

Федеральное государственное автономное образовательное
учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С. П. Королёва»

Факультет информатики
Кафедра технической кибернетики

Выпускная квалификационная работа бакалавра
на тему

**Выделение элементов изображения глазного дна с
использованием вейвлет-преобразования**

Выпускник _____ М. А. Ионкин
(подпись)

Научный руководитель _____ Н. Ю. Ильясова
(подпись)

Нормоконтролёр _____ С. В. Суханов
(подпись)

Рецензент _____

Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное
учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С. П. Королёва»

Факультет информатики
Кафедра технической кибернетики

УТВЕРЖДАЮ

Заведующий кафедрой

_____ В.А. Сойфер

«_____» _____ 20____ г.

**ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

студенту 6407 Б 302 группы *Ионкину Михаилу Анатольевичу*

Тема работы *Выделение элементов изображения глазного дна с использованием вейвлет-преобразования*

утверждена приказом по университету от «8» апреля 2016 г. № 474-ст.

Исходные данные: базы данных изображений глазного дна, метод вейвлет-преобразования, метод поля направлений, пакет на языке Java для выполнения дискретного вейвлет-преобразования над матрицами.

Структурные части работы:

- *Обзор и анализ существующих методов выделения сосудов.*
- *Исследование применимости существующих вейвлетов к выделению сосудов и конструирование новых методов выделения областей интереса на изображениях глазного дна. Реализация программного комплекса для осуществления вейвлет-преобразований.*
- *Программная реализация алгоритма построения поля направления. Адаптация и реализация алгоритма построения поля направления с использованием вейвлет-преобразования. Реализация алгоритма выделения центральных линий сосудов.*
- *Исследование выделяемых областей интереса.*

Научный руководитель
профессор каф. технической
кибернетики, д.т.н., доцент

_____ Н. Ю. Ильясова
(подпись)
«_____» 20____ г.

Задание принял к исполнению

_____ М. А. Ионкин
(подпись)

«_____» 20____ г.

РЕФЕРАТ

Выпускная квалификационная работа бакалавра: 85 с., 19 рисунков, одна таблица, 12 использованных источников, одно приложение.

Презентация: 10 слайдов в формате PDF.

ВЕЙВЛЕТ, ДВУМЕРНОЕ ВЕЙВЛЕТ-ПРЕОБРАЗОВАНИЕ, ОБРАБОТКА ИЗОБРАЖЕНИЙ, ЦЕНТРАЛЬНЫЕ ЛИНИИ СОСУДОВ, ПОЛЕ НАПРАВЛЕНИЙ

В работе исследуется применение вейвлет-преобразований к выделению областей интереса глазного дна: сосудов, экссудатов и макул. Оцениваются уже существующие вейвлеты и предлагаются новые.

Реализованы методы построения поля направления и метод выделения центральных линий сосудов.

Проведены исследования зависимости получаемых результатов от параметров вейвлет-преобразований.

По результатам работы создана программа на платформе Java Virtual Machine 8. В программе реализованы вейвлет-преобразования, методы построения полей направлений и методы выделений центральных линий сосудов, а также интерфейс, обеспечивающий удобный ввод и вывод на экран изображений.

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

CWT непрерывное вейвлет-преобразование;

DWT дискретное вейвлет-преобразование;

$2D\ CWM$ двумерный комплексный вейвлет Морле;

\hat{f} преобразование Фурье функции f ;

f^* комплексно-сопряженная к f функция;

Δf лапласиан функции f ;

$L^p(\mathbb{R}^n)$ пространство измеримых функций $\{f(x), x \in \mathbb{R}^n\}$, p -я ($p \geq 1$) степень которых абсолютно интегрируема;

$\|f\|$ норма функции f гильбертова пространства;

$\langle f, \phi \rangle$ скалярное произведение функций $f(x), \phi(x) \in L^2(\mathbb{R}^n)$;

$|x|$ евклидова норма вектора x .

СОДЕРЖАНИЕ

Введение	8
1 Обзор и анализ существующих методов выделения сосудов	9
1.1 Метод морфологических амёб	9
1.2 Метод с использованием вейвлета Морле	10
1.3 Итог обзора	10
2 Исследование применения вейвлет-преобразования к выделению областей интереса на изображении глазного дна	11
2.1 Одномерное вейвлет-преобразование	11
2.2 Двумерное вейвлет-преобразование	13
2.3 Способы двумерного непрерывного вейвлет-преобразования	14
2.4 Вейвлет Морле	16
2.5 Вейвлет, согласованный с моделью сосудов	16
2.6 Вейвлет Добеши 4-го порядка	18
2.7 Конструирование вейвлета, выделяющего сосуды	19
2.8 Конструирование вейвлета, выделяющего дискообразные объекты	20
2.9 Реализация вейвлет-преобразований	21
3 Выделение центральных линий сосудов	25
3.1 Предобработка	25
3.2 Алгоритм нахождения поля направлений с использованием проекционно-дисперсионного метода	26
3.3 Алгоритм нахождения поля направлений с использованием вейвлет-преобразования	26
3.4 Сравнение алгоритмов нахождения поля направлений . . .	28
3.5 Выделение центральных линий	30
4 Исследование выделяемых объектов	32
4.1 Исследование зависимости ширины выделяемых линий от параметра вейвлет-преобразования	32
4.2 Исследование качества выделения областей интересов . .	32
Заключение	36
Список использованных источников	37
Приложение А Исходный код программы	39
A.1 Лицензия	39
A.2 basic.Decompose.scala	39
A.3 basic.ArrayToolKit.scala	40
A.4 basic.Types.scala	40

A.5	basic.MatrixRotate.scala	42
A.6	basic.MatrixUpdate.scala	43
A.7	basic.Fourie.scala	43
A.8	basic.Integral.scala	44
A.9	basic.Constants.scala	45
A.10	basic.Statistic.scala	45
A.11	wavelets.AsVessel.scala	47
A.12	wavelets.Gauss.scala	47
A.13	wavelets.ICWavelet.scala	48
A.14	wavelets.FHAT.scala	48
A.15	wavelets.OldAsVessel.scala	48
A.16	wavelets.Gabor.scala	50
A.17	wavelets.Daubechies.scala	50
A.18	wavelets.Morlet.scala	51
A.19	wavelets.DWaveletTrait.scala	51
A.20	wavelets.MHAT.scala	52
A.21	wavelets.AsLongVessel.scala	52
A.22	wavelets.ACBoundWavelet.scala	53
A.23	transform.TransformTrait.scala	53
A.24	transform.DTransform.scala	56
A.25	transform.WaveletPacketTransform.scala	57
A.26	transform.CTransform.scala	58
A.27	transform.OldTransform.scala	58
A.28	transform.AncientEgyptianDecomposition.scala	59
A.29	main.WMain.scala	59
A.30	exceptions.BinaryAmountException.scala	61
A.31	exceptions.AmountItemsException.scala	61
A.32	tests.AnalysTest.scala	61
A.33	tests.CWTTest.scala	62
A.34	tests.DWTTest.scala	63
A.35	tests.ImageTest.scala	65
A.36	tests.AllTest.scala	66
A.37	accentuation.Line.scala	67
A.38	accentuation.Disk.scala	69
A.39	accentuation.Vessel.scala	70
A.40	image.Accentuation.scala	73
A.41	image.Generate.scala	73
A.42	image.Transform.scala	75
A.43	image.Analys.scala	77

A.44	image.Output.scala	80
A.45	image.Input.scala	81
A.46	image.Operation.scala	82

ВВЕДЕНИЕ

Выделение объектов интереса на изображении помогает врачу произвести диагностику таких заболеваний, как гипертония, сахарный диабет, атеросклероз, сердечно-сосудистые заболевания и инсульт, а также наиболее распространенную причину слепоты — диабетическую ретинопатию[1]. Уже созданы приборы (напр., Navitel), использующие сведения о выделенных сосудах.

Автоматизация этого процесса позволяет:

- перенести рутинную работу на компьютер;
- ускорить процесс диагностики;
- уменьшить фактор субъективности;
- уменьшить вероятность ошибки;
- удешевить диагностику.

С 1980-х годов вейвлеты стали все более активно применяться в различных отраслях [2, 3]. В [1] описано применение вейвлета Морле (см. подраздел 2.4) в процессе сегментации кровеносных сосудов, и приводится результат выделения сосудов. На взгляд автора, применение вейвлетов в медицине в русскоязычных источниках недостаточно освещено.

В первом разделе описаны существующие методы выделения сосудов, произведена их оценка.

Во втором разделе кратко изложена теория вейвлетов. Каждый описанный вейвлет оценивается на возможность выделения объектов интереса и тестируется. Предлагается к использованию для обработки изображений глазного дна два новых вейвлета.

В третьем разделе изложены используемые методы: метод поля направлений и метод выделения центральных линий, а также их реализации и примеры.

В четвертом разделе вейвлеты исследуются на качество выделения определенных областей интереса. Проведены исследования зависимости параметров вейвлета и параметров полученных после преобразования объектов, а также проведено сравнение изображений выделенных сосудов с эталонным выделением.

1 ОБЗОР И АНАЛИЗ СУЩЕСТВУЮЩИХ МЕТОДОВ ВЫДЕЛЕНИЯ СОСУДОВ

Согласно [4], можно выделить следующие классы методов обнаружения сосудов на изображениях глазного дна:

- а) методы, использующие свёртку изображений с двумерным направленным фильтром и последующую максимизацию (или минимизацию) результатов свертки по используемым направлениям;
- б) методы, использующие детектирование хребтов (осей сосудов);
- в) методы, использующие трекинг сосудов (один из вариантов трекинга – итерационный процесс выделения сосудов по сосудам, полученным на предыдущих итерациях). К преимуществам этого класса можно отнести высокую точность работы на тонких сосудах, и восстановление разрывных сосудов, а к недостаткам – сложность обработки ветвлений и пересечений;
- г) попиксельная классификация на основе вектора признаков с использованием машинного обучения.

К методам класса (а) можно отнести двумерное дискретное (и дискретизированное) вейвлет-преобразование. Однако, согласно [4], при использовании методов этого класса возможны ложные срабатывания.

1.1 Метод морфологических амёб

Метод морфологических амёб позволяет по множеству точек, заранее являемымися сосудами, построить их продолжения [4]. Таким образом, его можно отнести к классу методов, использующих трекинг сосудов. В методе можно регулировать достоверность получаемых на следующем шаге новых пикселей сосудов с помощью штрафной функции – типичные добавляемые точки будут с близкой яркостью. К недостаткам метода можно отнести:

- требование существования начального набора пикселей, являющихся сосудами;
- отсутствие учитывания формы выделяемого объекта, что может привести к ложным срабатываниям на объекты схожей яркости, расположенные непосредственно около сосудов.

В работе [5] начальные пиксели получаются посредством использования последовательного применения морфологического открытия и закрытия, вычитания из полученного изображения исходного и применения серии направленных фильтров Габора с различными масштабами, с применением функции максимизации яркости в каждом пикселе.

1.2 Метод с использованием вейвлета Морле

В работе [1] описано применение вейвлета Морле (см. подраздел 2.4) в процессе сегментации кровеносных сосудов. Для уменьшения ошибок на границах области глазного дна используется её расширение. Используется инверсный зеленый канал изображения. Подбираются оптимальные параметры Морле, и выполняются преобразования под различными углами, с максимизацией результата.

В недостаткам можно отнести:

- необходимость подбора параметров;
- недостаточное учитывание формы выделяемых объектов (сосудов), что может привести к ложным срабатываниям;
- зеленый канал лишь в некоторых базах изображений близок к оптимальному выделению. В общем случае, для каждой базы лучше подбирать индивидуальные коэффициенты компонентов цветов.

1.3 Итог обзора

Можно отметить, что в качестве предобработки или непосредственно в процессе выделения сосудов используют направленные преобразования. Однако, форма выделяемого объекта, а также его цвет, на взгляд автора, учитывается слабо.

Исследование применимости существующих вейвлетов к выделению сосудов и конструирование новых методов выделения областей интереса на изображениях глазного дна. Реализация программного комплекса для осуществления вейвлет-преобразований.

2 ИССЛЕДОВАНИЕ ПРИМЕНЕНИЯ ВЕЙВЛЕТ-ПРЕОБРАЗОВАНИЯ К ВЫДЕЛЕНИЮ ОБЛАСТЕЙ ИНТЕРЕСА НА ИЗОБРАЖЕНИИ ГЛАЗНОГО ДНА

2.1 Одномерное вейвлет-преобразование

2.1.1 Одномерные вейвлеты

Рассмотрение одномерных вейвлетов способствует пониманию двумерных вейвлетов, используемых в том числе при обработке изображений. Стоит отметить, что дискретные двумерные вейвлеты строятся по одномерным, а теория одномерных непрерывных вейвлетов с небольшими изменениями обобщается на двумерный случай.

Стефан Маллат в [6] определяет одномерный вейвлет как функцию $\psi(t) \in L^2(\mathbb{R})$ с нулевым средним:

$$\int_{\mathbb{R}} \psi(t) dt = 0, \quad (2.1)$$

с единичной нормой:

$$\int_{\mathbb{R}} |\psi(t)|^2 dt = 1, \quad (2.2)$$

сосредоточенную в окрестности точки $t = 0$.

Временно-частотные коэффициенты $\{\psi_{b,s}\}$ вычисляются путем масштабирования ψ по s и смещения по b :

$$\psi_{b,s}(t) = \frac{1}{\sqrt{s}} \psi\left(\frac{t-b}{s}\right), \quad b \in \mathbb{R}, s \in \mathbb{R}^+. \quad (2.3)$$

Дискретный случай более сложен для описания [6]:

- регулярность дискретной последовательности не определена;
- необходимо сохранение свойств ортогональности;
- для него нужно отдельно рассматривать граничные случаи.

Однако, дискретные алгоритмы требуют вычисления лишь конечного числа свёрток.

Переход от непрерывности к дискретности дает формулу для вычисления вейвлет-коэффициентов [7]:

$$\psi_{n,i}(t) = \frac{1}{\sqrt{2^i}} \psi\left(\frac{t}{2^i} - n\right), \quad n \in \mathbb{Z}, i \in \mathbb{N}. \quad (2.4)$$

Примечание - Вместо числа ‘2’ в формуле (2.4), вообще говоря, может стоять и другое положительное число.

2.1.2 Масштабная функция

Скейлинг-функцией, или масштабной функцией (см. [3, 6]) называют функцию φ , удовлетворяющую уравнению

$$\varphi(x) = \sqrt{2} \sum_{k \in \mathbb{Z}} h[k] \varphi(2x - k), \quad (2.5)$$

где $\{h[k]\}$ – некоторый набор коэффициентов, определяемых вейвлет-функцией.

Для скейлинг-функции должно выполняться условие нормировки:

$$\int_{\mathbb{R}} \varphi(x) dx = 1, \quad (2.6)$$

и, согласно [6] (пункт 7.1.4), соотношение

$$\hat{\psi}(\omega) = \frac{1}{\sqrt{2}} e^{-i\omega} \hat{h}^*(\frac{\omega + \pi}{2}) \hat{\varphi}(\frac{\omega}{2}). \quad (2.7)$$

2.1.3 Вейвлет Хаара

Один из самых известных вейвлетов – вейвлет Хаара. Он также один из самых первых вейвлетов. Задаётся формулой

$$\psi_{haar}(x) = \begin{cases} 1, & 0 \leq x < 0,5; \\ -1, & 0,5 \leq x < 1; \\ 0, & \text{иначе.} \end{cases} \quad (2.8)$$

Однако, часто применяют его дискретный вариант: каждой паре чисел исходного набора ставится в соответствие их полусумма и полуразность. Нормированные коэффициенты – числа $1/\sqrt{2}$ и минус $1/\sqrt{2}$.

Этот вейвлет позволяет детектировать границы, т.к. на однородном фоне в полуразности будет число, близкое к нулю, в отличие от полуразности на контрастном слое. Однако, автор не видит обоснования для его применения при выделении сосудов, т.к. это более протяженные объекты.

2.1.4 Семейство вейвлетов Добеши

Вейвлеты Добеши – дискретные вейвлеты четных порядков, что означает, что количество используемых коэффициентов четно. Вейвлет Добеши 2-го порядка совпадает с вейвлетами Хаара (дискретным).

Рассмотрим скейлинг-функцию вида

$$\varphi(x) = \sqrt{2} \sum_{k=0}^N d_k \varphi(2x - N). \quad (2.9)$$

Возьмём $N = 3$ и определим коэффициенты $\{d_k\}$ (см. [8]):

$$d_0 = \frac{1 + \sqrt{3}}{4}, \quad d_1 = \frac{3 + \sqrt{3}}{4}, \quad d_2 = \frac{3 - \sqrt{3}}{4}, \quad d_3 = \frac{1 - \sqrt{3}}{4}. \quad (2.10)$$

Согласно [3] вейвлет-коэффициенты будут однозначно определяться через коэффициенты масшабной функции:

$$\phi(x) = \sqrt{2} \sum_{k=0}^N g_k \varphi(2x - k), \quad (2.11)$$

где $g_k = (-1)^k d_{N-k}$ — вейвлет-коэффициенты.

Определенный вейвлет — вейвлет Добеши 4-го порядка.

Автор не видит предпосылок для качественного выделения объектов с использованием вейвлетов этого семейства.

2.1.5 Вейвлет Марра, или мексиканская шляпа

Вейвлет Марра используется для анализа неориентированных объектов. Описывается уравнениями вида

$$\psi_H(x) = (-\Delta)^n \exp\left\{-\frac{1}{2}|x|^2\right\}, \quad (2.12)$$

где $n = 1, 2, \dots$

Вейвлет Марра характеризуется тем, что при увеличении n все больше его моментов равны нулю [9]. Применение вейвлета Марра в обработке изображений описано, например, в [10] (в частности, с помощью этого преобразование происходило выделение границ).

2.2 Двумерное вейвлет-преобразование

Согласно [9], двумерный вейвлет есть комплексная функция $\psi(x) \in L^2(\mathbb{R}^2)$, для которой выполняется условие

$$c_\psi \equiv (2\pi)^2 \int_{\mathbb{R}^2} |\hat{\psi}(x)|^2 \frac{dx}{|x|^2} < \infty. \quad (2.13)$$

Если ψ — регулярная в \mathbb{R}^2 , то из (2.13) следует выполнение условия

$$\hat{\psi}(0) = 0 \Leftrightarrow \int_{\mathbb{R}^2} \psi(x) d^2x = 0. \quad (2.14)$$

Тривиальных подход выполнения двумерного дискретного вейвлета — последовательное применение одномерных: по всем строкам, по всем столбцам, или по всем строкам, а затем результат — по столбцам [3].

Вейвлет-коэффициенты двумерного преобразования можно найти из уравнения

$$\psi_{j_1, k_1, j_2, k_2}(x_1, x_2) = \psi_{j_1, k_1}(x_1) \psi_{j_2, k_2}(x_2). \quad (2.15)$$

Однако часто использую другой способ: строят вейвлет

$$2^j \psi(2^j x - k, 2^j y - l),$$

где $j, k, l \in \mathbb{Z}$ [3]. Для построения рассмотрим масштабную функцию вейвлета.

2.2.1 Построение двумерных дискретный вейвлетов

Пусть имеется скейлинг-функция φ и одномерный вейвлет $\psi(x - k)$. Тогда мы можем (см. [3]) сконструировать три двумерных дискретных вейвлета:

$$\begin{cases} \Psi_{j,k,l}^{[1]}(x,y) = 2^j \varphi(2^j x - k) \psi(2^j y - l), \\ \Psi_{j,k,l}^{[2]}(x,y) = 2^j \psi(2^j x - k) \varphi(2^j y - l), \\ \Psi_{j,k,l}^{[3]}(x,y) = 2^j \psi(2^j x - k) \psi(2^j y - l). \end{cases} \quad (2.16)$$

Один из этих вейвлетов хорошо обрабатывает вертикальные линии, другой — горизонтальные, ещё один — диагональные. Недостаток данных вейвлетов в том, что в такой трактовке они могут применяться лишь к квадратным (по размеру) изображениям. В зависимости от выбранного параметра j также лучше различаются мелкие или крупные детали [3].

2.3 Способы двумерного непрерывного вейвлет-преобразования

Пусть $f(x) \in L^2(\mathbb{R}^2)$ — функция, которую можно разложить по ортогональному базису $\psi(x)$.

Двумерное вейвлет-преобразование с вейвлетом ψ и сигналом $f(x)$

может определяться [1] уравнением вида

$$W_\psi^f(b,\theta,a) = c_\psi^{-1/2} \frac{1}{a} \int_{\mathbb{R}^2} f(x) \psi^*(r_{-\theta} \frac{x-b}{a}) d^2 x, \quad (2.17)$$

где r_θ — матрица вращения:

$$r_\theta = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix};$$

$c_\psi \in \mathbb{R}_+$ — нормализующая константа из уравнения (2.13);

$b \in \mathbb{R}^2$ — вектор смещения;

$\theta \in [0, 2\pi)$ — угол поворота;

$a \in \mathbb{R}_+$ — параметр масштаба.

Такой тип вейвлет-преобразования применяется, в частности, к ориентированным¹⁾ вейвлетам.

Запишем дискретизированный вариант данного преобразования:

$$Wd_\psi^f(y,x,\theta,a) = cd_\psi^{-1/2} \frac{1}{a} \sum_{k=0}^M \sum_{l=0}^N f(k,l) \psi^*(r_{-\theta} \frac{(k-y, l-x)}{a}), \quad (2.18)$$

где $cd_\psi \in \mathbb{R}_+$ — нормализующая константа из уравнения

$$cd_\psi \equiv (2\pi)^2 \sum_{k=0}^M \sum_{l=0}^N \frac{|\hat{\psi}(k,l)|^2}{|(k,l)|^2};$$

$(x,y) \in \mathbb{Z}^2$ — вектор смещения;

$\theta \in [0, 2\pi)$ — угол поворота;

$a \in \mathbb{R}_+$ — параметр масштаба;

(M, N) — размерность области определения функции f .

Другое вейвлет-преобразование [10] задается уравнением

$$W_\psi^f(b,a) = \frac{1}{\sqrt{a}} \int_{\mathbb{R}^2} f(x) \psi(\frac{x-b}{a}) d^2 x \quad (2.19)$$

и применяется к неориентированным вейвлетам.

