Anzahl an Komponenten in diesem Pixel. @ param initvalue Wert, der allen Komponenten des Pixels zugewiesen wird.

1.10.1.2 EB_PixelDescriptor (const EB_PixelDescriptor & pd)

Copy-Konstruktor

Copy-Konstruktor. Dieser Konstruktor erzeugt eine identische Kopie der übergebenen Instanz. Die Kopie wird mittels deep copy erzeugt. Beide Instanzen sind vollständig unabhängig.

Parameters: `pd` — Instanz, von der die Kopie erzeugt werden soll.

1.10.2 Operatoren

Names

1.10.2.1

EB_PixelDescriptor& **operator =** (const EB_PixelDescriptor & pd)

Zuweisungsoperator

Zuweisungsoperator. Dieser Operator erstellt eine Kopie der übergebenen Instanz in der aktuellen Instanz. Dabei wird eine deep copy durchgeführt, danach sind beide Instanzen vollständig unabhängig voneinander.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `pd` — Instanz, von der eine Kopie erzeugt werden soll.

1.10.2.2

float& **operator[]** (unsigned int index) const
throw(EBIIndexOutOfRangeEXP)

Indexoperator

Indexoperator. Mit diesem Operator kann man auf die einzelnen Komponenten des Pixels zugreifen. ist der Index ungültig, wird eine Exception geworfen.

Return Value: Intensitätswert des Pixels im entsprechenden Band.
Parameters: `index` — bestimmt, auf welche Komponente zugegriffen werden soll.

1.10.2.3

unsigned int **giveComponentCount** (void) const

Größe

Größe.

Return Value: Anzahl an Intensitätswerten in der aktuellen Instanz

1.10.3

public Methoden

Names

1.10.3.1

EB_PixelDescriptor& **changeComponentCount** (unsigned int newcount, float initvalue = 0.0)

Größenänderung

Größenänderung. Mit dieser Methode kann die Anzahl der Komponenten eines Pixels geändert werden. Es ist sowohl eine Verringerung als auch eine Erhöhung der Komponentenanzahl möglich. Der vorherige Inhalt des Pixels geht verloren.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `newcount` — neue Komponentenanzahl für diesen Pixel. @initvalue Initialisierungswert für alle Komponenten.

2 Filter

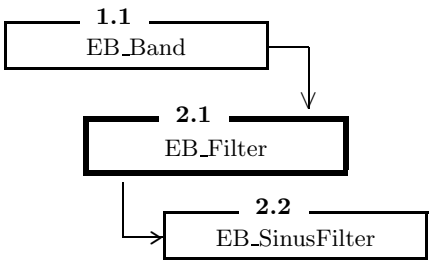
Names

typedef float **(*cffuncptr)** (unsigned int x, unsigned int y, unsigned int width, unsigned int height)
Callbackfunktion zur Berechnung von Filterkoeffizienten in Abhängigkeit von der Position (x, y) in der Filtermaske für die Klasse EB_Filter (→2.1, page 98*)*

2.1 class **EB_Filter** : public EB_Band

Diese Klasse stellt die Implementation von zweidimensionalen Filtern mit beliebigen Abmessungen dar

Inheritance



Public Members

Diese Klasse stellt die Implementation von zweidimensionalen Filtern mit beliebigen Abmessungen dar. Es werden nur Methoden zur Organisation von und zum Zugriff auf Filtermasken zur Verfügung gestellt. Die Filtermasken dürfen beliebige Abmessungen haben.

Author: Jürgen ËL BOSSO"Key

2.1.1

Konstruktoren und Destruktor

Names

virtual **~EB_Filter** () *Destruktor.*

2.1.1.1

EB_Filter (unsigned int x, unsigned int y)

Konstruktor

Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_Filter (→2.1.1.1, *page 99*) mit vorgegebenen Kantenlängen. Ist eine der Kantenlängen gerade, wird die nächstgrößere ungerade Zahl benutzt, also zum Beispiel 5 statt 4.

Parameters:
`x` — Breite der Filtermaske.
`y` — Höhe der Filtermaske.

2.1.1.2

EB_Filter (const EB_Filter &other)

Copy-Konstruktor

Copy-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_Filter (→2.1.1.1, *page 99*) als Kopie der übergebenen Instanz.

Parameters: `other` — Instanz, die zum Erstellen der Kopie benutzt wird.

2.1.2

Operatoren

Names

2.1.2.1 EB_Filter& **operator=** (const EB_Filter &other)
Zuweisungsoperator 100

2.1.2.1

EB_Filter& **operator=** (const EB_Filter &other)

Zuweisungsoperator

Zuweisungsoperator. Dieser Operator erzeugt eine Kopie der übergebenen in der aktuellen Instanz. Dies ist eine Deep Copy: Es existieren danach zwei völlig unabhängige Instanzen.

Return Value: Referenz auf die aktuelle Instanz
Parameters: `other` — Instanz, die kopiert werden soll.

2.1.3

public Methoden

Names

2.1.3.1 void **setValueSquareSymm** (unsigned int x, unsigned int y, float coefficient)
Einzelnen Filterkoeffizienten setzen 101

2.1.3.2 void **setValueXSymm** (unsigned int x, unsigned int y, float coefficient)
Einzelnen Filterkoeffizienten setzen 101

2.1.3.3 void **setValueYSymm** (unsigned int x, unsigned int y, float coefficient)

2.1.3.1

void **setValueSquareSymm** (unsigned int x, unsigned int y, float coefficient)

Einzelnen Filterkoeffizienten setzen

Einzelnen Filterkoeffizienten setzen. Diese Methode erlaubt es, effizient Filtermasken zu definieren, die symmetrisch bezüglich der x- und y-Achse sind. Der übergebene Wert wird an der angegebenen Stelle und an den entsprechenden an den Achsen gespiegelten Stellen, insgesamt also viermal in die Maske eingetragen.

Parameters:
`x` — Spalte der Filtermaske.
`y` — Zeile der Filtermaske.
`coefficient` — Wert für den Filterkoeffizienten an der entsprechenden Stelle.

2.1.3.2

void **setValueXSymm** (unsigned int x, unsigned int y, float coefficient)

Einzelnen Filterkoeffizienten setzen

Einzelnen Filterkoeffizienten setzen. Diese Methode erlaubt es, effizient Filtermasken zu definieren, die symmetrisch bezüglich der x-Achse sind. Der übergebene Wert wird an der angegebenen Stelle und an den entsprechenden an der x-Achse gespiegelten Stelle, insgesamt also zweimal in die Maske eingetragen.

Parameters:
`x` — Spalte der Filtermaske.
`y` — Zeile der Filtermaske.
`coefficient` — Wert für den Filterkoeffizienten an der entsprechenden Stelle.

2.1.3.3

void **setValueYSymm** (unsigned int x, unsigned int y, float coefficient)

Einzelnen Filterkoeffizienten setzen

Einzelnen Filterkoeffizienten setzen. Diese Methode erlaubt es, effizient Filtermasken zu definieren, die symmetrisch bezüglich der y-Achse sind. Der übergebene Wert wird an der angegebenen Stelle und an den entsprechenden an der y-Achse gespiegelten Stelle, insgesamt also zweimal in die Maske eingetragen.

Parameters:

`x` — Spalte der Filtermaske.
`y` — Zeile der Filtermaske.
`coefficient` — Wert für den Filterkoeffizienten an der entsprechenden Stelle.

2.1.3.4

void **setValueFunc** (cffuncptr function)

Filterkoeffizienten durch Funktion definieren

Filterkoeffizienten durch Funktion definieren. Diese Methode erlaubt es, Filtermasken entsprechend einer Funktion zu definieren. Die Funktion muß zwei int Argumente annehmen, die die Position des zu berechnenden Koeffizienten darstellen und den Filterkoeffizienten als float zurückliefern.

Parameters:

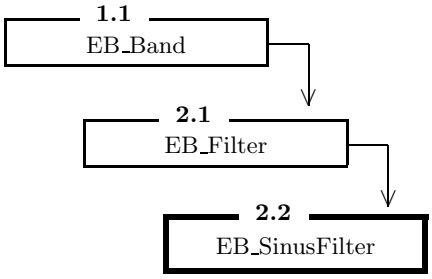
`function` — Zeiger auf die Funktion, die die Koeffizienten berechnet.
`y` — Zeile der Filtermaske.
`coefficient` — Wert für den Filterkoeffizienten an der entsprechenden Stelle.

2.2

class **EB_SinusFilter** : public EB_Filter

Diese Klasse stellt die Implementation von zweidimensionalen Sinusfiltern mit einstellbarer Frequenz und Phasenlage dar

Inheritance



Public Members

Protected Members

2.2.3 void **calculateFilterMask** (void)
Filtermaske berechnen 106

float	**xfreq**	*Frequenz in x-Richtung bezogen auf die Filterbreite.*
float	**yfreq**	*Frequenz in y-Richtung bezogen auf die Filteröhe.*
float	**xoff**	*Offset der Filterfunktion in x-Richtung.*
float	**yoff**	*Offset der Filterfunktion in y-Richtung.*

Diese Klasse stellt die Implementation von zweidimensionalen Sinusfiltern mit einstellbarer Frequenz und Phasenlage dar. Die Filterfunktion ist hier sin(x*xfreq+xoff)*sin(y*yfreq+yoff). Die Frequenz bezieht sich hierbei auf die Abmessungen der Filtermaske. Eine Frequenz von 1 in x-und y-Richtung würde also bedeuten, daß genau eine Periode der Funktion in der Maske enthalten wäre.

Author: Jürgen ËL BOSSO"Key

2.2.1

Konstruktoren und Destruktor

Names

2.2.1.1 **EB_SinusFilter** (unsigned int x, unsigned int y)
Konstruktor 104

2.2.1.2 **EB_SinusFilter** (unsigned int x, unsigned int y, float xf, float yf, float xo, float yo)

2.2.1.1

EB_SinusFilter (unsigned int x, unsigned int y)

Konstruktor

Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_SinusFilter (→2.2.1.1, *page 104*) mit vorgegebenen Kantenlängen. Die Frequenz ist für beide Dimensionen 1.0, der Offset ist für beide Dimensionen 0.0.

Parameters:
`x` — Breite der Filtermaske.
`y` — Höhe der Filtermaske.

2.2.1.2

EB_SinusFilter (unsigned int x, unsigned int y, float xf, float yf, float xo, float yo)

Konstruktor

Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_SinusFilter (→2.2.1.1, *page 104*) mit vorgegebenen Kantenlängen. Die Frequenz ist für beide Dimensionen 1.0, der Offset ist für beide Dimensionen 0.0.

Parameters:
`x` — Breite der Filtermaske.
`y` — Höhe der Filtermaske.
`xf` — Frequenz der Filterfunktion in x-Richtung bezogen auf die Breite der Filtermaske.
`yf` — Frequenz der Filterfunktion in y-Richtung bezogen auf die Höhe der Filtermaske.
`xo` — Offset der Filterfunktion in x-Richtung in Radiant.
`yo` — Offset der Filterfunktion in y-Richtung in Radiant.

2.2.1.3

EB_SinusFilter (const EB_SinusFilter &other)

Copy-Konstruktor

Copy-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_SinusFilter (→2.2.1.1, *page 104*) als Kopie der übergebenen Instanz.

Parameters: `other` — Instanz, die zum Erstellen der Kopie benutzt wird.

2.2.2

public Methoden

Names

2.2.2.1

void **setXFrequency** (float xf)

Frequenz setzen

Frequenz setzen. Diese Methode setzt die Frequenz des Filters in x-Richtung.

Parameters: `xf` — Frequenz der Filterfunktion in x-Richtung bezogen auf die Breite der Filtermaske.

2.2.2.2

void **setYFrequency** (float yf)

Frequenz setzen

Frequenz setzen. Diese Methode setzt die Frequenz des Filters in y-Richtung.

Parameters: `yf` — Frequenz der Filterfunktion in y-Richtung bezogen auf die Höhe der Filtermaske.

2.2.2.3

void **setXOffset** (float xo)

Offset setzen

Offset setzen. Diese Methode setzt den Offset der Filterfunktion in x-Richtung.

Parameters: `xo` — Offset der Filterfunktion in x-Richtung in Radiant.

2.2.2.4

void **setYOffset** (float yo)

Offset setzen

Offset setzen. Diese Methode setzt den Offset der Filterfunktion in y-Richtung.

Parameters: `yo` — Offset der Filterfunktion in y-Richtung in Radiant.

2.2.3

void **calculateFilterMask** (void)

Filtermaske berechnen

Filtermaske berechnen. Diese Methode berechnet bei der Konstruktion einer Instanz oder nach einer Parameteränderung die Werte der Filtermaske.

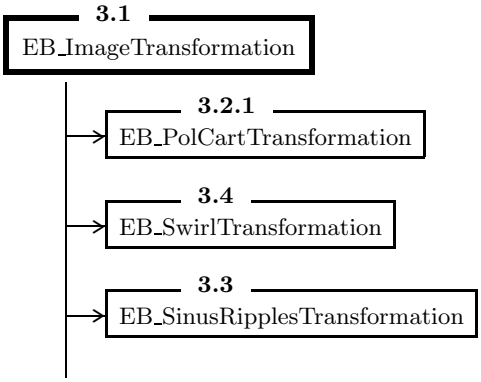
3 Koordinatentransformationen

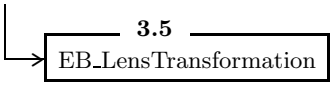
Names

3.1 class EB_ImageTransformation

Diese Klasse stellt eine Schnittstelle für Koordinatentransformationen in rechteckigen Intensitätsverteilungen wie zum Beispiel Bildern dar

Inheritance





Public Members

Protected Members

mutable unsigned int
width *Breite der Eingangsdaten.*

mutable unsigned int
height *Höhe der Eingangsdaten.*

mutable bool
haslut *Flag, das anzeigt, ob eine gültige LUT vorliegt.*

mutable unsigned int*
lut *Zeiger auf ein Feld von unsigned ints, das die LUT darstellt.*

mutable float
origx *Die Float-Koordinaten des Originalpixels zum Interpolieren!*

mutable float
origy *Die Float-Koordinaten des Originalpixels zum Interpolieren!*

Diese Klasse stellt eine Schnittstelle für Koordinatentransformationen in rechteckigen Intensitätsverteilungen wie zum Beispiel Bildern dar. Solche Transformationen sind beispielsweise Drehen, Skalieren, Spiegeln oder Konvertierung zwischen Koordinatensystemen, zum Beispiel polar <-> kartesisch. Diese Klasse enthält mehrere Methoden, die in abgeleiteten Klassen überladen werden müssen. Zum einen sind das die Methoden, die die Größe des Resultatbildes in Abhängigkeit von den Dimensionen des zu verarbeitenden Bildes berechnen. Daneben ist die Transformationsmethode selbst zu überladen. Zu den von der Schnittstelle bereitgestellten Funktionen gehört die selbstständig erstellte Look-Up-Tabelle, die die Ausführung derselben Transformation auf unterschiedlichen Eingangsdaten mit den gleichen Dimensionen beschleunigt.

