Ontwerpdocument

# Inleiding

In het kader van het project ‘Deblurring blurred text’ hebben wij een plug-in ontworpen voor het deblurren van geblurde tekstafbeeldingen volgens het algoritme van Pan et al. [2014]. De plug-in kan gebruikt worden in combinatie met het open source beeldbewerkingspakket GIMP. In dit document presenteren we het ontwerp van de plug-in en lichten we de gemaakte ontwerpbeslissingen toe. Daarbij willen we opmerken dat we het algoritme niet inhoudelijk behandelen. Hiervoor verwijzen we naar hoofdstuk 4 van de bijbehorende scriptie. We veronderstellen dat de lezer in hoofdlijnen bekend is met het algoritme van Pan et al. en de daarbij gebruikte terminologie.

Bij het ontwerpen hebben we de gangbare Rational Unified Process (RUP)-ontwerpaanpak gehanteerd zoals beschreven in het boek ‘applying UML and patterns’ van Craig Larman [2004].

In hoofdstuk 2 beschrijven we de requirements met behulp van een use case diagram en een uitwerking van de geïdentificeerde use cases. Vervolgens gaan we in hoofdstuk 3 in op de specifieke karakteristieken die gelden voor een GIMP plug-in. Deze leggen een aantal belangrijke randvoorwaarden vast, zoals de keuze van de programmeertaal. In hoofdstuk 4 beschrijven we de logische architectuur, gevolgd door de uitwerking van de domain layer in hoofdstuk 5. De uitwerking van de domain layer wordt beschreven aan de hand van een klassendiagram en een sequence diagram. In hoofdstuk 6 behandelen we de GUI layer. Omdat deze geprogrammeerd is in de taal C volstaan we hier met een communicatiediagram. Dit communicatiediagram geeft niet alleen een goed inzicht in de onderlinge communicatie tussen de modules, maar geeft ook een goed beeld van de modules zelf. Ten slotte beschrijven we in hoofdstuk 7 de uitwerking van de belangrijkste methodes. Deze helpen de lezer bij het begrijpen van de programmacode.

# Use cases

In deze paragraaf worden de requirements gespecificeerd met behulp van use cases. Eerst zal het use case diagram worden gepresenteerd waarmee een overzicht wordt gegeven van de complete op te leveren functionaliteit. Vervolgens worden de individuele use cases in detail uitgewerkt.

##  Use case diagram

In het onderstaande use case diagram zijn de twee use cases weergegeven waarmee de op te leveren functionaliteit beschreven wordt.

#

Figuur : Use case diagram

Merk op dat het systeem hier gedefinieerd is als zijnde het programma GIMP, de plug-in en de nog toe te lichten test suite.

Het diagram bevat twee use cases. De use case ‘deblur Pan et al’ beschrijft de functionaliteit met betrekking tot het deblurren van tekst en de use case ‘apply test suite’ beschrijft de functionaliteit met betrekking tot de test suite.

De belangrijkste actor is de user. Omdat het algoritme ook via een GIMP-script gerund moet kunnen worden is dit script als tweede system actor toegevoegd.

## Use case beschrijving

In deze paragraaf worden de twee use cases uit het use case diagram beschreven.

### Use case: Deblur Pan et al.

|  |
| --- |
| **Scope**: GIMP + plug-in |
| **Primary actor**: GIMP user + GIMP script |
| **Stakeholders + interests**:* User wil de geblurde tekst zo goed mogelijk deblurren. De verwerkingstijd bedraagt bij voorkeur enkele minuten, hoewel voor grote afbeeldingen 1 à 2 uur nog acceptabel is.
 |
| **Preconditions**: User heeft in GIMP een geblurde afbeelding geopend en optioneel een selectie gemaakt van de te deblurren tekst. Indien geen selectie is gemaakt wordt de gehele afbeelding gedeblurd. |
| **Postconditions**: De geblurde afbeelding of selectie is vervangen door de gedeblurde afbeelding in GIMP.  |
| **Main Succes Scenario:**1. User selecteert in GIMP de functie ‘Pan text deblurring’
2. User geeft een waarde voor de volgende parameters:
	* Kernelgrootte;
	* Lambda pixel;
	* Lambda grad;
	* Aantal iteraties;
	* Lambda TV;
	* Lambda L0;
	* Of er ringing removal moet worden toegepast;
	* Of er gamma correctie moet worden toegepast.

User bevestigt de opgegeven parameterwaarden.1. De plug-in deblurt de opgegeven afbeelding of selectie volgens het algoritme van Pan et al. met de opgegeven parameterwaarden.
2. De plug-in toont het resultaat ook wel de latente afbeelding genoemd, de originele afbeelding, de berekende kernel en het tussenresultaat ook wel de intermediary afbeelding genoemd.
3. User geeft aan dat het resultaat bruikbaar is.
4. GIMP vervangt in het origineel de opgegeven selectie door het deblurresultaat.
 |
| **Extensions**1a) De deblurfunctie kan ook via een script aangeroepen worden. De parameterwaarden voor stap 2 worden uit het script ingelezen en stap 5 wordt dan overgeslagen.2a) Voordat user de parameterwaarden bevestigt kan hij een preview laten genereren waarin voor een deel van de afbeelding ter grootte van 200 x 200 pixels het resultaat van het deblurren met de gekozen parameterwaarden wordt getoond. Hij kan dit op ieder moment gedurende stap 2 herhalen.2b) User kan op ieder moment gedurende stap 2 de parameterwaarden terugzetten naar de defaultwaarden.2c) User kan op ieder moment gedurende stap 2 een helppagina oproepen met daarin meer informatie over de plug-in, waaronder een toelichting van de parameters.2d) User kan op ieder moment gedurende stap 2 het deblurren annuleren. De originele afbeelding blijft dan ongewijzigd.4a) User kan besluiten de deblurresultaten niet te gebruiken. De originele afbeelding blijft dan ongewijzigd. |