¹⁾ т.е. с их помощью него можно определить “направление” объекта в заданной точке изображения (при условии, что направление существует)

2.4 Вейвлет Морле

Вейвлет Морле относится к классу вейвлетов, используемых для анализа ориентированных объектов (сегментов, краев, поля направления и т.д.) [9]. Двумерный комплексный вейвлет Морле (2D CWM) $\psi_M(x) : \mathbb{R}^2 \rightarrow \mathbb{C}$ задается уравнением [9]

$$\psi_M(x) = \exp\{ik_0x\} \exp\left\{-\frac{1}{2}|Ax|^2\right\}, \quad (2.20)$$

где $k_0 \in \mathbb{R}^2$ — волновое число, соответствующее вейвлету ψ ;

$$A = \text{diag}\left[\varepsilon^{-1/2}, 1\right], \varepsilon \geq 1.$$

Точность выполнения условия (2.14) при $\varepsilon = 1$ определяется величиной $\sqrt{2\pi} \exp\left\{-\frac{|k_0|^2}{2}\right\}$: для точности $4 \cdot 10^{-8}$ достаточно, чтобы $|k_0| \geq 6$. Поэтому вектор k_0 выбирается исходя из значения ε и требуемой точности решения задачи.

Заметим, что для любого вещественного ζ

$$|\psi_M(x)| < \zeta \Leftrightarrow \exp\left\{-\frac{1}{2}|Ax|^2\right\} < \zeta \Leftrightarrow \quad (2.21)$$

$$\Leftrightarrow -\frac{1}{2}|Ax|^2 < \ln \zeta \Leftrightarrow |Ax|^2 > \ln \frac{1}{\zeta^2} \Leftrightarrow \quad (2.22)$$

$$\Leftrightarrow \varepsilon x_1^2 + x_2^2 > \ln \frac{1}{\zeta^2} \Leftrightarrow \|(\varepsilon x_1, x_2)\| > -2 \ln \zeta. \quad (2.23)$$

Для аргумента $a^{-1}r(-\theta)(\varepsilon k, l)$ условие малости вейвлета будет:

$$\|a^{-1}r(-\theta)(\varepsilon k, l)\| > -2 \ln \zeta. \quad (2.24)$$

Пример использования и недостатки этого вейвлета были изложены в подразделе 1.2. Ниже на рисунках (2.1) и (2.2) приведен пример использования вейвлета Морле для выделения сосудов, с использованием разработанной автором программы.

2.5 Вейвлет, согласованный с моделью сосудов

В диссертации [11] изложено описание вейвлета, согласованного с моделью сосуда:

$$\phi(x_1, x_2) = \begin{cases} (1 - x_2^2) \exp(-0,25x_2^4), & x_1 \in [-d, d]; \\ 0, & \text{иначе.} \end{cases} \quad (2.25)$$

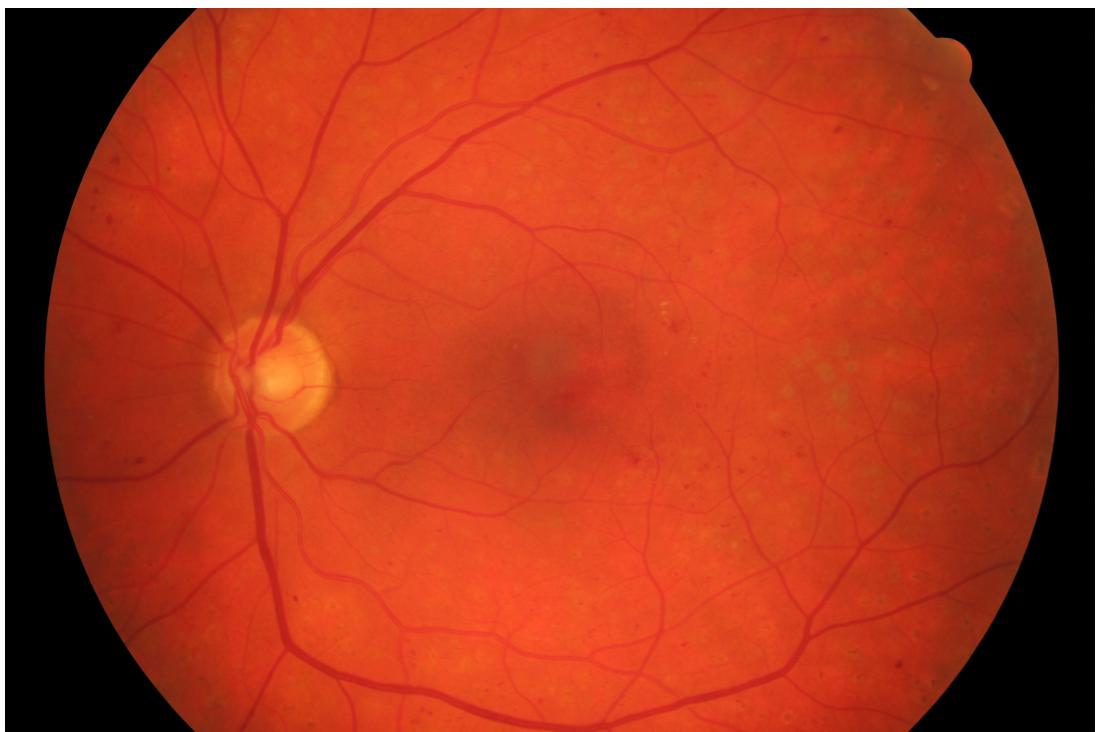


Рисунок 2.1 – Исходное изображение № 1

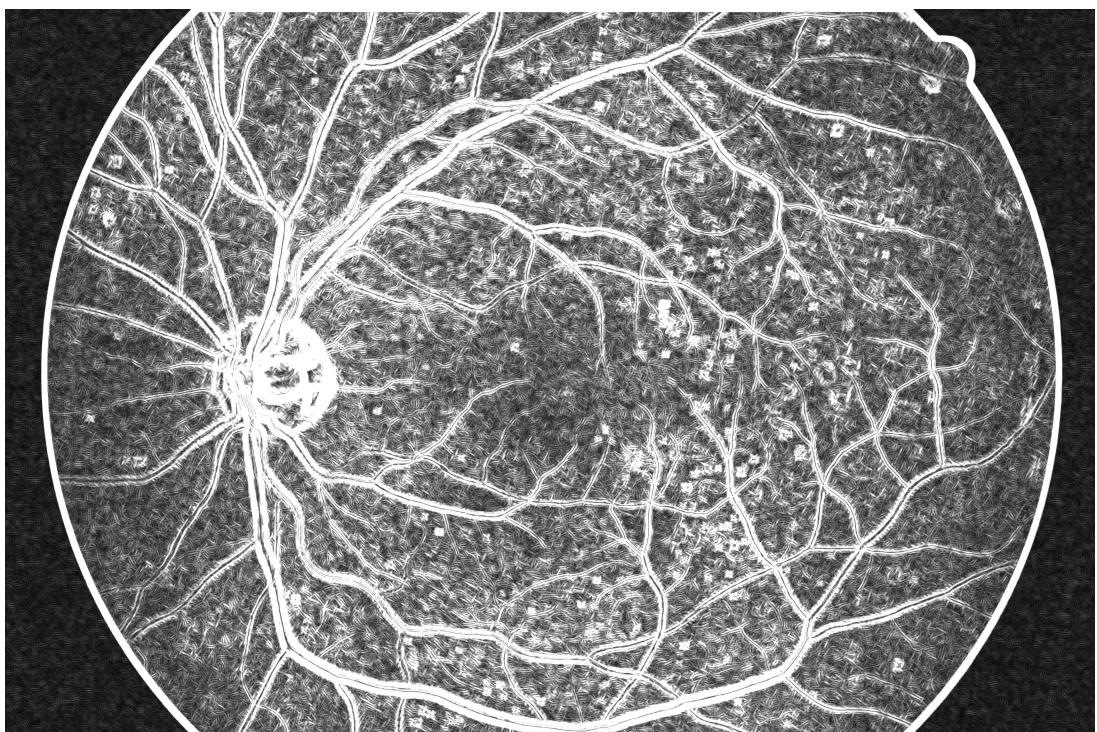


Рисунок 2.2 – Результат вейвлет-преобразования изображения № 1
вейвлетом Морле

где d — целое число, которое можно характеризовать как полуширину сосуда.

Проекция вейвлета на ось x_2 изображена на рисунке (2.3).

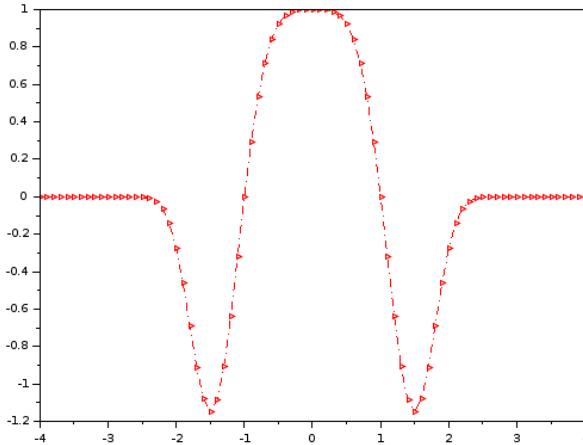


Рисунок 2.3 – Проекция вейвлета, согласованного с моделью сосудов, на ось x_2

К недостаткам этого вейвлета можно отнести то, что другой близкий по цвету (к сосуду) и достаточно объемный объект может быть воспринят как сосуд. Это связано с очень быстрым убыванием экспоненты. Например, на исходном изображении № 2 (рисунок 2.4) в центре и несколько ниже его есть пятна, близкие по цвету к сосудам.

На рисунках (2.4) и (2.5) приведен пример использования вейвлета, задаваемого уравнением (2.25).

2.6 Вейвлет Добеши 4-го порядка

Как уже говорилось ранее, дискретные многомерные вейвлеты образуются путем композиции одномерных. На рисунке (2.6) приведен пример использования вейвлета. Реализация этого (и некоторых других дискретных вейвлетов) есть в последних версиях Matlab, однако, набор параметров там достаточно мал, и полученное изображение нельзя сохранить с максимальным качеством (по крайней мере, в Matlab Toolbox). Поэтому, вейвлеты со 2-го по 8-й с шагом 2 порядков были реализованы (вейвлет 2-го порядка – вейвлет Хаара). На рисунке (2.6) отображен результат выполнения преобразования под углами от 0 до 180 градусов с шагом в 10 градусов (т.е. с поворотами изображения) и последующей максимизацией результата.



Рисунок 2.4 – Исходное изображение № 2

2.7 Конструирование вейвлета, выделяющего сосуды

Один из лучших результатов по выделению сосудов показал вейвлет, описанный в подразделе 2.5. Его отличительной особенностью является несимметричная область: она похожа на тонкий сосуд. Однако, такой вейвлет быстро убывает по модулю, и потому может выделить и достаточно большую область с цветом, аналогичным цвету сосудов.

Предлагаемый вейвлет задается функцией

$$\begin{aligned} \phi(x_1, x_2, \theta) = & \sum_{r=-d}^d f(x_1 + r \sin(\theta), x_2 + r \cos(\theta)) - f(x_1, x_2) - \\ & - \frac{d}{2} f(x_1 + (d+1) \sin(\theta), x_2 + (d+1) \cos(\theta)) - \\ & - \frac{d}{2} f(x_1 + (d+2) \sin(\theta), x_2 + (d+2) \cos(\theta)) - \\ & - \frac{d}{2} f(x_1 - (d+1) \sin(\theta), x_2 - (d+1) \cos(\theta)) - \\ & - \frac{d}{2} f(x_1 - (d+2) \sin(\theta), x_2 - (d+2) \cos(\theta)), \end{aligned} \quad (2.26)$$

где d — целое число, которое можно характеризовать как полуширину

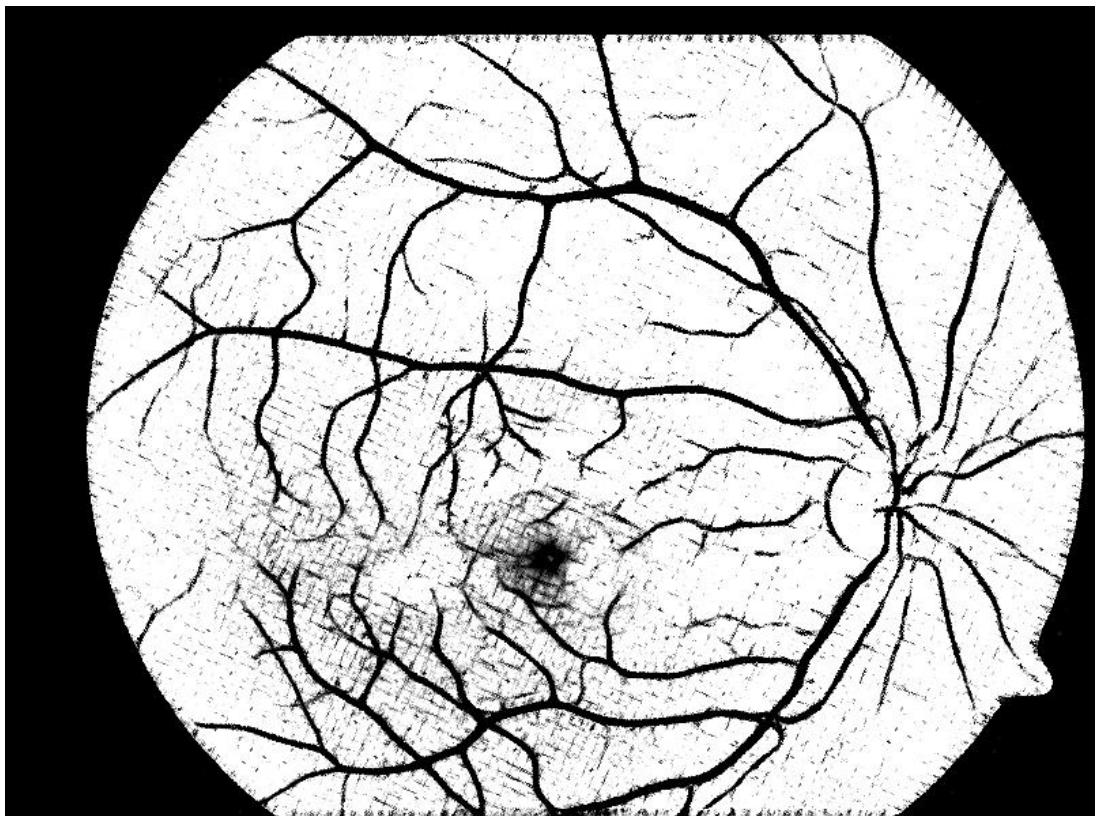


Рисунок 2.5 – Результат вейвлет-преобразования изображения № 2
вейвлетом, согласованным с моделью сосудов

сосуда.

Пример выделения сосудов сконструированным вейвлетом представлен на рисунке (2.7).

2.8 Конструирование вейвлета, выделяющего дискообразные объекты

Желтое пятно находится около центра глазного дна, и по форме близка к овалу. На изображениях глазного дна больного пациента могут встречаться экссудаты (рисунок 2.8). Также, после проведения лазерной коагуляции, на изображении глазного дна можно обнаружить лазеркоагуляты. Экссудаты по форме мало напоминают диск, но они, в отличие от сосудов не вытянуты, лазеркоагуляты же похожи на диск.

Выделение этих объектов предлагается с использованием ненаправленного вейвлета

$$\phi(x, y) = \begin{cases} L \exp\left(-\frac{x^2+y^2}{\sigma^2}\right), & x^2 + y^2 \leq r^2; \\ -\frac{R^2-r^2}{R^2} \sum_{x^2+y^2 \leq r^2} \phi(x, y), & \text{иначе.} \end{cases} \quad (2.27)$$

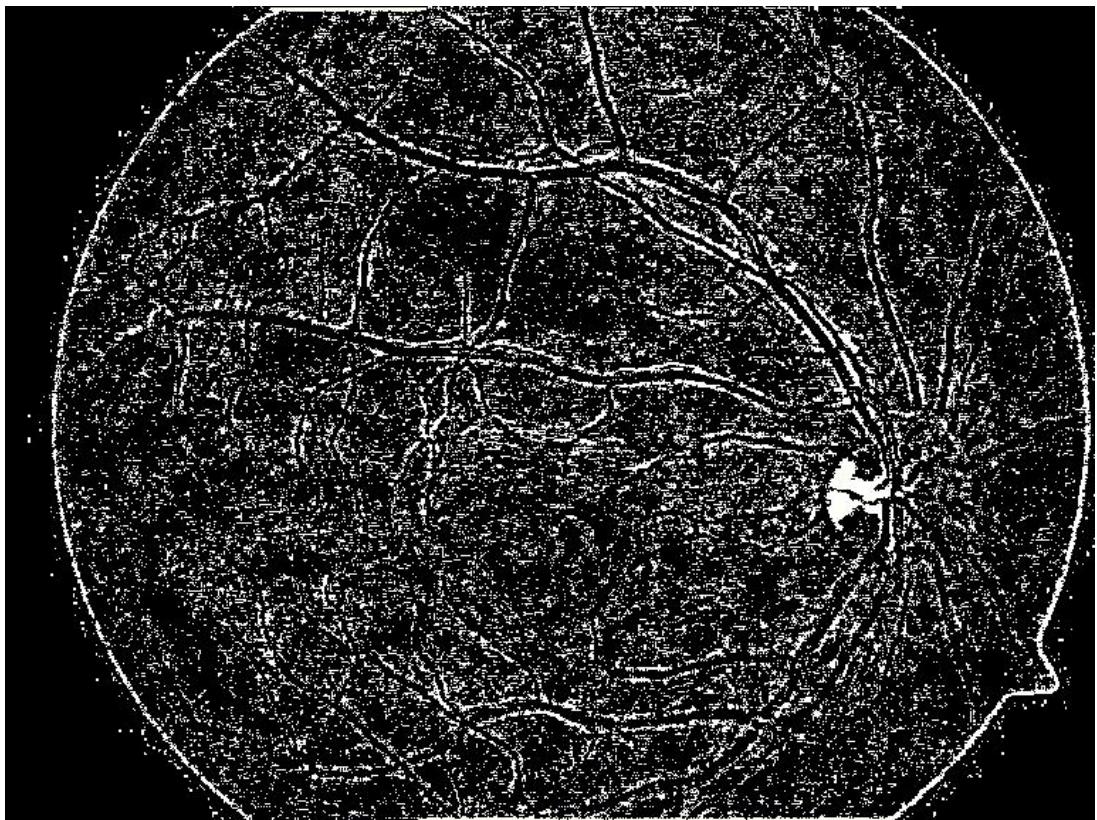


Рисунок 2.6 – Результат вейвлет-преобразования изображения № 1
вейвлетом Добеши 4-го порядка

где L — нормировочный коэффициент;

r — параметр, характеризующий радиус выделяемого диска;

R — параметр, характеризующий радиус маски;

σ — некоторый параметр.

Примечание - Более точно, центр был сформирован из положительных чисел, а по краям от него до радиуса R были симметрично расположены отрицательные числа так, чтобы сумма всех коэффициентов была равна нулю.

В процессе преобразований L был равен 8, а σ – 5.

Результат выполнения вейвлет-преобразования представлен на рисунке (2.9).

2.9 Реализация вейвлет-преобразований

Для удобного выполнения вейвлет-преобразований над изображениями, был создан интерфейс ввода изображений (с возможностью выбора компонентов цвета, используемых для преобразования изображения в матрицу, а также, с возможностью инвертировать полученную матрицу) и вывода их на экран. Непосредственно в программе легко сохранить их

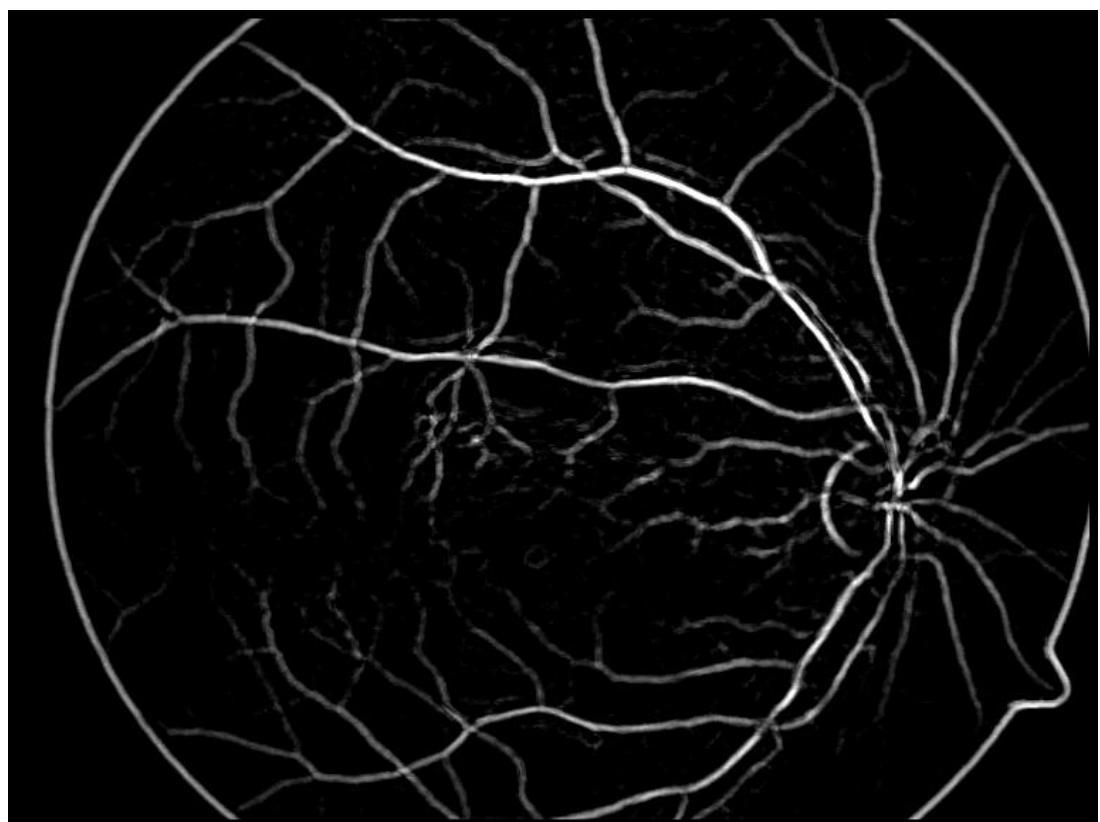


Рисунок 2.7 – Результат вейвлет-преобразования сконструированным
для выделения сосудов вейвлетом