Author: Jürgen ËL BOSSO"Key

3.1.1

Konstruktoren und Destruktor

Names

3.1.1.1

EB_ImageTransformation (void)

Parameterloser Konstruktor

Parameterloser Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_ImageTransformation (→3.1.1.1, *page 109*).

3.1.1.2

EB_ImageTransformation (const EB_ImageTransformation &source)

Copy-Konstruktor

Copy-Konstruktor. Dieser Konstruktor erzeugt eine identische Kopie der übergebenen Instanz. Die Kopie wird mittels deep copy erzeugt. Beide Instanzen sind vollständig unabhängig.

Parameters: `source` — Instanz, von der die Kopie erzeugt werden soll.

3.1.2

Operatoren

Names

3.1.2.1

EB_ImageTransformation& **operator =** (const EB_ImageTransformation &source)

Zuweisungsoperator

Zuweisungsoperator. Dieser Operator erzeugt eine Kopie der übergebenen in der aktuellen Instanz. Dies ist eine Deep Copy: Es existieren danach zwei völlig unabhängige Instanzen.

Return Value: Referenz auf die aktuelle Instanz
Parameters: `other` — Instanz, die kopiert werden soll.

3.1.3

public Methoden

Names

3.1.3.1

void **setSourceDimensions** (unsigned int w, unsigned int h) const

Eingangsdimensionen setzen

Eingangsdimensionen setzen. Mit dieser Methode werden dee aktuellen Instanz die Dimensionen der zu transformierenden Daten bekanntgegeben. Unterscheiden sie sich von den vorher gültigen Werten, wird die eventuell bereits berechnete Look-Up-Tabelle ungültig.

Parameters: `w` — Breite der Eingangsdaten.
`h` — Höhe der Eingangsdaten.

3.1.3.2

virtual unsigned int **giveDestinationHeight** (void) const

Höhe des Resultats

Höhe des Resultats. Diese Methode ist pur virtuell. Sie muß von abgeleiteten Klassen überladen werden. Das Resultat dieser Funktion muß die Höhe des Resultats der Transformation sein.

Return Value: Von den Dimensionen der Eingagsdaten abhängige Höhe des Resultats.

3.1.3.3

virtual unsigned int **giveDestinationWidth** (void) const

Breite des Resultats

Breite des Resultats. Diese Methode ist pur virtuell. Sie muß von abgeleiteten Klassen überladen werden. Das Resultat dieser Funktion muß die Breite des Resultats der Transformation sein.

Return Value: Von den Dimensionen der Eingagsdaten abhängige Breite des Resultats.

3.1.3.4

virtual unsigned int **transform** (unsigned int x, unsigned int y) const

Transformation

Transformation. Diese Methode ist das Herzstück der Klasse. Sie ist pur virtuell und muß von abgeleiteten Klassen überladen werden. Dabei ist es so, daß die übergebenen Koordinaten einen

Punkt im Resultat spezifizieren. Die Methode muß dann den linear kodierten (die Eingangsdaten werden dabei zeilenweise durchnumeriert) Punkt in den Eingangsdaten berechnen, dessen Inhalt an dieser Stelle in das Resultat eingetragen werden soll.

Return Value: linear kodierte Koordinate des Punktes in der Eingabedaten, dessen Inhalt an die angegebene Stelle des Resultats kopiert werden soll.

Parameters: `x` — x-Koordinate im Resultat.
`y` — y-Koordinate im Resultat.

3.1.3.5

unsigned int **transform** (unsigned int index) const

Transformation

Transformation. Hier ist es so, daß die übergebenen Koordinaten einen Punkt im Resultat linear kodieren. Die Methode liefert den linear kodierten Punkt in den Eingangsdaten, dessen Inhalt an dieser Stelle in das Resultat eingetragen werden soll.

Return Value: linear kodierte Koordinate des Punktes in der Eingabedaten, dessen Inhalt an die angegebene Stelle des Resultats kopiert werden soll.

Parameters: `index` — linear kodierte Koordinate des Resultats.

3.1.3.6

unsigned int **transformWithLUT** (unsigned int x, unsigned int y) const

Transformation mit LUT

Transformation mit LUT. Die übergebenen Koordinaten kodieren einen Punkt im Resultat. Die Methode liefert den linear kodierten Punkt in den Eingangsdaten, dessen Inhalt an dieser Stelle in das Resultat eingetragen werden soll. Ist noch keine LUT erstellt, wird dies für alle Elemente des Resultats nachgeholt. In der LUT stehen danach die linear kodierten Koordinaten der jeweils zugehörigen Punkte der Eingabedaten.

Return Value: linear kodierte Koordinate des Punktes in der Eingabedaten, dessen Inhalt an die angegebene Stelle des Resultats kopiert werden soll.

Parameters: `x` — x-Koordinate im Resultat.
`y` — y-Koordinate im Resultat.

3.1.3.7

unsigned int **transformWithLUT** (unsigned int index) const

Transformation mit LUT

Transformation mit LUT. Hier spezifizieren die übergebenen Koordinaten linear kodiert einen Punkt im Resultat. Die Methode liefert den linear kodierten Punkt in den Eingangsdaten, dessen Inhalt an dieser Stelle in das Resultat eingetragen werden soll. Ist noch keine LUT erstellt, wird dies für alle Elemente des Resultats nachgeholt. In der LUT stehen danach die linear kodierten Koordinaten der jeweils zugehörigen Punkte der Eingabedaten.

Return Value: linear kodierte Koordinate des Punktes in der Eingabedaten, dessen Inhalt an die angegebene Stelle des Resultats kopiert werden soll.

Parameters: `index` — linear kodierte Koordinate des Resultats.

3.2

Polar-kartesische Transformationen

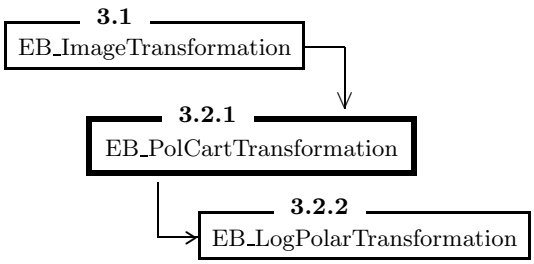
Names

3.2.1

class **EB_PolCartTransformation** : public EB_ImageTransformation

Diese Klasse dient als Basisklasse für Spielarten der kartesisch-polaren Koordinatentransformation

Inheritance



Public Members

Protected Members

double	**inner**	*Minimaler Transformationsradius.*
double	**outer**	*Maximaler Transformationsradius.*
double	**sample**	*Abtastrate für den Winkel.*
double	**startangle**	*Startwinkel.*
double	**span**	*Winkelbereich.*

Diese Klasse dient als Basisklasse für Spielarten der kartesisch-polaren Koordinatentransformation. Sie implementiert dazu hauptsaechlich Hilfsfunktionen zur Parametrisierung solcher Funktionen. Es lassen sich dabei der Start- und Endwinkel einer solchen Transformation ebenso wie minimaler und maximaler Radius sowie die Abtastrate im Winkel festlegen.

Author: Jürgen ËL BOSSO"Key

3.2.1.1

Konstruktoren und Destruktor

Names

Destruktor.

3.2.1.1.1

EB_PolCartTransformation (void)

Parameterloser Konstruktor

Parameterloser Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_PolCartTransformation (→3.2.1.1.1, *page 115*).

3.2.1.1.2

EB_PolCartTransformation (const EB_PolCartTransformation &source)

Copy-Konstruktor

Copy-Konstruktor. Dieser Konstruktor erzeugt eine identische Kopie der übergebenen Instanz. Die Kopie wird mittels deep copy erzeugt. Beide Instanzen sind vollständig unabhängig.

Parameters: `source` — Instanz, von der die Kopie erzeugt werden soll.

3.2.1.2

Operatoren

Names

3.2.1.2.1 EB_PolCartTransformation&
operator = (const EB_PolCartTransformation &source)
Zuweisungsoperator 116

3.2.1.2.1

EB_PolCartTransformation& **operator =** (const EB_PolCartTransformation &source)

Zuweisungsoperator

Zuweisungsoperator. Dieser Operator erzeugt eine Kopie der übergebenen in der aktuellen Instanz. Dies ist eine Deep Copy: Es existieren danach zwei völlig unabhängige Instanzen.

Return Value: Referenz auf die aktuelle Instanz
Parameters: `other` — Instanz, die kopiert werden soll.

3.2.1.3

public Methoden

Names

3.2.1.3.1

void **setInnerRadius** (double r)

Minimaler Radius

Minimaler Radius. Diese Methode setzt den minimalen Radius der Transformation fest. Das bedeutet, daß der Inhalt auf diesem Radius die unterste Zeile im Resultat bildet.

Parameters: `r` — Minimaler Radius. Wird interpretiert als Verhältnis zum größten noch vollständig in den Eingangsdaten liegenden Radius, der durch die Hälfte der Höhe oder Breite der Eingangsdaten - je nachdem, was kleiner ist - bestimmt wird. Ist dieser Wert kleiner als 0, wird der Betrag verwendet, ist dieser größer als der maximale Radius, wird der minimale auf 0.0 gesetzt.

3.2.1.3.2

void **setOuterRadius** (double r)

Maximaler Radius

Maximaler Radius. Diese Methode setzt den maximalen Radius der Transformation fest. Das bedeutet, daß der Inhalt auf diesem Radius die oberste Zeile im Resultat bildet.

Parameters: `r` — Maximaler Radius. Wird interpretiert als Verhältnis zum größten noch vollständig in den Eingangsdaten liegenden Radius, der durch die Hälfte der Höhe oder Breite der Eingangsdaten - je nachdem, was kleiner ist - bestimmt wird. Ist dieser Wert kleiner als 0, wird der Betrag verwendet, ist dieser kleiner als der minimale oder größer als 1.0, wird er auf 1.0 gesetzt

3.2.1.3.3

void **setSampling** (double s)

Abtastrate

Abtastrate. Diese Methode setzt die Winkelabtastrate fest. Eine 1.0 bedeutet dabei, daß das Bild mit einer Abtastrate von einem Grad pro Bildspalte des Resultats abgetatset wird. Eine 2.0 würde eine Abtastrate von 2 Grad pro Bildspalte des Resultats ergeben.

Parameters: `s` — Abtastrate in Grad pro Bildspalte des Resultatbildes.

3.2.1.3.4

void **setStartingAngle** (double a)

Startwinkel

Startwinkel. Diese Methode legt den Winkel des Resultates fest, der der mittelsten Spalte des Resultats entspricht. Dabei bedeutet 0 Grad vom Bildmittelpunkt gerade nach oben. Die Zählweise ist mathematisch positiv.

Parameters: `a` — Startwinkel in Radiant. Liegt der Wert nicht innerhalb des Intervalls (-2PI, 2PI), wird 0.0 benutzt.

3.2.1.3.5

void **setStartingAngleInGrad** (double a)

Startwinkel

Startwinkel. Diese Methode legt den Winkel des Resultates fest, der der mittelsten Spalte des Resultats entspricht. Dabei bedeutet 0 Grad vom Bildmittelpunkt gerade nach oben. Die Zählweise ist mathematisch positiv.

Parameters: `a` — Startwinkel in Grad. Liegt der Wert nicht innerhalb des Intervalls (-360, 360), wird 0.0 benutzt.

3.2.1.3.6

void **setSpan** (double a)

Winkelöffnung

Winkelöffnung. Diese Methode legt den Winkelbereich des Resultates fest. Das Resultat enthält das transformierte Bild, welches sich rechts und links des Startwinkels bis zum hier angegebenen Winkel erstreckt. Bei einem Startwinkel von PI/8 und einem Öffnungswert von PI/4 würde also der Bereich von -3/8PI bis 5/8PI transformiert.

Parameters: `a` — Öffnungswinkel in Radiant. Liegt der Betrag des Wertes nicht innerhalb des Intervalls (0, PI), wird 0.0 benutzt.

3.2.1.3.7

void **setSpanInGrad** (double a)

Winkelöffnung

Winkelöffnung. Diese Methode legt den Winkelbereich des Resultates fest. Das Resultat enthält das transformierte Bild, welches sich rechts und links des Startwinkels bis zum hier angegebenen Winkel erstreckt. Bei einem Startwinkel von 45Grad und einem Öffnungswert von 90Grad würde also der Bereich von 45Grad bis 135Grad transformiert.

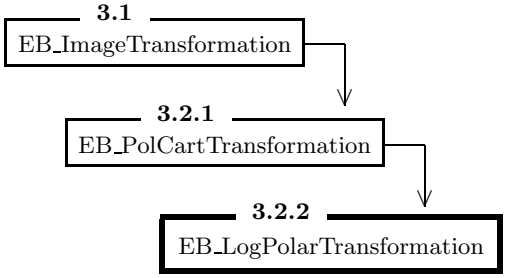
Parameters: `a` — Öffnungswinkel in Grad. Liegt der Betrag des Wertes nicht innerhalb des Intervalls (0, 180), wird 0.0 benutzt.

3.2.2

class **EB_LogPolarTransformation** : public EB_PolCartTransformation

Diese Klasse realisiert eine spezielle Transformation aus Polar- in kartesische Koordinaten, wie sie zur Berechnung eines Panoramabildes aus Bildern von omnidirektionalen Kameras gebraucht wird

Inheritance



Public Members

Diese Klasse realisiert eine spezielle Transformation aus Polar- in kartesische Koordinaten, wie sie zur Berechnung eines Panoramabildes aus Bildern von omnidirektionalen Kameras gebraucht wird. Diese Transformation berücksichtigt im speziellen eine Klasse von Kameras, die die Rundumsicht mittels eines konvexen, parabolischen Spiegels erzeugen.

Author: Jürgen ËL BOSSO"Key

3.2.2.1

Konstruktoren und Destruktor

Names

3.2.2.1.1 **EB_LogPolarTransformation** (void)
Parameterloser Konstruktor 120

virtual ~ **EB_LogPolarTransformation** (void)
Destruktor.

3.2.2.1.1

EB_LogPolarTransformation (void)

Parameterloser Konstruktor

Parameterloser Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_LogPolarTransformation (→3.2.2.1.1, *page 120*).