### Use case : apply test suite

|  |
| --- |
| **Scope**: text deblurring tools |
| **Primary Actor**: User |
| **Stakeholders + interests**:* User wil beschikken over een set afbeeldingen waarmee de effectiviteit van een algoritme of tool voor text deblurring kan worden bepaald. Dit vanuit het oogpunt van informatiebeveiliging.
 |
| **Preconditions**: User beschikt over een tool voor text deblurring |
| **Postconditions**: User heeft inzicht in de effectiviteit van de gebruikte tool en/of het algoritme voor deblurring van geblurde tekst in het kader van informatiebeveiliging. |
| **Main Succes Scenario:**1. User past het algoritme toe op de geblurde afbeeldingen in de test suite.
2. User vergelijkt de ontblurde afbeeldingen met de originelen om de effectiviteit te beoordelen
3. User beoordeelt handmatig hoe effectief het algoritme is voor het deblurren van tekst die geblurd is in het kader van informatiebeveiliging.
 |
| **Opmerkingen**:1. De nadruk bij deze use case ligt op het verzamelen van voor informatiebeveiliging representatieve geblurde tekstafbeeldingen en bijbehorende originelen.
2. Automatisering van de test suite, waarbij de afbeeldingen in een batch worden ontblurd en vergeleken, is vanwege de verscheidenheid aan algoritmes niet zinvol en behoort daarom niet tot de scope van deze use case.
3. Om het toepassen van het algoritme zo efficiënt mogelijk te laten verlopen zullen alleen volledig geblurde afbeeldingen worden opgenomen, dus geen afbeeldingen waarin slechts een gedeelte geblurd is.
4. De gebruikte blurmethodes zijn beperkt tot motion blur, gaussian blur en pixelization. Bij de eerste twee methodes wordt uitgegaan van uniforme blur. Pixelization is inherent niet-uniform.
 |

# Gimp plug-in

Omdat het algoritme als een GIMP plug-in geïmplementeerd moet worden, beschrijven we hier de generieke kenmerken van een GIMP plug-in en de randvoorwaarden die daaruit volgen.

GIMP plug-ins kunnen in de volgende talen gebouwd worden:

* Python
* GIMP script
* C

Omdat het algoritme erg complex is en veel zware matrixberekeningen bevat is de triviale keuze om C te gebruiken. C wordt namelijk als enige van de drie mogelijkheden gecompileerd naar machinetaal waardoor de performance naar verwachting beter zal zijn. Ook biedt C betere mogelijkheden tot rechtstreekse geheugentoegang en parallellisatie van berekeningen, en heeft C meer ondersteunende bibliotheken voor dit soort berekeningen.

Een aandachtspunt van C is dat dit een low level procedurele taal is waarbij de programmeur zelf verantwoordelijk is voor een groot deel van het geheugenmanagement. Om dit te ondervangen hebben we gekozen om alleen de GUI-componenten in C te bouwen en het algoritme in C++. We hebben een proof-of-concept uitgevoerd om aan te tonen dat deze constructie werkt.

De voordelen van C++ ten opzichte van C zijn:

* Het is een objectgeoriënteerde taal waardoor de voordelen van objectgeoriënteerd ontwerpen benut worden zoals een heldere en beter onderhoudbare structuur van de code.
* C++ kent meer bibliotheken dan C.
* C++ kent een betere foutafhandeling.
* C++ kent referenties waardoor gebruik van pointers niet meer nodig is. Hierdoor is C++ veiliger voor programmeurs die geen ervaring met pointers hebben.
* C++ kent meer mogelijkheden voor terugkeerwaardes van functies, waardoor je minder afhankelijk bent van het gebruik van pointers voor het teruggeven van de resultaten.
* C++ heeft een betere typechecking dan C.
* In C++ kun je gebruik maken van const reference parameters om te voorkomen dat de waarden van variabelen onbedoeld gewijzigd worden.

GIMP start een plug-in op in een eigen run-time omgeving waarbij gebruik wordt gemaakt van een gedeeld geheugen voor de overdracht van data.

Om de plug-in vanuit GIMP aan te kunnen roepen, moet de plug-in de volgende struct bevatten:

**GimpPlugInInfo PLUG\_IN\_INFO = {**

 **NULL, /\* init\_proc \*/**

 **NULL, /\* quit\_proc \*/**

 **query, /\* query\_proc \*/**

 **run, /\* run\_proc \*/**

**};**

Deze struct specificeert de methoden die aangeroepen moeten worden voor de initialisatie, het afsluiten (quit), het uitvoeren (run) en het registreren van de plug-in (query). De query-procedure wordt aangeroepen bij het opstarten van GIMP om alle plug-ins die in de plug-in folder staan te registreren in de Procedural Database (PDB). Hierbij wordt onder andere de volgende informatie aan GIMP doorgegeven:

* De naam van de plug-in;
* De omschrijving van de plug-in;
* De Help-functies;
* De plaats in het menu;
* De gebruikte parameters.

Als GIMP een plug-in activeert dan roept GIMP de run-procedure aan. Deze krijgt als parameters een pointer naar de inputvariabelen, zoals de input afbeelding, en een pointer naar de outputvariabelen zoals het resultaat. De data waar de pointers naar verwijzen staan in een geheugen dat door GIMP en de plug-in gedeeld worden. Als de plug-in klaar is met de bewerking dan stuurt deze het flush-commando naar GIMP als teken dat de resultaten verder verwerkt kunnen worden in GIMP.

# Logische architectuur

Omdat de plug-in als een apart programma wordt opgestart, kan voor de plug-in de volgende standaard logische architectuur gebruikt worden:

Figuur Systeem architectuur voor de TextDeblurPan plug-in

Systeem

GIMP Application

PDB

TextDeblurPan plugin

textDeblurPanGui

textDeblurPan

Utilities layer

Test layer

Het complete systeem bestaat uit GIMP en de plug-in. Deze twee communiceren met elkaar via de Procedural Database (PDB) waarin de plug-in wordt geregistreerd.

Binnen de architectuur van de plug-in worden vier layers onderscheiden: een presentatie layer (textDeblurPanGui), een domain layer (textDeblurPan), een utilities layer en een test layer.

De presentatie layer ‘textDeblurGui’ regelt de communicatie met de gebruiker en daarnaast ook de communicatie met GIMP, inclusief het renderen van de resultaten via het flush commando. De presentatie layer is geschreven in C. Een uitzondering geldt voor de klassen DeblurAdapter en progressBarUpdater die communiceren met de domain layer die in C++ geschreven is.

De domain layer ‘textDeblur’ bevat de uitwerking van het algoritme en is geschreven in C++. Deze layer is onafhankelijk van de presentatie layer en kan dus eventueel ook door andere GUI’s hergebruikt worden.

De utilities layer bestaat uit de C++ bibliotheek OpenCV.

De test layer bevat de unittests voor de domain layer. Omdat deze layer alleen gebruikt wordt voor unittesten wordt deze niet mee gecompileerd in de uiteindelijke plug-in. In figuur 2 is dit aangegeven door de test layer met een onderbroken lijn naast de andere layers te tekenen.

Merk op dat de plug-in geen datalaag heeft. De resultaten van het algoritme worden dus alleen teruggeven aan GIMP en niet door de plug-in opgeslagen.

# domain layer

In dit hoofdstuk beschrijven we het ontwerp van de domain layer. Eerst beschrijven we in de paragrafen 1 en 2 respectievelijk het klassendiagram en het sequence diagram voor de deblur-methode. In paragraaf 3 gaan we in op de gebruikte bibliotheken en in paragraaf 4 beschrijven we de overwegingen met betrekking tot de performance.