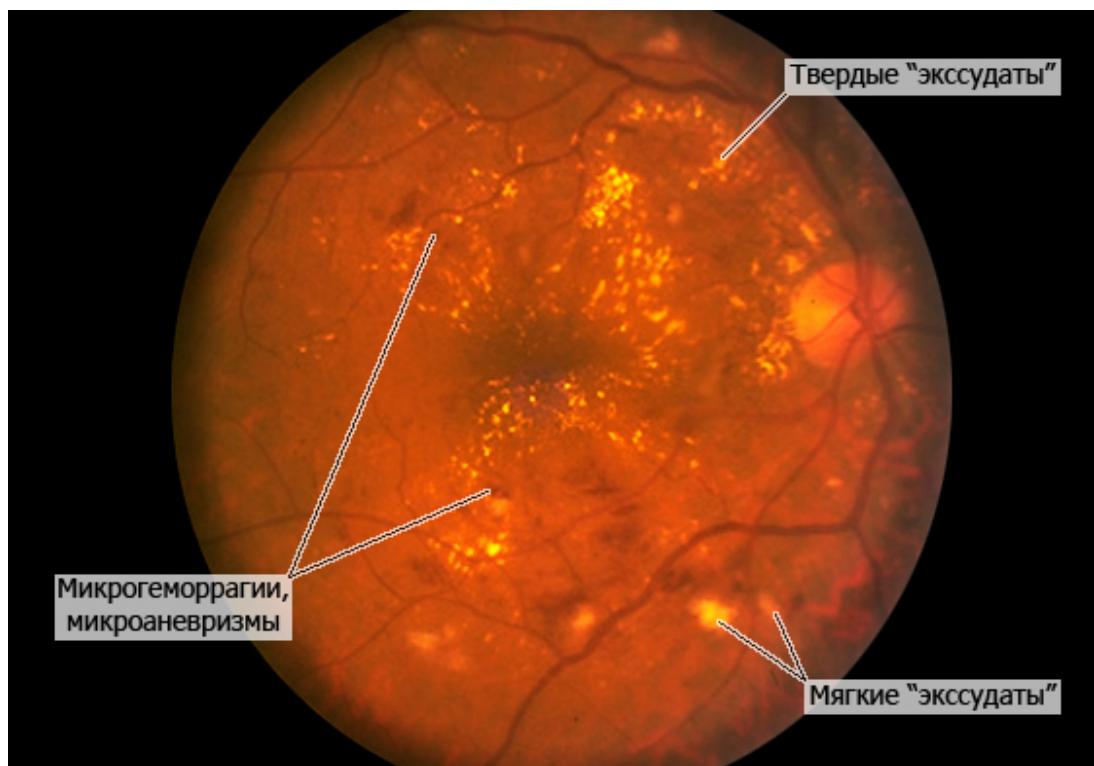


Рисунок 2.8 – Глазное дно при непролиферативной диабетической ретинопатии

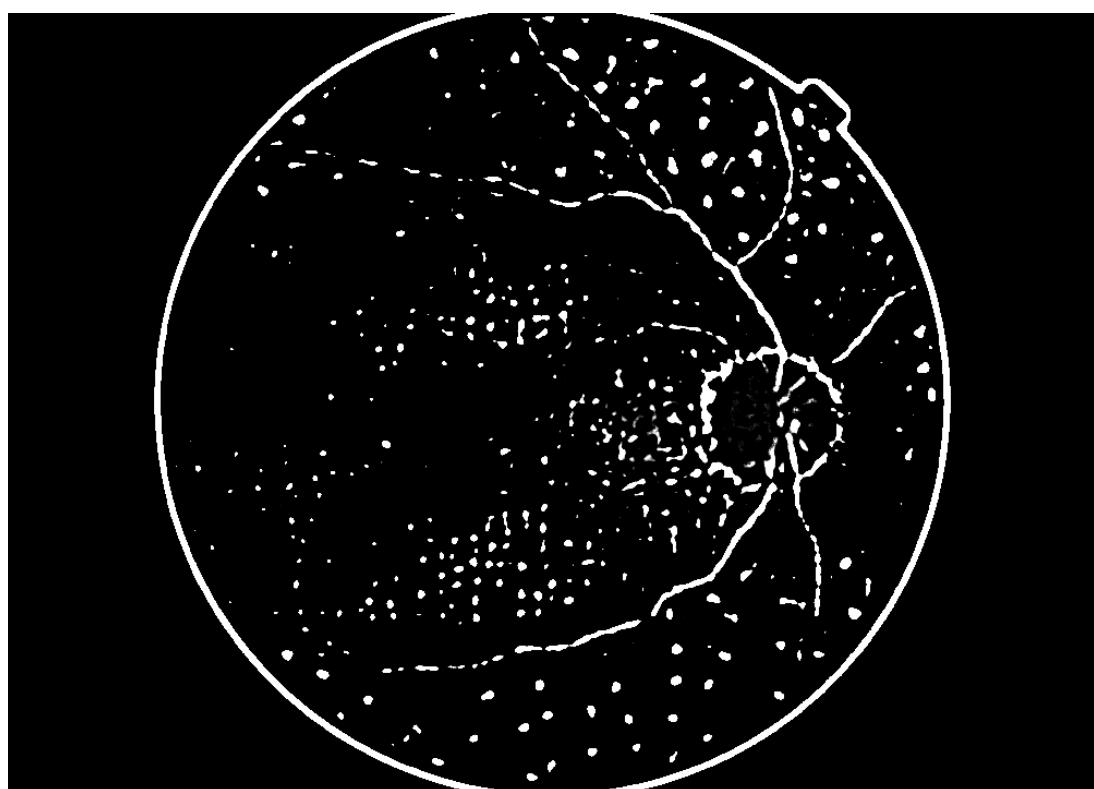


Рисунок 2.9 – Результат выделения дискообразных объектов

в файл большинства форматов изображений.

2.9.1 Дискретное преобразование

Автор использовал проект *JWave*, написанный на языке *Java* Christian Scheiblich (cscheiblich@gmail.com), и расположенный на сайте <https://github.com/cscheiblich/JWave> под лицензией **MIT license**. Автором была исправлена ошибка, связанная с неправильным выполнением преобразования при размерах изображения, не равных степени двойки (по ширине и высоте). Был произведен существенный рефакторинг пакета с целью упрощения кода и оптимизации вычислений.

Реализованное семейство вейвлетов Добеши было протестировано.

2.9.2 Непрерывное вейвлет-преобразование

Непрерывное вейвлет-преобразование изображений, осуществляющее посредством Wavelet Toolbox, существует в версии Matlab 2014-го года, но не существует в версии 2008-го года. Однако, набор углов для выполнения преобразования там очень ограничен: 0, 45, 90 и 135 градусов. Кроме того, в Wavelet Toolbox отсутствует возможность сохранения изображений в оригинальном масштабе. Таким образом, существующая система не подходила для выделения объектов на изображении глазного дна.

Автором были реализованы классы, осуществляющие ввод, вывод и поворот изображений, преобразование изображений в матрицу и обратно. Также, реализованы классы, осуществляющие вейвлет-преобразование, численное интегрирование, и все описанные выше двумерные вейвлеты. Алгоритм численного интегрирования (двумерный вариант метода Симпсона) был протестирован.

3 ВЫДЕЛЕНИЕ ЦЕНТРАЛЬНЫХ ЛИНИЙ СОСУДОВ

Для выделения центральных линий сосудов было использовано поле направлений. Поле направлений совпадает с полем градиентов в каждой точке, повернутое на 90° , и, в отличие от поля градиентов, определено в диапазоне $[0, 180^\circ)$ [12]. Поле направлений в данной работе реализовывалось двумя способами: с использованием вейвлет-преобразования (при получении вейвлет-образа) и с использованием проекционно-дисперсионного метода, изложенного в [12].

3.1 Предобработка

Вейвлет-преобразование над изображением осуществлялось посредством преобразованием над матрицей, достаточно хорошо отражающей выделяемые объекты. Сосуды обычно хорошо выделяются инвертированным зеленым компонентом изображения, но, вообще говоря, выбор только зеленой компоненты не оптимальен – для многих баз данных можно подобрать сочетания, которые лучше отражают, например, сосуды, до или после инверсии. В связи с этим был реализован интерфейс с возможностью выбора коэффициентов преобразования цветовых каналов изображения в матрицу, а также с возможностью инвертировать зеленый канал текущего изображения (рисунок 3.1).

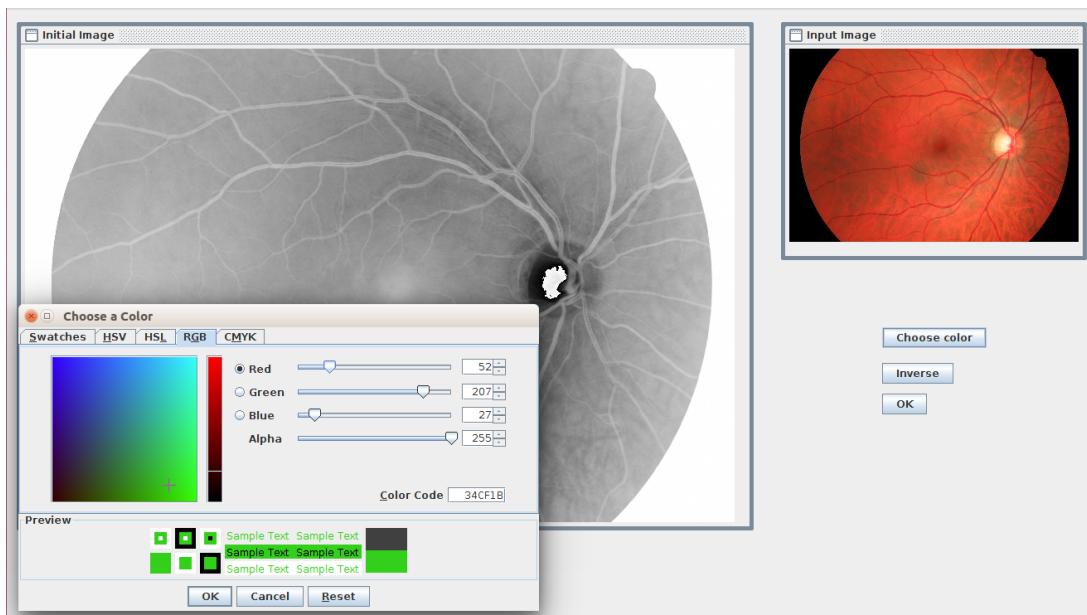


Рисунок 3.1 – Результат предобработки

3.2 Алгоритм нахождения поля направлений с использованием проекционно-дисперсионного метода

Проекционно-дисперсионный метод в своей основе использует нахождение дисперсии функции яркости.

Алгоритм состоит в следующем:

- для каждой точки матрицы для каждого угла из набора в полярных координатах строится отрезок длиной $2R + 1$: в направлении этого угла длиной R , и в противоположном направлении длиной R ;
- точки каждого отрезка определяют множество точек, для которого находится его дисперсия;
- угол, при котором достигнут минимум дисперсии, объявляется направляющим углом в данной точке;
- полученное поле направлений фильтруется медианным фильтром.

К достоинствам метода относится его устойчивость (при $R > 0$ поле направлений зависит от набора точек), к недостаткам – необходимость настройки параметра R и ширины маски, а также (относительно многих других алгоритмов нахождения поля направлений) долгое время работы.

На рисунке (3.4) приведен пример нахождения поля направлений тестового изображения с использованием проекционно-дисперсионного алгоритма при $R = 5$ и ширине маски в 3 пикселя. На рисунке (3.4) приведен пример нахождения поля направлений изображения глазного дна с использованием проекционно-дисперсионного алгоритма при $R = 5$ и ширине маски в 3 пикселя.

3.3 Алгоритм нахождения поля направлений с использованием вейвлет-преобразования

С использованием вейвлет-преобразований поле находилось следующим образом:

- для набора углов производится вейвлет-преобразование;
- для каждого угла из набора: если значение ближе к белому цвету (принимаемому за 255), то если результат преобразования для текущего угла больше любого из результатов для всех уже полученных углов, то направлением в данной клетке принималось за текущий угол, и максимальным результат преобразования в клетке обновлялся;

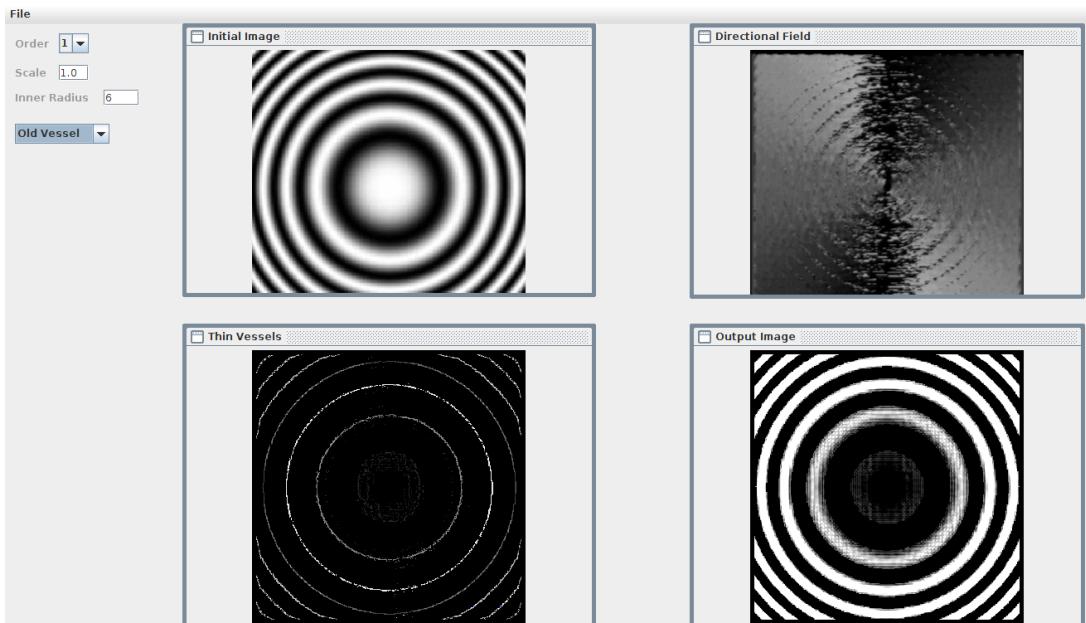


Рисунок 3.2 – Пример нахождения поля направления тестового изображения с использованием вейвлет-преобразования. В левом верхнем углу – исходное изображение, в правом верхнем – поле направлений, в левом нижнем – центральные линии, в правом нижнем – результат преобразования

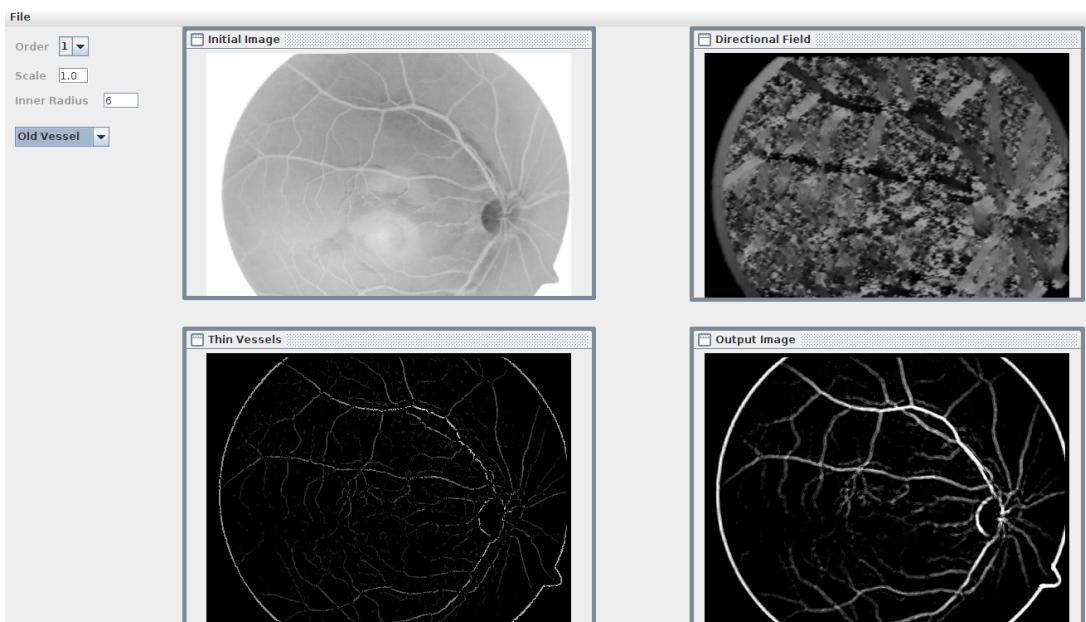


Рисунок 3.3 – Пример нахождения поля направления изображения глазного дна с использованием вейвлет-преобразования. В левом верхнем углу – исходное изображение, в правом верхнем – поле направлений, в левом нижнем – центральные линии, в правом нижнем – результат преобразования

- если же значение в клетке ближе к черному цвету (принимаемому за 0), то если результат преобразования для текущего угла меньше любого из результатов для всех полученных углов, то направлением в данной клетке принималось за текущий угол, и минимальный результат преобразования в клетке обновлялся;

- ко всем полученным углам прибавлялись 90 градусов.

Были испробованы разные комбинации, в частности, искался только максимум вейвлет-преобразования с обновлением углов при максимуме. Один из способов давал результат, близкий к результату проекционно-дисперсионного метода – в этом случае искался минимум функции яркости. Однако, при выполнении вейвлет-преобразования для нахождения сосудов, заданных инверсным (обычно зеленым) каналом, выполняется максимизация образа, и потому данный способ не был использован. Выложенный способ лучше других позволяет выделять центральные линии сосудов.

Преимуществом данного алгоритма является его встроенность в процесс вейвлет-преобразования: учитывается его особенность, экономится время нахождения поля направлений (по сравнению с алгоритмом, реализованным проекционно-дисперсионным методом).

На рисунке (3.4) приведен пример нахождения поля направлений тестового изображения с использованием вейвлет-преобразования. На рисунке (3.5) приведен пример нахождения поля направлений изображения глазного дна с использованием вейвлет-преобразования.

3.4 Сравнение алгоритмов нахождения поля направлений

На рисунках (3.2) и (3.4) заметно, что использование дисперсионного метода привело к меньшим шумам на изображении центральных линий, по сравнению с использованием алгоритма, основанного на вейвлет-преобразовании. Однако, на рисунках (3.3) и (3.5) центральные линии практически идентичны.

Существуют и другие методы нахождения поля направлений, однако, некоторые из них (дифференциальные методы) неустойчивы в зашумленных картинках, либо, на взгляд автора, излишне сложны (методы параметрической аппроксимации).

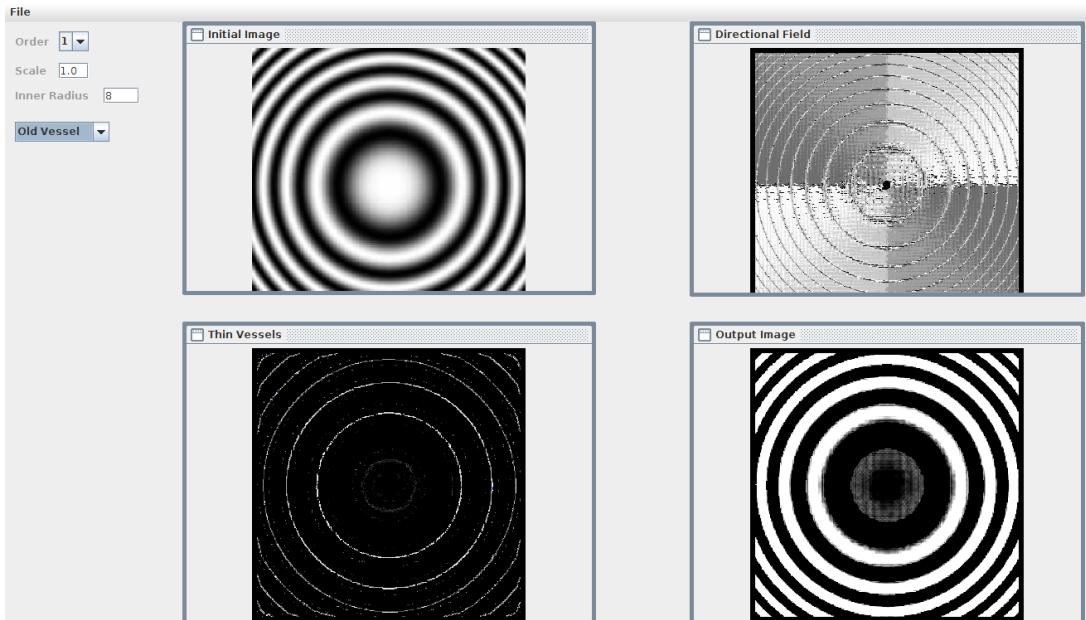


Рисунок 3.4 – Пример нахождения поля направления тестового изображения с использованием вейвлет-преобразования. В левом верхнем углу – исходное изображение, в правом верхнем – поле направлений, в левом нижнем – центральные линии, в правом нижнем – результат преобразования

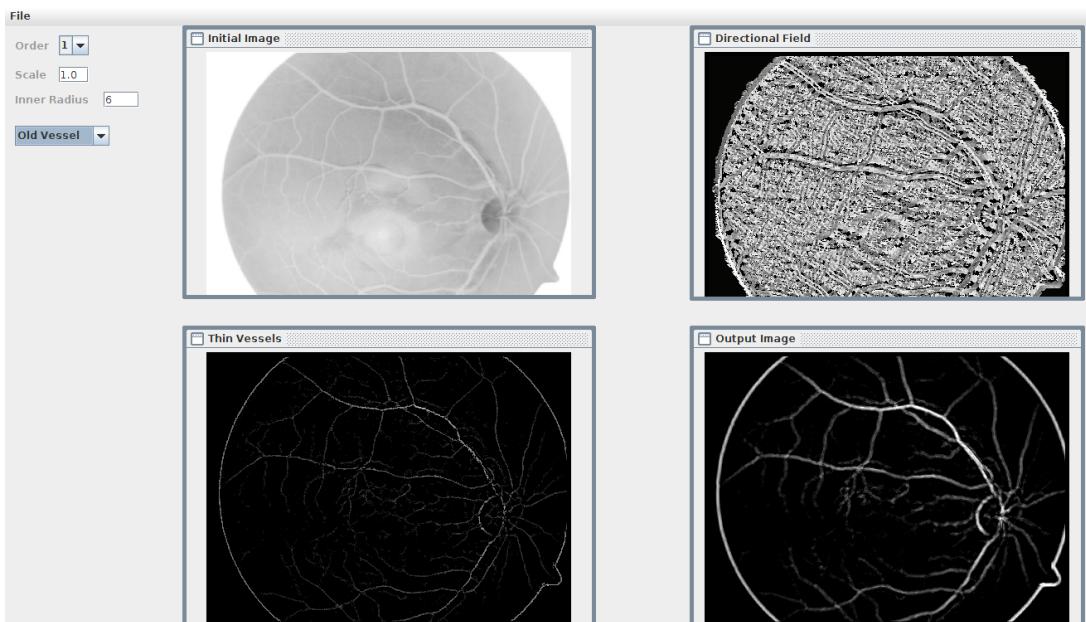


Рисунок 3.5 – Пример нахождения поля направления изображения глазного дна с использованием вейвлет-преобразования. В левом верхнем углу – исходное изображение, в правом верхнем – поле направлений, в левом нижнем – центральные линии, в правом нижнем – результат преобразования

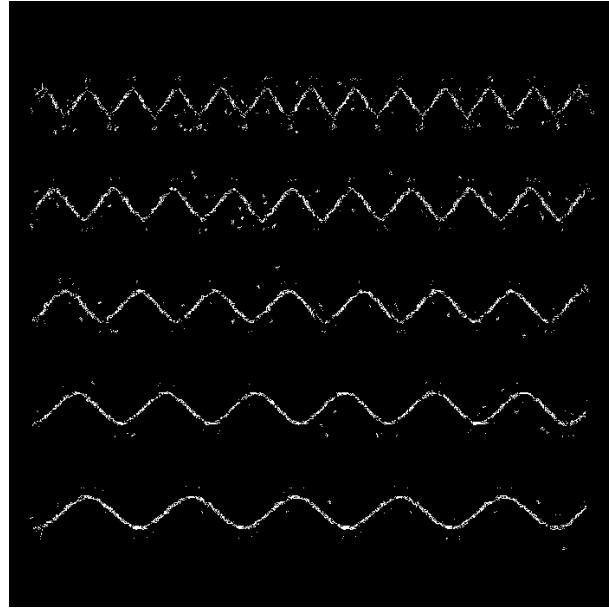


Рисунок 3.6 – Центральные линии

3.5 Выделение центральных линий

Выделение центральных линий используется при решении задачи лазерной коагуляции: области около линий не должны быть повреждены. Для эффективного решения этой проблемы, находится поле направлений, по нему для достаточно ярких пикселей определяется диаметр сосуда (или другого объекта), выделяются центральные линии, и на расстоянии, кратном диаметру сосуда выделяется граница области, которую нельзя задевать при коагуляции.

