**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по производственной практике (НИР)**

Тема: Доработка компонентов САПР полиграфической системы на основе аддитивной технологии

|  |  |  |
| --- | --- | --- |
| Студент гр. 2301 |  | Мезенцев Р.С. |
| Руководитель |  | Сольницев Р.И. |

Санкт-Петербург

2017

**ЗАДАНИЕ**

**на производственную практику (НИР)**

|  |  |  |
| --- | --- | --- |
| Студент Мезенцев Р.С. | | |
| Группа 2301 | | |
| Тема практики: Доработка компонентов САПР полиграфической системы на основе аддитивной технологии | | |
| Задание на практику:   1. Обзор литературы 2. Исследование алгоритмов обработки изображений 3. Реализация градиентного метода в среде MATLAB 4. Реализация алгоритма на платформе .Net 5. Тестирование программного обеспечения 6. Написание пояснительной записки | | |
| Сроки прохождения практики: 16.10.2017– 24.12.2017 | | |
| Дата сдачи отчета: 28.12.2017 | | |
| Дата защиты отчета: 28.12.2017 | | |
|  | | |
| Студент |  | Мезенцев Р.С. |
| Руководитель |  | Сольницев Р.И. |

**Аннотация**

В данной работе была проведена доработка компонентов САПР полиграфической системы на основе аддитивной технологии. Доработки включают в себя как улучшение уже используемых алгоритмов обработки изображений, так и добавление новых.

**Summary**

The purpose of the practice is the CAD components of the printing system based on additive technology were refined. Improvements include both improving the already used image processing algorithms, as well as adding new ones.

# СОДЕРЖАНИЕ

[Введение 5](#_Toc503104960)

[1. Исследование алгоритмов 6](#_Toc503104961)

[1.1. Оператор Робертса 8](#_Toc503104962)

[1.2. Оператор Превитта 9](#_Toc503104963)

[1.3. Оператор Собеля 10](#_Toc503104964)

[2. Реализация градиентного метода в среде MATLAB. 11](#_Toc503104965)

[3. Реализация алгоритма на платформе .Net 20](#_Toc503104966)

[4. Тестирование 21](#_Toc503104967)

[Заключение 24](#_Toc503104968)

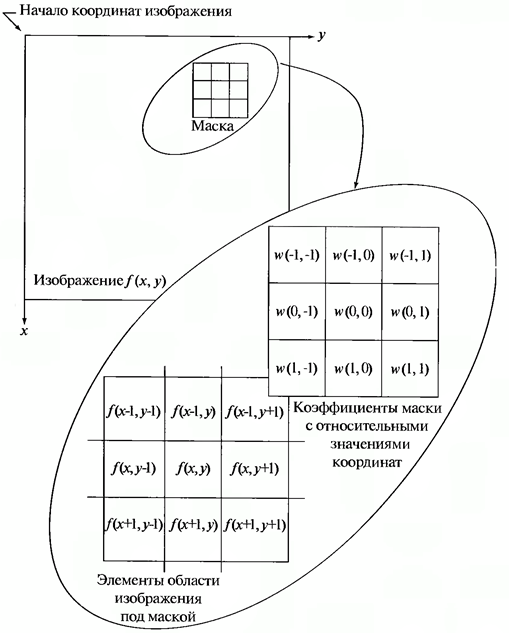
[Список литературы 25](#_Toc503104969)

# Введение

Целью работы является доработка существующих алгоритмов набора .Net библиотек SmearsMaker, а также добавление новой функциональности. Необходимо добавить возможность формирования мазков с использованием градиентных методов, корректировку настроек работы алгоритмов, а также отображение статуса текущей работы на клиентском пользовательском интерфейсе.

# Исследование алгоритмов

Схема пространственной фильтрации иллюстрируется на рисунке 1.

  
Рисунок 1 **–** Схема пространственной фильтрации

Процесс основан на простом перемещении маски фильтра от точки к точке изображения; в каждой точке (x,y) отклик фильтра вычисляется с использованием предварительно заданных связей. В случае линейной пространственной фильтрации отклик задается суммой произведения коэффициентов фильтра на соответствующие значения пикселей в области, покрытой маской фильтра. Для маски 3х3 элемента, показанной на рисунке 1, результат (отклик)***R*** линейной фильтрации в точке (x,y) изображения составит:

**image**(1.1)

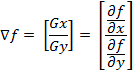
что, как видно, есть сумма произведений коэффициентов маски на значения пикселей непосредственно под маской. В частности заметим, что коэффициент w(0,0) стоит при значении f(x,y), указывая тем самым, что маска центрирована в точке (x,y). При обнаружении перепадов яркости используются дискретные аналоги производных первого и второго порядка. Для простоты изложения будут рассмотрены одномерные производные.   
 Первая производная одномерной функции f(x) определяется как разность значений соседних элементов:

**image**(1.2)

Здесь использована запись в виде частной производной для того, чтобы сохранить те же обозначения в случае двух переменных ***f(x,y)***, где придется иметь дело с частными производными по двум пространственным осям. Использование частной производной не меняет существа рассмотрения.  
  