## Klassendiagram

Op basis van het deelonderzoek over het algoritme van Pan et al., de MATLAB-code en het artikel van Pan et al. [2014] hebben we het volgende klassendiagram voor de domain layer opgesteld:



Figuur : Klassendiagram van de domain layer. Voor de overzichtelijkheid is de superklasse ImageParameter onder de subklassen afgebeeld in plaats van erboven.

In het ontwerp maken we gebruik van het controllerpatroon dat door de klasse FrameworkController wordt geïmplementeerd. Deze klasse is als een singleton klasse geïmplementeerd omdat deze maar één keer geïnstantieerd hoeft te worden. De FrameworkController is naast de communicatie met de GUI ook verantwoordelijk voor de implementatie van het hoofdalgoritme. We hebben deze verantwoordelijkheden bewust samengenomen in één klasse om het ontwerp zo eenvoudig mogelijk te houden. Mocht de plug-in in de toekomst uitgebreid worden met andere algoritmen dan kan - afhankelijk van de gekozen oplossing - overwogen worden om deze verantwoordelijkheden te scheiden.

De FrameworkController is een subklasse van de superklasse Observable, zodat we in de GUI layer gebruik kunnen maken van het observerpatroon. Hiermee is het mogelijk om in de GUI door middel van een voortgangsbalk de voortgang van het deblurren weer te geven.

De FrameworkController heeft onder andere als attributen een LatentImage-object, een Kernel-object en BlurredImage-object. Deze drie objecten stellen de parameters voor uit de generieke deblurformule y = k\*x + n die toegelicht wordt in hoofdstuk 2 van de scriptie. In deze formule is y de geblurde afbeelding, k de kernel en x de latente afbeelding.

Omdat deze parameters veelal dezelfde bewerkingen ondergaan en ze alle drie een afbeeldingmatrix voorstellen, hebben we een generieke klasse ImageParameter gemaakt waarvan de klassen LatentImage, Kernel en BlurredImage subklassen zijn. Op deze manier wordt duplicatie van code voorkomen. Klasse ImageParameter heeft als attribuut een Mat-object heeft waarmee een afbeelding kan worden opgeslagen. De klasse bevat daarnaast generieke wiskundige bewerkingen zoals de Fast Fourier Transform (FFT), berekeningen met complexe getallen en elementsgewijze vermenigvuldiging.

De klasse BlurredImage is verantwoordelijk voor de bewerkingen op de geblurde afbeelding zoals het resizen van de afbeelding en het aanpassen van de afbeelding aan de optimale FFT-grootte. Omdat de geblurde afbeelding in meerdere vormen gebruikt wordt, zoals geconverteerd naar grijswaarden of voorzien van padding, bevat de klasse een aantal attributen om deze vormen naast elkaar vast te kunnen houden.

De klasse Kernel is verantwoordelijk voor de berekeningen die uitgevoerd worden op de kernel. De belangrijkste methode van de klasse Kernel is de methode calculateKernel die de kernel berekent op basis van de geblurde afbeelding en intermediary latente afbeelding.

De klasse LatentImage is verantwoordelijk voor de bewerkingen die op de intermediary en de latente afbeelding worden uitgevoerd. We hebben deze bewust samengenomen omdat er geen wezenlijk verschil is tussen de intermediary afbeelding en de latente afbeelding, behalve dat de intermediary afbeelding een tussenproduct is. De drie attributen, h, v en u staan voor de hulpvariabelen die Pan et al. in hun algoritme gebruiken.

Voor meer details wordt verwezen naar hoofdstuk 7 van dit document en naar het deelonderzoek over het algoritme dat in hoofdstuk 4 van de scriptie opgenomen is.

## Sequence diagram

## Vanuit de GUI layer wordt een instantie van klasse FrameworkController opgevraagd waarop de deblur-methode wordt aangeroepen. Het onderstaande sequence diagram geeft de verdere interactie tussen de klassen weer bij de uitvoering van de deblur-methode.

##

Figuur : Sequence diagram van de deblur-methode

Merk op dat de resultaten van de deblur-methode, zijnde de latente afbeelding, de kernel en de intermediary latente afbeelding, via pointers wordt teruggegeven. Deze pointers zijn bij de aanroep van de deblur-methode als parameters meegegeven. Voor de leesbaarheid zijn deze outputparameters weergegeven als aparte variabelen.

Eveneens omwille van de leesbaarheid is de aanroep van de update-methode als onderdeel van het observerpatroon achterwege gelaten. Deze is immers volgens de standaardconventie geïmplementeerd.

De klasse FrameworkController kent ook de methode ‘getMaxKernelSize’ die door de GUI wordt aangeroepen. Deze methode berekent de maximale kernelgrootte voor de gegeven afmetingen van de geblurde afbeelding. Deze waarde wordt in de GUI gebruikt als bovengrens voor de input widgets voor de kernelgrootte. De maximale kernelgrootte wordt opgezocht in een array die wordt gecreëerd met de methode ‘initializeMaxKernelSize’. Bij deze methode is er geen verdere interactie met de andere klassen.

## Ondersteunde bibliotheken.

In het ontwerp hebben we de keuze gemaakt voor de volgende ondersteunende bibliotheken.

### OpenCV

OpenCV staat voor Opensource Computer Vision en is gericht op het zo efficiënt mogelijk uitvoeren van afbeeldingsmanipulaties en videobewerkingen.

OpenCV bevat het Mat-object dat afbeeldingen kan representeren in verschillende formaten zoals kleur- en grijswaarden in 8-, 16- of 32-bits kleurendiepte. Het Mat-object kent bovendien een veelvoud aan kleurcoderingen, waaronder RGB en Lab. Hierdoor is de plug-in voor praktisch alle typen afbeeldingen geschikt en kan het ook eenvoudig gebruikt worden in combinatie met andere GUI-layers.

Daarnaast zijn praktisch alle voorkomende matrixbewerkingen al geïmplementeerd in OpenCV. Deze kunnen rechtstreeks op het Mat-object aangeroepen worden.

Een ander voordeel van OpenCV is dat deze intern dezelfde datastructuur heeft voor het opslaan van afbeeldingen als GIMP. Hierdoor is het mogelijk om het hele datasegment van GIMP te linken met een pointer naar een Mat-object van OpenCV, zonder de data zelf te hoeven kopiëren.

OpenCV scoort in vergelijking met andere bibliotheken hoog met betrekking tot performance [Sinha z.d.].

OpenCV wordt verspreid onder de Berkeley Software Distribution (BSD)-licentie en kan gebruikt worden onder de voorwaarde van vermelding van de auteurs.