Нахождение диаметра сосуда и его центра в каждом достаточно ярком пикселе осуществляется следующим образом:

- в направления, перпендикулярные направлению в данной точке, отходят прямые, задаваемые в полярных координатах, с дискретизацией, до тех пор, пока все пиксели на отрезке достаточно яркие;
- для нахождения диаметра, полученные результаты складываются;
- с помощью диаметра определяются координаты от текущей точки до середины отрезка, и середина отрезка выделяется.

Пример выделения центральных линий приведен на рисунке (3.6).

На рисунке (3.7) представлен интерфейс для выполнения преобразований, и результаты.

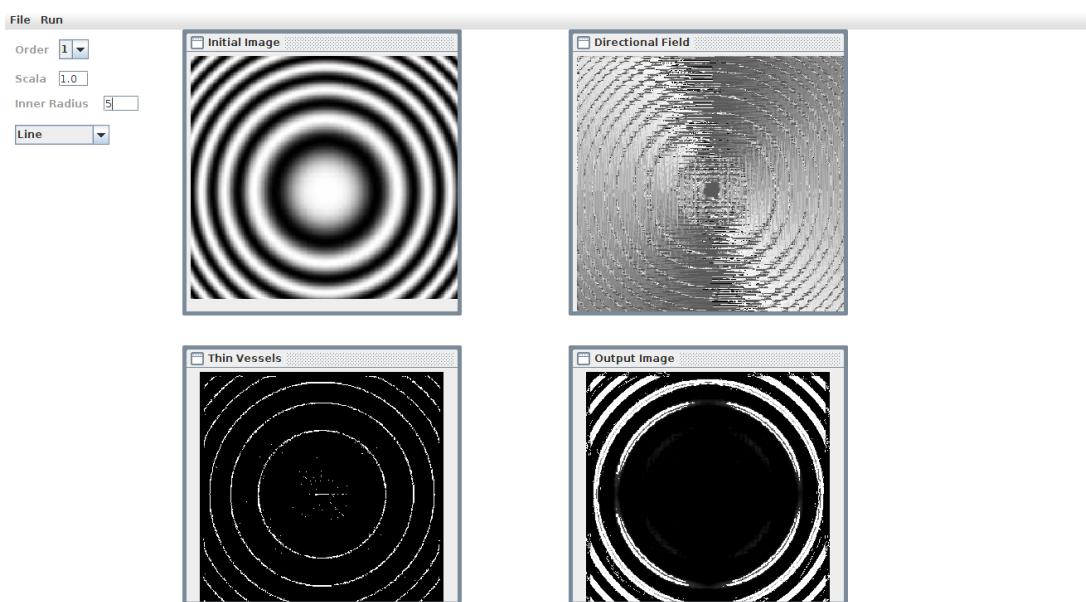


Рисунок 3.7 – Интерфейс для выполнения преобразований

4 ИССЛЕДОВАНИЕ ВЫДЕЛЯЕМЫХ ОБЪЕКТОВ

4.1 Исследование зависимости ширины выделяемых линий от параметра вейвлет-преобразования

В данной работе были реализованы вейвлеты, выделяющие сосуды и объекты дискообразной формы. В интерфейсе программы есть возможность менять два параметра: масштаб вейвлет-преобразования и радиус выделяемых объектов.

Исследуем зависимость между радиусом выделяемых объектов, полученных в результате вейвлет-преобразования, и параметром d формулы (2.26). На рисунке (4.1) отображены результаты при различных значениях параметра d .

Видим, что при небольших значениях d выделяются лишь тонкие сосуды, а при самых больших выделяются все кольца изображения.

4.2 Исследование качества выделения областей интересов

Использовалась общедоступная база данных изображений глазного дна **High-Resolution Fundus (HRF) Image Database**, содержащая 45 изображений глазного дна с разрешением 3504×2336 пикселей в формате *JPG*. Для каждого изображения глазного дна есть изображение идеально выделенных сосудов и маска к изображению, закрывающая фон глазного дна.

Будем кратко обозначать изображение с идеально выделенными (белым цветом на фоне чёрного) сосудами как *IdealImage*, а изображение, полученное после вейвлет-преобразования и бинаризации – за *WTImage*. Пример *IdealImage* представлен на рисунке (4.2), а пример *WTImage*, выполненного с помощью вейвлета, согласованного с моделью сосудов – на рисунке 4.3.

Обозначим количество белых пикселей на *IdealImage* за *AllWhite*, а количество черных – за *AllBlack*. Обозначим также за *ErrorWhite* количество случаев, в которых на *IdealImage* пиксель белый, а на *WTImage* – чёрный. Аналогично, обозначим за *ErrorBlack* количество случаев, в которых на *IdealImage* пиксель чёрный, а на *WTImage* – белый. Результаты сравнений других пяти изображений в указанных обозначениях и с использованием сконструированного для выделения сосудов вейвлета представлены в таблице 1.

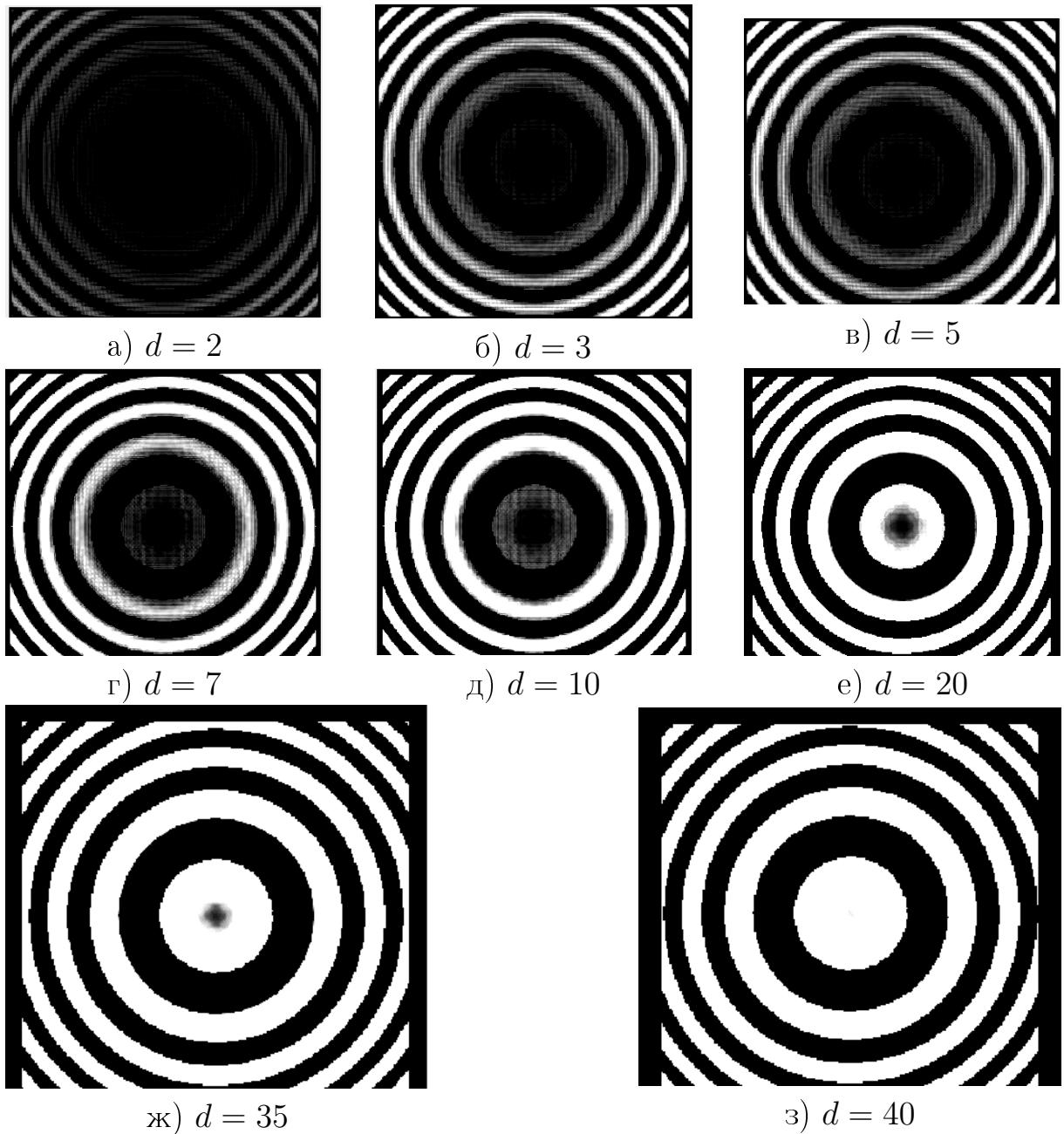


Рисунок 4.1 – Зависимость между радиусом выделяемых объектов и параметром d

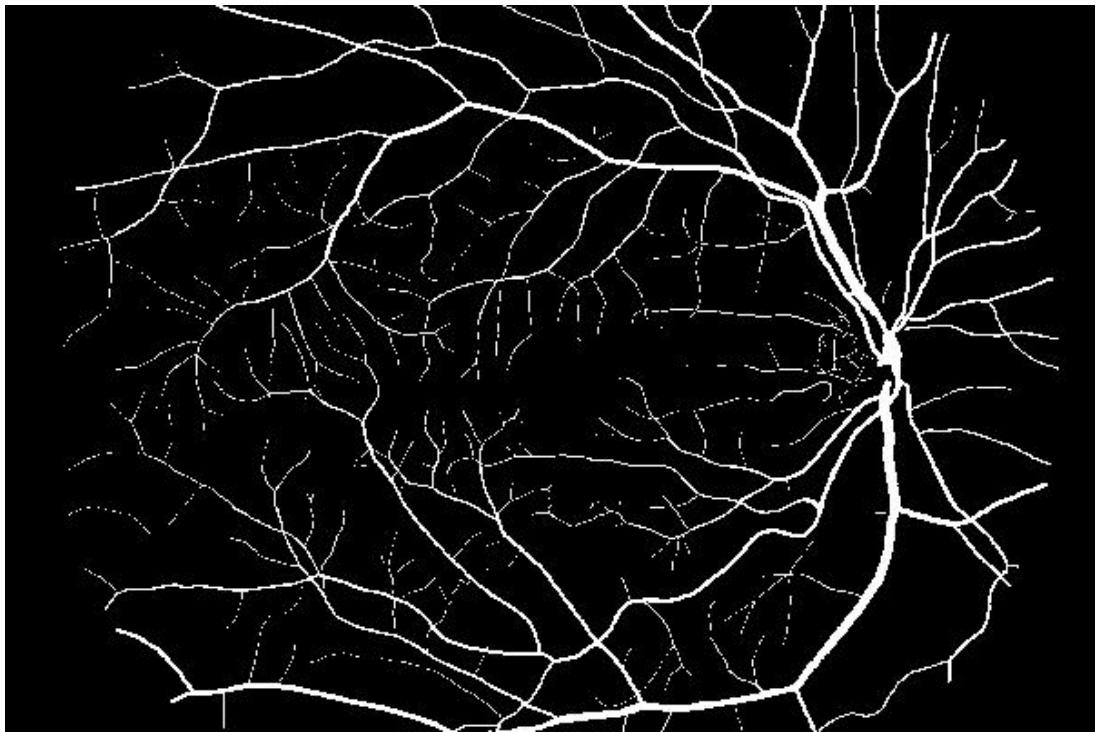


Рисунок 4.2 – Идеальное выделение сосудов

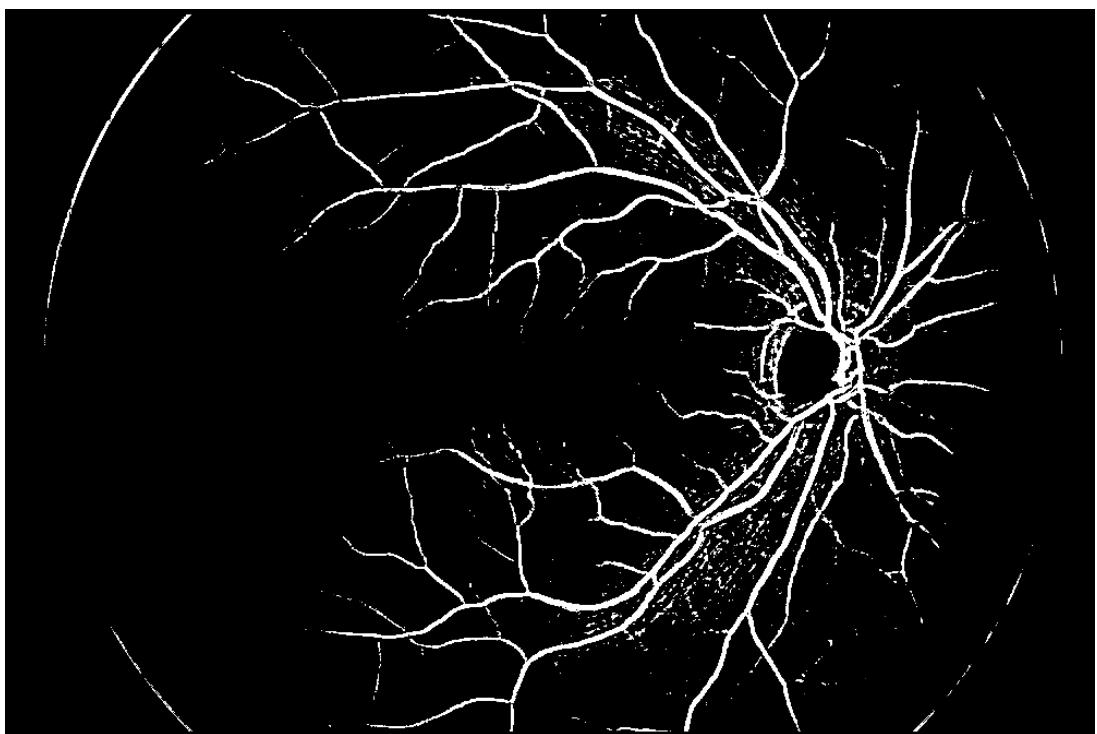


Рисунок 4.3 – Результат, полученный с помощью вейвлета,
согласованного с моделью сосудов

Таблица 1 – Результаты сравнений изображений

Наименование исходного изображения	$\frac{ErrorWhite}{AllWhite} \times 100\%$	$\frac{ErrorBlack}{AllBlack} \times 100\%$
01_dr.jpg	9,11	28,2
01_g.jpg	13,9	15,5
01_h.jpg	10,6	30,3
02_dr.jpg	12,1	29,9
02_g.jpg	13,8	17,5

ЗАКЛЮЧЕНИЕ

По результатам работы были:

- Проанализированы методы выделения сосудов.
- Исследованы различные виды вейвлетов. Дано оценка успешности их применения для выделения определенного класса объектов: сосудов или круглых пятен. Предложены и протестированы два новых вейвлета: один для выделения сосудов, и один для выделения объектов типа “диск”. Каждый из исследуемых вейвлетов был реализован в программе.
 - Исследованы и реализованы методы построения поля направлений, реализован метод выделения центральных линий сосудов.
 - Проведено исследование зависимости выделяемых объектов от параметра вейвлетов. Проведено сравнение выделенных с помощью вейвлет-преобразования сосудов, и сосудов, выделенных экспертами.

Таким образом, работа выполнена в полном объёме.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Blood vessels segmentation in nonmydriatic images using wavelets and statistical classifiers / Jorge J. G. Leandro, Joao V. B. Soares, Roberto M. Cesar Jr., Herbert F. Jelinek // Computer Graphics and Image Processing, 2003. SIBGRAPI 2003. XVI Brazilian Symposium on / Computer Graphics and Image Processing, 2003. SIBGRAPI 2003. XVI Brazilian Symposium on. — IEEE, 2003. — October. — pp. 262–269. — URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1241018&isnumber=27822> (дата обращения: 28.05.2016).
- 2 Нагорнов О.В. Вейвлет-анализ в примерах: Учебное пособие / О.В. Нагорнов, В.Г. Никитаев, В.М. Простокишин и др. — М. : НИ-ЯУ МИФИ, 2010. — 120 с. — URL: http://library.mephi.ru/Data-IRBIS/book-mephi/Nagornov_Vejvlet-analiz_v_primerah_2010.pdf (дата обращения: 28.05.2016).
- 3 Дремин И.М. Вейвлеты и их использование /И.М. Дремин, О.В. Иванов, В.А. Нечитайло // Успехи физических наук. — 2001. — Май. — Т. 171, № 5. — С. 465–502. — URL: https://ufn.ru/ufn01/ufn01_5/Russian/r015a.pdf (дата обращения: 28.05.2016).
- 4 Насонов А.В. Применение метода морфологических амёб для выделения сосудов на изображениях глазного дна / А.В. Насонов, А.А. Черноморец, А.С. Крылов, А.С. Родин // Труды 13-й международной конференции «Цифровая обработка сигналов и её применение» (DSP'2011). — Т. 2. — Москва, 2011. — С. 158–161.
- 5 Крылов А.С. Компьютерный анализ изображений глазного дна / А.С. Крылов, А.В. Насонов, А.С. Семашко и др. // Труды VIII Российско-Баварской конференции по биомедицинской инженерии. — 2012. — 424 с.
- 6 Mallat S. A Wavelet Tour of Signal Processing. — 3 edition. — Academic Press, 2009.
- 7 Shensa M.J. The discrete wavelet transform: Wedding the à trous and Mallat algorithms, IEEE Trans. Signal Process. — 1992.
- 8 Демьянович Ю.К. Введение в теорию вейвлетов / Ю.К. Демьянович, В.А. Ходаковский. — Санкт-Петербург: Петер. гос. ун-т путей сообщения, 2007. — 49 с. — URL: http://www.math.spbu.ru/parallel/pdf/dh_theory.pdf (дата обращения: 28.05.2016).
- 9 Antoine J-P. The Continuous Wavelet Transform in Image Processing, Institut de Physique Theorique Universite Catholique de Louvain. — Louvain-la-Neuve, Belgium, 2007.

- 10 Sarvaiya Jignesh N, Patnaik Dr. Suprava. Automatic Image Registration Using Mexican Hat Wavelet, Invariant Moment, and Radon Transform // International Journal of Advanced Computer Science and Applications (IJACSA), Special Issue on Image Processing and Analysis. — 2011. — URL: <http://dx.doi.org/10.14569/SpecialIssue.2011.010111> (дата обращения: 28.05.2016).
- 11 Ильясова Н.Ю. Системы компьютерного анализа диагностических изображений кровеносных сосудов: дисс. докт. технич. наук: 19.12.14: защищена 01.04.2015 / Ильясова Наталья Юрьевна. С.-Пб.: 2014. – 346 с.
- 12 Ильясова Н.Ю. Информационные технологии анализа изображений в задачах медицинской диагностики / Н.Ю. Ильясова, А.В. Куприянов, А.Г. Храмов. — М.: Радио и связь, 2012. — 424 с.

ПРИЛОЖЕНИЕ А.

ИСХОДНЫЙ КОД ПРОГРАММЫ

A.1 Лицензия

В данной работе автор использует лицензию **MIT license**. Текст лицензии расположен на сайте <https://opensource.org/licenses/MIT>.