3.2.2.2

public Methoden

Names

3.2.2.2.1 unsigned int **giveDestinationHeight** (void) const
Höhe des Resultats 121

3.2.2.2.2 unsigned int **giveDestinationWidth** (void) const
Breite des Resultats 121

3.2.2.2.3 unsigned int **transform** (unsigned int x, unsigned int y) const
Omni->Panorama-Konversion 121

3.2.2.2.1

unsigned int **giveDestinationHeight** (void) const

Höhe des Resultats

Höhe des Resultats.

Return Value: Von den Dimensionen der Eingagsdaten abhängige Höhe des Resultats.

3.2.2.2.2

unsigned int **giveDestinationWidth** (void) const

Breite des Resultats

Breite des Resultats.

Return Value: Von den Dimensionen der Eingagsdaten abhängige Breite des Resultats.

3.2.2.2.3

unsigned int **transform** (unsigned int x, unsigned int y) const

Omni->Panorama-Konversion

Omni->Panorama-Konversion. Diese Methode führt eine quadratische Transformation aus Polar- in kartesische Koordinaten durch. Man beachte die nichtlineare Transformation und die mathematisch negative Abfolge der Winkel im Ergebnis. Beides ist durch die Methode der Gewinnung des Rundumblickes bei den vorhandenen Omnikameras begründet.

Return Value: linear kodierte Koordinate des Punktes in der Eingabedaten, dessen Inhalt an die angegebene Stelle des Resultats kopiert werden soll.

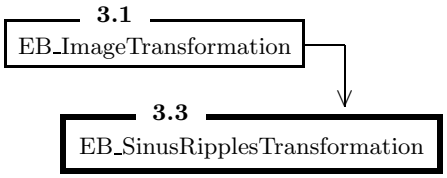
Parameters: `x` — x-Koordinate im Resultat.
`y` — y-Koordinate im Resultat.

3.3

class **EB_SinusRipplesTransformation** : public EB_ImageTransformation

Diese Klasse dient eher Effekten als klassischer Bildverarbeitung

Inheritance



Public Members

Diese Klasse dient eher Effekten als klassischer Bildverarbeitung. Der Grund ihrer Implementation war ein Versuch, zu veranschaulichen, wie mächtig der Mechanismus der Transformationsinterfaces ist. Die vorliegende Klasse implementiert den von Photoshop o.ä. her bekanten Ripples-Effekt dabei ist es so, daß verschiedene Sinusfunktionen überlagert werden können. Bei jeder dieser Sinusfunktionen ist es so, daß Phase, Frequenz, Amplitude und Winkel zur unteren Bildkante getrennt eingestellt werden können. Der visuelle Effekt dieser Transformation erscheint als abwechselnde Streckung und Stauchung des Bildinhaltes. Eine Frequenz von 1 bedeutet dabei, daß exakt ein gestreckter und ein gestauchter Bereich im Bild auftauchen. Hat die Phase den Wert 0, taucht der gestauchte Bereich ganz links im Bild auf. Die Amlitude oder das Displacement bestimmt, wie groß die maximale Stauchung oder Streckung sein darf. Der Winkel schließlich bestimmt, in welche Richtung sich die Wellen ausbreiten, ein Wert von 0.0 bestimmt hier die Ausbreitung von links entlang der unteren Bildkante.

Author: Jürgen ËL BOSSO"Key

3.3.1

Konstruktoren und Destruktor

Names

3.3.1.1

EB_SinusRipplesTransformation (void)

Parameterloser Konstruktor

Parameterloser Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_SinusRipplesTransformation (→3.3.1.1, *page 123*), die noch keine Funktionen definiert hat

3.3.1.2

EB_SinusRipplesTransformation (const EB_SinusRipplesTransformation &source)

Copy-Konstruktor

Copy-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_SinusRipplesTransformation (→3.3.1.1, *page 123*)als genaue Kopie der übergebenen Instanz.

3.3.2

public Methoden

Names

3.3.2.1

unsigned int **giveDestinationHeight** (void) const

Höhe des Resultats

Höhe des Resultats.

Return Value: Von den Dimensionen der Eingagsdaten abhängige Höhe des Resultats.

3.3.2.2

unsigned int **giveDestinationWidth** (void) const

Breite des Resultats

Breite des Resultats.

Return Value: Von den Dimensionen der Eingagsdaten abhängige Breite des Resultats.

3.3.2.3

void **addSinus** (double o, double f, double md, double alpha)

Funktion hinzufügen

Funktion hinzufügen. Diese Methode fügt eine weitere Funktion zu der Liste von unterschiedlich parametriereten Sinusfunktionen hinzu, die in ihrer Gesamtheit die Transformation bestimmen.

Parameters:
`o` — Hat die Phase den Wert 0.0, taucht der erste gestauchte Bereich ganz links im Bild auf.
`f` — Eine Frequenz von 1 bedeutet dabei, daß exakt ein gestreckter und ein gestauchter Bereich im Bild auftauchen.
`md` — Die Amlitude oder das Displacement bestimmt, wie groß die maximale Stauchung oder Streckung sein soll.
`alpha` — Der Winkel bestimmt, in welche Richtung sich die Wellen ausbreiten, ein Wert von 0.0 bestimmt hier die Ausbreitung von links entlang der unteren Bildkante.

3.3.2.4

void **clearSinusList** (void)

Funktionen entfernen

Funktionen entfernen. Diese Mathode entfernt alle bisher definierten Sinusfunktionen aus der Liste.

3.3.2.5

unsigned int **transform** (unsigned int x, unsigned int y) const

Ripples-Effekt

Ripples-Effekt. Diese Methode stellt den von Photoshop o.ä. her bekanten Ripples-Effekt nach. Es werden verschiedene Sinusfunktionen überlagert. Bei jeder dieser Sinusfunktionen ist es so, daß Phase, Frequenz, Amplitude und Winkel zur unteren Bildkante getrennt eingestellt werden können. Der visuelle Effekt dieser Transformation erscheint als abwechselnde Streckung und Stauchung des Bildinhaltes.

Return Value: linear kodierte Koordinate des Punktes in der Eingabedaten, dessen Inhalt an die angegebene Stelle des Resultats kopiert werden soll.

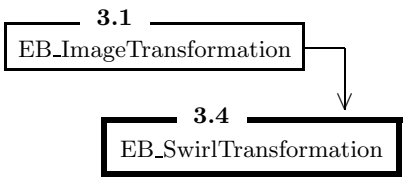
Parameters: `x` — x-Koordinate im Resultat.
`y` — y-Koordinate im Resultat.

3.4

class **EB_SwirlTransformation** : public EB_ImageTransformation

Diese Klasse realisiert eine bloße Effekttransformation

Inheritance



Public Members

Diese Klasse realisiert eine bloße Effekttransformation. Sie stellt den aus den verschiedensten Bildverarbeitungsprogrammen bekannten Swirleffekt nach. Dabei werden Bildinhalte nicht um den gleichen Winkel sondern um einen variablen Winkel gedreht. Der Winkel wird dabei aus dem Abstand der Pixel vom Drehzentrum bestimmt.

Author: Jürgen ËL BOSSO"Key

3.4.1

Konstruktoren und Destruktor

Names

3.4.1.1

EB_SwirlTransformation (void)

Parameterloser Konstruktor

Parameterloser Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_SwirlTransformation (→3.4.1.1, *page 126*), die noch keine Punkte definiert hat

3.4.2

public Methoden

Names

3.4.2.1

void **setCenter** (int x, int y)

Kreisring definieren

Kreisring definieren. Diese Methode definiert zusammen mit setInnerRadius (→3.4.2.2, *page 127*) und setBreadth (→3.4.2.3, *page 128*) die Region, in der Bildinhalte verändert werden. Hiermit wird der Mittelpunkt des zu transformierenden Kreisringes festgelegt.

Parameters:
`x` — x-Koordinate des Zentrums des Kreisringes.
`y` — y-Koordinate des Zentrums des Kreisringes.

3.4.2.2

void **setInnerRadius** (unsigned int r)

Kreisring definieren

Kreisring definieren. Diese Methode definiert zusammen mit setCenter (→3.4.2.1, *page 127*) und setBreadth (→3.4.2.3, *page 128*) die Region, in der Bildinhalte verändert werden. Hiermit wird der innere Radius des zu transformierenden Kreisringes festgelegt.

Parameters:
`r` — Innerer Radius des zu transformierenden Kreisringes.

3.4.2.3

void **setBreadth** (unsigned int b)

Kreisring definieren

Kreisring definieren. Diese Methode definiert zusammen mit setInnerRadius (→3.4.2.2, *page 127*) und setCenter (→3.4.2.1, *page 127*) die Region, in der Bildinhalte verändert werden. Hiermit wird die Breite und damit implizit der äußere Radius des zu transformierenden Kreisringes festgelegt.

Parameters: `b` — Breite des zu transformierenden Kreisringes.

3.4.2.4

void **setInnerAngle** (double a)

Winkel setzen

Winkel setzen. Diese Methode setzt den Winkel, der am inneren Radius des Kreisringes zur Rotation der Bildinhalte benutzt werden soll. Die Winkel werden linear zwischen dem mit dieser und dem mit setOuterAngle (→3.4.2.5, *page 128*) gesetzten Wert interpoliert.

Parameters: `a` — Am inneren Radius des Kreisringes wird dieser Wert benutzt.

3.4.2.5

void **setOuterAngle** (double a)

Winkel setzen

Winkel setzen. Diese Methode setzt den Winkel, der am äußeren Radius des Kreisringes zur Rotation der Bildinhalte benutzt werden soll. Die Winkel werden linear zwischen dem mit setOuterAngle (→3.4.2.5, *page 128*) und dem mit dieser Methode gesetzten Wert interpoliert.

Parameters: `a` — Am äußeren Radius des Kreisringes wird dieser Wert benutzt.

3.4.2.6

unsigned int **giveDestinationHeight** (void) const

Höhe des Resultats

Höhe des Resultats.

Return Value: Von den Dimensionen der Eingagsdaten abhängige Höhe des Resultats.

3.4.2.7

unsigned int **giveDestinationWidth** (void) const

Breite des Resultats

Breite des Resultats.

Return Value: Von den Dimensionen der Eingagsdaten abhängige Breite des Resultats.

3.4.2.8

unsigned int **transform** (unsigned int x, unsigned int y) const

Swirl-Operation

Swirl-Operation. Diese Methode führt die aus den verschiedenen Bildbearbeitungsprogrammen bekannte Swirl-Operation aus. Dabei werden die Inhalte eines Kreisringeses des Bildes abhängig vom Abstand vom Mittelpunkt dieses Kreisringes mit variablen Winkeln rotiert.

Return Value: linear kodierte Koordinate des Punktes in der Eingabedaten, dessen Inhalt an die angegebene Stelle des Resultats kopiert werden soll.

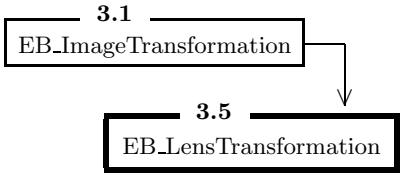
Parameters: `x` — x-Koordinate im Resultat.
`y` — y-Koordinate im Resultat.

3.5

class **EB_LensTransformation** : public EB_ImageTransformation

Diese Klasse implementiert den allgemein bekannten Linsenoperator auf Bildern

Inheritance



Public Members

Protected Members

int	**cx**	*x-Koordinate des Bereichsmittelpunktes.*
int	**cy**	*x-Koordinate des Bereichsmittelpunktes.*
double	**scalefactor**	*Maximaler Skalierungsfaktor.*
double	**scalerest**	*Interner Faktor.*
double	**maxradius**	*Radius des beeinflussten Bereiches.*
double	**effectiveness**	*Effektivität.*

Diese Klasse implementiert den allgemein bekannten Linsenoperator auf Bildern. Dabei wird der Bildinhalt an einer bestimmten Koordinate mit einem festen Faktor skaliert. Dieser Faktor variiert mit der Entfernung von diesem Punkt innerhalb eines kreisförmigen Bereichsmit gegebenem Radius. Bereiche Außerhalb des Radius werden nicht beeinflußt.

Author: Jürgen ËL BOSSO"Key

3.5.1

Konstruktoren und Destruktor

Names

3.5.1.1

EB_LensTransformation (void)

Parameterloser Konstruktor

Parameterloser Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_LensTransformation (→3.5.1.1, *page 131*). Dabei werden die Parameter so eingestellt, daß amBild nichts geändert wird.

3.5.2

public Methoden

Names

3.5.2.1

unsigned int **giveDestinationHeight** (void) const

Höhe des Resultats

Höhe des Resultats. Diese Methode liefert die Höhe des Resultats der Transformation.

Return Value: Von den Dimensionen der Eingagsdaten abhängige Höhe des Resultats.

3.5.2.2

unsigned int **giveDestinationWidth** (void) const

Breite des Resultats

Breite des Resultats. Diese Methode liefert die Breite des Resultats der Transformation.

Return Value: Von den Dimensionen der Eingagsdaten abhängige Breite des Resultats.

3.5.2.3

void **setMaxScaling** (double s)

Maximum der Verzerrung

Maximum der Verzerrung. Mittels dieser Methode wird die maximale Verzerrung in der Mitte des kreisförmigen Bereichs festgelegt.

Parameters: `s` — Faktor, mit dem die Mitte skaliert werden soll.

3.5.2.4

void **setMidPoint** (int x, int y)

Mittelpunkt festlegen

Mittelpunkt festlegen. Diese Methode legt den Mittelpunkt des beeinflussten Bereiches fest.

Parameters: `x` — x-Koordinate des Mittelpunkts.
`y` — y-Koordinate des Mittelpunkts.

3.5.2.5

void **setRadius** (double r)

Radius festlegen

Radius festlegen. Diese Methode legt den Radius des beeinflussten Bereiches fest.

Parameters: `r` — Radius des beeinflussten Bereiches.

3.5.2.6

void **setEffectiveness** (double r)

Effektivität festlegen

Effektivität festlegen. Diese Methode legt fest, wie stark der Skalierungsfaktor bis zur Grenze des beeinflussten Bereichs abnimmt. Ist der Faktor gleich 1.0, so ist der Skalierungsfaktor an dieser Grenze genau 1.

Parameters: `r` — Je näher dieser Wert der 0 kommt, desto näher liegt der Skalierungsfaktor an der Grenze des Bereichs an dem in der Mitte.

3.5.2.7

unsigned int **transform** (unsigned int x, unsigned int y) const

Transformation

Transformation. Diese Transformation implementiert eine ortsabhängige Skalierung. Die Skalierung findet in einem kreisförmigen Bereich statt. Der Skalierungsfaktor variiert über den Radius dieses Bereiches.