### Log4C

Om relevante informatie over de uitvoering van de plug-in te kunnen loggen hebben we het Log4C-framework gebruikt. Log4C is een C-implementatie van de Log4J-functionaliteit. De logging kan optioneel geconfigureerd worden door middel van een configuratiebestand. Standaard hebben we de logging-output naar de console gezet. Logging is vooral praktisch ingeval de plug-in via GIMP-scripts in batch wordt aangeroepen.

Log4C wordt verspreid onder de GNU Lesser General Public License (LGPL)

### CUTE

Voor de unittests hebben we gebruik gemaakt van het CUTE-testframework. CUTE heeft een add-on voor de door ons gebruikte ontwikkelomgeving Eclipse waarmee de resultaten van de unittests overzichtelijk weergegeven kunnen worden.

## Performance overwegingen

Omdat deblurring rekenintensief is, hebben we het algoritme zo efficiënt mogelijk gecodeerd. Dit hebben we onder andere gedaan door waar mogelijk pointers te gebruiken als methodeparameters. Hiermee wordt onnodig kopiëren van data voorkomen. De performance van de huidige implementatie is acceptabel en binnen de grens zoals bij de use case is aangegeven.

Omdat de huidige implementatie single threaded is, kan de performance verder verbeterd worden door gebruik te maken van parallellisatie. OpenCV biedt hiervoor standaard de volgende mogelijkheden:

* GPU-versnelling door middel van CUDA. Dit is het parallelle programmeermodel van NVIDIA. Het nadeel van CUDA is dat dit model alleen te gebruiken is op NVIDIA grafische kaarten en niet op AMD-kaarten.
* GPU-versnelling door middel van OpenCL. Dit is een parallel programmeermodel dat oorspronkelijk van Apple was, maar inmiddels ook geschikt is voor Linux en Windows.

OpenCL kan normaalgesproken ook multi-threading op de CPU ondersteunen, echter in OpenCV is dit niet geïmplementeerd. Daarom is multi-threading niet eenvoudig te realiseren voor onze plug-in.

Naast parallellisatie is het ook mogelijk om vectorisatie uit te voeren. Hierbij worden identieke bewerkingen op een array parallel uitgevoerd op meerdere CPU-kernels. In theorie heeft dit een groot effect op de plug-in vanwege de vele matrixbewerkingen. De GCC-compiler biedt de mogelijkheid om dit automatisch in te regelen met de flag “ftree-vectorize”. Echter uit testen met en zonder vectorisatie blijk dat de verwerkingstijd niet merkbaar korter wordt. Vermoedelijk komt dit doordat de OpenCV-bibliotheek hiervoor al geoptimaliseerd is.

Daarnaast is het mogelijk om in de GCC-compiler de optimalisatie flag “-O3” te zetten waarbij een veelvoud van optimalisaties wordt toegepast [Free Software Foundation z.d.]. Echter ook hier bleek uit testen dat dit geen merkbare invloed heeft op de verwerkingstijd.

# Gui layer

In dit hoofdstuk beschrijven we de GUI layer. Eerst beschrijven we de structuur van de GUI layer in paragraaf 1. In paragraaf 2 wordt het ontwerp van de dialoogvensters gegeven.

## Interactie gui layer

Omdat de GUI layer is opgebouwd in C laten we het klassendiagram achterwege. In plaats daarvan gebruiken we het UML communicatiediagram. Deze geeft naast de interactie tussen de modules ook een goed overzicht van de modules zelf.



Figuur : Communicatie diagram van de Gui layer

Vanuit GIMP wordt de run-methode aangeroepen in de module DeblurPanPlugin. De run-methode heeft als parameters pointers naar de input- en outputwaarden. Voor de overzichtelijkheid zijn deze niet in de afbeelding opgenomen. Na aanroep van de run-methode wordt het dialoogscherm voor het opgeven van de deblur-parameters aangeroepen met de functie showDialog in de module DeblurDialog. Omdat de event afhandeling in GIMP maar één dataveld aankan is ervoor gekozen om alle relevante inputparameters samen te brengen in de struct deblurParam. Deze struct wordt ook door GIMP opgeslagen zodat bij het commando “repeat least action” weer dezelfde parameters worden gebruikt. De informatie over het proces en de pointers naar de afbeeldingen worden in de struct deblurImagedata opgeslagen. Beide structs worden als variabele meegeven aan de methode deblurFromGimp in de module DeblurAdapter. De methode deblurFromGimp creëert eerst een instantie van de klasse FrameworkController en een instantie van ProgressBarUpdater. De laatste wordt als observer geregistreerd in stap 2.3 bij de FrameworkController. Na uitvoering van de deblur-methode in de FrameworkController worden de resultaten van de deblur-methode met de functie showResults van de module ResultDialog weergegeven. Als laatste wordt het flush-commando aan GIMP gegeven om de outputparameters verder in GIMP te verwerken.

Als GIMP de plug-in aanroept in de modus ‘non interactive’ (ingeval van scripting) of ‘run with last values’ (ingeval van redo) dan worden de stappen 1, 2.1, 2.2 en 2.4 overgeslagen.

Indien in de deblurDialog gebruik gemaakt wordt van een preview, dan wordt de preview opgebouwd met de functie deblurPreview in de module DeblurPanPlugin. Dit is voor de overzichtelijkheid van het communicatiediagram achterwege gelaten. De preview wordt gegenereerd met de methode deblurFromGimp waarbij de inhoud van het preview canvas als geblurde afbeelding wordt meegegeven.

## Dialoog schermen

De dialoogschermen zijn geschreven met het GimpGTK framework dat een extensie is van het normale GTK framework voor GUI’s in C. Het ontwerp van de GUI is gebaseerd op de generieke GIMP-stijl voor GUI’s zodat de GUI intuïtief aanvoelt voor de ervaren GIMP-gebruiker. Onderstaande afbeelding toont de GUI. De GUI bevat een standaard preview widget. Omdat de berekening van de preview een aantal seconden in beslag neemt, wordt de preview alleen berekend als de gebruiker deze activeert en op de update preview-knop klikt. De preview heeft een eigen voortgangsbalk om verwarring te voorkomen.



Figuur : TextDeblurPan gui voor invoeren van de deblurparametersi

In het rechtergedeelte van het scherm kan de gebruiker de parameterwaarden invoeren door middel van een slider of door de waarde rechtstreeks in te typen. Het gebruik van een slider geeft de gebruiker een visuele indicatie van in welk gedeelte van het bereik de parameterwaarde zich bevindt.

Onderaan in de GUI wordt een voortgangsbalk getoond, naast de standaard voortgangsbalk in het hoofdscherm van GIMP. Merk op dat figuur 6 alleen de plug-in GUI laat zien en niet het GIMP hoofdscherm. De gebruiker kan via de GIMP-statusbar de bewerking op de standaard GIMP-manier eventueel afbreken indien nodig. Met de resetknop worden de waarden teruggezet naar de standaardwaarden zoals Pan et al. deze aanbevelen. De helpknop opent een browserscherm waarin informatie over de plug-in en het gebruik ervan wordt getoond. Met de OK-knop wordt het deblurren in gang gezet.