A.2 basic.Decompose.scala

```
1 package basic
2
3 import math._
4 import basic.Types._
5
6 /** some math methods
7  * @author Ionkin Mikhail
8 *
9 * first author of this class: Christian Scheiblich (cscheiblich@gmail.com)
10 * @see https://github.com/cscheiblich/JWave tools.MathToolKit
11 * @see tests.MathTest.mathToolKitTests
12 */
13 object Decompose {
14
15     /** The method converts a positive integer to the ancient Egyptian
16      multipliers
17      * which are actually the multipliers to display the number by a sum
18      * of the
19      * largest possible powers of two.
20      * @see tests.MathTest#mathToolKitTests
21      * e.g.:
22      * {{ decompose(254).equals( List(128, 64, 32, 16, 8, 4, 2)) } }})
23
24     def decompose(number: Int): List[Int] = {
25         if (number <= 0) return List[Int]()
26         val log = log2(number)
27         val num = 1 << log
28         if (log > 0) num::decompose(number - num)
29         else          List(num)
30     }
31
32     /** splits the given length of the data array to a possible number of
33      blocks in
34      * block size and then handles the rest as the ancient egyptian
35      * decomposition:
36      * e. g. 127 by block size 32 ends up as: 32 | 32 | 32 | 16 | 8 | 4 |
37      * 2 | 1.
38      * @param number
39      *      the number that should be decompose; greater than block size
```

```

35     * @param maxBlockSize
36     *      the block size as a type of  $2^p | p=\{1,2,4,\dots\}$  that is first used
37     *      blocks until a rest is left; smaller than parameter number.
38     * @see tests.MathTest#mathToolKitTests
39     */
40 def decompose(number: Int, maxBlockSize: Int): List[Int] = {
41   if (number <= 0) return List[Int]()
42   val log = log2(number)
43   val num = 1 << log           // usually block size. @see #decompose(#number)
44   val noOfBlocks = if (num >= maxBlockSize) num / maxBlockSize else 1
45   val elemToAdd = num / noOfBlocks
46   val curList = (for (i <- 0 until noOfBlocks) yield elemToAdd).toList
47   if (log > 0) curList:::decompose(number - num, maxBlockSize)
48   else         curList
49 }
50 }
```

A.3 basic.ArrayToolKit.scala

```

1 package basic
2
3 import basic.Types._
4
5 object ArrayToolKit {
6
7   def copyPart(ar: A, ind: Int, length: Int): A = {
8     val res = new A(length)
9     for (i <- 0 until length) res(i) = ar(i+ind)
10    res
11  }
12
13  def copyColumn(mat: M, j: Int): A = mat.map {_(j)}
14
15  def copyColumn(ar: A, mat: M, j: Int): Unit =
16    for (i <- 0 until mat.length) mat(i)(j) = ar(i)
17
18  /** @see Array.copy */
19  def copyArray[E](ar1: Array[E], pos1: Int, ar2: Array[E], pos2: Int,
20                  length: Int) {
21    val delta = pos1 - pos2
22    for (i <- pos2 until pos2+length) ar2(i) = ar1(i+delta)
23  }
24 }
```

A.4 basic.Types.scala

```

1 package basic
2
3 object Types {
4   type T = Double
5   type B = Boolean
6   type BI = java.awt.image.BufferedImage
7   def sqr(x: T) = x*x
```

```

8  def norm2(x: T, y: T) = sqr(x) + sqr(y)
9  def pow4(x: T) = sqr(sqr(x))
10 def log2(number: Int): Int = {
11   var res = 0
12   while ((number >> res) > 0) res +=1
13   (res-1)
14 }
15
16 /** exist k in N_0 : 2^k = x */
17 def isBinary(x: Int) = (1 << log2(x)) == x
18
19 type A = Array[Double]
20 def sum(f: T => T, x1: T, x2: T, h: T) =
21   (x1 until x2 by h).map {f(_)}.sum
22 type ABool = Array[Boolean]
23 type AIInt = Array[Int]
24 type CollectionBI = List[BI]
25 import scala.collection.immutable.IndexedSeq
26 type ISInt = IndexSeq[Int]
27 def createA(n: Int) = new A(n)
28 def createAIInt(n: Int) = new AIInt(n)
29 def printAr(ar: A) {
30   for (x <- ar) printf(s"$x ")
31   println
32 }
33
34 type M = Array[A]
35 type MBool = Array[ABool]
36 type MIInt = Array[AIInt]
37
38 /** mat in R^{m,n} => m*n */
39 def productSize(mat: M): Int = mat.length*mat(0).length
40 def createM(m: Int, n: Int) = Array.ofDim[T](m, n)
41 def createMBool(m: Int, n: Int) = Array.ofDim[Boolean](m, n)
42 def print(mat: M) {
43   for (str <- mat) {
44     for (x <- str) printf("%3.4f ", x)
45     println
46   }
47 }
48 type MT[X] = Array[Array[X]]
49
50 def map(mat: M, fun: T => T): M =
51   mat.map{_.map{ x => fun(x) }}
52 def map(mat: MIInt, fun: Int => Int): MIInt =
53   mat.map{_.map{ x => fun(x) }}
54 def map(mat: M, fun: T => Int): MIInt =
55   mat.map{_.map{ x => fun(x) }}
56 def map(mat: MIInt, fun: Int => T): M =
57   mat.map{_.map{ x => fun(x) }}
58
59 def printMat(mat: M) {
60   for (str <- mat) printAr(str)
61   println
62 }
```

```

63
64  def createMInt(m: Int, n: Int) = Array.ofDim[Int](m, n)
65  def createWhiteMat(m: Int, n: Int): M =
66    map(createMInt(m, n), (x: Int) => 255.0)
67 }

```

A.5 basic.MatrixRotate.scala

```

1 package basic
2 import math._
3 import Types._
4 import basic.Constants._
5
6 object MatrixRotate {
7
8   /**
9    * rotate matrix
10   * @param mat -- matrix to rotate
11   * @param theta -- rotate angle, from -180 to 180.
12   * @param m -- usually, mat.length
13   * @param n -- usually, mat(0).length
14   * @see math definition as code
15   * {{{
16   * input: (y,x)
17   * [m00 m01 m02] [x] = [m00x + m01y + m02]
18   * [m10 m11 m12] [y] = [m10y + m11x + m12]
19   * [0 0 0] [1] = [0 + 0 + 1]
20   *
21   * 1. (y,x) => (y+h, x+w)
22   * 2. (y,x) => [(c,-s); (s, c)]*(y,x)
23   * 3. (y,x) => (y-h/2, x-w/2)
24   *
25   * [1 0 w] [x] = [x + 0 + w]
26   * [0 1 h] [y] = [0 + y + h]
27   * [0 0 0] [1] = [0 + 0 + 1]
28   *
29   * [c -s 0] [1 0 -w/2] [c -s -cw/2+sh/2]
30   * [s c 0] * [0 1 -h/2] = [s c -sw/2-ch/2]
31   * [0 0 1] [0 0 1] [0 0 1]
32   *
33   * [1 0 w] [c -s -cw/2+sh/2] [c -s -cw/2+sh/2+w]
34   * [0 1 h] * [s c -sw/2-ch/2] = [s c -sw/2-ch/2+h]
35   * [0 0 1] [0 0 1] [0 0 1]
36   *
37   * [c -s -cw/2+sh/2+w] [x] [cx - sy -cw/2+sh/2+w]
38   * [s c -sw/2-ch/2+h] [y] = [sx + cy -sw/2-ch/2+h]
39   * [0 0 1] [1] [1]
40   * }}}
41   */
42  def rotate(mat: M, theta: T, m: Int, n: Int): M = {
43    val angle: T = Pi*theta/180
44    val c = cos(angle); val s = sin(angle)
45
46    val hM: T = 0.5*m; val hN: T = 0.5*n
47    val addX = -c*hN + s*hM + n

```

```

48     val addY = -s*hN - c*hM + m
49
50     val res: M = createM(2*m, 2*n)
51     for (y <- 0 until m; x <- 0 until n) {
52         val X = (c*x - s*y + addX).round.toInt
53         val Y = (s*x + c*y + addY).round.toInt
54         res(Y)(X) = mat(y)(x)
55     }
56     res
57 }
58
59 def invRotate(mat: M, theta: T, m: Int, n: Int): M = {
60     val angle: T = Pi*theta/180+Pi
61     val c = cos(angle); val s = sin(angle)
62
63     val hM: T = 0.5*m; val hN: T = 0.5*n
64     val addX = -c*hN + s*hM + n
65     val addY = -s*hN - c*hM + m
66
67     val res: M = createWhiteMat(m, n)
68     for (y <- 0 until m; x <- 0 until n) {
69         val X = (c*x - s*y + addX).round.toInt
70         val Y = (s*x + c*y + addY).round.toInt
71         res(y)(x) = mat(Y)(X)
72     }
73     res
74 }
75 }
```

A.6 basic.MatrixUpdate.scala

```

1 package basic
2
3 import math._
4 import Types._
5 import basic.Constants._
6
7 object MatrixUpdate {
8
9     /** @return 255.0*(x-mn)/(mx-mn)) or bordered value */
10    def contrast(mat: M, mn: T, mx: T): M =
11        map( mat,
12            (x: T) =>
13                if (x < mn) 0.0
14                else
15                    if (x > mx) 255.0
16                    else 255.0*(x-mn)/(mx-mn))
17 }
```

A.7 basic.Fourie.scala

```

1 package basic
2
3 import math._
```

```

4 import basic.Types._
5
6 /** some math methods
7 * @author Ionkin Mikhail
8 *
9 * first author of this class: Christian Scheiblich (cscheiblich@gmail.com)
10 * @see https://github.com/cscheiblich/JWave tools.MathToolKit
11 * @see tests.MathTest.mathToolKitTests
12 */
13 object Fourie {
14     /** Returns a sampled array of #fun waves for given number of
15      * oscillations.
16      * @author Christian Scheiblich (cscheiblich@gmail.com)
17      * update: Ionkin Mikhail (ionkinmikhail@gmail.com)
18      *
19      * @param samplingRate
20      *      should be great than 2 and likely to be of 2^p | p E N
21      * @param noOfOscillations
22      *      should be of natural numbers except zero
23      * @param fun
24      *      is function, e.g. sin or cos
25      * @return sampled array keeping a number of @fun waves
26      */
27     @Deprecated
28     def createOscillation(samplingRate: Int, noOfOscillations: Int, fun: T
29                           => T): A = {
30         val sR      = max(samplingRate, 2)
31         val noOfosc = max(noOfOscillations, 2)
32         val argMulti = 2*Pi*noOfosc/sR
33         for (i <- Array.range(0, sR)) yield fun(argMulti*i)
34     }
35     /** @see a def #createOscillation with a #fun = sin */
36     @Deprecated
37     def createSinOscillation(samplingRate: Int, noOfOscillations: Int) =
38         createOscillation(samplingRate, noOfOscillations, sin)
39     /** @see a def #createOscillation with a #fun = cos */
40     @Deprecated
41     def createCosOscillation(samplingRate: Int, noOfOscillations: Int) =
42         createOscillation(samplingRate, noOfOscillations, cos)
43     }

```

A.8 basic.Integral.scala

```

1 package basic
2 import basic.Types._
3
4 object Integral {
5
6     /** return min n: n = 2*N+1 & (x2-x1)/h0 <= n */
7     def getN(x1: T, x2: T, h0: T): Int =
8         ((x2 - x1)/h0).ceil.toInt + 1
9
10    /** @see "Simpsons rule" in Internet */

```

```

11  def simpson(f: T => T, x1: T, x2: T, h0: T): T = {
12    val n = getN(x1, x2, h0)
13    val h = (x2-x1)/n
14    val s1 = sum(f, x1+h, x2, 2*h)
15    val s2 = sum(f, x1+2*h, x2, 2*h)
16    (h/3)*(4*s1 + 2*s2 + f(x1) + f(x2))
17  }
18
19 /** Simpsons rule for 2D
20 * @see simpson
21 */
22 def simpson2(f: (T, T) => T, xBeg: (T, T), xEnd: (T, T), h0: T): T = {
23   val n1 = getN(xBeg._1, xEnd._1, h0)
24   val h1 = (xEnd._1-xBeg._1)/n1
25   val n2 = getN(xBeg._2, xEnd._2, h0)
26   val h2 = (xEnd._2-xBeg._2)/n2
27   def locSimpson(g: T => T): T = simpson(g, xBeg._2, xEnd._2, h2)
28   def sum(f: (T, T) => T, x1: T, x2: T, h: T): T = {
29     var sum: T = 0
30     for (x <- x1 until x2 by h)
31       sum += locSimpson(f(x, _))
32     sum
33   }
34   val s1: T = sum(f, xBeg._1+h1, xEnd._1, 2*h1)
35   val s2: T = sum(f, xBeg._1+2*h1, xEnd._1, 2*h1)
36   (h1/3)*(4*s1 + 2*s2 + locSimpson(f(xBeg._1, _)) + locSimpson(f(xEnd._1, _)))
37 }
38 }
```

A.9 basic.Constants.scala

```

1 package basic
2
3 import math._
4
5 object Constants {
6   val thetas = (0.0 until 360.0 by 1.0)
7   val angles = thetas.map{_*Pi/180.0}
8   val coss = angles.map{cos(_)}
9   val sins = angles.map{sin(_)}
10 }
```

A.10 basic.Statistic.scala

```

1 package basic
2
3 import math._
4 import basic.Types._
5
6 object Statistic {
7
8   def minMaxAverage(mat: M): (T, T, T) = {
9     var min: T = mat(0)(0)
```

```

10     var max: T = min
11     var averange: T = 0
12     for (i <- 0 until mat.length; j <- 0 until mat(0).length){
13         val x = mat(i)(j)
14         if (x < min) min = x
15         else
16             if (x > max) max = x
17             averange += x
18     }
19     (min, max, averange/productSize(mat))
20 }
21
22 def disp(mat: M, aver: T) = {
23     mat.map { _.map { y => sqr(y-aver) }.sum }.sum/
24         productSize(mat)
25 /** X => EX */
26 def aver(mat: M): T = mat.map {_.sum}.sum / productSize(mat)
27
28 /** (X,Y) => E[X.*Y] - EX*EY
29 * @see tests.MathTest#MathToolKitTests
30 */
31 def correlation(mat1: M, mat2: M): T = {
32     val aver1 = aver(mat1); val aver2 = aver(mat2)
33     var sum: T = 0
34     for (i <- 0 until mat1.length; j <- 0 until mat1(0).length)
35         sum += mat1(i)(j)*mat2(i)(j)
36     sum / productSize(mat1) - aver1*aver2
37 }
38
39 def mxMat(mat: M, sy: Int, sx: Int): M = {
40     val m = mat.length; val n = mat(0).length
41     val sumStr = createM(m, n)
42     val dxs = (-sx to sx)
43     for (y <- sy until m-sy){
44         sumStr(y)(sx) = dxs.map{dx => mat(y)(sx+ dx)}.sum
45         for (x <- sx+1 until n-sx)
46             sumStr(y)(x) = sumStr(y)(x-1) + (mat(y)(x+sx) - mat(y)(x-sx-1))
47     }
48
49     val sumCol = createM(m, n)
50     val dys = (-sy to sy)
51     for (x <- sx until n-sx){
52         sumCol(sy)(x) = dys.map{dy => sumStr(sy+dy)(x)}.sum
53         for (y <- sy+1 until m-sy) {
54             val dif = sumStr(y+sy)(x) - sumStr(y-sy-1)(x)
55             sumCol(y)(x) = sumCol(y-1)(x) + dif
56         }
57     }
58
59     val S: T = (2*sx+1)*(2*sy+1)
60     val mx = sumCol.map{_.map{_-/S}}
61     mx
62 }
63
64 def mx2Mat(mat: M, sy: Int, sx: Int): M = {

```

```

65     val m = mat.length; val n = mat(0).length
66     val sumStr = createM(m, n)
67     val dxs = (-sx to sx)
68     for (y <- sy until m-sy) {
69       sumStr(y)(sx) = dxs.map{dx => sqr(mat(y)(sx+ dx))}.sum
70       for (x <- sx+1 until n-sx)
71         sumStr(y)(x) = sumStr(y)(x-1) + (sqr(mat(y)(x+sx)) - sqr(mat(y)(x-sx-1)))
72     }
73     val sumCol = createM(m, n)
74     val dys = (-sy to sy)
75     for (x <- sx until n-sx) {
76       sumCol(sy)(x) = dys.map(dy => sumStr(sy+dy)(x)).sum
77       for (y <- sy+1 until m-sy)
78         sumCol(y)(x) = sumCol(y-1)(x) + (sumStr(y+sy)(x) - sumStr(y-sy-1)(x))
79     }
80     val S: T = (2*sx+1)*(2*sy+1)
81     val mx2 = sumCol.map{_.map{/_/S}}
82     mx2
83   }
84
85   def dispMat(mat: M, sy: Int, sx: Int): M = {
86     val mx = mxMat(mat, sy, sx)
87     val mx2 = mx2Mat(mat, sy, sx)
88     val m = mat.length; val n = mat(0).length
89     val res = createM(m, n)
90     for (i <- 0 until m; j <- 0 until n)
91       res(i)(j) = mx2(i)(j) - sqr(mx(i)(j))
92     res
93   }
94 }
```

A.11 wavelets.AsVessel.scala

```

1 package wavelets
2 import basic.Types._
3 import math._
4 import basic.Integral
5
6 class AsVessel(d: T = 3, a: T) extends ACBoundedWavelet(d, a) {
7   override val wavename = "AsVessel"
8   override def psi(x: T): T = exp(-0.5*sqr(x))
9 }
```

A.12 wavelets.Gauss.scala

```

1 package wavelets
2 import math._
3 import basic.Types._
4
5 class Gauss extends ICWavelet {
6   override val wavename = "Gauss"
7   /** wavelet for blood vessel */
```

```

8     override def psi(x: T, y: T): T =
9         sin(x+y)*exp(-0.5*norm2(x, y))
10    }

```

A.13 wavelets.ICWavelet.scala

```

1 package wavelets
2
3 import basic.Types._
4 import basic.Constants._
5
6 abstract class ICWavelet {
7     val wavename: String
8     val h: T = 0.1
9
10    /** 2D wavelet-function */
11    def psi(x: T, y: T) : T = ????
12    /** int_{x0,y0}^{x0+1, y0+1} psi(x, y) dx dy */
13    protected def integral(xBeg: (T, T), xEnd: (T, T)): T =
14        basic.Integral.simpson2(psi, xBeg, xEnd, h)
15
16    /** 2D core of wavelet */
17    def core(sx: Int, sy: Int, theta: Int, a: T): M = {
18        // r(-theta)/a
19        val cs = coss(theta)/a // cos -sin
20        val sn = sins(theta)/a // sin cos
21        val res = createM(2*sy+1, 2*sx+1)
22        for (y <- -sy to sy; x <- -sx to sx) {
23            val newX = (cs*x + sn*y).round.toInt
24            val newY = (-sn*x + cs*y).round.toInt
25            res(y+sy)(x+sx) = integral((newX, newY), (newX+1, newY+1))/a
26        }
27        res
28    }
29 }

```

A.14 wavelets.FHAT.scala

```

1 package wavelets
2 import math._
3 import basic.Types._
4
5 class FHAT(d: T = 3, a: T) extends ACBoundedWavelet(d, a) {
6     override val wavename = "FHAT"
7     override def psi(x: T): T =
8         if (abs(x) <= a) 1
9         else if (abs(x) <= 2*a+1) -(2*a+1)/(2*a+2)
10        else 0
11    }

```

A.15 wavelets.OldAsVessel.scala

```

1 package wavelets
2

```

```

3 import math._
4 import basic.Types._
5
6 class OldAsVessel(s0: Int = 5) {
7   /** transform with specified norms */
8   def specTransform(mat: M, a: T): (M, MInt) = {
9     val m = mat.length; val n = mat(0).length
10    val s1 = (s0/a).ceil.toInt
11    val s2 = 2
12    val s = s1+s2
13    val rs = -s to s
14    val dir = createMInt(m, n)
15    val res = createM(m, n)
16    for (theta <- 0 until 180 by 10) {
17      val angle = Pi*theta/180
18      val cs = cos(angle); val sn = sin(angle)
19      val xs = rs.map{r => (r*cs).round.toInt}
20      val ys = rs.map{r => (r*sn).round.toInt}
21
22      for (y <- s until m-s; x <- s until n-s) {
23        val g = (-s1 to s1).map{i => mat(ys(i+s)+y)(xs(i+s)+x)}.sum -
24          mat(y)(x) -
25          (s1.toDouble/2)*(-s until -s1).map{i => mat(ys(i+s)+y)(xs(i+s)
26            +x)}.sum -
27          (s1.toDouble/2)*(s1+1 to s).map{i => mat(ys(i+s)+y)(xs(i+s)+x)
28            }.sum
29        /*if (g > res(y)(x)) {
30          res(y)(x) = g
31          dir(y)(x) = theta + 90
32        }*/
33        if (mat(y)(x) < 128) {
34          if (res(y)(x) < g) {
35            //res(y)(x) = g
36            //dir(y)(x) = theta + 90
37          } else {
38            dir(y)(x) = theta + 90
39            res(y)(x) = g
40          }
41        } else {
42          if (res(y)(x) > g){
43            //res(y)(x) = g
44            //dir(y)(x) = theta
45          } else {
46            dir(y)(x) = theta + 90
47            res(y)(x) = g
48          }
49        }
50      val resInt = res.map{_.map{_.toInt}}
51      //val trueDir = image.Analys.direction(resInt)
52      //for (y <- s until m-s; x <- s until n-s) res(y)(x) = 255 - res(y)(
53        x)
54      (res, dir)
55    }

```

54 }

A.16 wavelets.Gabor.scala

```
1 package wavelets
2 import math._
3 import basic.Types._
4
5 class Gabor(sigma: T = 7) extends ICWavelet {
6   override val wavename = "Gabor"
7   val sigma2 = sigma*sigma
8   override def psi(x: T, y: T): T =
9     exp(-norm2(x, y)/(2*sigma2))*cos(10*(x+y))
10 }
```

A.17 wavelets.Daubechies.scala

```
1 package wavelets
2
3 import basic.Types._
4 import scala.collection.mutable.ArraySeq
5
6 class Daubechies(order: Int) extends WaveletTransformTrait {
7
8   val motherWavelength      = order << 1
9   val transformWavelength    = 2
10  override def getScaling: A = Daubechies.Daub(order)//.par
11 }
12
13 object Daubechies {
14   lazy val Daub = new M(5)
15   Daub(1) = Array(
16     7.071067811865475244008443621048490392848359376884740365883398e-01,
17     7.071067811865475244008443621048490392848359376884740365883398e-01) //
18     .par
19   Daub(2) = Array(
20     4.829629131445341433748715998644486838169524195042022752011715e-01,
21     8.365163037378079055752937809168732034593703883484392934953414e-01,
22     2.241438680420133810259727622404003554678835181842717613871683e-01,
23     -1.294095225512603811744494188120241641745344506599652569070016e-01) //
24     .par
25   Daub(3) = Array(
26     3.326705529500826159985115891390056300129233992450683597084705e-01,
27     8.068915093110925764944936040887134905192973949948236181650920e-01,
28     4.598775021184915700951519421476167208081101774314923066433867e-01,
29     -1.350110200102545886963899066993744805622198452237811919756862e-01,
30     -8.544127388202666169281916918177331153619763898808662976351748e-02,
31     3.522629188570953660274066471551002932775838791743161039893406e-02) //
32     .par
33   Daub(4) = Array(
34     2.303778133088965008632911830440708500016152482483092977910968e-01,
```

```

35      7.148465705529156470899219552739926037076084010993081758450110e-01,
36      6.308807679298589078817163383006152202032229226771951174057473e-01,
37      -2.798376941685985421141374718007538541198732022449175284003358e-02,
38      -1.870348117190930840795706727890814195845441743745800912057770e-01,
39      3.084138183556076362721936253495905017031482172003403341821219e-02,
40      3.288301166688519973540751354924438866454194113754971259727278e-02,
41      -1.059740178506903210488320852402722918109996490637641983484974e-02) //
        .par
42  }