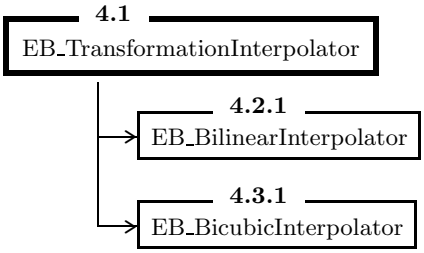
4 Interpolatoren

Names

4.1 class EB_TransformationInterpolator

Diese Klasse stellt eine Schnittstelle zu beliebigen Interpolatoren bereit

Inheritance



Public Members

	giveIndices (void)	*Positionen ermitteln, Diese Methode liefert einen Vektor, der nach Aufruf der Methode EB_TransformationInterpolator::calculate (→4.1.3,* page 136*) die Positionen der für eine spezifische Koordinate benötigten Nachbarn enthält* 137
4.1.6	EB_Vector <float> & **giveFactors** (void)	*Gewichte ermitteln, Diese Methode liefert einen Vektor, der nach Aufruf der Methode EB_TransformationInterpolator::calculate (→4.1.3,* page 136*) die Gewichte der für eine spezifische Koordinate benötigten Nachbarn enthält* 137

Protected Members

unsigned int **members**	*Maximale Anzahl an Nachbarn, die ein bestimmtes Interpolationsschema benötigt.*
EB_Vector <unsigned int> **indices**	*Vektor, in dem die jeweiligen Positionen der zur Interpolation benutzten Pixel liegen.*
EB_Vector <float> **factors**	*Vektor, in dem die jeweiligen Gewichte der zur Interpolation benutzten Pixel liegen.*
unsigned int **width**	*Breite des bearbeiteten Bildes (wird für die lineare Kodierung benötigt)*
unsigned int **height**	*Höhe des bearbeiteten Bildes (wird für die lineare Kodierung benötigt)*

Diese Klasse stellt eine Schnittstelle zu beliebigen Interpolatoren bereit. Instanzen von von dieser Klasse abgeleiteten Klassen benutzt zum Beispiel die Methode EB_Image::transform (→1.3.3.26, *page 59*) der Klasse EB_Image (→1.3, *page 39*) um die Ergebnisse von Koordinatentransformationen auf Bildern zu verbessern. Die Schnittstelle ist so konzipiert, daß hier noch gar keine Verarbeitung stattfindet. Hier wird lediglich das Interface für konkrete Interpolationsalgorithmen definiert. Darüber hinaus stellt diese Klasse einige administrative Methoden bereit, die viele Interpolatoren benötigen oder die die Interpolation beschleunigen. Die Interpolation wird auch von den von dieser Klasse abgeleiteten Klassen nicht auf dem Bild vollzogen. Es erfolgt lediglich die Berechnung der Koordinaten der in die Interpolation einzubezieheneden Koordinaten sowie der Gewichte, mit dem die einzelnen Pixel in das Ergebnis eingehen. Das Ergebnis sind zwei Vektoren, auf die der Aufrufer danach zurückgreifen kann.

Author: Jürgen EL BOSSO Key

4.1.1

EB_TransformationInterpolator (unsigned int samplecount)

Konstruktor

Konstruktor. Dieser Konstruktor reserviert lediglich den zur Speicherung der Koordinaten und Gewichte nötigen Speicher.

Parameters: `samplecount` — Maximale Anzahl der zur Interpolation eines Pixels herangezogenen Nachbarn. Diese Anzahl legt die Länge der Vektoren fest.

4.1.2

void **setSourceDimensions** (unsigned int w, unsigned int h)

Eingangsdimensionen setzen

Eingangsdimensionen setzen. Mit dieser Methode werden dee aktuellen Instanz die Dimensionen der zu transformierenden Daten bekanntgegeben.

Parameters: `w` — Breite der Eingangsdaten.
`h` — Höhe der Eingangsdaten.

4.1.3

virtual void **calculate** (float x, float y)

Interpolation berechnen

Interpolation berechnen. Diese Methode ist pur virtuell und muß von abgeleiteten Klassen überladen werden. Dabei sollte die Funktion aus den gebrochenen Pixelkoordinaten die Koordinaten aller zur Interpolation nötigen Nachbarn berechnen und linear kodiert im Vektor indices (→ *page 135*) ablegen. Die zugehörigen Gewichte müssen im entsprechenden Element des Vektorsfactors (→ *page 135*) abgelegt werden.

Parameters: `x` — gebrochenzahlige Pixelkoordinate.
`y` — gebrochenzahlige Pixelkoordinate.

4.1.4

unsigned int **giveSampleCount** (void)

Interpolatormächtigkeit

Interpolatormächtigkeit. Dahinter verbirgt sich letztlich nur die Aussage, wie viele Pixel zur Interpolation herangezogen werden sollen.

Return Value: s Anzahl der zur Interpolation benutzten Nachbarn.

4.1.5

EB_Vector <unsigned int> & **giveIndices** (void)

Positionen ermitteln, Diese Methode liefert einen Vektor, der nach Aufruf der Methode EB_TransformationInterpolator::calculate (→4.1.3, page 136*) die Positionen der für eine spezifische Koordinate benötigten Nachbarn enthält*

Return Value: Referenz auf einen Vektor, der die Positionen der zur Interpolation zu benutzenden Pixel linear kodiert enthält

4.1.6

EB_Vector <float> & **giveFactors** (void)

Gewichte ermitteln, Diese Methode liefert einen Vektor, der nach Aufruf der Methode EB_TransformationInterpolator::calculate (→4.1.3, page 136*) die Gewichte der für eine spezifische Koordinate benötigten Nachbarn enthält*

Return Value: Referenz auf einen Vektor, der die Gewichte der zur Interpolation zu benutzenden Pixel linear kodiert enthält

4.2

Bilineare Interpolatoren

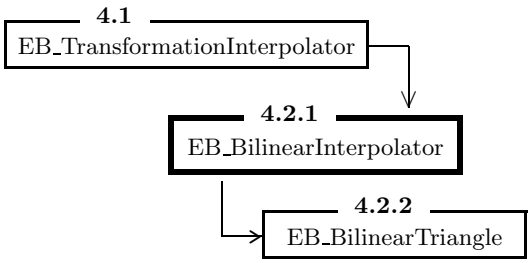
Names

4.2.1

class **EB_BilinearInterpolator** : public EB_TransformationInterpolator

Diese Klasse ist die Basisklasse für die verschidenen bilinearen Interpolationsverfahren

Inheritance



Public Members

Diese Klasse ist die Basisklasse für die verschidenen bilinearen Interpolationsverfahren. Sie reserviert den für die Gewichte und Positionen der Nachbarpixel benötigten Speicher und berechnet die Positionen der Nachbarn. Die Berechnung der Gewichte ist die Aufgabe von von dieser Klasse abgeleiteten Klassen.

Author: Jürgen EL BOSSO Key

4.2.1.1

EB_BilinearInterpolator (void)

Konstruktor

Konstruktor. Dieser Konstruktor übergibt die Menge an benötigten Positionen und Gewichten an den Basisklassenkonstruktor.

4.2.1.2

void **calculate** (float x, float y)

Bilineare Interpolation

Bilineare Interpolation. Diese Methode berechnet lediglich die Positionen der zur Interpolation benutzten Nachbarn. Zur Berechnung der Gewichte wird die virtuelle Methode EB_BilinearInterpolator::BilinearFunc (→4.2.1.3, *page 139*) aufgerufen, die in von dieser Klasse abgeleiteten Klassen überladen werden muß.

Parameters: `x` — gebrochenzahlige Pixelkoordinate.
`y` — gebrochenzahlige Pixelkoordinate.

4.2.1.3

virtual float **BilinearFunc** (float x)

Berechnung der Gewichte für die Interpolation

Berechnung der Gewichte für die Interpolation. Diese Methode muß in abgeleiteten Klassen für spezielle bilineare Interpolatoren überladen werden.

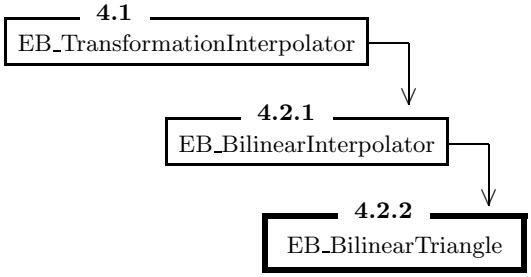
Return Value: Gewicht für die Interpolation.
Parameters: `x` — Parameter für die jeweils benutzte Funktion.

4.2.2

class **EB_BilinearTriangle** : public EB_BilinearInterpolator

Diese Klasse realisiert eine Interpolation mittels der Dreiecksfunktion

Inheritance



Public Members

		EB_BilinearTriangle (void) *Konstruktor*	
	virtual	**~EB_BilinearTriangle** (void) *Destruktor*	
4.2.2.1	float	**BilinearFunc** (float x)	*Berechnung der Gewichte für die Interpolation mittels der Dreiecksfuktion* 140

4.2.2.1

float **BilinearFunc** (float x)

Berechnung der Gewichte für die Interpolation mittels der Dreiecksfuktion

Return Value: Gewicht für die Interpolation.
Parameters: `x` — Argument der Funktion.

4.3

Bikubische Interpolatoren

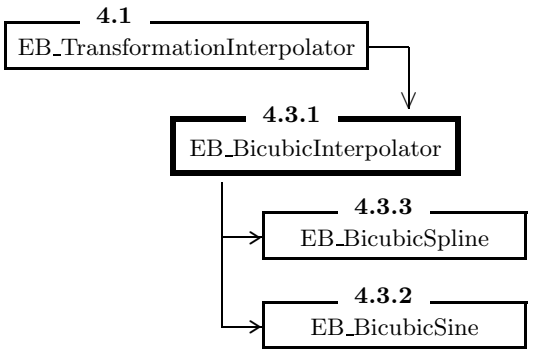
Names

4.3.1

class **EB_BicubicInterpolator** : public EB_TransformationInterpolator

Diese Klasse ist die Basisklasse für die verschidenen bikubischen Interpolationsverfahren

Inheritance



Public Members

Destruktor.

Diese Klasse ist die Basisklasse für die verschidenen bikubischen Interpolationsverfahren. Sie reserviert den für die Gewichte und Positionen der Nachbarpixel benötigten Speicher und berechnet die Positionen der Nachbarn. Die Berechnung der Gewichte ist die Aufgabe von von dieser Klasse abgeleiteten Klassen.

Author: Jürgen EL BOSSO Key

4.3.1.1

EB_BicubicInterpolator (void)

Konstruktor

Konstruktor. Dieser Konstruktor übergibt die Menge an benötigten Positionen und Gewichten an den Basisklassenkonstruktor.

4.3.1.2

void **calculate** (float x, float y)

Bikubische Interpolation

Bikubische Interpolation. Diese Methode berechnet lediglich die Positionen der zur Interpolation benutzten Nachbarn. Zur Berechnung der Gewichte wird die virtuelle Methode EB_BicubicInterpolator::BicubicFunc (→4.3.1.3, *page 143*) aufgerufen, die in von dieser Klasse abgeleiteten Klassen überladen werden muß.

Parameters: `x` — gebrochenzahlige Pixelkoordinate.
`y` — gebrochenzahlige Pixelkoordinate.

4.3.1.3

virtual float **BicubicFunc** (float x)

Berechnung der Gewichte für die Interpolation

Berechnung der Gewichte für die Interpolation. Diese Methode muß in abgeleiteten Klassen für spezielle bikubische Interpolatoren überladen werden.

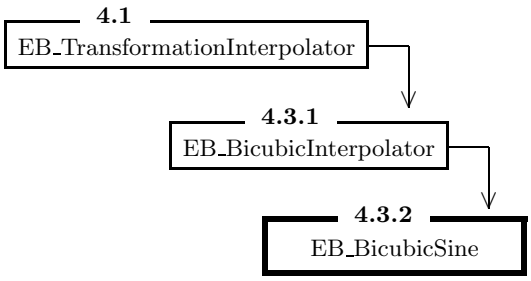
Return Value: Gewicht für die Interpolation.
Parameters: `x` — Parameter für die jeweils benutzte Funktion.

4.3.2

class **EB_BicubicSine** : public EB_BicubicInterpolator

Diese Klasse realisiert eine Interpolation mittels eines bikubischen Sinus

Inheritance



Public Members

		EB_BicubicSine (void) *Konstruktor*	
	virtual	**~EB_BicubicSine** (void) *Destruktor*	
4.3.2.1	float	**BicubicFunc** (float x) *Berechnung der Gewichte für die Interpolation mittels eines Sinus*	144

4.3.2.1

float **BicubicFunc** (float x)

Berechnung der Gewichte für die Interpolation mittels eines Sinus

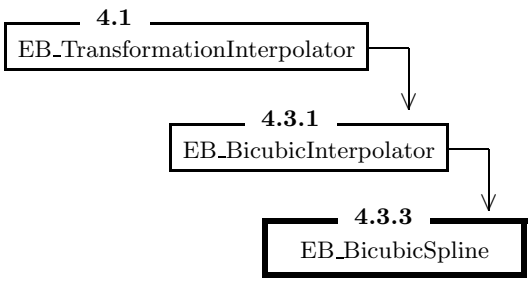
Return Value: Gewicht für die Interpolation.
Parameters: `x` — Argument der Funktion.

4.3.3

class **EB_BicubicSpline** : public EB_BicubicInterpolator

Diese Klasse realisiert eine Interpolation mittels des bikubischen Splineverfahrens

Inheritance



Public Members

		EB_BicubicSpline (void)	*Konstruktor*
	virtual	**~EB_BicubicSpline** (void)	*Destruktor*
4.3.3.1	float	**BicubicFunc** (float x)	*Berechnung der Gewichte für die Interpolation mittels einer Spline* 145
	double	**fac1**	*Faktor der Interpolationsfunktion.*
	double	**fac2**	*Faktor der Interpolationsfunktion.*

4.3.3.1

float **BicubicFunc** (float x)

Berechnung der Gewichte für die Interpolation mittels einer Spline

Return Value: Gewicht für die Interpolation.
Parameters: `x` — Argument der Funktion.

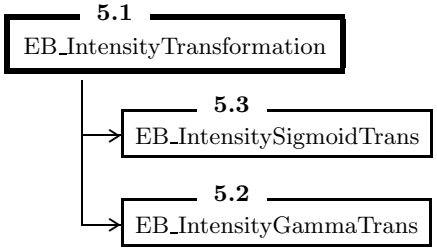
5 Pixeloperationen (Intensitätstransformationen)

Names

5.1 class EB_IntensityTransformation

Diese Klasse stellt eine Schnittstelle für einwertige Funktionen dar, die zum Beispiel auch zu Intensitätstransformationen in Bilder benutzt werden können (Gammakorrektur o

Inheritance



Public Members

Diese Klasse stellt eine Schnittstelle für einwertige Funktionen dar, die zum Beispiel auch zu Intensitätstransformationen in Bilder benutzt werden können (Gammakorrektur o.ä.).