 Аналогично, вторая производная определяется как разность соседних значений первой производной:

**image**(1.3)

Вычисление первой производной цифрового изображения основано на различных дискретных приближениях двумерного градиента. По определению, градиент изображения f(x,y) в точке (x,y) — это вектор:

(1.4)

Как известно из курса математического анализа, направление вектора градиента совпадает с направлением максимальной скорости изменения функции f в точке (x,y) [2].

Важную роль при обнаружении контуров играет модуль этого вектора, который обозначается ∇f и равен

image(1.5)

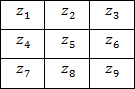
Эта величина равна значению максимальной скорости изменения функции f в точке (x,y), причем максимум достигается в направлении вектора ∇f. Величину ∇f также часто называют градиентом.  
 Направление вектора градиента также является важной характеристикой. Обозначим α(x,y) угол между направлением вектора ∇f в точке (x,y) и осью x. Как известно из математического анализа,

image(1.6)

Отсюда легко найти направление контура в точке (x,y), которое перпендикулярно направлению вектора градиента в этой точке. А вычислить градиент изображения можно, вычислив величины частных производных ∂f/∂x и ∂f/∂y для каждой точки.

## Оператор Робертса

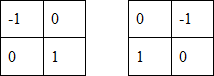
Пусть область 3х3, показанная на рисунке ниже (см. рис. 2), представляет собой значения яркости в окрестности некоторого элемента изображения.

  
Рисунок 2. Окрестность 3х3 внутри изображения

Один из простейших способов нахождения первых частных производных в точке image состоит в применении следующего перекрестного градиентного оператора Робертса [2]:

image(1.7)  
и  
image (1.8)

Эти производные могут быть реализованы путем обработки всего изображения с помощью оператора, описываемого масками на рисунке 3, используя процедуру фильтрации, описанную ранее.

  
Рисунок 3. Маски оператора Робертса

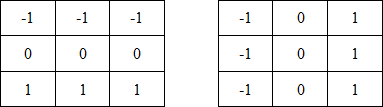
Реализация масок размерами 2х2 не очень удобна, т.к. у них нет четко выраженного центрального элемента, что существенно отражается на результате выполнения фильтрации. Но этот «минус» порождает очень полезное свойство данного алгоритма – высокую скорость обработки изображения.

## Оператор Превитта

Оператор Превитта, так же как и оператор Робертса, оперирует с областью изображения 3х3, представленной на рисунке 2, только использование такой маски задается другими выражениями [2]:

image(1.9)  
и  
image (1.10)

В этих формулах разность между суммами по верхней и нижней строкам окрестности 3х3 является приближенным значением производной по оси ***x***, а разность между суммами по первому и последнему столбцам этой окрестности – производной по оси ***y***. Для реализации этих формул используется оператор, описываемый масками на рисунке 4, который называется оператором Превитта.

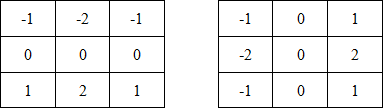
  
Рисунок 4. Маски оператора Превитта

## Оператор Собеля

Оператор Собеля тоже использует область изображения 3х3, отображенную на рисунке 2. Он довольно похож на оператор Превитта, а видоизменение заключается в использовании весового коэффициента 2 для средних элементов [2]:

image(1.11)  
и  
image (1.12)

Это увеличенное значение используется для уменьшения эффекта сглаживания за счет придания большего веса средним точкам.   
  
 Маски, используемые оператором Собеля, отображены на рисунке 5.