```

A.18 wavelets.Morlet.scala

```

1 package wavelets
2 import math._
3 import basic.Types._
4
5 /** psi(x, y) = cos(x*k0x+y*k0y)*exp(-norm2(kx*x, y)) */
6 class Morlet(k0x: T = -1, k0y: T = 3, eps: T = 8) extends ICWavelet {
7   override val wavename = "Morlet"
8   val kx = 1.0 / sqrt(eps)
9   /** Im-part of 2D wavelet-function */
10  override def psi(x: T, y: T) : T =
11    cos(x*k0x+y*k0y)*exp(-norm2(kx*x, y))
12 }

```

A.19 wavelets.DWaveletTrait.scala

```

1 package wavelets
2
3 import basic.Types._
4
5 trait WaveletTransformTrait {
6
7   /** The wavelength of the base or so called mother wavelet and its
8     * matching scaling function */
8   val motherWavelength: Int
9
10  /** The minimal wavelength of a signal that can be transformed */
11  val transformWavelength: Int
12
13  def getScaling: A
14  val scalingDeCom: A = getScaling
15  lazy val waveletDeCom: A = {
16    val res = new A(motherWavelength)
17    for (i <- 0 until motherWavelength)
18      res(i) = if ((i&1)==0) scalingDeCom(motherWavelength - 1 - i)
19                  else -scalingDeCom(motherWavelength - 1 - i)
20    res
21  }
22
23  def scaling(j: Int): T = scalingDeCom(j)
24  def wavelet(j: Int): T = waveletDeCom(j)
25
26  /** @param id:

```

```

27     *      1 -- Reverse
28     *      2 -- Forward
29   */
30   def waveletTransform(array1: A, lengthAr: Int, id: Int): A = {
31     val array2 = new A(lengthAr)
32     val h = lengthAr >> 1
33     for(i <- 0 until h; j <- 0 until motherWavelength) {
34       val k = ((i << 1) + j) % lengthAr
35       if (id == 1){
36         array2(i) += array1(k) * scaling(j) // low pass filter for
            the energy (approximation)
37         array2(i + h) += array1(k) * wavelet(j) // high pass filter for
            the details
38     } else { //if (id == 2)
            // adding up energy from low pass (approximation) and details
            from high pass filter
39         array2(k) += array1(i) * scaling(j) +
40                     array1(i+h) * wavelet(j)
41     }
42   }
43 }
44 array2
45 }
46
47 def waveletReverse(arrHilb: A, arrHilbLength: Int): A =
48   waveletTransform(arrHilb, arrHilbLength, 1)
49
50 def waveletForward(arrTime: A, arrTimeLength: Int): A =
51   waveletTransform(arrTime, arrTimeLength, 2)
52 }

```

A.20 wavelets.MHAT.scala

```

1 package wavelets
2 import math._
3 import basic.Types._
4
5 class MHAT(d: T = 2, a: T) extends ACBoundedWavelet(d, a) {
6   override val wavename = "MHAT"
7   val norm = 2.0 / sqrt( 3.0*sqrt(Pi) )
8   override def psi(x: T): T = {
9     val l1 = (x-0.5); val l2 = (x+0.5)
10    val s1 = pow4(l1); val s2 = pow4(l2)
11    norm*(l2*exp(-s2*0.25) - l1*exp(-0.25*s1))
12  }
13 }

```

A.21 wavelets.AsLongVessel.scala

```

1 package wavelets
2 import basic.Types._
3 import math._
4 import basic.Integral
5
6 class AsLongVessel(d: T = 4, a: T) extends ACBoundedWavelet(d, a) {

```

```

7   override val wavename = "AsLongVessel"
8   override def psi(x: T) = (d-abs(x))*exp(-sqr(x))
9 }
```

A.22 wavelets.ACBoundedWavelet.scala

```

1 package wavelets
2
3 import basic.Types._
4 import math._
5 import basic.Integral
6
7 abstract class ACBoundedWavelet(d: T, a: T) extends ICWavelet {
8   val dDivA: T = d/a
9   def psi(x: T): T
10  /** 2D wavelet-function */
11  override def psi(x: T, y: T) : T =
12    if (abs(y) < dDivA) psi(x) else 0.0
13 }
```

A.23 transform.TransformTrait.scala

```

1 package transform
2
3 import basic.Types._
4 import basic.ArrayToolKit._
5 import exceptions._
6
7 /**
8  * @author Mikhail Ionkin (ionkinmikhail@gmail.com)
9  *
10 * I'm using a project
11 * [1] JWave (by Christian Scheiblich (cscheiblich@gmail.com))
12 * @see https://github.com/cscheiblich/JWave
13 */
14 trait TransformTrait {
15
16  /** [1]:
17   * Performs the reverse transform from frequency or Hilbert domain to
18   * time
19   * domain for a given array depending on the used transform algorithm
20   * by
21   * inheritance.
22   */
23  def reverse1D(arrToReverse: A): A = {
24    val n = arrToReverse.length
25    if (!isBinary(n))
26      throw new BinaryAmountException("arrToReverse", n)
27    reverse1D(arrToReverse, log2(n))
28  }
29  /** @see reverse1D(arrToReverse) */
30  def reverse1D(arrToReverse: A, level: Int): A
31  /** [1]:
```

```

31     * Performs the forward transform from time domain to frequency or
32     * Hilbert
33     * domain for a given array depending on the used transform algorithm
34     * by
35     * inheritance.
36     */
37     def forward1D(arrToForward: A): A = {
38       val n = arrToForward.length
39       if (!isBinary(n))
40         throw new BinaryAmountException("arrToForward", n)
41       forward1D(arrToForward, log2(n))
42     }
43     /** @see forward1D(arrToForward) */
44     def forward1D(arrToForward: A, level: Int): A
45
46     protected def twoOnString(inp: M, lvlM: Int, lvlN: Int, fun1D: (A, Int
47       ) => A): M =
48       for(str <- inp) yield fun1D(str, lvlN)
49
50     protected def twoOnColumn(inp: M, lvlM: Int, lvlN: Int, fun1D: (A, Int
51       ) => A): M =
52       val res = createM(inp.length, inp(0).length)
53       for(j <- 0 until inp(0).length)
54         copyColumn(fun1D(copyColumn(inp, j), lvlM), res, j)
55       res
56     }
57
58     /**
59      * @param transformID:
60      *   str -- transform of strings
61      *   col -- transform of columns
62      *   mat -- transform of strings and columns
63      */
64     protected def two(mat1: M, lvlM: Int, lvlN: Int, fun1D: (A, Int) => A,
65       transformID: String): M =
66       transformID match {
67         case "str" =>
68           for(str <- mat1) yield fun1D(str, lvlN)
69         case "col" => {
70           val res = createM(mat1.length, mat1(0).length)
71           for(j <- 0 until mat1(0).length)
72             copyColumn(fun1D(copyColumn(mat1, j), lvlM), res, j)
73           res
74         }
75       }
76
77     /**
78      * Performs the 2-D forward transform from time domain to frequency or
79      * Hilbert
80      * domain for a given matrix depending on the used transform algorithm

```

```

        by
80     * inheritance.
81     */
82     def forward2D(matTime: M): M =
83         forward2D(matTime, log2(matTime.length), log2(matTime(0).length))
84
85     /** @see forward2D(matTime) */
86     def forward2D(matTime: M, transformID: String): M =
87         forward2D(matTime, log2(matTime.length), log2(matTime(0).length),
88                 transformID)
88
89     /** @see forward2D(arrToForward) */
90     def forward2D(matTime: M, lvlM: Int, lvlN: Int): M =
91         two(matTime, lvlM, lvlN, forward1D, "mat")
92
93     /** @see forward2D(arrToForward) */
94     def forward2D(matTime: M, lvlM: Int, lvlN: Int, transformID: String):
95         M =
96         two(matTime, lvlM, lvlN, forward1D, transformID)
97
98     /** [1]
99      * Performs the 2-D reverse transform from frequency or Hilbert or
100     time domain
100    * to time domain for a given matrix depending on the used transform
101     algorithm
101    * by inheritance.
102    */
103    def reverse2D(matFreq: M): M =
104        reverse2D(matFreq, log2(matFreq.length), log2(matFreq(0).length))
105
106    /** @see reverse2D(arrToReverse) */
107    def reverse2D(matFreq: M, lvlM: Int, lvlN: Int): M =
108        two(matFreq, lvlM, lvlN, reverse1D, "mat")
109
110
111    /** @see reverse2D(matFreq) */
112    def reverse2D(matFreq: M, transformID: String): M =
113        reverse2D(matFreq, log2(matFreq.length), log2(matFreq(0).length),
114                  transformID)
114
115    /** @see reverse2D(arrToReverse) */
116    def reverse2D(matFreq: M, lvlM: Int, lvlN: Int, transformID: String):
117        M =
117        two(matFreq, lvlM, lvlN, reverse1D, transformID)
118    /** [1]
119      * Generates from a 2-D decomposition a 1-D time series.
120      */
121    def decompose(times: A): M = ???
122    /** @see decompose(times) */
123    def decompose(times: A, level: Int): M = ???
124
125    /** [1]
126      * Generates from a 1-D signal a 2-D output, where the second
126      dimension are

```

```

127     * the levels of the wavelet transform. The first level should keep
128     * the
129     * original coefficients. All following levels should keep each step
130     * of the
131     * decomposition of the Fast Wavelet Transform. However, each level of
132     * the
133     * this decomposition matrix is having the full set, full energy and
134     * full
135     * details, that are needed to do a full reconstruction. So one can
136     * select a
137     * level filter it and then do reconstruction only from this single
138     * line!
139     * BY THIS METHOD, THE _HIGHEST_ LEVEL IS _ALWAYS_ TAKEN FOR
140     * RECONSTRUCTION!
141     */
142     def recompose(matDecompose: M): A = ???
143
144     /** recompose(matDecompose) */
145     def recompose(matDecompose: M, level: Int): A = ???
146 }

```

A.24 transform.DTransform.scala

```

1 package transform
2
3 import java.awt.color._
4 import java.awt.image.ColorConvertOp
5 import java.awt.image.BufferedImage
6 import math._
7 import scala.collection.mutable.ArraySeq
8 import scala.collection.immutable.IndexedSeq
9 import basic.Types._
10 import image._
11
12 object DTransform {
13
14     def daubechies(
15         mat: M,
16         order: Int = 2,
17         transformID: String = "mat"): M = {
18         val wavelet = new wavelets.Daubechies(order)
19         val trans = new transform.AncientEgyptianDecomposition(wavelet)
20         val resMat: M = trans.forward2D(mat, transformID)
21         resMat
22     }
23
24     def daubechiesForwardAndReverse(
25         mat: M,
26         order: Int = 2,
27         transformID: String = "mat"): M = {
28         val wavelet = new wavelets.Daubechies(order)
29         val trans = new transform.AncientEgyptianDecomposition(wavelet)
30         val resMat: M = trans.forward2D(mat, transformID)
31         val invresMat: M = trans.reverse2D(resMat, transformID)
32         invresMat
33     }

```

```
33     }
34 }
```

A.25 transform.WaveletPacketTransform.scala

```
1 package transform
2
3 import basic.Types._
4 import basic.ArrayToolKit._
5 import exceptions._
6 import wavelets.WaveletTransformTrait
7
8 class WaveletPacketTransform(wavelet: WaveletTransformTrait) extends
9   TransformTrait {
10   override def reverse1D(arrToReverse: A, level: Int): A = {
11     val n = arrToReverse.length
12     val arrTime = arrToReverse.clone()
13     val steps = log2(n)
14     if (!isBinary(n))
15       throw new BinaryAmountException("arrToReverse", n)
16     var h = wavelet.transformWavelength << (steps - level)
17     while(h <= arrTime.length && h >= wavelet.transformWavelength) {
18       val g = n / h // ... -> 8 -> 4 -> 2 -> 1
19       val iBuf = new A(h)
20       for(p <- 0 until g) {
21         copyArray(arrTime, p * h, iBuf, 0, h)
22         val oBuf = wavelet.waveletReverse(iBuf, h)
23         copyArray(oBuf, 0, arrTime, p * h, h)
24       }
25       h <<= 1
26     }
27     arrTime
28   }
29   override def forward1D(arrTime: A, level: Int): A = {
30     val n = arrTime.length
31     if (!isBinary(n))
32       throw new BinaryAmountException("arrTime", n)
33     val arrHilb = arrTime.clone()
34     var h = n
35     var l = 0
36     while (l < level && h >= wavelet.transformWavelength) {
37       val g = n / h
38       val iBuf = new A(h)
39       for(p <- 0 until g) {
40         copyArray(arrHilb, p * h, iBuf, 0, h)
41         val oBuf = wavelet.waveletForward(iBuf, h)
42         copyArray(oBuf, 0, arrHilb, p * h, h)
43       }
44       h >>= 1
45       l += 1
46     }
47     arrHilb
48   }
49 }
```

A.26 transform.CTransform.scala

```
1 package transform
2 import wavelets._
3 import basic.Types._
4 import image._
5 import math._

7 class CTransform(wavelet: ICWavelet, dx: Int = 10, dy: Int = 5) {
8
9     def transform(mat: M, theta: Int, a: T): M = {
10        val sx = (dx/a).floor.toInt
11        val sy = (dy/a).floor.toInt
12        val core: M = wavelet.core(sx, sy, theta, a)
13        val coef: T = core.map{_.sum}.sum
14        assert{abs(coef) > 0.01 && abs(coef) < 100}
15        val m = mat.length;    val n = mat(0).length
16        val res = createM(m, n)
17        for (y <- sy until m-sy; x <- sx until n-sx){
18            var sum: T = 0
19            for (k <- -sy to sy; l <- -sx to sx)
20                sum += mat(k+y)(l+x) * core(k+sy)(l+sx)
21            res(y)(x) = sum / coef
22        }
23        res
24    }
25
26    def WT(img: BI, theta: Int, a: T): M =
27        transform(
28            map(Input.getColorsComponents(img, 2), (x: Int) => x.toDouble),
29            theta,
30            a)
31 }
```

A.27 transform.OldTransform.scala

```
1 package transform
2
3 import math._
4 import basic.Types._
5 import image.Input._
6 import image._

8 object OldTransform {
9     /** universal def to wavelt's transform.
10      *  @param transf -- some (wavelet) transform */
11     def wavelet(img: BI, transf: (M, T) => (M, MInt), nameWavelet: String,
12                a: T): (BI, BI, BI) = {
13         val mat: M = getColorsComponents(img, 2).map{_.map{_.toDouble}}
14         val m = mat.length; val n = mat(0).length
15         val (res, dir) = transf(mat, a)
16
17         val resIntImg = res.map{_.map{_.toInt}}
18         Analys.mediate(dir, resIntImg)
```

```

18     val thinyImg = Operation.toImage(resIntImg)
19
20     val dirImg = Operation.toImage(dir)
21
22     val resImg = Operation.matrixToImage(res)
23     Operation.constrast(resImg, 140, 20)
24
25     (resImg, dirImg, thinyImg)
26 }
27
28 def asVesselSpecTransform(img: BI, oldAsVes: wavelets.OldAsVessel): (
29   BI, BI, BI) =
30   wavelet(img, oldAsVes.spectTransform, "asVesselSpec", a = 1)
31 }
```

A.28 transform.AncientEgyptianDecomposition.scala

```

1 package transform
2
3 import basic.Types._
4 import basic.ArrayToolKit._
5
6 class AncientEgyptianDecomposition(wavelet: wavelets.
7   WaveletTransformTrait)
8 extends TransformTrait {
9
10    val transform = new WaveletPacketTransform(wavelet)
11
12    private def oneD(array1: A, fun: A => A, maxLvl: Int = 8): A = {
13      val n = array1.length
14      val array2 = new A(n)
15      var offSet = 0
16      val lvls = basic.Decompose.decompose(n, maxLvl)
17      for(x <- lvls) {
18        val arr1Sub: A = copyPart(array1, offSet, x)
19        val arr2Sub: A = fun(arr1Sub)
20        Array.copy(arr2Sub, 0, array2, offSet, x)
21        offSet += x
22      }
23      array2
24    }
25
26    override def forward1D(arrTime: A) =
27      oneD(arrTime, transform.forward1D)
28    override def reverse1D(arrToReverse: A) =
29      oneD(arrToReverse, transform.reverse1D)
30
31    override def forward1D(arrToForward: A, level: Int): A = forward1D(
32      arrToForward)
33    override def reverse1D(arrToReverse: A, level: Int): A = reverse1D(
34      arrToReverse)
35 }
```

A.29 main.WMain.scala

```

1 package main
2
3 import org.junit.runner.RunWith
4 import org.scalatest.junit.JUnitRunner
5 import parser.FormulaParser
6 import basic.Types._
7
8 object WMain {
9     def main(args: Array[String]) {
10         val dirImage = "/home/misha/Documents/all/images/"
11         val imgNames = Array("01_dr", "01_g", "01_h", "02_dr", "02_g")
12         val names = imgNames.map{dirImage+_+".jpg"}
13         val imgs = names.map{image.Input.getImage(_)}
14         val scaleImgs = imgs.map{image.Operation.scale(_, 0.25)}
15         val mats = scaleImgs.map{image.Input.getColorsComponents(_, 0.3, 0.7
16             , 0.1)}
17         def inv(mat: MInt) = mat.map{_.map{255 - _}}
18         val invMats = mats.map{inv(_)}
19         val invimgs = invMats.map{image.Operation.toImage(_)}
20
21         val dirOut = "/home/misha/out/"
22         val wave = new wavelets.OldAsVessel(4)
23
24         val tifDir = "/home/misha/Documents/all/manual1/"
25         val tifNames = imgNames.map{tifDir + _ + ".tif"}
26         val tifImgs = tifNames.map{image.Input.getTifImage(_)}
27         val tifScaleImgs = tifImgs.map{image.Operation.scale(_, 0.25)}
28
29         for (i <- 0 until imgs.length) {
30             val res = transform.OldTransform.asVesselSpecTransform(invimgs(i),
31                         wave)
32             println("dirOut"+imgNames(i)+".jpg")
33             image.Output.saveImage(res._1, dirOut+imgNames(i)+".jpg", "jpg")
34             image.Output.saveImage(res._2, dirOut+imgNames(i)+"_dir.jpg", "jpg
35             ")
36             image.Output.saveImage(res._3, dirOut+imgNames(i)+"_thiny.jpg", "
37                 jpg")
38             val mat = map(image.Input.getColorsComponents(res._1, 2), (x: Int
39                 => if (x<5) 0 else 255)
40             val binary = image.Operation.toImage(mat, java.awt.image.
41                 BufferedImage.TYPE_BYTE_GRAY)
42             val resCompare = image.Analys.compareBinaryImage(binary,
43                     tifScaleImgs(i))
44             val matThiny = map(image.Input.getColorsComponents(res._3, 2), (x:
45                 Int) => if (x<40) 0 else 255)
46             val binaryThiny = image.Operation.toImage(matThiny, java.awt.image
47                 .BufferedImage.TYPE_BYTE_GRAY)
48             image.Output.visible(binaryThiny, "tit")
49             val resCompareThiny = image.Analys.compareBinaryImage(binaryThiny,
50                     tifScaleImgs(i))
51             println(resCompare)
52             println(resCompareThiny)
53         }
54         /*
55         val name = "/home/misha/2.jpg"

```

```

46     val img = image.Input.getImage(name)
47     val mat = image.Input.getColorsComponents(img, 0.3, 0.7, 0.1)
48     val invMat = mat.map{_.map{255 - _}}
49     val imgRes = image.Operation.toImage(invMat, img.getType)
50     //val imgScale = image.Operation.scale(imgRes, 400.0/imgRes.getWidth
51     )
51     val res = wavelets.OldTransform.asVesselSpecTransform(imgRes, wave)
52     image.Output.saveImage(res._1, name+4, "jpg")
53     */
54     /*
55     val name1 = "/home/misha/images/bestRes.png"
56     val img1 = image.Input.getImage(name1)
57     val mat1 = image.Input.getColorsComponents(img1, 2)
58     val img1Gray = image.Operation.toImage(mat1, 10)
59     val name2 = "/home/misha/idealRes"// "/home/misha/02_dr.tif"
60     val imgTif = image.Input.getImage(name2)
61     val img2 = image.Operation.scale(imgTif, 1000.0/imgTif.getWidth)
62     image.Output.saveImage(imgTif, "/home/misha/02_dr_gray.jpg", "jpg")
63     val res = image.Analys.compareBinaryImage(img1Gray, img2)
64     println(res)
65     */
66   }
67 }
```

A.30 exceptions.BinaryAmountException.scala

```

1 package exceptions
2
3 class BinaryAmountException(valueName: String, n: Int, msg: String=null,
4   cause: Throwable=null)
5   extends AmountItemsException(msg, cause) {
6   override val _msg =
7     s"The number of items error: ${valueName}.length=${n} must be 2^k,
8       where k is integer"
9   override def getMessage = _msg
10 }
```

A.31 exceptions.AmountItemsException.scala

```

1 package exceptions
2
3 class AmountItemsException(msg: String, cause: Throwable = null)
4   extends java.lang.Exception (msg, cause) {
5   val _msg = s"The number of items error: ${msg}"
6   override def getMessage = s"The number of items error: ${msg}"
7 }
```

A.32 tests.AnalysTest.scala

```

1 package tests
2
3 import basic.Types._
4 import org.junit.runner.RunWith
5 import org.scalatest.junit.JUnitRunner
```

```

6 import math._
7 import image._
8
9 @RunWith(classOf[JUnitRunner])
10 object AnalysTest {
11   val dir = "/home/misha/"
12
13   /** vessels accentuation test (0 until 180 by 10 degree) */
14   def vesselSegmentTest {
15     val fileName = dir+"126.jpg"
16     val res = image.Accentuation.AllVesselSegment(fileName)
17     val resImg = Operation.scale(res, 700.0/res.getWidth)
18     // Output.saveImage(res, dir+"02_dr_out.jpg", Input.format)
19     Output.visible(resImg, "Vessel Segment Test")
20   }
21
22   def allLinesTest {
23     val fileName = dir+"01_g.jpg"
24     val img2 = Input.getImage(fileName)
25     val img: BI = Operation.scale(img2, 700.0/img2.getWidth)
26     val res = accentuation.Vessel.allLines(img, 6)
27     image.Output.visible(res, "All Lines Test")
28   }
29
30   def allLinesDirectlyAndRadsTest {
31     val img2: BI = Input.getImage("/home/misha/1.jpg")
32     val img: BI = Operation.scale(img2, 1)
33     val res = accentuation.Vessel.allLinesDirectlyAndRads(img, 5)
34     val dir = res._2
35   }
36
37   /** line (0 degree) accentuation test */
38   def lineSegmentTest {
39     val fileName = dir+"02_dr.jpg"
40     val res = Accentuation.simpleVesselSegment(fileName)
41     val resImg = Operation.scale(res, 700.0/res.getWidth)
42     image.Output.visible(resImg, "Line Segment Test")
43   }
44
45   /** disk accentuation test */
46   def diskTest{
47     val fileName = dir+"126.jpg"
48     val img = image.Input.getImage(fileName)
49     val res = accentuation.Disk.disk(img)
50     val resImg = image.Operation.scale(res, 1)
51     image.Output.visible(resImg, "Disc Test")
52   }
53 }
```