Author: Jürgen ËL BOSSO"Key

5.1.1

Konstruktoren und Destruktor

Names

5.1.1.1 **EB_IntensityTransformation** (void)
Parameterloser Konstruktor 147

virtual ~ **EB_IntensityTransformation** (void)
Destruktor.

5.1.1.1

EB_IntensityTransformation (void)

Parameterloser Konstruktor

Parameterloser Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_IntensityTransformation (→5.1.1.1, *page 147*).

5.1.2

public Methoden

Names

5.1.2.1 virtual float **transform** (float intensity) const
Transformation 148

5.1.2.1

virtual float **transform** (float intensity) const

Transformation

Transformation. Diese Methode ist das Herzstück der Klasse. Sie ist pur virtuell und muß von abgeleiteten Klassen überladen werden. Hier muß die entsprechende, die Transformation beschreibende Funktion implementiert werden.

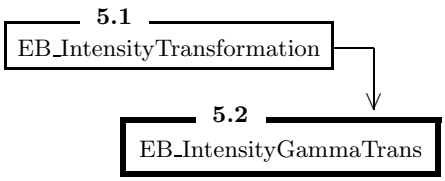
Return Value: Resultat der Transformationsfunktion.
Parameters: `intensity` — Argument der Transformationsfunktion.

5.2

class **EB_IntensityGammaTrans** : public EB_IntensityTransformation

Diese Klasse berechnet die Funktion der Gammakorrektur

Inheritance



Public Members

Protected Members

float	**factor**	*Kehrwert des Parameters der Gammafunktion.*

Author: Jürgen ËL BOSSO"Key

5.2.1

Konstruktoren und Destruktor

Names

5.2.1.1

EB_IntensityGammaTrans (void)

Parameterloser Konstruktor

Parameterloser Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_IntensityGammaTrans (→5.2.1.1, *page 149*).

5.2.2

public Methoden

Names

5.2.2.1

void setGamma (float f)

Parameter setzen

Parameters: `gamma` — Parameter für die Gamma-Korrektur: `>1` heller, `<1` dunkler

5.2.2.2

virtual float **transform** (float intensity) const

Gammakorrektur

Gammakorrektur. Zur Transformation wird angenommen, dass die Argumente im Intervall 0,1 liegen.

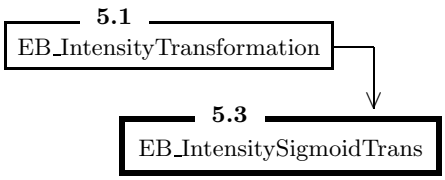
Return Value: Ergebnis der Gammafunktion.
Parameters: `intensity` — Argument für die Gammafunktion.

5.3

class **EB_IntensitySigmoidTrans** : public EB_IntensityTransformation

Diese Klasse berechnet die Sigmoidfunktion

Inheritance



Public Members

Protected Members

float	**xstretch**	*Streckung in x-Richtung.*
float	**ystretch**	*Streckung in y-Richtung.*
float	**xshift**	*Verschiebung in x-Richtung.*
float	**yshift**	*Verschiebung in y-Richtung.*

Diese Klasse berechnet die Sigmoidfunktion. Sie kann über entsprechende Parameter in x- und y-Richtung verschoben, gestaucht und gestreckt werden.

Author: Jürgen ËL BOSSO"Key

5.3.1

Konstruktoren und Destruktor

Names

5.3.1.1

EB_IntensitySigmoidTrans (void)

Parameterloser Konstruktor

Parameterloser Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_IntensitySigmoidTrans (→5.3.1.1, *page 151*).

5.3.2

public Methoden

Names

5.3.2.1

void **setXShift** (float s)

Parameter setzen

Parameters: `Verschiebung` — in x-Richtung

5.3.2.2

void **setYShift** (float s)

Parameter setzen

Parameters: `Verschiebung` — in y-Richtung

5.3.2.3

void **setXStretch** (float s)

Parameter setzen

Parameters: `Streckung` — in x-Richtung

5.3.2.4

void **setYStretch** (float s)

Parameter setzen

Parameters: `Streckung` — in y-Richtung

5.3.2.5

virtual float **transform** (float intensity) const

Sigmoidfunktion

Return Value: Ergebnis der Sigmoidfunktion.
Parameters: `intensity` — Argument für die Sigmoidfunktion.

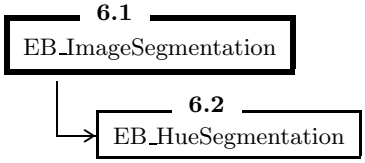
6 Farbsegmentationsoperationen

Names

6.1 class EB_ImageSegmentation

Diese Klasse stellt eine Schnittstelle für die farbbasierte Segmentation dar

Inheritance



Public Members

Protected Members

EB_PixelDescriptor
color *Farbe, gegen die die Distanz berechnet werden soll.*

Diese Klasse stellt eine Schnittstelle für die farbbasierte Segmentation dar. Diese Segmentation geschieht nicht in dieser Klasse, sie dient lediglich dazu, den Abstand zwischen zwei Farben zu berechnen. Dabei ist die konkrete Wahl des Abstandmaßes in einer abgeleiteten Klasse zu treffen.

Author: Jürgen ËL BOSSO"Key

6.1.1 Konstruktoren und Destruktor

Names

6.1.1.1 **EB_ImageSegmentation** (void)
Parameterloser Konstruktor 155

virtual ~ **EB_ImageSegmentation** (void)
Destruktor.

6.1.1.1 EB_ImageSegmentation (void)

Parameterloser Konstruktor

Parameterloser Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_ImageSegmentation (→6.1.1.1, *page 155*).

6.1.2 public Methoden

Names

6.1.2.1 void **setColor** (const EB_PixelDescriptor &compcolor)
Vergleichsfarbe setzen 156

6.1.2.2 virtual float **computeDistance** (EB_PixelDescriptor &p) const
Distanzberechnung 156

6.1.2.1

void **setColor** (const EB_PixelDescriptor &compcolor)

Vergleichsfarbe setzen

Vergleichsfarbe setzen. Diese Methode legt die Vergleichsfarbe zum späteren Berechnen der Distanz fest.

6.1.2.2

virtual float **computeDistance** (EB_PixelDescriptor &p) const

Distanzberechnung

Distanzberechnung. Diese Methode ist das Herzstück der Klasse. Sie ist pur virtuell und muß von abgeleiteten Klassen überladen werden. Die Methode muß mit einem entsprechenden Distanzmaß den Abstand zwischen der übergebenen und der Vergleichsfarbe berechnen und dieses Resultat auf das Intervall zwischen 0.0 und 1.0 normiert zurückgeben.

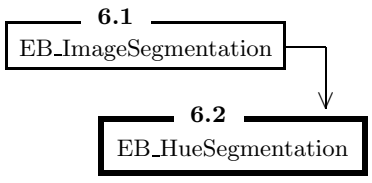
Return Value: Abstand von der Vergleichsfarbe.
Parameters: `p` — Farbe, für die die Differenz berechnet werden soll.

6.2

class **EB_HueSegmentation** : public EB_ImageSegmentation

Diese Klasse berechnet den Abstand der H-Komponente des HSI-Farbraumes aus dem übergebenen Pixel im RGB-Farbraum von einem Referenzwert

Inheritance



Public Members

Diese Klasse berechnet den Abstand der H-Komponente des HSI-Farbraumes aus dem übergebenen Pixel im RGB-Farbraum von einem Referenzwert. Dabei wird eine minimale Sättigung in Rechnung gestellt, die die Farbe mindestens haben muß. Liegt der Sättigungswert darunter, wird die Differenz pauschal auf das Maximum gesetzt

Author: Jürgen ËL BOSSO"Key

6.2.1

Konstruktoren und Destruktor

Names

6.2.1.1

EB_HueSegmentation (void)

Parameterloser Konstruktor

Parameterloser Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse EB_HueSegmentation (→6.2.1.1, *page 157*). Der Referenzwert und die Sättigungsschwelle werden dabei jeweils auf 0.0 gesetzt.

6.2.2

public Methoden

Names

6.2.2.1

void **setReferenceH** (float h)

Referenzwert für H setzen

Referenzwert für H setzen. Diese Methode setzt den Wert, zu dem die Differenz des jeweiligen H-Wertes berechnet werden soll.

Parameters: `h` — Referenzwert, wird auf das Intervall zwischen 0.0 und 1.0 beschränkt

6.2.2.2

void **setThresholdS** (float s)

Schwelwert für Ssetzen

Schwelwert für Ssetzen. Diese Methode setzt den minimalen Sättigungswert. Unterschreitet eine Farbe diesen Wert, wird der Abstand vom Referenzwert pauschal mit dem Maximum, 1.0, bewertet.

Parameters: `s` — Schwellwert, wird auf das Intervall zwischen 0.0 und 1.0 beschränkt

6.2.2.3

float **computeDistance** (EB_PixelDescriptor &p) const

Distanzberechnung

Distanzberechnung. Diese Methode berechnet die H-Komponente der Farbe und interpretiert dafür die ersten drei Bestandteile der übergebenen Farbe als Rot, Grün und Blau im RGB-Farbraum. Ist die Sättigung nicht zu gering, wird anschließend die Differenz zwischen dem errechneten und dem Referenzwert gebildet und zurückgegeben.

Return Value: Abstand des berechneten H-Wertes vom Referezwert.
Parameters: `p` — Farbe, für die die Differenz berechnet werden soll.

7

Unterstützte Bildformate

Names

7.1

Modul zum Laden und Speichern im PNM-Format.

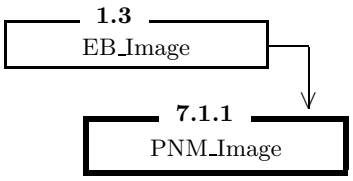
Names

7.1.1

class **PNM_Image** : public EB_Image

Diese Klasse dient zum Laden und Speichern von Bilddaten

Inheritance



Public Members

Protected Members

int	**pnmtype**	*Typ der Daten.*
int	**charmax**	*Maximaler Intensitätswert.*

Diese Klasse dient zum Laden und Speichern von Bilddaten. Dabei ist es nur möglich, Bilder mit einem oder drei Bändern zu speichern. Ein Bild mit einem Band wird dabei beim Speichern als Grauwertbild im PGM-Format und eins mit drei Bändern als Darstellung im RGB-Farbraum im PPM-Format betrachtet. Beim Laden ergibt ein Grauwertbild im PGM-Format entsprechend eine Instanz mit einem Band und ein Farbbild im PPM-Format eine Instanz mit drei Bändern.

Author: Jürgen EL BOSSO Key

7.1.1.1

Konstruktoren und Destruktor

Names

virtual ~ **PNM_Image** () *Destruktor.*

7.1.1.1.1

PNM_Image (void)

Default-Konstruktor

Default-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse PNM_Image (→7.1.1.1.1, *page 161*) mit einer Breite und Höhe von eins, einem Band und dem Dynamikbereich von 0.0 bis 1.0.

7.1.1.1.2

PNM_Image (unsigned int x, unsigned int y, unsigned int usedbands, float color = -1.0, float min = 0.0, float max = 1.0)

Konstruktor

Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse PNM_Image (→7.1.1.1.1, *page 161*).

Parameters:

`x` — Breite des Bildes.
`y` — Höhe des Bildes.
`usedbands` — Anzahl an Bändern im Bild.
`color` — Intensitätswert, mit dem alle Pixel aller Bänder initialisiert werden. Liegt dieser Wert außerhalb des durch min und max bestimmten Intervalls, findet keine Initialisierung statt. Voreinstellung ist -1.0
`min` — Minimaler Intensitätswert in jedem Band. Voreinstellung ist 0.0
`max` — Maximaler Intensitätswert in jedem Band. Voreinstellung ist 1.0

7.1.1.1.3

PNM_Image (const EB_Image & Image)

Copy-Konstruktor

Copy-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse PNM_Image (→7.1.1.1.1, *page 161*) als Kopie von der übergebenen Instanz.

Parameters:

`Image` — Instanz, von der die Kopie erzeugt wird.

7.1.1.1.4

PNM_Image (const char *name)

File-Konstruktor

File-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der KlassePNM_Image (→7.1.1.1.1, *page 161*), indem er versucht, den Inhalt der Datei mit dem angegebenen Namen als im PPM- oder PGM-Format gespeicherte Bilddaten zu interpretieren.

Parameters: `name` — Name einer Datei, aus der die Bilddaten gelesen werden sollen.

7.1.1.2

public Methoden

Names

7.1.1.2.1

virtual EB_Image& **load** (const char *name) throw(EBIOutOfMemoryEXP, EBIPNMLoaderEXP, EBICouldNotLoadEXP, EBICouldNotOpenLoadFileEXP)

Bild laden

Bild laden. Diese Methode versucht, den Inhalt der Datei mit dem angegebenen Namen als im PPM- oder PGM-Format gespeicherte Bilddaten zu interpretieren und die Daten entsprechend in die aktuelle Instanz einzutragen. Kann aus der angegebenen Datei nicht gelesen werden, wird eine Exception geworfen.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `name` — Name einer Datei, aus der die Bilddaten gelesen werden sollen.

7.1.1.2.2

virtual void **save** (const char *name) throw(EBIWrongNumberOfBandsEXP, EBICouldNotOpenSaveFileEXP)

Bild speichern

Bild speichern. Diese Methode speichert die Bilddaten in der aktuellen Instanz abhängig von der Anzahl der Bänder im PPM- oder PGM-Format in eine Datei mit dem angegebenen Namen. Hat die aktuelle Instanz nicht genau ein oder drei Bänder oder kann eine Datei mit dem angegebenen Namen nicht geschrieben werden, wird eine Exception geworfen

Parameters: `name` — Name einer Datei, in der die Daten gespeichert werden sollen.

7.2

extern C extern **Modul zum Laden und Speichern im JPEG-Format.**

Names

#define **DEFAULTQUALITY**
Voreinstellung für die Qualität beim Speichern.

7.2.1 class **JPG_Image** : public EB_Image
Diese Klasse dient zum Laden und Speichern von Bilddaten 163

7.2.1

class **JPG_Image** : public EB_Image

Diese Klasse dient zum Laden und Speichern von Bilddaten

Inheritance

Public Members

Protected Members

unsigned int	**quality**	*Die bei der Kompression zu benutzende Qualität.*
bool	**corrupted**	*Flag zeigt an, ob die Daten beim Laden korrumpiert wurden (true)*

Diese Klasse dient zum Laden und Speichern von Bilddaten. Dabei ist es nur möglich, Bilder mit einem oder drei Bändern zu speichern. Ein Bild mit einem Band wird dabei beim Speichern als Grauwertbild und eins mit drei Bändern als Darstellung im RGB-Farbraum betrachtet. Beim Laden ergibt ein Grauwertbild entsprechend eine Instanz mit einem Band und ein Farbbild eine Instanz mit drei Bändern. Von den verschiedenen Möglichkeiten, auf das Ergebnis der Kompression Einfluß zu nehmen, wurde in der vorliegenden Klasse nur die prozentuale Einstellung der Qualität berücksichtigt.