Figuur : Deblur resultatenscherm

Om de gebruiker te helpen met de parameterkeuze zijn standaard de meest voorkomende waarden ingevuld. Het bereik van de parameters is gebaseerd op de gebruikte waarden door Pan et al., waarbij we een factor tien lager en hoger zitten om eventueel extreme situaties aan te kunnen. De maximale waarde van de kernelgrootte is afhankelijk van de afmetingen van de geblurde afbeelding en wordt dynamisch ingesteld in de GUI. Alle boven- en ondergrenzen van de variabelen zijn opgeslagen als constanten zodat deze eenvoudig aangepast kunnen worden.

Na het deblurren wordt onderstaande resultatenscherm getoond met daarin de volgende informatie:

* De geblurde afbeelding om het resultaat mee te vergelijken.
* De kernel. Deze geeft informatie over de vorm van de blur. Ingeval van motion blur laat de kernel het pad van de beweging zien. Ook kan de gebruiker aan de kernel zien of de afmetingen van de kernel groot genoeg zijn.
* De intermediary latente afbeelding. Dit is het ‘kale’ resultaat zonder de laatste non blind-deconvolution stap. Het geeft een indicatie van hoe succesvol het algoritme heeft gewerkt.
* De latente afbeelding. Dit is het eindresultaat.

Met behulp van de Cancel- of OK-knop kan de gebruiker beslissen of de resultaten wel of niet naar GIMP worden doorgestuurd.

## Variable structures

Zoals in paragraaf 6.1 werd aangegeven maakt de GUI gebruik van een tweetal structs om relevante informatie op te slaan. Beide structs zijn gedeclareerd in de headerfile van de module DeblurPanPlugin. Deze structs zijn hieronder verder uitgewerkt.

### deblurParam

De struct deblurParam bevat de parameters voor het deblurren. De struct heeft de onderstaande velden. Merk op dat de parameters van het algoritme niet inhoudelijk in het ontwerpdocument worden behandeld. Daarvoor verwijzen we naar hoofdstuk 4 van de scriptie.

|  |  |
| --- | --- |
| * (gint) kernelSize
 | : Bepaalt de grootte van de kernel |
| * (gdouble) lambdaPixel
 | : Lambda-coëfficient voor pixelintensiteit |
| * (gdouble) lambdaGrad
 | : Lambda-coëfficient voor pixelgradiënt |
| * (gint) numberIterations
 | : Aantal loops tussen stap 1 en stap 2 van het algoritme. |
| * (gboolean) preview
 | : Wel of geen preview activeren in het dialoogvenster |
| * (gdouble) lambdaTV
 | : Lambda-coëfficient voor de LaPlacian deblur |
| * (gdouble) lambdaL0
 | : Lambda-coëfficient voor pixelintensiteit bij de LaPlacian deblur |
| * (gboolean) weightRing
 | : Wel of geen verwijderen van ringing toepassen |
| * (GtkProgressBar\*) progressPreview
 | : Pointer naar de voortgangsbalk voor de preview |
| * (GtkProgressBar\*) progressMain
 | : Pointer naar de voortgangsbalk in het dialoogscherm |
| * (gboolean) gammaCorrectionBoolean
 | : Wel of geen gammacorrectie toepassen |
| * (gint) kernelMaxSize
 | : Maximaal toegestane kernelgrootte |

### DeblurImageData

De struct DeblurImageData bevat de procesinformatie en pointers naar de verschillende afbeeldingen. Deze struct bevat de volgende velden:

|  |  |
| --- | --- |
| * (gint) imageHeight
 | : De hoogte van de geblurde regio |
| * (gint) imageWidth
 | : De breedte van de geblurde regio |
| * (gint) x
 | : x-coördinaat van de linkerbovenhoek |
| * (gint) y
 | : y-coördinaat van de rechterbovenhoek |
| * (GimpDrawable\*) drawable
 | : Gimp drawable  |
| * (GimpPixelRgn\*) latentPixelRgn
 | : Pixel region voor de latente afbeelding |
| * (GdkPixbuf\*) kernelPixbuf
 | : Pixel buffer voor de kernel |
| * (GdkPixbuf\*) blurredPixbuf
 | : Pixel buffer voor de geblurde afbeelding |
| * (GdkPixbuf\*) latentPixbuf
 | : Pixel buffer voor de latente afbeelding |
| * (GdkPixbuf\*) intermediaryPixbuf
 | : Pixel buffer voor de intermediary latente afbeelding |
| * (GimpPixelRgn\*) blurredPixelRgn
 | : Pixel region voor de geblurde afbeelding |
| * (RunMode) runMode;
 | : de runMode: interactive, preview of non interactive |

De pixel buffers worden gebruikt voor het resultatenscherm, waarin geschaalde afbeeldingen van de geblurde afbeelding, de kernel, de intermediary latente afbeelding en de latente afbeelding worden getoond. De twee pixel regions voor respectievelijk de geblurde en de latente afbeelding worden door GIMP gebruikt. De pixel regions, pixel buffers en openCV gebruiken allemaal dezelfde dataopslagstructuur voor de afbeelding.

# Uitwerking deblur algoritme

In dit laatste hoofdstuk van het ontwerpdocument wordt een aantal algoritmen beschreven met als doel de lezer te ondersteunen bij het analyseren van de code. De beschrijving is gebaseerd op het deelonderzoek over het algoritme van Pan et al., de MATLAB-code en het artikel van Pan et al. [2014].

De indeling van dit hoofdstuk is gebaseerd op een onderscheid in klassen, zodat de beschrijving goed aansluit bij de structuur van het ontwerp. De beschrijving is beperkt tot de algoritmen die essentieel zijn voor een goed begrip van de code.

## Klasse FrameworkController

Zoals aangegeven bij de beschrijving van de domain layer, bevat de FrameworkController een implementatie van het centrale controllerpatroon en de hoofdimplementatie van het algoritme. Als de deblur-methode wordt aangeroepen op de klasse FrameworkController, dan wordt eerst het framework geïnitialiseerd. Hierbij worden alle relevante variabelen en objecten gecreëerd die in het algoritme worden gebruikt. Zoals eerder aangegeven is de FrameworkController een singleton, zodat deze niet meerdere keren geïnstantieerd kan worden.