A.33 tests.CWTTest.scala

```

1 package tests
2
3 import basic.Types._
4 import org.junit.runner.RunWith
```

```

5 import org.scalatest.junit.JUnitRunner
6 import math._
7 import image._
8
9 @RunWith(classOf[JUnitRunner])
10 object CWTTest {
11
12     val dir = "/home/misha/"
13
14     def cwtTests {
15         import wavelets._
16         val filename = dir + "01_dr.jpg"
17         val img = image.Input.getImage(filename)
18         for (id <- 1 to 1; a <- 0.4 to 3 by 0.4) {
19             def cwt(wave: ICWavelet) =
20                 image.Transform.cwt(wave, a, id, img)
21             cwt(new AsLongVessel(3, a))
22             cwt(new Morlet(3, a))
23             cwt(new Gauss)
24             //cwt(new AsVessel(3, a))
25             cwt(new MHAT(3, a))
26             //cwt(new FHAT(3, a))
27             //cwt(new Gabor)
28         }
29     }
30 }
```

A.34 tests.DWTTest.scala

```

1 package tests
2
3 import basic.Types._
4 import org.junit.runner.RunWith
5 import org.scalatest.junit.JUnitRunner
6 import math._
7 import image.Transform._
8 import image.Output._
9
10 @RunWith(classOf[JUnitRunner])
11 object DWTTest {
12
13     val dir = "/home/misha/"
14
15     def assertEquals(x: T, y: T, eps: T) =
16         assert{ abs(x-y) < eps }
17
18     def dwtTests{
19         Daubechies1DTest
20         Daubechies2DTest
21         DaubechiesFullImageTest
22         DaubechiesForwardImageTest
23     }
24
25     def Daubechies1DTest{
26         val n = 250
```

```

27     val ar: A = new A(n)
28     for (i <- 0 until n) ar(i) = i
29     for (order <- 1 to 4) {
30       val db = new wavelets.Daubechies(order)
31       val aed = new transform.AncientEgyptianDecomposition(db)
32       val forw = aed.forward1D(ar)
33       val revr = aed.reverse1D(forw)
34       val dif = (0 until ar.length).map{i => abs(ar(i)-revr(i))}.sum
35       assertEquals(dif, 0.0, 1e-10)
36     }
37   }
38
39 def Daubechies2DTest{
40   val mat: M = Array(
41     Array(1, 2, 3, 20, 25),
42     Array(4, 5, 6, 20, 26),
43     Array(7, 8, 9, 20, 27),
44     Array(10, 11, 12, 20, 28))
45   val order = 2
46   for (order <- 1 to 4) {
47     val db = new wavelets.Daubechies(order)
48     val aed = new transform.AncientEgyptianDecomposition(db)
49     val forw = aed.forward2D(mat)
50     val revr = aed.reverse2D(forw)
51
52     val cor = basic.Statistic.correlation(mat, revr)
53     val aver = basic.Statistic.aver(mat)
54     val disp = basic.Statistic.disp(mat, aver)
55     assertEquals(cor, disp, 1e-10)
56   }
57 }
58
59 def DaubechiesFullImageTest{
60   import transform.DTransform._
61   import image.Input._
62   val inpName = dir + "1.jpg"
63   val img: BI = getImage(inpName)
64   val mat = getColorsComponents(img, 2).map{_.map{_.toDouble}}
65   for (ord <- 1 to 4) {
66     val resDB: M = daubechiesForwardAndReverse(mat, order = ord)
67     val cor: T = basic.Statistic.correlation(mat, resDB)
68     val aver: T = basic.Statistic.aver(mat)
69     val disp: T = basic.Statistic.disp(mat, aver)
70     assertEquals(cor, disp, eps = 20)
71   }
72 }
73
74 def DaubechiesForwardImageTest {
75   val inpName = dir + "1.jpg"
76   val img = image.Input.getImage(inpName)
77   val resImg: BI = DaubechiesForwardImage(img, order = 1, "mat")
78   visible(resImg, "Daubechies Forward Image Test")
79 }
80
81 def DaubechiesForwardImageWithRotateTest {

```

```

82     val inpName = dir + "1.jpg"
83     val img = image.Input.getImage(inpName)
84     val (imgTr, imgTheta) =
85         DaubechiesForwardImageWithRotate(img, ord = 1)
86     saveImage(imgTr, s"${dir}tr_im001.jpg", "jpg")
87     saveImage(imgTheta, s"${dir}theta_im001.jpg", "jpg")
88   }
89 }
```

A.35 tests.ImageTest.scala

```

1 package tests
2
3 import basic.Types._
4 import org.junit.runner.RunWith
5 import org.scalatest.junit.JUnitRunner
6 import math._
7 import image._
8
9 @RunWith(classOf[JUnitRunner])
10 object ImageTest {
11   // basic directory of images
12   val dir = "/home/misha/"
13
14   /** test on the allocation of the field direction */
15   def directionTest {
16     val name = dir + "126.jpg"
17     val img = Input.getImage(name)
18     val direct: BI = Analys.direction(img)
19     Output.saveImage(direct, s"${name}_field_out.jpg", Input.format)
20   }
21
22   /** test load, scale and visible .tif image */
23   def imageTifTest {
24     val name = dir + "01_g.tif"
25     val img: BI = Input.getTifImage(name)
26     val resImg = Operation.scale(img, 1000.0/img.getWidth)
27     Output.visible(resImg, "Test load a tif image")
28   }
29
30   /** converted image to binary (1 byte gray color) */
31   private def toBinary(img: BI): BI = {
32     import java.awt.image.BufferedImage
33     val mat = map(Input.getColorsComponents(img, 2), (x: Int) => if (x < 1
34       60) 0 else 255)
35     Operation.toImage(mat, BufferedImage.TYPE_BYTE_GRAY)
36   }
37
38   /** compare two gray images. 2th image is ideal, 1th -- is my result
39   */
40   def compareTest {
41     val imgWT = Input.getImage(dir+"02_dr_out.jpg")
42     val resWt = toBinary(imgWT)
43
44     val imgTif: BI = Input.getTifImage(dir+"02_dr.tif")
```

```

43
44     val resCompare = Analys.compareBinaryImage(resWt, imgTif)
45     println(s"Compare Image Test: result = $resCompare")
46 }
47
48 /** img <- img & mask */
49 def maskTest{
50     val inpName = dir + "01_g_full.jpg"
51     val img = Input.getImage(inpName)
52
53     val inpNameMask = dir+"01_g_mask_full.jpg"
54     val tif: BI = Input.getImage(inpNameMask)
55     val tifScala: BI = Operation.scale(tif, 600.0/tif.getWidth)
56     Output.saveImage(tifScala, dir+"g_mask.jpg", "jpg")
57     val mask = Input.getImage(inpNameMask)
58
59     val colComps = Input.getColorsComponents(img, 0.0, 0.3, 0.7).map{_.map{255 - _}}
60     for (i <- 0 until mask.getHeight; j <- 0 until mask.getWidth)
61         colComps(i)(j) = colComps(i)(j) & mask.getRGB(j, i)
62     val greenImg = Operation.toImage(colComps)
63     Operation.constrast(greenImg)
64     val resImg = Operation.scale(greenImg, 600.0/greenImg.getWidth)
65     Output.saveImage(resImg, dir+"01_g.jpg","jpg")
66     Output.visible(resImg, "Mask Test")
67 }
68 }
```

A.36 tests.AllTest.scala

```

1 package tests
2
3 import basic.Integral
4 import basic.Types._
5 import org.junit.runner.RunWith
6 import org.scalatest.junit.JUnitRunner
7 import math._
8
9 @RunWith(classOf[JUnitRunner])
10 object AllTest {
11     def assertEquals(x: T, y: T, eps: T) =
12         assert{ abs(x-y) < eps }
13
14     def mathTest {
15         integralTests
16         mathToolKitTests
17         constantTest
18     }
19
20     def constantTest {
21         assertEquals( basic.Constants.sins(30), 0.5, 1e-6)
22         assertEquals( basic.Constants.coss(60), 0.5, 1e-6)
23         assertEquals( basic.Constants.coss(90), 0, 1e-6)
24     }
25 }
```

```

26  def integralTests{
27    assert (Integral.getN(0, 2, 0.1) == 21)
28    assert (abs(Integral.simpson(sin, 0, Pi, 0.01) - 2) < 1e-4)
29    def fun(x: T, y: T) = sin(x) * sin(y)
30    assertEquals( Integral.simpson2(fun, (0, 0), (Pi, Pi), 0.01), 4, 1e
31      -4)
32  }
33
34  def mathToolKitTests {
35    import basic.Statistic._
36    import basic.Decompose._
37    val mat1: M = Array(Array(1, 2), Array(3, 4))
38    assert{ abs(aver(mat1)-2.5) < 1e-10}
39    val mat2: M = Array(Array(2, 2), Array(3, 4))
40    assertEquals(
41      correlation(mat1, mat2),
42      (31.0/4 - (10.0/4)*(11.0/4)),
43      1e-10)
44    assert{ decompose(254).equals( List(128, 64, 32, 16, 8, 4, 2)) }
45    assert{ decompose(3).equals( List(2, 1)) }
46    assert{ decompose(254, 32).equals( List(32, 32, 32, 32, 32, 32, 32,
47      16, 8, 4, 2)) }
48    assert{ decompose(3, 1).equals( List(1, 1, 1)) }
49  }

```

A.37 accentuation.Line.scala

```

1 package accentuation
2
3 import basic.Types._
4 import image._
5 import math._
6 import basic.Constants._
7
8 object Line {
9
10  /**
11   * @param img -- image to accentuation
12   * @param innerRadius -- inner radius of line-filter
13   */
14  def lineSegment(
15    img: BI,
16    innerRadius: Int = 9): BI = {
17    val mat = map(Input.getColorComponents(img, 3), (x: Int) => x.
18     toDouble)
19    val resMat = lineSegment(mat, innerRadius)
20    Operation.matrixToImage(resMat, img.getType)
21  }
22
23  def lineSegmentWithRotate(
24    img: BI, theta: Int,
25    innerRadius: Int = 9): BI = {
26    val mat = map(Input.getColorComponents(img, 3), (x: Int) => x.

```

```

        toDouble)
26    val resMat = lineSegment(mat, innerRadius, theta)
27    val res = Operation.matrixToImage(resMat, img.getType)
28    res
29 }
30
31 private def lineSegment(mat: M, innerRadius: Int, fun: (ISInt, Int,
32   Int) => T): M = {
33   val inR = innerRadius
34   val outR = 2*inR // outer radius of line-filter
35   val m = mat.length; val n = mat(0).length
36   val core = (-outR to outR).map{i => if (abs(i) < inR) 1 else -1}
37   def locMask(i: Int, j: Int): T = fun(core, i, j)
38   val res = createM(m, n)
39   (outR until m-outR).par.map {
40     i =>
41       for(j <- outR until n-outR)
42         res(i)(j) = mat(i)(j)*locMask(i, j)
43   }
44   res
45 }
46
47 protected def lineSegment(mat: M, innerRadius: Int): M = {
48   val inR = innerRadius
49   val outR = 2*innerRadius // outer radius of line-filter
50   def locMask(core: ISInt, i: Int, j: Int): T = {
51     val norm = core.sum
52     assert{abs(norm) > 1e-2 && abs(norm) < 100}
53     var sum: T = 0
54     for (x <- -outR to outR)
55       sum += mat(i)(x+j)*core(x+outR)
56     sum/abs(norm)
57   }
58   lineSegment(mat, inR, locMask(_, _, _))
59 }
60
61 protected def lineSegment(mat: M, innerRadius: Int, theta: Int): M = {
62   val inR = innerRadius
63   val outR = 2*innerRadius // outer radius of line-filter
64   def locMask(core: ISInt, i: Int, j: Int): T = {
65     val norm = core.sum
66     assert{abs(norm) > 1e-2 && abs(norm) < 100}
67     var sum: T = 0
68     for (r <- -outR to outR) {
69       val dx = r*coss(theta).round.toInt
70       val dy = r*sins(theta).round.toInt
71       sum += mat(i+dy)(j+dx)*core(r+outR)
72     }
73     sum/abs(norm)
74   }
75   lineSegment(mat, inR, locMask(_, _, _))
76 }

```

A.38 accentuation.Disk.scala

```
1 package accentuation
2
3 import basic.Types._
4 import image._
5 import math._
6
7 object Disk {
8
9     /** disk accentuation
10      *  use a 3th color-components as default
11      *  for use other color-components, use function #disk(mat: M, _, _)
12      *  @param img: image to accentuation disc-like objects
13      *  @param diskRadius -- radius of objects
14      *  @param halfWidth2dMask -- half-width of 2d window
15      *  @see #disk(mat: M, _, _)
16      *  @return image with accentuation's object-as-disk
17      */
18     def disk(img: BI, diskRadius: Int = 10): BI = {
19         val mat = map(Input.getColorsComponents(img, 3), (x: Int) => (x).
20             toDouble)
21         val resMat = disk(mat, diskRadius)
22         Operation.matrixToImage(resMat, img.getType)
23     }
24
25     /**
26      *  disks accentuation
27      *  use function:
28      *      if (x*x+y*y < r*r)
29      *          8*exp(-0.04*(x*x+y*y))
30      *      else
31      *          -1
32      *  use a parallel map
33      *  @param mat: matrix to accentuation disc-like objects
34      *  @param r -- radius of objects
35      *  @param halfWidth2dMask -- half-width of 2d window
36      *  @see #disk(img: BI, _, _)
37      *  @return matrix with accentuation's object-as-disk
38      */
39     def disk(mat: M, r: Int): M = {
40         val m = mat.length; val n = mat(0).length
41         val r2 = r*r
42         val core: M = {
43             val innerDisk = createM(2*r+1, 2*r+1)
44             for (y <- -r to r; x <- -r to r)
45                 if (x*x+y*y <= r2)
46                     innerDisk(y+r)(x+r) = 8*exp(-0.04*(x*x+y*y))
47             val anti = -7
48             val R0st: (Int, T) = {
49                 var norm = innerDisk.map(_.sum).sum
50                 var locR = r+1
51                 var ost = norm
52                 println(norm, locR)
```

```

52         while (norm > 1) {
53             ost = norm
54             norm = max(norm + anti*2*(2*locR+1), 0.5)
55             locR += 1
56         }
57         (locR - 1, ost)
58     }
59     val R = R0st._1; var ost = R0st._2
60     println(R, ost)
61     val outDisk = createM(2*R+1, 2*R+1)
62     val R2 = R*R
63     for (y <- -R to R; x <- -R to R){
64         val s2 = x*x+y*y
65         if (s2 <= r2) outDisk(y+R)(x+R) = innerDisk(y+r)(x+r)
66         if (s2 > r2 && s2 < R2) outDisk(y+R)(x+R) = anti
67         if (s2 == R2) {
68             if (ost >= 1) {
69                 outDisk(y+R)(x+R) = -1
70                 ost -= 1
71             }
72         }
73     }
74     outDisk
75 }
76 val R = (core.length-1)/2
77 val norm = core.map{_.sum}.sum
78 println(norm)
79 assert{abs(norm) > 0.01}
80 def locMask(i: Int, j: Int): T = {
81     var sum: T = 0
82     for (y <- -R to R; x <- -R to R)
83         sum += mat(y+i)(x+j)*core(y+R)(x+R)
84     sum
85 }
86 val res = createM(m, n)
87 (R until m-R).par.map {
88     i =>
89         for(j <- R until n-R)
90             res(i)(j) = mat(i)(j)*locMask(i, j)
91     }
92     res
93 }
94 }
```

A.39 accentuation.Vessel.scala

```

1 package accentuation
2
3 import basic.Types._
4 import image._
5 import math._
6 import basic.Constants._
7
8 object Vessel {
9 }
```

```

10  def allLinesDirectlyAndRads(
11      img: BI,
12      innerRadius: Int = 8): (BI, BI, BI) = {
13  val mat = Input.getColorsComponents(img, 3)
14  val (vesselM, directMInt, thinVesselMat): (M, MInt, MInt) =
15      allLinesDirectlyAndRads(mat, innerRadius)
16  /*
17  val m = img.getHeight; val n = img.getWidth
18  for (y <- 0 until m; x <- 0 until n)
19      if (vesselM(y)(x) > 50)
20      {
21          val r = vesselLengthMat(y)(x)>>1
22          if (r < 20) {
23              val theta = directMInt(y)(x) + 90
24              val rc = (2*r*coss(theta)).round.toInt
25              val rs = (2*r*sins(theta)).round.toInt
26              val (x1, y1) = ((x+rc).max(0).min(n-1), (y+rs).max(0).min(m-1)
27                  )
28              val (x2, y2) = ((x-rc).max(0).min(n-1), (y-rs).max(0).min(m-1)
29                  )
30              vesselM(y1)(x1) = 255
31              vesselM(y2)(x2) = 255
32          }
33      }
34  */
35  val vesImg = Operation.matrixToImage(vesselM, img.getType)
36  Output.visible(vesImg, "ves")
37  val directImg = Operation.toImage(directMInt, img.getType)
38  Output.visible(directImg, "dir")
39  val thinVesselImg = Operation.toImage(thinVesselMat, img.getType)
40  Output.visible(thinVesselImg, "thin")
41  (vesImg, directImg, thinVesselImg)
42
43  def allLinesDirectlyAndRads(mat: MInt, d1: Int): (M, MInt, MInt) = {
44      val d2 = 4*d1/3
45      val m = mat.length; val n = mat(0).length
46      val core = (-d2 to d2).map{i => if (abs(i) < d1) 1 else -3}
47      val norm = core.sum
48      assert{abs(norm) > 1e-2 && abs(norm) < 100}
49      def locMask(i: Int, j: Int, theta: Int): T = {
50          var sum: T = 0
51          for (r <- -d2+1 to d2-1) {
52              val dx = r*coss(theta).round.toInt
53              val dy = r*sins(theta).round.toInt
54              sum += mat(i+dy)(j+dx)*core(r+d2)
55          }
56          sum / abs(norm)
57      }
58      val resTranform: M = createM(m, n)
59      val trueDir: MInt = createMInt(m, n)
60      def allLocMask(i: Int, j: Int) {
61          import image.Analys._
62          var mx = Double.MinValue

```

```

63         var mn = Double.MaxValue
64         for (theta <- 0 until 180 by 10) {
65             val lm = locMask(i, j, theta)
66             if (mat(i)(j) < 128) {
67                 if (mx < lm) {
68                     mx = lm
69                     resTranform(i)(j) = mx
70                     trueDir(i)(j) = theta
71                 }
72             } else {
73                 if (mn > lm) {
74                     mn = lm
75                     //resTranform(i)(j) = mn
76                     trueDir(i)(j) = theta
77                 }
78             }
79         }
80     }
81
82     (d2 until m-d2).par.map {
83         i =>
84             for(j <- d2 until n-d2)
85                 allLocMask(i, j)
86     }
87
88     val res = resTranform.map{_.map{_.toInt}}
89     Analys.mediate(trueDir, res)
90
91     (resTranform, trueDir, res)
92 }
93
94 def allLines(
95     img: BI,
96     d1: Int = 8): BI = {
97     val mat = map(Input.getColorComponents(img, 3), (x: Int) => x.
98                  toDouble)
99     val vesselM = allLines(mat, d1)
100    Operation.matrixToImage(vesselM, img.getType)
101 }
102
103 def allLines(mat: M, d1: Int): M = {
104     val d2 = d1*2
105     val m = mat.length; val n = mat(0).length
106     val core = (-d2 to d2).map{i => if (abs(i) < d1) 1 else -1}
107     val norm = core.sum
108     assert{abs(norm) > 1e-2 && abs(norm) < 100}
109
110     def allLocMask(i: Int, j: Int) =
111         (0 until 180 by 10).map{locMask(i, j, _)}.max / abs(norm)
112
113     def locMask(i: Int, j: Int, theta: Int): T = {
114         val norm = core.sum
115         var sum: T = 0
116         for (r <- -d2+1 to d2-1) {
117             val dx = r*coss(theta).round.toInt

```

```

117         val dy = r*sins(theta).round.toInt
118         sum += mat(i+dy)(j+dx)*core(r+d2)
119     }
120     sum
121 }
122
123 val res = createM(m, n)
124 (d2 until m-d2).par.map {
125     i =>
126         for(j <- d2 until n-d2)
127             res(i)(j) = allLocMask(i, j)
128     }
129
130 res
131 }
132 }
```

A.40 image.Accentuation.scala

```

1 package image
2
3 import basic.Types._
4 import math._
5
6 object Accentuation {
7
8     private def accent(fileName: String, fun: BI => BI): BI =
9         fun(Input.getImage(fileName))
10
11    /** vessels accentuation */
12    def AllVesselSegment(fileName: String): BI =
13        accent(fileName, accentuation.Vessel.allLines(_))
14
15    /** vessels accentuation */
16    def AllVesselFieldAndRadsSegment(fileName: String): (BI, BI, BI) = {
17        val img = Input.getImage(fileName)
18        accentuation.Vessel.allLinesDirectlyAndRads(img)
19    }
20
21    /** line (0 degree) accentuation */
22    def simpleVesselSegment(fileName: String): BI =
23        accent(fileName, accentuation.Line.lineSegment(_))
24 }
```