Author: Jürgen EL BOSSO Key

7.2.1.1

Konstruktoren und Destruktor

Names

7.2.1.1.1

JPG_Image (void)

Default-Konstruktor

Default-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse JPG_Image (→7.2.1.1.1, *page 165*) mit einer Breite und Höhe von eins, einem Band und dem Dynamikbereich von 0.0 bis 1.0.

7.2.1.1.2

JPG_Image (unsigned int x, unsigned int y, unsigned int usedbands, float color = -1.0, float min = 0.0, float max = 1.0)

Konstruktor

Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse JPG_Image (→7.2.1.1.1, *page 165*).

Parameters:

`x` — Breite des Bildes.
`y` — Höhe des Bildes.
`usedbands` — Anzahl an Bändern im Bild.
`color` — Intensitätswert, mit dem alle Pixel aller Bänder initialisiert werden. Liegt dieser Wert außerhalb des durch min und max bestimmten Intervalls, findet keine Initialisierung statt. Voreinstellung ist -1.0
`min` — Minimaler Intensitätswert in jedem Band. Voreinstellung ist 0.0
`max` — Maximaler Intensitätswert in jedem Band. Voreinstellung ist 1.0

7.2.1.1.3

JPG_Image (const EB_Image & Image)

Copy-Konstruktor

Copy-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse JPG_Image (→7.2.1.1.1, *page 165*) als Kopie von der übergebenen Instanz. Die Qualität, die beim Speichern des Inhaltes der aktuellen Instanz benutzt wird, wird auf den Wert DEFAULTQUALITY (→ *page 163*) gesetzt.

Parameters: `Image` — Instanz, von der die Kopie erzeugt wird.

7.2.1.1.4

JPG_Image (const char *name)

File-Konstruktor

File-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der KlasseJPG_Image (→7.2.1.1.1, *page 165*), indem er versucht, den Inhalt der Datei mit dem angegebenen Namen als im JPEG-Format gespeicherte Bilddaten zu interpretieren.

Parameters: `name` — Name einer Datei, aus der die Bilddaten gelesen werden sollen.

7.2.1.2

public Methoden

Names

7.2.1.2.1

void **setCorrupted** (void)

Daten als korrupt markieren

Daten als korrupt markieren. Diese Methode wird von der Methode JPG_Image::load (→7.2.1.2.3, *page 167*) aufgerufen, wenn die Daten, die geladen wurden in irgendeiner Weise korrumpiert waren.

7.2.1.2.2

void **setQuality** (unsigned int newq)

Qualität setzen

Qualität setzen. Diese Methode ermöglicht es, die beim Speichern benutzte Qualität und somit die Kompressionsrate zu bestimmen.

Parameters: `newq` — Wert, der die Qualität bestimmt. Ist der Wert größer als 100, wird 100 benutzt.

7.2.1.2.3

virtual EB_Image& **load** (const char *name) throw(EBICouldNotOpenLoadFileEXP, EBIJPGLoaderEXP, EBIJPGPrematureEndOfDataEXP)

Bild laden

Bild laden. Diese Methode versucht, den Inhalt der Datei mit dem angegebenen Namen als im JPEG-Format gespeicherte Bilddaten zu interpretieren und die Daten entsprechend in die aktuelle Instanz einzutragen. Kann aus der angegebenen Datei nicht gelesen werden, wird eine Exception geworfen.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `name` — Name einer Datei, aus der die Bilddaten gelesen werden sollen.

7.2.1.2.4

virtual void **save** (const char *name) throw(EBIWrongNumberOfBandsEXP, EBICouldNotOpenSaveFileEXP)

Bild speichern

Bild speichern. Diese Methode speichert die Bilddaten in der aktuellen Instanz im JPEG-Format in eine Datei mit dem angegebenen Namen. Hat die aktuelle Instanz nicht genau drei Bänder oder kann eine Datei mit dem angegebenen Namen nicht geschrieben werden, wird eine Exception geworfen

Parameters: `name` — Name einer Datei, in der die Daten gespeichert werden sollen.

7.3

extern **Modul zum Laden und Speichern im PNG-Format.**

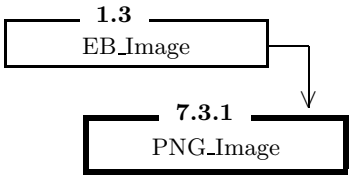
Names

7.3.1

class **PNG_Image** : public EB_Image

Diese Klasse dient zum Laden und Speichern von Bilddaten

Inheritance



Public Members

Diese Klasse dient zum Laden und Speichern von Bilddaten. Dabei ist es nur möglich, Bilder mit einem, drei oder vier Bändern zu speichern. Ein Bild mit einem Band wird dabei beim Speichern als Grauwertbild, eins mit drei Bändern als Darstellung im RGB-Farbraum und eins mit vier Bändern als RGB-Bild mit Alphakanal betrachtet. Beim Laden ergibt ein Grauwertbild entsprechend eine Instanz mit einem Band und ein Farbbild eine Instanz mit drei Bändern.

Author: Jürgen EL BOSSO Key

7.3.1.1

Konstruktoren und Destruktor

Names

virtual ~ **PNG_Image** () *Destruktor.*

7.3.1.1.1

PNG_Image (void)

Default-Konstruktor

Default-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse PNG_Image (→7.3.1.1.1, *page 169*) mit einer Breite und Höhe von eins, einem Band und dem Dynamikbereich von 0.0 bis 1.0.

7.3.1.1.2

PNG_Image (unsigned int x, unsigned int y, unsigned int usedbands, float color = -1.0, float min = 0.0, float max = 1.0)

Konstruktor

Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse PNG_Image (→7.3.1.1.1, *page 169*).

Parameters:
`x` — Breite des Bildes.
`y` — Höhe des Bildes.
`usedbands` — Anzahl an Bändern im Bild.
`color` — Intensitätswert, mit dem alle Pixel aller Bänder initialisiert werden. Liegt dieser Wert außerhalb des durch min und max bestimmten Intervalls, findet keine Initialisierung statt. Voreinstellung ist -1.0
`min` — Minimaler Intensitätswert in jedem Band. Voreinstellung ist 0.0
`max` — Maximaler Intensitätswert in jedem Band. Voreinstellung ist 1.0

7.3.1.1.3

PNG_Image (const EB_Image & Image)

Copy-Konstruktor

Copy-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse PNG_Image (→7.3.1.1.1, *page 169*) als Kopie von der übergebenen Instanz.

Parameters: `Image` — Instanz, von der die Kopie erzeugt wird.

7.3.1.1.4

PNG_Image (const char *name)

File-Konstruktor

File-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der KlassePNG_Image (→7.3.1.1.1, *page 169*), indem er versucht, den Inhalt der Datei mit dem angegebenen Namen als im PNG-Format gespeicherte Bilddaten zu interpretieren.

Parameters: `name` — Name einer Datei, aus der die Bilddaten gelesen werden sollen.

7.3.1.2

public Methoden

Names

7.3.1.2.1

virtual EB_Image& **load** (const char *name) throw(EBIPNGLoaderEXP, EBICouldNotOpenLoadFileEXP)

Bild laden

Bild laden. Diese Methode versucht, den Inhalt der Datei mit dem angegebenen Namen als im PNG-Format gespeicherte Bilddaten zu interpretieren und die Daten entsprechend in die aktuelle Instanz einzutragen. Kann aus der angegebenen Datei nicht gelesen werden, wird eine Exception geworfen.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `name` — Name einer Datei, aus der die Bilddaten gelesen werden sollen.

7.3.1.2.2

virtual void **save** (const char *name) throw(EBIWrongNumberOfBandsEXP, EBICouldNotOpenSaveFileEXP)

Bild speichern

Bild speichern. Diese Methode speichert die Bilddaten in der aktuellen Instanz im PNG-Format in eine Datei mit dem angegebenen Namen. Hat die aktuelle Instanz nicht genau drei Bänder oder kann eine Datei mit dem angegebenen Namen nicht geschrieben werden, wird eine Exception geworfen

Parameters: `name` — Name einer Datei, in der die Daten gespeichert werden sollen.

7.4

Modul zum Laden und Speichern einer Instanz der Klasse EB_Image (→1.3, *page 39*).

Names

7.4.1 class **GZ_Image** : public EB_Image
Diese Klasse dient zum Laden und Speichern von Bilddaten 172

7.4.1

class **GZ_Image** : public EB_Image

Diese Klasse dient zum Laden und Speichern von Bilddaten

Inheritance

Public Members

Diese Klasse dient zum Laden und Speichern von Bilddaten. Die Bilddaten werden binär in eine Datei geschrieben und on the fly mittels gzip komprimiert. Gespeichert werden in der Datei in dieser Reihenfolge die Breite und Höhe des Bildes, der minimale und der maximale Intensitätswert, die Anzahl der Bänder und schließlich die Intensitätswerte jedes Bandes.

Author: Jürgen EL BOSSO Key

7.4.1.1

Konstruktoren und Destruktor

Names

7.4.1.1.1

GZ_Image (void)

Default-Konstruktor

Default-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse GZ_Image (→7.4.1.1.1, *page 173*) mit einer Breite und Höhe von eins, einem Band und dem Dynamikbereich von 0.0 bis 1.0.

7.4.1.1.2

GZ_Image (unsigned int x, unsigned int y, unsigned int usedbands, float color = -1.0, float min = 0.0, float max = 1.0)

Konstruktor

Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse GZ_Image (→7.4.1.1.1, *page 173*).

Parameters:
`x` — Breite des Bildes.
`y` — Höhe des Bildes.
`usedbands` — Anzahl an Bändern im Bild.
`color` — Intensitätswert, mit dem alle Pixel aller Bänder initialisiert werden. Liegt dieser Wert außerhalb des durch min und max bestimmten Intervalls, findet keine Initialisierung statt. Voreinstellung ist -1.0
`min` — Minimaler Intensitätswert in jedem Band. Voreinstellung ist 0.0
`max` — Maximaler Intensitätswert in jedem Band. Voreinstellung ist 1.0

7.4.1.1.3

GZ_Image (const EB_Image & Image)

Copy-Konstruktor

Copy-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse GZ_Image (→7.4.1.1.1, *page 173*) als Kopie von der übergebenen Instanz.

Parameters: `Image` — Instanz, von der die Kopie erzeugt wird.

7.4.1.1.4

GZ_Image (const char *name)

File-Konstruktor

File-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der KlasseGZ_Image (→7.4.1.1.1, *page 173*), indem er versucht, den Inhalt der Datei mit dem angegebenen Namen als im beschriebenen Format gespeicherte Bilddaten zu interpretieren.

Parameters: `name` — Name einer Datei, aus der die Bilddaten gelesen werden sollen.

7.4.1.2

public Methoden

Names

7.4.1.2.1

virtual EB_Image& **load** (const char *name) throw(EBICouldNotOpenLoadFileEXP)

Bild laden

Bild laden. Diese Methode versucht, den Inhalt der Datei mit dem angegebenen Namen als im beschriebenen Format gespeicherte Bilddaten zu interpretieren und die Daten entsprechend in die aktuelle Instanz einzutragen. Kann aus der angegebenen Datei nicht gelesen werden, wird eine Exception geworfen.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `name` — Name einer Datei, aus der die Bilddaten gelesen werden sollen.

7.4.1.2.2

virtual void **save** (const char *name) throw(EBICouldNotOpenSaveFileEXP)

Bild speichern

Bild speichern. Diese Methode speichert die Bilddaten in der aktuellen Instanz im beschriebenen Format in eine Datei mit dem angegebenen Namen. Kann eine Datei mit dem angegebenen Namen nicht geschrieben werden, wird eine Exception geworfen

Parameters: `name` — Name einer Datei, in der die Daten gespeichert werden sollen.

7.5

Modul zum Laden und Speichern im TIFF-Format.

Names

7.5.1

class **TIFF_Image** : public EB_Image

Diese Klasse dient zum Laden und Speichern von Bilddaten

Inheritance

Public Members

Diese Klasse dient zum Laden und Speichern von Bilddaten. Dabei ist es nur möglich, Bilder mit einem oder drei Bändern zu speichern. Ein Bild mit einem Band wird dabei beim Speichern als Grauwertbild und eins mit drei Bändern als Darstellung im RGB-Farbraum betrachtet. Beim Laden ergibt ein Grauwertbild entsprechend eine Instanz mit einem Band und ein Farbbild eine Instanz mit drei Bändern.

Author: Jürgen EL BOSSO Key

7.5.1.1

Konstruktoren und Destruktor

Names

7.5.1.1.1

TIFF_Image (void)

Default-Konstruktor

Default-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse TIFF_Image (→7.5.1.1.1, *page 177*) mit einer Breite und Höhe von eins, einem Band und dem Dynamikbereich von 0.0 bis 1.0.

7.5.1.1.2

TIFF_Image (unsigned int x, unsigned int y, unsigned int usedbands, float color = -1.0, float min = 0.0, float max = 1.0)

Konstruktor

Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse TIFF_Image (→7.5.1.1.1, *page 177*).

Parameters:

`x` — Breite des Bildes.
`y` — Höhe des Bildes.
`usedbands` — Anzahl an Bändern im Bild.
`color` — Intensitätswert, mit dem alle Pixel aller Bänder initialisiert werden. Liegt dieser Wert außerhalb des durch min und max bestimmten Intervalls, findet keine Initialisierung statt. Voreinstellung ist -1.0
`min` — Minimaler Intensitätswert in jedem Band. Voreinstellung ist 0.0
`max` — Maximaler Intensitätswert in jedem Band. Voreinstellung ist 1.0

7.5.1.1.3

TIFF_Image (const EB_Image & Image)

Copy-Konstruktor

Copy-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse TIFF_Image (→7.5.1.1.1, *page 177*) als Kopie von der übergebenen Instanz.

Parameters: `Image` — Instanz, von der die Kopie erzeugt wird.

7.5.1.1.4

TIFF_Image (const char *name)

File-Konstruktor

File-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der KlasseTIFF_Image (→7.5.1.1.1, *page 177*), indem er versucht, den Inhalt der Datei mit dem angegebenen Namen als im TIFF-Format gespeicherte Bilddaten zu interpretieren.

Parameters: `name` — Name einer Datei, aus der die Bilddaten gelesen werden sollen.