### Attributen

deblurParameters: Een struct om alle deblurparameters vast te houden.

maxKernelSize Een array met daarin de maximale kernelgrootte als functie van de afbeeldingsgrootte

### Algoritme s

#### Algoritme : deblur

Het algoritme van de methode deblur bevat de volgende stappen

Calculate kernel and intermediate

latent image

Calculate latent

 image based on

Laplacian priors

Calculate

latent image

Calculate

kernel

Repeat 5x

Iteration # < Iteration Max

Calculate latent

 image based on

Laplacian priors

Initialize

Parameters

- K Initial

- Iteration Max

- Kernel and image sizesteps

Converteer

Afbeeling naar

Zwart/wit

Increase kernel and image size

Stap 1

Stap 2

|  |
| --- |
| **Input**: geblurde afbeelding *afb*, gamma *g*, lamdba pixel *lp*, lambda grad *lg,* kernel size *sizek* en number of iterations #i*ter* |
| initialiseer controller (*afb, sizek, g*), zie algoritme initializeDeblur**For** i=Max aantal iteraties **to** 0 Ks = resize kernel Bs = resize blurred image LatentImage ← initieer latent image ter grootte van Bs lambdaPixelLoop ← *lp* lambdaGradLoop ← *lg* For j = 1 to *#iter* latentImage ← calculate latente image(Ks, Bs, lambdaPixelLoop, lambdaGradLoop), zie algoritme calculateLatent Kernel = calculate kernal (Ls, Bs, parameters), zie algoritme calculateKernel adjust lambdaPixelLoop en lambdaGradLoop parameterend forKs ← cleanKernel met threshold “max pixel waarde van Ks / threshold”**end** bereken latente afbeelding met laplacian prior based on K, zie algoritme calculateLaPlacianKopieer x, K en Bs naar struct *deblurResults* |
| **Output**: Latente afbeelding en relevante informatie opgeslagen in de struct *deblurResults* |

#### Algoritme InitializeDeblur

Dit algoritme is onderdeel van de deblur-methode. Het creëert objecten voor de kernel, de latente afbeelding en de geblurde afbeelding en berekent het aantal iteraties in de buitenste lus.

|  |
| --- |
| **input:** afbeelding + kernel size, gamma |
| maxIterations ← floor(log(5/Kernel size)/log(0.7)K ← new Kernel object met grootte kernel sizeL ← new LatentImage objectB ← new Blurred object (afbeelding ,gamma) |
| **Output**: objecten K, L B en variabele maxIteration |

## Klasse kernel

Klasse die de kernel representeert. Aangezien de kernel als afbeelding kan worden gezien is deze als subklasse van de klasse ImageParameter geïmplementeerd.

### Attributen

kernelSize : de door de gebruiker opgegeven kernelgrootte

threshold: de grens waarboven de gradiënten in de latente afbeelding uitgesloten worden bij de berekening van de kernel

### Algoritme

#### Algoritme : resizeForStep kernel

Vergroot de afmetingen van de kernel conform de coarse-to-fine-benadering.

|  |
| --- |
| **input**: iteration step |
| **if** kernel afbeeling is null **then**size ← $ceil\left(Kernel\\_size⋅\left(\sqrt{2}\right)^{iterationstep}\right)$**if** (size = even) **then** size← size + 1**end if** Creëer afbeelding met nullen van size x size j ←( sizes-1) / 2set pixel (j,j) en (j,j+1) van kernel op 0,5**else** resize kernel met factor ${\left(\sqrt{2}\right)^{iterationstep}}/{\left(\sqrt{2}\right)^{iterationstep-1}}$ normaliseer kernel, zie algoritme normalize**end if** |
| **output**: resized kernel |

#### Algoritme : pruneKernel

Verwijdert kleine ruis in de kernel. Is als private methode geïmplementeerd.

|  |
| --- |
| **Input**: kernel afbeeling |
| cclist ← lijst met array van pixels met een waarde > 0 en die geconnecteerd zijn met elkaar, omliggend incl hoeken**for** **each** cc **in** cclist **if** sum van alle pixels < 0.1 **then**  pixel waarde ← 0 **end** **if****end** |
| **output**: kernel geschoond voor ruis |

#### Algoritme: Normalize

Verwijdert alle negatieve waarden in de kernel en normaliseert de kernel zodat de som van de waarden 1 wordt.

|  |
| --- |
| **input**: kernel afbeelding |
| **for** **each** pixel **in** afbeeling **if** pixel waarde < 0 **then**  pixel waarde ← 0 **end** **if****end**som ← som van alle pixel waarden in de afbeelding**for** **each** pixel **in** afbeeling pixel waarde ← pixel waarde / som**end** |
| **output**: kernel geschoond voor negatieve waarden en genormaliseerd naar 1 |

#### Algoritme : CalculateKernel

Berekent de kernel op basis van het algoritme van Pan et al.

|  |
| --- |
| **input**: x en y gradiënt van geblurde afbeelding (*dxB* *dyB),* x en y gradiënt van latente afbeelding (*dxL* *dyL),* weight *(w)* en psf grootte *(s)* |
| b ← $F^{-1}\left(\overline{F\left(dxL\right)}F\left(dxB\right)+\overline{F\left(dyL\right)}F\left(dyB\right)\right)$p ← $\overline{F\left(dxL\right)}F\left(dxL\right)+\overline{F\left(dyL\right)}F\left(dyL\right)$imgSize = grootte van F(dxB)psf← matrix ter grootte van imgSize waarbij alle pixels de waarde 1/(aantal pixels) hebbenr ← b – computeAx(psf, p, s,w)p2 ←rrsold ← som van kwadratische waarde van alle elementen in r**for** i=1 **to** 20 ap← computeAx(p2, p, s, w) alpha ← rsold/ sum( waardes van p2 \* waardes van ap) psf = psf + alpha\*p2 r ← r – alpha \* ap rsnew = som van kwadratische waarde van alle elementen in r **if** sqrt(rsnew) < 1e-5 **then** break **end** **if** p ← r + rsnew/rsold\*p2 rsold ← rsnew**end**psf ← psf cleanKernel met threshold 0,5psf ← prune kernelpsf ← normalised psf**if laatste step van inner loop**psf ← adjust\_psf\_center |
| **Output**: psf |

#### Algoritme : computeAx

Dit algoritme ondersteunt het calculateKernel-algoritme. Is als private methode geïmplementeerd.

|  |
| --- |
| **input**: matrix *x*, matrix *p*, size *s*, weight *w* |
| xf ← psf2otf(x, s)y ← otf2psf(p \* xf, size)y ← y + w \* x |
| **output**: matrix y |

#### Algoritme : adjustPSFCenter

Dit algoritme verplaatst het zwaartepunt van de kernel naar het midden van de kernel.

|  |
| --- |
| **input**: kernel |
| Mx ¬ kopieer kernelMy ¬ kopieer kernel**for each** pixel **in** Mx pixelwaarde ¬ pixelwaarde \* kolomnummer**end for****for each** pixel **in** My pixelwaarde ¬ pixelwaarde \* rijnummer**end for**xc1 ¬ som van de pixelwaarden in Mxxc2 ¬ (aantal kolommen in Mx + 1) / 2yc1 ¬ som van de pixelwaarden in Myyc2 ¬ (aantal rijen in My + 1) / 2xshift ¬ afronden(xc2 – xc1)yshift ¬ afronden(yc2 – yc1)kernel ¬verplaats alle kolommen met xshift plaatsen naar rechtskernel ¬ verplaats alle rijen met yshift plaatsen naar benedenkernel ¬ vul de vrijgekomen rijen en kolommen met 0-waarden |
| **output**: kernel met het zwaartepunt in het midden |

## Klasse BlurredImage

Klasse die de parameter voor de geblurde afbeelding representeert. De geblurde afbeelding wordt geconverteerd naar grijswaarden, waarbij eerst eventueel gammacorrectie wordt toegepast.