A.41 image.Generate.scala

```

1 package image
2
3 import java.awt._
4 import java.awt.image._
5 import java.awt.geom.AffineTransform
6
7 import math._
8 import basic.Types._
```

```

9
10 object Generate {
11
12     val dir = "/home/misha/"
13
14     /** generate simple image for wavelet test
15      * @param isVisible Do visible result image?
16      * default value isVisible = false
17      */
18     def generateImageMat(isVisible: Boolean = false): M = {
19         // B is amplitude
20         // C is frequency
21         val B: T = 5; val C: T = 10;
22         val size = 100
23         val picture: M = basic.Types.createWhiteMat(size, size)
24         for (y <- 0 until size; x <- 0 until size)
25             for (A <- 30 to 190 by 30)
26                 picture(y)(x) =
27                     if (abs(y - A - B*sin(x/C)) < 6) 255-A
28                     else min(picture(y)(x), 255)
29         if (isVisible){
30             val gRes: MInt = Operation.toColorMInt(picture)
31             val grayImg: BI = Operation.toImage(gRes, Input.imgType)
32             Output.visible(grayImg, "outImg")
33             Output.saveImage(grayImg, s"${dir}outGenerateImg.jpg", Input.
34                             format)
35         }
36         picture
37     }
38
39     def generateRectCol(){
40         val m = 255; val n = 20;
41         val mat = Array.ofDim[Int](m,n)
42         for (y <- 0 until m){
43             val eignte = 45*255/180 // = 66
44             val col = if (y % eignte != 0) y else 0
45             for (x <- 0 until n)
46                 mat(y)(x) = col
47         }
48         Output.saveImage(mat, s"${dir}colors.jpg", Input.format, 5)
49     }
50
51     def generateImageMat(amplit: T, freq: T, w: Int, h: Int): BI = {
52         val picture: M = basic.Types.createWhiteMat(w, h)
53         for (y <- 0 until h; x <- 0 until w)
54             for (A <- 30 to 190 by 30)
55                 picture(y)(x) =
56                     if (abs(y - A - amplit*sin(x/freq)) < 6) 255-A
57                     else min(picture(y)(x), 255)
58         val gRes: MInt = Operation.toColorMInt(picture)
59         val newRGB = (gRes, gRes, gRes)
60         Operation.createImage(newRGB, Input.imgType)
61     }

```

A.42 image.Transform.scala

```
1 package image
2
3 import basic.Types._
4 import image._
5
6 object Transform {
7
8     def DaubechiesForwardImage(img: BI, order: Int, trans: String): BI = {
9         import image.Input._
10        val mat: M =
11            map(getColorsComponents(img, 2), (x: Int) => x.toDouble)
12        val g: MInt =
13            map(transform.DTransform.daubechies(mat, order, trans), (x: T) =>
14                x.toInt)
15        val resImg: BI = Operation.toImage(g, img.getType)
16        resImg
17    }
18
19    /**
20     * @param ord -- order of Daubechies
21     */
22    def DaubechiesForwardImageWithRotate(img: BI, ord: Int = 1): (BI, BI)
23        = {
24        import transform.DTransform._
25        import image.Operation._
26        val m = img.getHeight; val n = img.getWidth
27        val resTr = createMInt(m, n)
28        val resTheta = createMInt(m, n)
29        for (theta <- 0 until 180 by 10) {
30            println(theta)
31            val imgT: BI = rotate(img, theta)
32            val matt = image.Input.getColorsComponents(imgT, 2).
33                map{_.map{255 - _.toDouble}}
34            val gT: MInt = daubechies(matt, order = 1, transformID = "str").
35                map{_.map{_.toInt}}
36            val resImgT: BI = Operation.toImage(gT, img.getType)
37            val resImg: BI = inverseRotate(resImgT, -theta, img.getWidth, img.
38                getHeight)
39            val locMat = image.Input.getColorsComponents(resImg, 2)
40            for (i <- 0 until m; j <- 0 until n)
41                if (locMat(i)(j) > resTr(i)(j)){
42                    resTr(i)(j) = locMat(i)(j)
43                    resTheta(i)(j) = theta
44                }
45            val imgTr = Operation.toImage(resTr, img.getType)
46            val imgTheta = Operation.toImage(resTheta, img.getType)
47            (imgTr, imgTheta)
48        }
49        /**
50         * @param a -- scala
```

```

50     * @param id -- wave rotate => 1
51     *           image rotate => 2
52   */
53 def cwt(wave: wavelets.ICWavelet, a: T, id: Int, img: BI): (BI, BI, BI
54   ) = {
55   val cwt = new transform.CTransform(wave)
56   def trMor: M => M = cwt.transform(_, 0, a)
57   val wavename = wave.wavename
58
59   import image.Input._
60   val matInt = getColorsComponents(img, 2)
61
62   val mat = matInt.map{_.map{_.toDouble}}
63   val m = img.getHeight; val n = img.getWidth;
64
65   def locTr: Int => M = if (id == 1) locTr1 else locTr2
66   def locTr1(theta: Int): M =
67     cwt.transform(mat, theta, a)
68   def locTr2(theta: Int): M = {
69     import Operation._
70     val imgR: BI = rotate(img, theta)
71     val matR: M = map(getColorsComponents(imgR, 2), (x: Int) => x.
72       toDouble)
73     val matTrR: MInt = map(trMor(matR), (x: T) => x.abs.toInt)
74     val imgTrR: BI = toImage(matTrR, img.getType)
75     val imgTr: BI = inverseRotate(imgTrR, -theta, img.getWidth, img.
76       getHeight)
77     map(getColorsComponents(imgTr, 2), (x: Int) => x.toDouble)
78   }
79
80   val resTr = createM(m, n)
81   val resTheta = image.Analys.direction(matInt, 5, 5)
82   //val resTheta = createMInt(m, n)
83   for (theta <- 0 until 180 by 10){
84     val lTr = locTr(theta)
85
86     for (i <- 0 until m; j <- 0 until n){
87       if (mat(i)(j) >= 128) {
88         if (lTr(i)(j) > resTr(i)(j)){
89           resTr(i)(j) = lTr(i)(j)
90           // resTheta(i)(j) = theta
91         }
92       } else {
93         if (lTr(i)(j) < resTr(i)(j)){
94           resTr(i)(j) = lTr(i)(j)
95           // resTheta(i)(j) = theta
96         }
97       }
98     }
99   }
100
101  val white = Analys.white
102  def mediate(directly: MInt, resMatImg: MInt){
103    val mediateMat = createMBool(m, n)
104    for (y <- 0 until m; x <- 0 until n)

```

```

102         if (resMatImg(y)(x) > white) {
103             val (yMed, xMed): (Int, Int) =
104                 Analys.getMediateLine(resMatImg, x, y, directly(y)(x))
105             if (sqr(yMed-y)+sqr(xMed-x) <= 2)
106                 mediateMat(yMed)(xMed) = true
107         }
108
109         for (y <- 0 until m; x <- 0 until n)
110             if (!mediateMat(y)(x))
111                 resMatImg(y)(x) = 0
112     }
113
114     val res = resTr.map{_.map{_.toInt}}
115     val imgTr: BI = Operation.toImage(res)
116     val imgTheta: BI = Operation.toImage(resTheta)
117     mediate(resTheta, res)
118     val imgTr2: BI = Operation.toImage(res)
119     (imgTr, imgTheta, imgTr2)
120   }
121 }
```

A.43 image.Analys.scala

```

1 package image
2
3 import basic.Types._
4 import math._
5 import java.awt.image._
6 import basic.Constants._
7
8 object Analys {
9   val white = 20
10
11   private def toInt(x: T) = x.round.toInt
12
13   def direction(mat: MInt, R: Int = 5, steptheta: Int = 5): MInt = {
14     val m = mat.length; val n = mat(0).length
15     val res = createMInt(m, n)
16     (0 until m).par.map {
17       i =>
18         for(j <- 0 until n)
19           res(i)(j) = dir(i, j)
20     }
21
22   def dir(i: Int, j: Int): Int = {
23     var maxDisp: Int = Int.MinValue
24     var minDisp: Int = Int.MaxValue
25     var resTheta: Int = 0
26     for (theta <- 0 until 180 by steptheta) {
27       val angle = theta*Pi / 180
28       val rs = (-R to R)
29       val ys = rs.map{r => (i + (r*sin(angle)).floor.toInt).max(0).min
30                     (m-1)}
31       val xs = rs.map{r => (j + (r*cos(angle)).floor.toInt).max(0).min
32                     (n-1)}
```

```

31         val myx = for (i <- 0 until 2*R+1) yield mat(ys(i))(xs(i))
32         val mx = myx.sum
33         val mx2 = myx.map{x => x*x}.sum
34         val s = (2*R.toInt+1).toDouble
35         val disp = (mx2.toDouble/s - (mx*mx).toDouble/(s*s)).floor.toInt
36         if (disp < minDisp) {
37             minDisp = disp
38             resTheta = theta
39         }
40     }
41
42     resTheta
43 }
44
45 (1 until m-1).par.map {
46     i =>
47         for(j <- 1 until n-1)
48             res(i)(j) = locMed(i, j)
49 }
50
51 def locMed(i: Int, j: Int) : Int = {
52     var sum = 0
53     for (dy <- -1 to 1; dx <- -1 to 1)
54         sum += res(i+dy)(j+dx)
55     sum / 9
56 }
57
58 res
59 }
60
61 def mediate(directly: MInt, resMatImg: MInt){
62     val m = directly.length; val n = directly(0).length
63     val mediateMat = createMBool(m, n)
64     for (y <- 0 until m; x <- 0 until n)
65         if (resMatImg(y)(x) > white) {
66             val (yMed, xMed): (Int, Int) =
67                 AnalyS.getMediateLine(resMatImg, x, y, directly(y)(x))
68             if (sqr(yMed-y)+sqr(xMed-x) <= 2)
69                 mediateMat(yMed)(xMed) = true
70         }
71
72         for (y <- 0 until m; x <- 0 until n)
73             if (!mediateMat(y)(x))
74                 resMatImg(y)(x) = 0
75             else 255
76     }
77
78 def direction(img: BI): BI = {
79     val mat = image.Input.getColorsComponents(img, 2)
80     val dir: MInt = direction(mat)
81     image.Operation.toImage(dir)
82 }
83
84 /**
85 * @param img1 -- my image: type == 10, all values is -1 or 0

```

```

86     * @param img2 -- ideal image: type == 10, all values is -1 or 0
87     * @return pair (errorWhite/white, errorBlack/black)
88     */
89     def compareBinaryImage(img1: BI, img2: BI): (T, T) = {
90         val pixs1 = Input.getPixels(img1)
91         val pixs2 = Input.getPixels(img2)
92         if (pixs1.length != pixs2.length)
93             throw new Exception("Probably, Image Format Exception or Size
94                         Image Exception")
95         val n = pixs1.length
96         var white = 0
97         var errorWhite = 0
98         var black = 0
99         var errorBlack = 0
100        val m1: Byte = -1 // probably, it is white
101        val m0: Byte = 0 // probably, it is black
102        for (i <- 0 until n) {
103            if (pixs2(i) == m1) {
104                white += 1
105                if (pixs1(i) != m1)
106                    errorWhite += 1
107                } else if (pixs2(i) == m0)
108                {
109                    black += 1
110                    if (pixs1(i) != m0)
111                        errorBlack += 1
112                }
113            assert{white+black == n}
114            (errorWhite.toDouble / white, errorBlack.toDouble / black)
115        }
116
117    /**
118     * @param imgMat -- components of image, access to element in format (
119     * h,w)
120     * @param angle -- (directly of blood vessel - pi/2), in [-pi/2, pi/2)
121     * .
122     * @param x -- current value pixel's width
123     * @param y -- current value pixel's height
124     * @return (r1, r2),
125     * where #r1 -- radius of the motion in the direction of the #angle
126     * from the point (#x, #y) until the last black dot in, #r2 -- in
127     * the direction of the (#angle+Pi).
128     */
129     def radsWhiteLine(imgMat: MInt, theta: Int, x: Int, y: Int): (Int, Int
130         ) = {
131         val m = imgMat.length; val n = imgMat(0).length
132         /** @see image.ImageWaveletInterface.white */
133         def isWhite(x: Int, y: Int) = imgMat(y)(x) > white
134         def inMatrix(x: Int, y: Int) = (x >= 0 && y >= 0 && x < n && y < m)
135         def r(theta: Int): Int = {
136             val angle = theta*Pi/180
137             val cosa = cos(angle); val sinA = sin(angle)
138             var r = 0
139             var kx = x; var ky = y

```

```

136     while (inMatrix(kx, ky) && isWhite(kx, ky)) {
137         kx = x+toInt(r*cosA)
138         ky = y+toInt(r*sinA)
139         r += 1
140     }
141     r
142 }
143
144 (r(theta)-1, r(theta+180)-1)
145 }
146
147 def lengthWhiteLine(imgMat: MInt, theta: Int, x: Int, y: Int): Int = {
148     val rads = radsWhiteLine(imgMat, theta, x, y)
149     rads._2 + rads._1
150 }
151
152 /**
153 * @param imgMat -- components of image, access to element in format (
154 * h,w)
155 * @param theta -- directly of line, in [0, 180].
156 * @param x -- current value pixel's width
157 * @param y -- current value pixel's height
158 * @return (yMediate, xMediane) -- coordiane of mediate white line:
159 *      (theta - 90[degree] = directly(line)) & ((y,x) in line)
160 */
161 def getMediateLine(imgMat: MInt, x: Int, y: Int, theta: Int): (Int,
162     Int) = {
163     val (r1, r2) = radsWhiteLine(imgMat, theta+90, x, y)
164     val cosA = coss(theta+90); val sinA = sins(theta+90)
165     val xMed = x + toInt((r1-r2)*cosA/2)
166     val yMed = y + toInt((r1-r2)*sinA/2)
167     (yMed, xMed)
168 }
169 }
```

A.44 image.Output.scala

```

1 package image
2
3 import java.awt._
4 import java.io.File
5 import javax.imageio.ImageIO
6 import javax.swing._
7 import java.awt.image._
8 import basic.Types._
9
10 object Output {
11
12     def saveImage(im: BufferedImage, fileName: String, format: String):
13         Unit =
14         ImageIO.write(im, format, new File(fileName))
15     def saveImage(mat: MInt, fileName: String, format: String, imgType:
```

```

        Int): Unit = {
16    val outImg = Operation.toImage(mat, imgType)
17    saveImage(outImg, fileName, format)
18  }
19
20  def saveImage(mat: M, fileName: String, format: String, imgType: Int): Unit =
21    saveImage(Operation.toColorMInt(mat), fileName, format, imgType)
22
23  /** visualization image through frame */
24  def visible(image: BufferedImage, title: String) {
25    val frame = new JFrame()
26    val icon = new ImageIcon(image)
27    val label = new JLabel(icon)
28    frame.getContentPane().add(label, BorderLayout.CENTER)
29    frame.pack()
30    frame.setName(title)
31    frame.setTitle(title)
32    frame.setVisible(true)
33    frame.setDefaultCloseOperation( WindowConstants.DISPOSE_ON_CLOSE )
34  }
35
36  def visualisationAndSaveMat(mat: M, frameName: String, fileName: String) {
37    val r = 0.213; val g = 0.715; val b = 0.072;
38    val rDivG = r/g; val bDivG = b/g
39    val gRes = Operation.toColorMInt(mat)
40    def multi(mat: MInt, a: T): MInt = map(mat, (x: Int) => (x*a).
41      toInt)
42    val newRGB = (multi(gRes, rDivG), gRes, multi(gRes, bDivG))
43    val grayImg = Operation.createImage(newRGB, Input.imgType)
44    Output.visible(grayImg, frameName)
45    Output.saveImage(grayImg, fileName, Input.format)
46  }

```

A.45 image.Input.scala

```

1 package image
2
3 import java.io.File
4 import javax.imageio.ImageIO
5 import java.awt.image.-
6
7 import basic.Types.-
8
9 object Input {
10   val format = "jpg"
11   val imgType = BufferedImage.TYPE_INT_RGB
12   val imgColor = java.awt.Color.BLACK
13
14 /**
15  * @param name name of file
16  * @return image from file with name @name
17 */

```

```

18  def getImage(name: String): BI = ImageIO.read(new File(name))
19
20 def getTifImage(name: String): BI = {
21   import com.sun.media.jai.codec._
22   import java.awt.image.renderable.ParameterBlock
23   import javax.media.jai._
24   val stream = new FileSeekableStream(name)
25   val decodeParam = new TIFFDecodeParam(){this.
26     setDecodePaletteAsShorts(true)}
27   val params = new ParameterBlock() {this.add(stream)}
28   JAI.create("tiff", params).getAsBufferedImage
29 }
30 /**
31 * @param img some image
32 * @return all pixels as array from raster data
33 */
34 def getPixels(img: BufferedImage): Array[Byte] =
35   img.getRaster.getDataBuffer().asInstanceOf[DataBufferByte].getData()
36
37 def getColorsComponents(img: BI, colorID: Int): MInt = {
38   val n = img.getWidth; val m = img.getHeight
39   val shift = (colorID-1)*8
40   val res = createMInt(m, n)
41   for (y <- 0 until m; x <- 0 until n) {
42     res(y)(x) = (img.getRGB(x, y) >> shift)&255
43   }
44 }
45
46 def getColorsComponents(img: BI, cb: T, cg: T, cr: T) = {
47   val n = img.getWidth; val m = img.getHeight
48   val res = createMInt(m, n)
49   for (y <- 0 until m; x <- 0 until n) {
50     val rgb = img.getRGB(x, y)
51     val b = rgb & 255
52     val g = (rgb >> 8) & 255
53     val r = (rgb >> 16) & 255
54     res(y)(x) = (b*cb+g*cg+r*cr).floor.toInt
55   }
56   res
57 }
58 }
```

A.46 image.Operation.scala

```

1 package image
2
3 import java.awt._
4 import java.awt.image._
5 import java.awt.geom.AffineTransform
6
7 import math._
8 import basic.Types._
9
```

```

10 object Operation {
11
12     /** forall cell in mat: cell => cell.toInt.max(0).min(255) */
13     def toColorMInt(mat: M): MInt = map(mat, (x: T) => x.toInt.max(0).min(
14         255))
15
16     /** x => 255*(x-1)/(h-1) */
17     def contrast(img: BI, h: Int = 200, l: Int = 50) {
18         val m = img.getHeight; val n = img.getWidth
19         def toNew(x: Int) =
20             if (x > h) 255 else if (x < l) 0 else 255*(x-1)/(h-1)
21         for (y <- 0 until m; x <- 0 until n) {
22             val rgb = img.getRGB(x, y)
23             val b = rgb & 255
24             val g = (rgb >> 8) & 255
25             val r = (rgb >> 16) & 255
26             val newRgb = toNew(b) + (toNew(g) << 8) + (toNew(r) << 16)
27             img.setRGB(x, y, newRgb)
28         }
29     }
30
31     /** full copy of image */
32     def deepCopy(bi: BI): BI = {
33         val cm: ColorModel = bi.getColorModel()
34         val isAlphaPremultiplied = cm.isAlphaPremultiplied()
35         val raster: WritableRaster = bi.copyData(null)
36         new BI(cm, raster, isAlphaPremultiplied, null)
37     }
38
39     /** simple rotate image on theta degree.
40      * To rotate use #rotate */
41     private def simpleRotate(img: BI, theta: T): BI = {
42         val w = img.getWidth; val h = img.getHeight
43         val angle: T = theta*Pi/180
44         val at = new AffineTransform()
45
46         at.translate(w, h)
47         at.rotate(angle)
48         at.translate(-w/2, -h/2)
49
50         affineTransform(img, at, w, h)
51     }
52
53     /** rotate image on theta degree (theta in (-180; 180])
54      * resImg bigger img: (w, h)==size(img) -> (R,R)==size(resImg)
55      **/
```

55 def rotate(img: BI, theta: T): BI = {
56 val resImg = simpleRotate(img, theta)
57 val curW = resImg.getWidth; val curH = resImg.getHeight;
58 val R = sqrt(sqr(img.getWidth)+sqr(img.getHeight)).toInt;
59 resImg.getSubimage((curW-R)/2, (curH-R)/2, R, R)
60 }
61
62 /** rotate image from theta degree (theta in (-180; 180])
63 * new image as init image: (oldW,oldH) -> (R,R)==size(img) -> (oldW,

```

        oldH)
64     */
65     def inverseRotate(img: BI, theta: T, oldW: Int, oldH: Int): BI = {
66       val resImg = simpleRotate(img, theta)
67       val curW = resImg.getWidth; val curH = resImg.getHeight;
68       resImg.getSubimage((curW-oldW)/2, (curH-oldH)/2, oldW, oldH)
69     }
70
71     /** @return scaled image */
72     def scale(img: BI, scale: T): BI = {
73       val newW = (img.getWidth * scale).round.toInt
74       val newH = (img.getHeight * scale).round.toInt
75       val res: BI = new BI(newW, newH, img.getType)
76       val graph = res.createGraphics
77       val transf = AffineTransform.getScaleInstance(scale, scale)
78       graph.drawRenderedImage(img, transf)
79       res
80     }
81
82     /**
83      * @param trans some affine transform
84      * @param img used to obtain information about the image
85      * @return new image on white background before transform
86      */
87     def affineTransform(img: BI, trans: AffineTransform, w: Int, h: Int): BI =
88       {
89         val resImg = new BI(2*w, 2*h, img.getType)
90         val riGraphic = resImg.createGraphics()
91         riGraphic.setBackground(Input.imgColor)
92         riGraphic.clearRect(0, 0, 2*w, 2*h)
93         resImg.getGraphics.asInstanceOf[Graphics2D].drawImage(img, trans,
94           null)
95         resImg
96       }
97
98     /** Let supMat == (red, green, blue) - components
99      * @return red<<16 + green<<8 + blue
100     */
101     def createImage(supMat: (MInt, MInt, MInt), imgType: Int): BI = {
102       val r = supMat._1; val g = supMat._2; val b = supMat._3
103       val h = r.length; val w = r(0).length;
104       val img = new BI(w, h, imgType)
105       for(x <- 0 until w; y <- 0 until h){
106         val c = b(y)(x) +
107             (g(y)(x) << 8) +
108             (r(y)(x) << 16)
109         img.setRGB(x, y, c)
110       }
111     }
112     /** @return mat<<16 + mat<<8 + mat
113     */
114     def toImage(mat: MInt, imgType: Int = Input.imgType): BI = {
115       val newRGB = (mat, mat, mat)

```

```

116     val grayImg = Operation.createImage(newRGB, imgType)
117     grayImg
118   }
119
120  /** @see this.toImage */
121  def matrixToImage(mat: M, imgType: Int = Input.imgType): BI =
122    toImage(Operation.toColorMInt(mat), imgType)
123
124  /** @see #basic.MathToolKit.correlation */
125  def correlation(img1: BI, img2: BI, colorID: Int): T = {
126    val mat1 = map(Input.getColorsComponents(img1, colorID), (x: Int) =>
127      x.toDouble)
128    val mat2 = map(Input.getColorsComponents(img2, colorID), (x: Int) =>
129      x.toDouble)
130    val cor = basic.Statistic.correlation(mat1, mat2)
131    cor
132  }
133
134  /** @see #basic.MathToolKit.disp */
135  def disp(img: BI, colorID: Int): T = {
136    val mat = map(Input.getColorsComponents(img, colorID), (x: Int) => x
137      .toDouble)
138    val aver = basic.Statistic.aver(mat)
139    basic.Statistic.disp(mat, aver)
140  }

```