7.5.1.2

public Methoden

Names

7.5.1.2.1 virtual EB_Image&
load (const char *name)
throw(EBICouldNotOpenLoadFileEXP,
EBIWrongFileTypeEXP)
Bild laden 179

7.5.1.2.2 virtual void **save** (const char *name) throw(EBIWrongNumberOfBandsEXP,
EBICouldNotOpenSaveFileEXP,
EBIOutOfMemoryEXP)
Bild speichern 179

7.5.1.2.1

virtual EB_Image& **load** (const char *name) throw(EBICouldNotOpenLoadFileEXP, EBIWrongFileTypeEXP)

Bild laden

Bild laden. Diese Methode versucht, den Inhalt der Datei mit dem angegebenen Namen als im TIFF-Format gespeicherte Bilddaten zu interpretieren und die Daten entsprechend in die aktuelle Instanz einzutragen. Kann aus der angegebenen Datei nicht gelesen werden, wird eine Exception geworfen.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `name` — Name einer Datei, aus der die Bilddaten gelesen werden sollen.

7.5.1.2.2

virtual void **save** (const char *name) throw(EBIWrongNumberOfBandsEXP, EBICouldNotOpenSaveFileEXP, EBIOutOfMemoryEXP)

Bild speichern

Bild speichern. Diese Methode speichert die Bilddaten in der aktuellen Instanz im TIFF-Format in eine Datei mit dem angegebenen Namen. Hat die aktuelle Instanz nicht genau ein oder drei Bänder oder kann eine Datei mit dem angegebenen Namen nicht geschrieben werden, wird eine Exception geworfen

Parameters: `name` — Name einer Datei, in der die Daten gespeichert werden sollen.

7.6

Modul zum Laden und Speichern im Windows-BMP-Format.

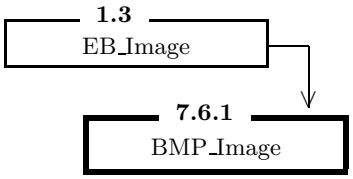
Names

7.6.1

class **BMP_Image** : public EB_Image

Diese Klasse dient zum Laden und Speichern von Bilddaten

Inheritance



Public Members

Diese Klasse dient zum Laden und Speichern von Bilddaten. Dabei ist es nur möglich, Bilder mit einem oder drei Bändern zu speichern. Ein Bild mit einem Band wird dabei beim Speichern als Grauwertbild mit entsprechender Palette und eins mit drei Bändern als Darstellung im RGB-Farbraum gespeichert. Es werden jeweils Bilder mit einer plane gespeichert und beim Laden erwartet. Beim Laden wird immer ein dreibändiges Bild erzeugt. Handelt es sich bei der Datei um ein Bild mit Palette werden die entsprechenden Farben eingetragen.

Author: Jürgen EL BOSSO Key

7.6.1.1

Konstruktoren und Destruktor

Names

7.6.1.1.1

BMP_Image (void)

Default-Konstruktor

Default-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse BMP_Image (→7.6.1.1.1, *page 181*) mit einer Breite und Höhe von eins, drei Bändern und dem Dynamikbereich von 0.0 bis 1.0.

7.6.1.1.2

BMP_Image (unsigned int x, unsigned int y, float color = -1.0, float min = 0.0, float max = 1.0)

Konstruktor

Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse BMP_Image (→7.6.1.1.1, *page 181*).

Parameters:

`x` — Breite des Bildes.
`y` — Höhe des Bildes.
`color` — Intensitätswert, mit dem alle Pixel aller Bänder initialisiert werden. Liegt dieser Wert außerhalb des durch min und max bestimmten Intervalls, findet keine Initialisierung statt. Voreinstellung ist -1.0
`min` — Minimaler Intensitätswert in jedem Band. Voreinstellung ist 0.0
`max` — Maximaler Intensitätswert in jedem Band. Voreinstellung ist 1.0

7.6.1.1.3

BMP_Image (const EB_Image & Image)

Copy-Konstruktor

Copy-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der Klasse BMP_Image (→7.6.1.1.1, *page 181*) als Kopie von der übergebenen Instanz.

Parameters:

`Image` — Instanz, von der die Kopie erzeugt wird.

7.6.1.1.4

BMP_Image (const char *name)

File-Konstruktor

File-Konstruktor. Dieser Konstruktor erzeugt eine Instanz der KlasseBMP_Image (→7.6.1.1.1, *page 181*), indem er versucht, den Inhalt der Datei mit dem angegebenen Namen als im beschriebenen Format gespeicherte Bilddaten zu interpretieren.

Parameters: `name` — Name einer Datei, aus der die Bilddaten gelesen werden sollen.

7.6.1.2

public Methoden

Names

7.6.1.2.1

virtual EB_Image& **load** (const char *name) throw(EBICouldNotOpenLoadFileEXP, EBIWrongFileTypeEXP, EBIWrongNumberOfBandsEXP)

Bild laden

Bild laden. Diese Methode versucht, den Inhalt der Datei mit dem angegebenen Namen als im BMP-Format gespeicherte Bilddaten zu interpretieren und die Daten entsprechend in die aktuelle Instanz einzutragen. Kann aus der angegebenen Datei nicht gelesen werden, ist es keine Datei im BMP-Format oder ist das Format ein anderes als 1 plane und 8/24 Bits wird eine Exception geworfen.

Return Value: Referenz auf die aktuelle Instanz.
Parameters: `name` — Name einer Datei, aus der die Bilddaten gelesen werden sollen.

7.6.1.2.2

virtual void **save** (const char *name) throw(EBICouldNotOpenSaveFileEXP, EBIWrongNumberOfBandsEXP)

Bild speichern

Bild speichern. Diese Methode speichert die Bilddaten in der aktuellen Instanz im BMP-Format in eine Datei mit dem angegebenen Namen. Kann eine Datei mit dem angegebenen Namen nicht geschrieben werden oder enthält die aktuelle Instanz nicht genau eins oder drei Bänder, wird eine Exception geworfen

Parameters: `name` — Name einer Datei, in der die Daten gespeichert werden sollen.

8 Exceptions

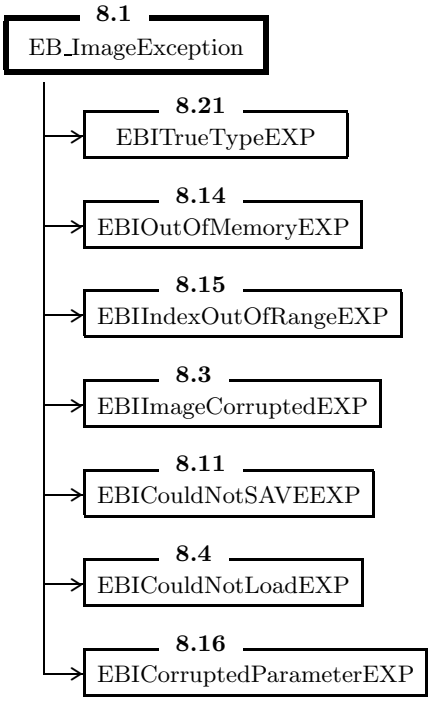
Names

8.1

class **EB_ImageException**

Wurzelexception

Inheritance



Public Members

EB_ImageException (void)

Konstruktor

virtual ~ **EB_ImageException** (void)

Destruktor

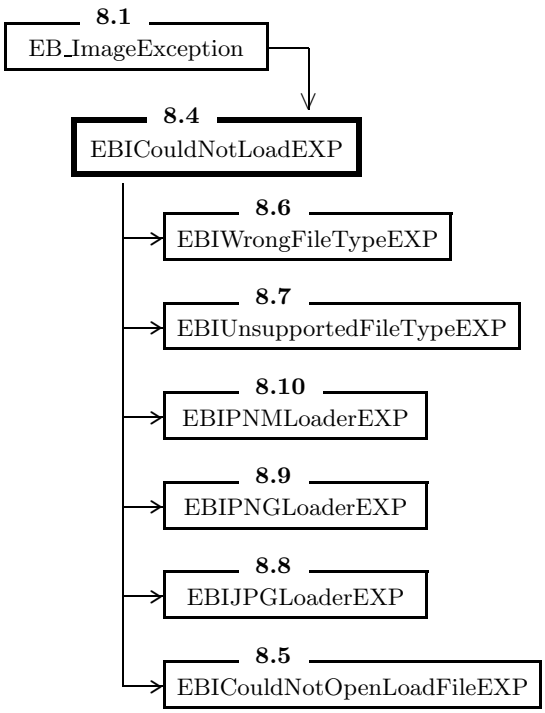
Wurzelexception. Von dieser Exception werden alle Exceptions abgeleitet, die direkt von EB_Image (→1.3, *page 39*) oder von ihr abgeleiteten Klassen geworfen werden.

8.4

class **EBICouldNotLoadEXP** : public EB_ImageException

Diese Exception wird geworfen, wenn beim Laden einer vorhandenen Bilddatei ein Problem auftrat

Inheritance

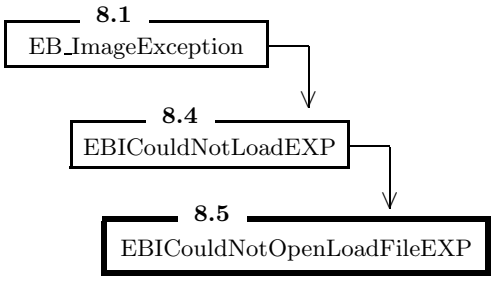


8.5

class **EBICouldNotOpenLoadFileEXP** : public EBICouldNotLoadEXP

Diese Exception wird geworfen, wenn versucht wurde, eine nicht existente Bilddatei zu laden

Inheritance

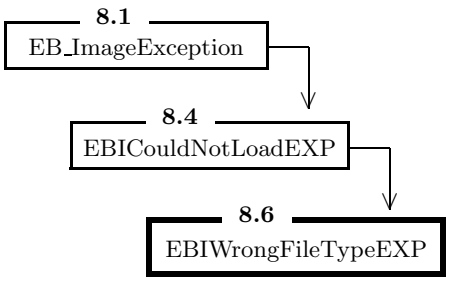


8.6

class **EBIWrongFileTypeEXP** : public EBICouldNotLoadEXP

Diese Exception wird geworfen, wenn der Typ der Bilddatei nicht von der benutzten Loader-Klasse verstanden wird

Inheritance

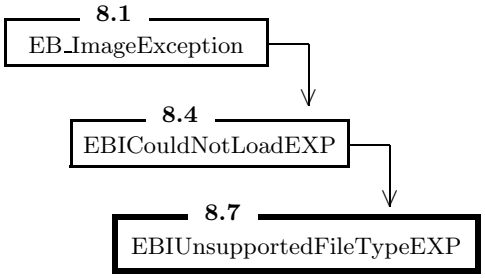


8.7

class **EBIUnsupportedFileTypeEXP** : public EBICouldNotLoadEXP

Diese Exception wird geworfen, wenn das Format der Bilddatei nicht verstanden wurde

Inheritance

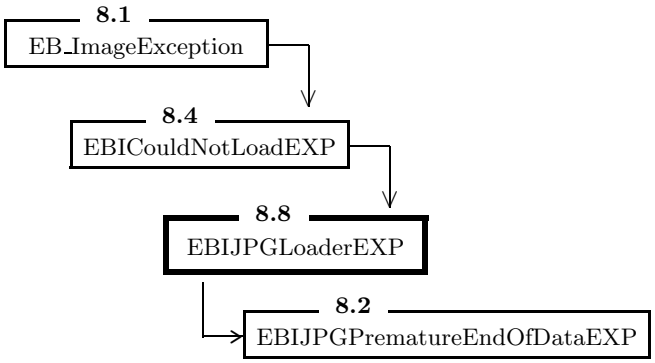


8.8

class **EBIJPGLoaderEXP** : public EBICouldNotLoadEXP

Diese Exception wird von der Klasse JPG_Image (→7.2.1, page 163*) geworfen, wenn beim Laden ein für diese Klasse spezifisches Problem auftritt*

Inheritance

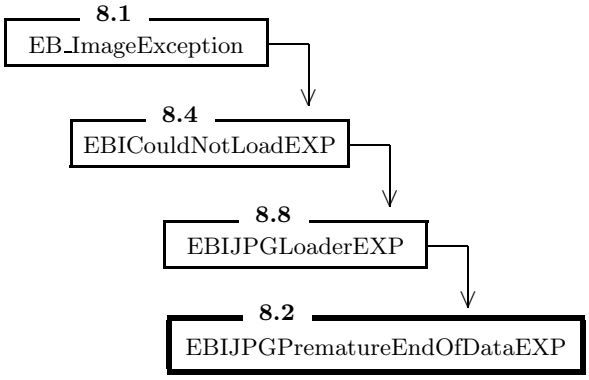


8.2

class **EBIJPGPrematureEndOfDataEXP** : public EBIJPGLoader-EXP

Diese Exception wird von der Klasse JPG_Image (→7.2.1, page 163*) geworfen, wenn die sattsam bekannte Condition premature end of*

Inheritance



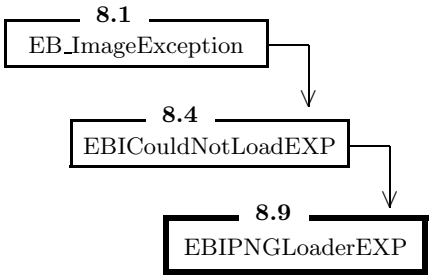
Diese Exception wird von der Klasse JPG_Image (→7.2.1, *page 163*) geworfen, wenn die sattsam bekannte Condition premature end of ... auftritt.