### Atributen

imageOriginal : ge blurde afbeelding geconverteerd naar double-waarden

ImageOriginalOneChannel: imageOriginal geconverteerd naar grijswaarden

imagePadded imageOriginal geresized naar de optimale FFT-grootte met behulp van de wrap boundary liu-methode.

GammaCorrectie: c orrectiefactor voor de afbeeldingsintensiteit

SizeRescaled: a fmeting van de imagePadded die geherschaald is naar de optimale grootte voor FFT

### Algoritmes

#### Algoritme : new Blurred

Creëert en initialiseert een object dat de geblurde afbeelding representeert.

|  |
| --- |
| **input**: afbeelding uit GIMP, gammacorrectie |
| imageOriginal ← copy waardes van input afbeeldingimageOrignal ← corrigeer voor gamma als gamma <> 1imageOriginalOneChannel ← converteer imageOrignal naar grijswaarden |
| **output**: blurred image naar grijswaarden geconverteerd en gecorrigeerd voor gamma. |

#### Algoritme: resizeForFFT

Schaalt de afbeelding naar de optimale grootte voor FFT, rekening houdend met de grootte van de kernel.

|  |
| --- |
| in**put**: afbeeling, factor, kernel siz |
| sizeImage ←originele grootte x factorwidth ← Width van sizeImage + kernel size -1height ← height van sizeImage + kernel size -1size[] ← [optimal\_size(Width), optimal\_size(heigt)]imageResized ← creëer afbeelding met grootte size[1] x size[2]imageResized ← plak image geschaald met factorimage\_resized ← wrapBoundaryLiu(size[]) |
| **Output**: blurred image, geresized en aangepast voor de optimale FFT grootte. |

####  Algoritme : wrapBoundaryLiu Algoritme

Vult de extra ruimte die ontstaan is door de schaling van de geblurde afbeelding naar de optimale FFT-grootte. Hiervoor wordt gebruik gemaakt van de methode wrapBoundaryLiu. Bij deze methode wordt de padding in drie gebieden A,B en C opgedeeld zoals aangegeven in Figuur 7. Per gebied worden de randen ingevuld door de randen van de geblurde afbeelding te spiegelen. Het binnenste gedeelte van het gebied wordt door extrapolatie van een discrete sinustransformatie ingevuld.



Figuur : opdeling van de padding in 3 gebieden en kopieren van de randen van de gebieden

|  |
| --- |
| **Input:**  afbeelding, [Ha, Wa] hoogte en breedte van de aangepaste afbeelding |
| Hi ← hoogte input afbeeldingWi ← breedte input afbeeldingA← grid met nullen van Ha-Hi hoog bij Wi breedB ← grid met nullen van Hi hoog en Wa-Wi breedC ← grid met nullen van Ha-Hi hoog en Wa-Wi breedA rij1 ← laatste rij van input afbeeldingA laatste rij ← rij 1 van input afbeelding kolommen A ← geëxtrapoleerd tussen eerste en laatste rij mbv DSTB kolom1 ← laatste kolom van input afbeeldingB laatste kolom ← eerste kolom van input afbeelding rijen B ← geëxtrapoleerd tussen eerste en laatste kolom mbt DSTC rij1 ← laatste rij van BC laatste rij ← rij 1 van Bkolommen C ← geëxtrapoleerd tussen eerste en laatste rij mbt DSTOutput ← Combineer input afbeelding met A, B en C zoals in figuur aangegeven  |
| **Output**: afbeelding met afmeting Ha bij Wa met daarin de input afbeelding aangevuld met geëxtrapoleerde waardes |

## Klasse LatentImage

Klasse die de parameter voor de latente afbeelding representeert.

### Atributen

u : matrix voor de hulpvariabele u die de latente afbeelding representeert

h : matrix voor de hulpvariabele die de horizontale gradiënt van de latente afbeelding representeert.

v : matrix voor de hulpvariabele die de verticale gradiënt van de latente afbeelding representeert.

### Algoritmen

#### Algoritme : initialize

Creëert en initialiseert een object dat de latente afbeelding representeert.

|  |
| --- |
| **input**: grootte van latente afbeelding |
| size ← opgegeven grootteimage ← nieuwe afbeelding met alle waarden op 0 ter grootte van size x size |
| **output**: zwarte latente afbeelding met grootte size x size |

#### Algoritme : calculateLatenteImage(kernel, blurred\_adj, parameters)

Berekent de latente afbeelding op basis van de geblurde afbeelding en de kernel.

|  |
| --- |
| **Input**: kernel k, blurred afbeelding b, lambda pixel lp en lambda grad lg |
| Initialize: l ← bmuMax ← 1E5betaMax ← 2³size[] ← [hoogte, breedte] van bKfft ← psf2otf(k, size)denomin2 ← abs(psf2otf(fx,size))²+abs(psf2otf(fy,size))²nomin1 ← conj(Kfft).\*fft(b)den\_ker ← abs(Kfft)²beta ← 2 \* lp**while** beta < betaMax mu ← 2\* lg u ← solveU **while** mu < muMax  Denom ← den\_ker + mu \* denomin2 + beta g ← solveG dg ← calucate gradientG fg ← fft(dg) sfft ← normin1 + mu\*fg + beta\*fft(u) / denomrin l ← real(ifft(fs)) mu ← 2 mu **end** beta ← 2 beta**end** |
| **output: l**atent afbeelding l |

Opmerkingen:

psf2otf = Converteert de kernel van het ruimtedomein naar het frequentiedomein

FFT = 2D Fast Fourier Transform

conj = de complex conjugate van een FFT-transformatie

#### Algoritme : solveG

Berekent de hulpvariabelen h en v voor de gradiënten van de latente afbeelding. Is als private methode geïmplementeerd.

|  |
| --- |
| **Input**: latente afbeelding *s*, lambda gradiënt *lg*, mu  |
| h ← x gradiënt van s met extra kolom met de waarde s eerste kolom – s laatste kolomv ← y gradiënt van s met extra rij met de waarde s eerste rij – s laatste rijt← h² + v²**for** **each** pixel in t **if** waarde pixel < lg/mu **then** zet corresponderende waarde van v en g op 0 **end** **if****end** |
| **output**: variabelen h en v  |

Merk op dat de afmetingen van h en v gelijk moeten zijn aan die van de latente afbeelding om de FFT-berekening uit te kunnen voeren. Om die reden worden deze circulair berekend. Dit houdt in dat de gradiënt van de laatste rij of kolom wordt berekend uit het verschil met de eerste rij of kolom.

#### Algoritme : gradientG

Berekent de gecombineerde gradiënt van g. Is als private methode geïmplementeerd.

|  |
| --- |
| **Input**: de h en v matrixes van G |
| dgx ← laatste kolom van h – eerste kolom van h aangevuld met x gradient van hdgy ← laatste rij van v – eerste rij van v aangevuld met y gradient van vdg = dgx + dgy |
| **output**: variabele dg  |

Merk op dat de afmetingen van dgx en dgx gelijk moeten zijn aan die van h en v om de FFT-berekeningen uit te kunnen voeren. Om die reden worden deze circulair berekend.

#### Algoritme :solveU

Berekent de hulpvariabele u. Is als private methode geïmplementeerd.

|  |
| --- |
| **input** latente afbeelding *s*, lambda pixel *lp*, beta |
| u ← s²**for** **each** pixel in u **if** waarde pixel < lp / beta **then** zet corresponderende waarde van v en g op 0 **end** **if****end** |
| **output:** variabele u |

#### Algoritme : calculateLaPlacian

Berekent de latente afbeelding volgens de LaPlacian-methode op basis van de geblurde afbeelding met bekende kernel.

|  |
| --- |
| **Input**: blurredImage b, kernel k, lambda\_tv tv, lambda\_l0, weight\_ring |
| h ← hoogte van bw ← breedte van bchannels ← aantal channels van bplanes[] ← split b per channel**for** **each** channel imageResized ← resizeForFFT(planes[i], k size, factor =1) latent[i] ← deblurring\_adm\_aniso(imageResized, k, tv)**end**latent ← voeg latent[i] samenlatent ← resize terug naar h en w (dus verwijder padding)**if** (weight\_ring != 0) Latent\_Lo = calculateLatent(imageResized, kernel, lambda\_L0,2) Latent\_Lo = resize terug naar h en w (dus verwijder padding) diff = latent – Latent\_Lo bf\_diff = bilateral\_filter(diff,3,0.1) (check if opencv bilateralFilter returns same results) result = latent – wieght\_ring \* bf\_diff**else** result = latent**end if** |
| **output** : de definitieve latente afbeelding. |

#### Algoritme : deblurring\_adm\_aniso

Hulpmethode van de methode calculateLaplacian. Berekent de latente afbeelding met een Hyper-Laplacian prior en bekende kernel.

|  |
| --- |
| **Input**: imageResized b, kernel k, lambda\_tv tv) |
| i ← bsize ← (hoogte van b, breedte van b)otfK ← psf2otf(k,size)nomin1 = conj(otfK).\*fft2(b)denom1 ← abs(otfK).^2denomin2 ← abs(psf2otf(fx,size))²+abs(psf2otf(fy,size))²beta = 1/ tvbeta\_min = 0.001beta\_rate = 2\*sqrt(2)**while**( beta>beta\_min) bereken h en v met I gamma = 1/(2\*beta) denom = denom1 + gamma \* denom2 Wh = max(abs(h) – beta\*tv, 0).\* sign(h) Wv = max(abs(v) – beta\*tv,0).\* sign(v) wxx ← gradientG(wh, wv) out ← (nomin1 + gamma\*fft2(wxx))./Denom I ← real(ifft2(out)) beta = beta /2**end while** |
| **output**: image I |

## Klasse: ImageParameter

Generieke klasse voor een parameter die een afbeelding representeert. Afbeeldingen kunnen zowel RGB- als grijswaarden hebben.

### Atributen

Image: : Een matrix die een afbeelding in RGB- of grijswaarden representeert.

### Algoritmen

Omdat de methoden van de klasse ImageParameter generieke wiskundige bewerkingen zijn, wordt hier volstaan met een summiere beschrijving van de belangrijkste methoden die niet triviaal zijn. Voor meer details wordt verwezen naar de programmacode van deze klasse.

#### Algoritme: getXgrad

Geeft de x-gradiënt van een afbeelding berekend als xn – xn-1. .

#### Algoritme: getYgrad

Geeft de y-gradient van een afbeelding berekend als yn – yn-1 .

#### fft2 en ifft2

De methode fft2 berekent de 2D Fast Fourier Transform van een matrix. De methode ifft2 is de inverse hiervan.

#### ConjFFT

Berekent de 2D Fast Fourier Transform van de afbeelding en geeft alleen de complex conjugate terug.

#### psf2otf(size, kernel, otf)

Transformeert de kernel van het ruimtedomein naar het frequentiedomein, waarbij de grootte wordt aangepast aan de opgegeven grootte.

#### Otf2psf (otf, size)

De inverse van de psf2otf-methode. Transformeert de kernel van het frequentiedoemin naar het ruimtedomein, waarbij de grootte wordt aangepast aan de opgegeven grootte.

#### conv2(filter)

Voert een 2D-convolutie uit op het image-attribuut met het opgegeven filter.

#### dst(src) en idst(src)

De methode dst(src) voert een discrete sinustransformatie uit op de opgegeven matrix src. De methode idst(src) is de inverse hiervan.

**Literatuurlijst**

Larman, C. 2004. *Applying UML and patterns, an introduction to object-oriented analysis and design and iterative development* (3rd ed.). Prentice Hall PTR, Upper Saddle River, NJ.

Pan, J., Hu, Z., Su, Z., and Yang, M. H. 2014. Deblurring text images via L0-regularized intensity and gradient prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 2901-2908. doi:10.1109/CVPR.2014.371

Pan, J., Hu, Z., Su, Z., and Yang, M. H. 2014. *Deblurring text images via L0-regularized intensity and gradient prior*. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2901-2908. doi:10.1109/CVPR.2014.371

**Niet-wetenschappelijke publicaties**

Free Software Foundation. z.d. 3.10 Options That Control Optimization. Geraadpleegd op 30 mei 2016 van <https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html>.

Sinha, U. z.d. OpenCV vs VXL vs LTI: Performance Test. Geraadpleegd op 30 mei 2016 van <http://aishack.in/tutorials/opencv-vs-vxl-vs-lti-performance-test>.