8.9

class **EBIPNGLoaderEXP** : public EBICouldNotLoadEXP

Diese Exception wird von der Klasse PNG_Image (→7.3.1, page 168*) geworfen, wenn beim Laden ein für diese Klasse spezifisches Problem auftritt*

Inheritance

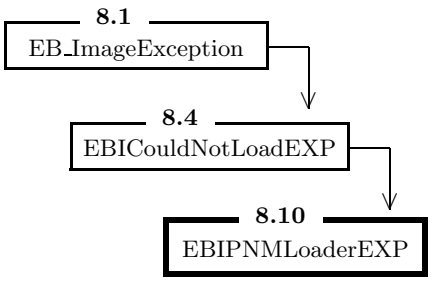


8.10

class **EBIPNMLoaderEXP** : public EBICouldNotLoadEXP

Diese Exception wird von der Klasse PNM_Image (→7.1.1, page 159*) geworfen, wenn beim Laden ein für diese Klasse spezifisches Problem auftritt*

Inheritance

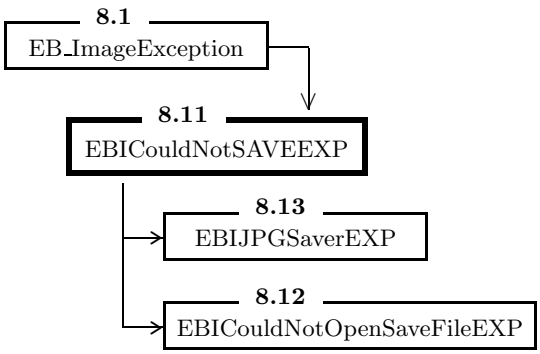


8.11

class **EBICouldNotSAVEEXP** : public EB_ImageException

Diese Exception wird geworfen, wenn beim Speichern einer Bilddatei ein Problem auftrat

Inheritance

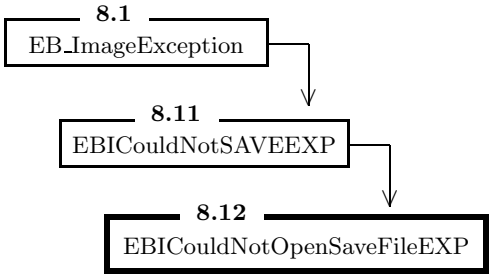


8.12

class **EBICouldNotOpenSaveFileEXP** : public EBICouldNotSAVE-EXP

Diese Exception wird geworfen, wenn die Datei nicht zum Speichern geöffnet werden konnte

Inheritance

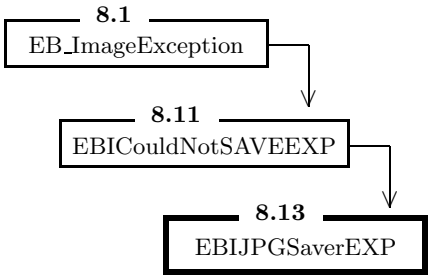


8.13

class **EBIJPGSaverEXP** : public EBICouldNotSAVEEXP

Diese Exception wird von der Klasse JPG_Image (→7.2.1, page 163*) geworfen, wenn beim Speichern ein für diese Klasse spezifisches Problem auftrat*

Inheritance

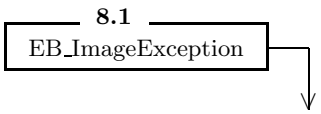


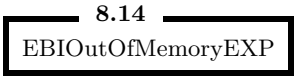
8.14

class **EBIOutOfMemoryEXP** : public EB_ImageException

Diese Exception wird geworfen, wenn im Zuge irgendeiner Operation beim Reservieren von Speicher ein Problem auftrat

Inheritance



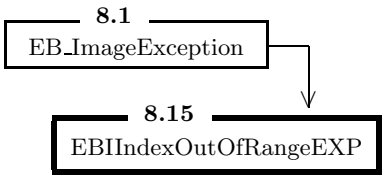


8.15

class **EBIIndexOutOfRangeEXP** : public EB_ImageException

Diese Exception wird geworfen, wenn versucht wurde, auf ein nicht verfügbares Bildelement (Band oder Pixel) zuzugreifen

Inheritance

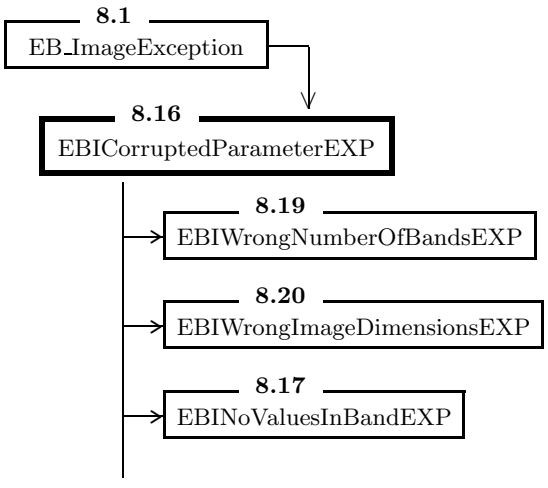


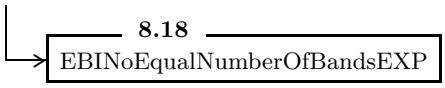
8.16

class **EBICorruptedParameterEXP** : public EB_ImageException

Diese Exception wird geworfen, wenn Parameter außerhalb des Toleranzbandes liegen und die jeweilige Methode keine sichere Rückfallposition kennt

Inheritance



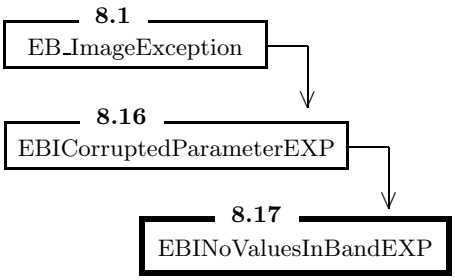


8.17

class **EBINoValuesInBandEXP** : public EBICorruptedParameter-EXP

Diese Exception wird geworfen, wenn eine Operation Inhalte in einem bestimmten Band voraussetzt, in dem jedoch keine vorhanden sind

Inheritance

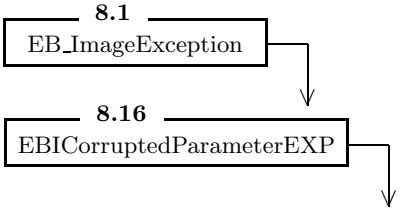


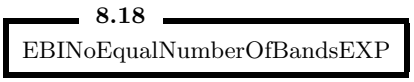
8.18

class **EBINoEqualNumberOfBandsEXP** : public EBICorruptedParameterEXP

Diese Exception wird geworfen, wenn eine Operation auf zwei Bildern die gleiche Anzahl an Bändern in ihnen voraussetzt, diese sich aber unterscheiden

Inheritance



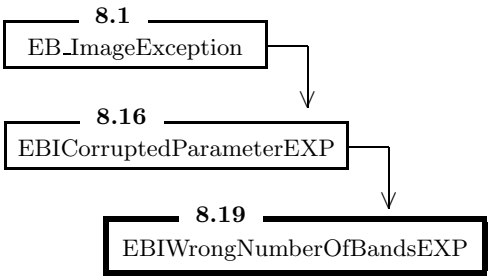


8.19

class **EBIWrongNumberOfBandsEXP** : public EBICorruptedParameterEXP

Diese Exception wird geworfen, wenn für eine Operation eine bestimmte Anzahl Bänder vorausgesetzt wird und diese Voraussetzung aber nicht erfüllt wird

Inheritance

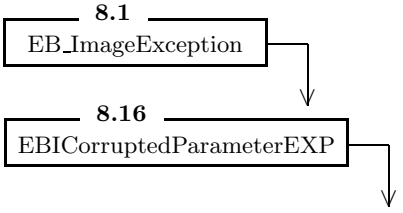


8.20

class **EBIWrongImageDimensionsEXP** : public EBICorruptedParameterEXP

Diese Exception wird geworfen, wenn die Dimensionen des Bildes die gewählte Operation nicht zulassen

Inheritance

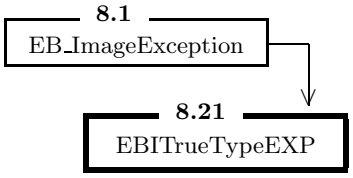


8.21

class **EBITrueTypeEXP** : public EB_ImageException

Diese Exception wird geworfen, wenn die TrueType-Bibliothek ein nicht tolerierbares Problem anzeigte

Inheritance

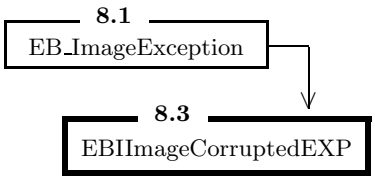


8.3

class **EBIImageCorruptedEXP** : public EB_ImageException

Diese Exception wird geworfen, wenn bei der Verarbeitung klassenintern ein Fehler auftrat

Inheritance



Diese Exception wird geworfen, wenn bei der Verarbeitung klassenintern ein Fehler auftrat. Sie sollte also eigentlich nie kommen

9 Unterstützende Funktionen

Names

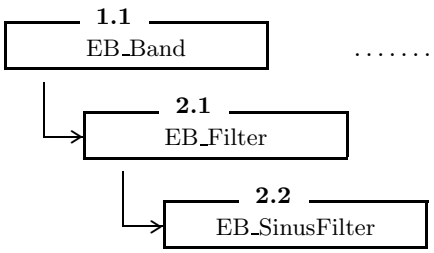
9.1 EB_Image **loadEB_Image** (char *filename) throw(EBICouldNotLoadEXP)

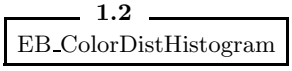
Bild laden

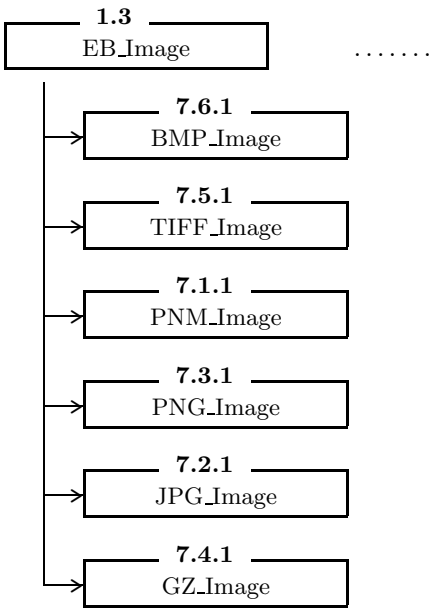
Bild laden. Diese Funktion nutzt Funktionalität der Klasse EB_Image (→1.3, *page 39*) und von ihr abgeleiteter Klassen um auf Massenspeicher ausgelagerte Bilder bearbeiten zu können. Der Antrieb hinter dieser Funktion war, die Notwendigkeit für den Anwender beziehungsweise Entwickler auszuräumen, erst das Format, in dem die Daten vorliegen zu ermitteln, damit er die entsprechende Klasse zum Laden auswählen kann. Diese Methode prüft intern die übergebene Datei auf das vorliegende Format und lädt die Daten anschließend wenn eine dazu fähige Klasse vorliegt. Ist das nicht der Fall oder können die Daten aus sonstigen Gründen nicht geladen werden, wird eine Exception geworfen.

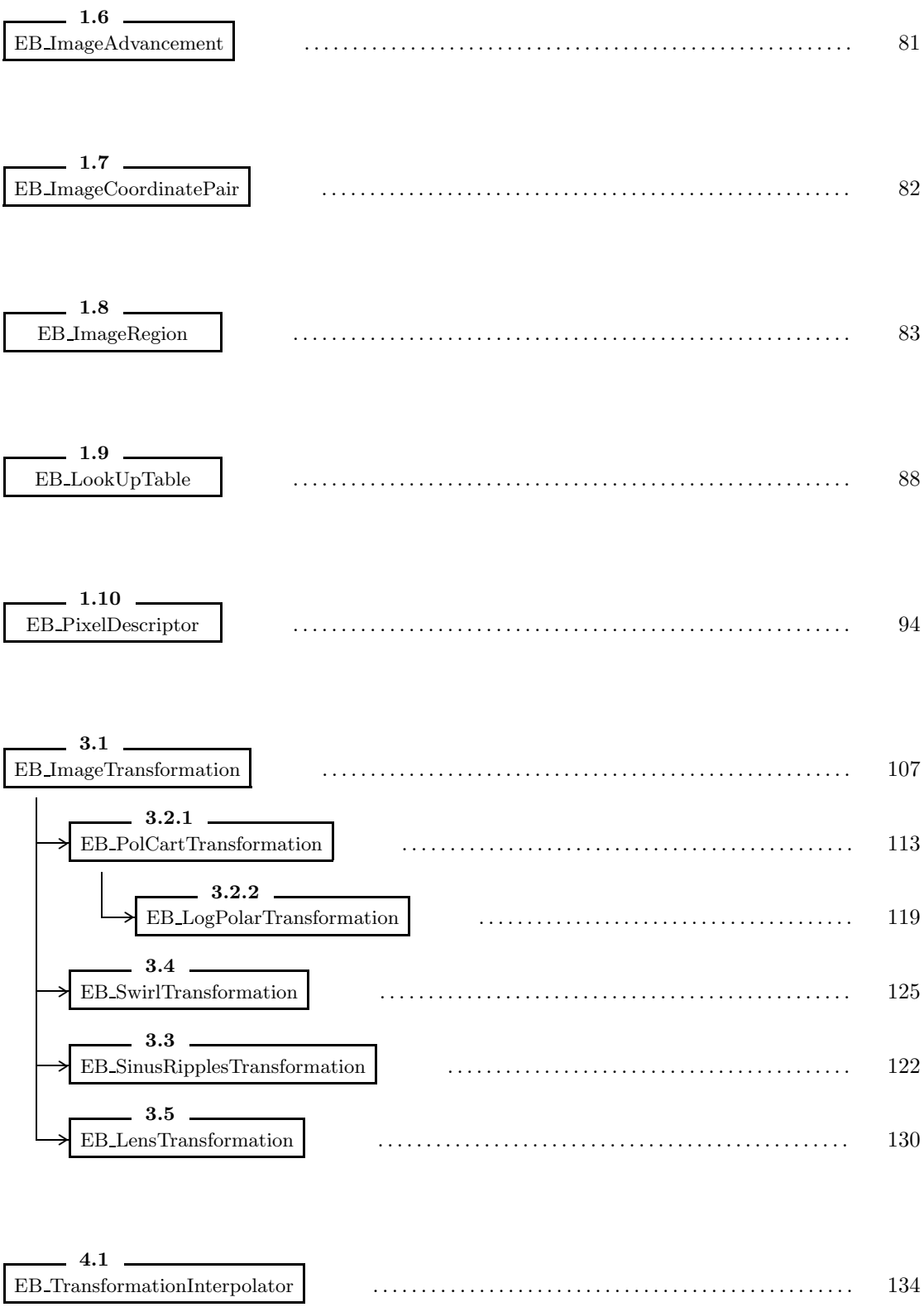
Return Value: Referenz auf eine Instanz der Klasse EB_Image (→1.3, *page 39*)
Parameters: `name` — Name einer zu ladenden Datei

Class Graph








1.6
EB_ImageAdvancement
81
1.7
EB_ImageCoordinatePair
82
1.8
EB_ImageRegion
83
1.9
EB_LookUpTable
88
1.10
EB_PixelDescriptor
94
3.1
EB_ImageTransformation
107
3.2.1
EB_PolCartTransformation
113
3.2.2
EB_LogPolarTransformation
119
3.4
EB_SwirlTransformation
125
3.3
EB_SinusRipplesTransformation
122
3.5
EB_LensTransformation
130
4.1
EB_TransformationInterpolator
134