  
Рисунок 5**.** Маски оператора Собеля

Рассмотренные выше маски применяются для получения составляющих градиента image. Для вычисления величины градиента эти составляющие необходимо использовать совместно[1]:

image (1.14)  
или  
image (1.15)

# Реализация градиентного метода в среде MATLAB.

Наша система машинной живописи использует кисть с круглым кончиком, непрозрачные акриловые краски и белый холст. Время высыхания красок составляет около часа, так что они могут смешиваться друг с другом на границе мазков, но их вязкость препятствует полному перемешиванию. Это позволяет нам исключить эффекты диффузии из моделирования и использовать простую модель мазка (рис.6). В нашем алгоритме мазок представлен как след от окружности, двигающейся по траектории, состоящей из прямых соединенных между собой отрезков.

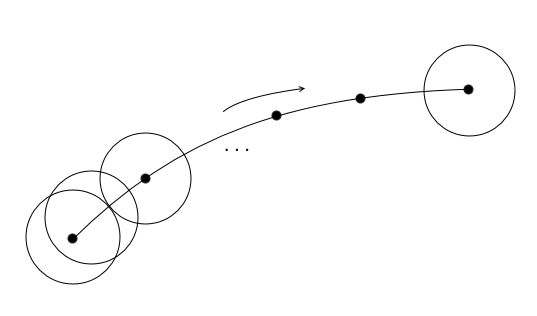


Рис. 6. Геометрическая интерпретация выбранной модели мазка

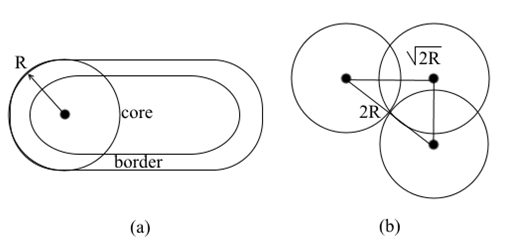


Рис. 7. Модель мазка: (a) Ядро и граница, (b) распределение начальных точек

С целью накладывать мазки на холст плотно, но и без излишнего перекрытия, мы представили каждый из них в виде ядра (core) и граничной части (border) (рис.7, а). Радиус ядра равне *Rc*, радиус мазка равен *R*. Радиус ядра мазка составляет примерно половину радиуса кисти (задается в настройках). Мы размещаем начальные точки на регулярной сетке с шагом, например,  (можно также задавать R, это тоже нужно задавать в настойках) (Рис.7, b).

В ходе работы используются два холста: один холст canvas, на котором рисуются мазки, и второй холст canvas2, на котором рисуется карта мазков.

Исходно, холст canvas белый, canvas2 черный.

В ходе экспериментов мы проверили два типа измерения ошибки. Первый из них это 1-норма в цветовом пространстве RGB



Где *c* – цвет мазка,  - цвет пикселя и *n* число задействованных пикселей исходного изображения.

Вторая мера ошибки это 2-норма (евклидова норма) в цветовом пространстве CIE-Lab:



Несмотря на то, что цветовое пространство CIE-Lab было разработано для достижения большей точности в контексте восприятия человеком, мы обнаружили, что субъективная разница между итоговыми изображениями зависит больше от других настроек алгоритма, чем от выбранной меры погрешности.

**Алгоритм**

1. Загрузка изображения.

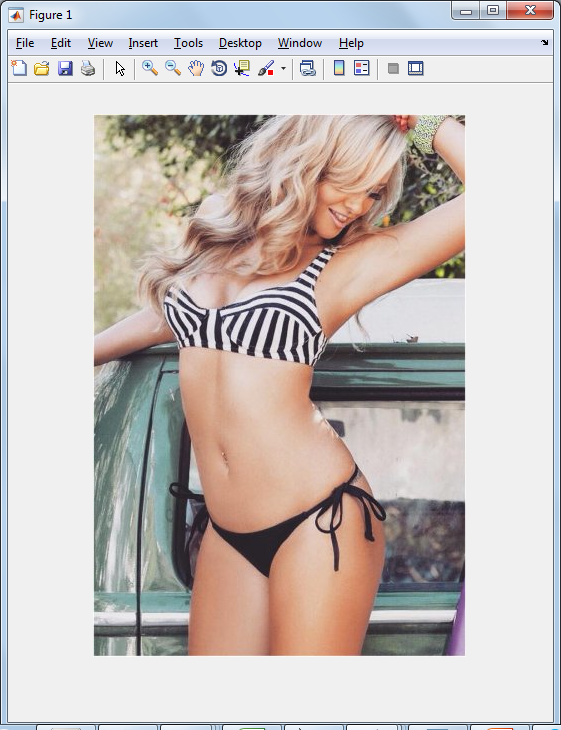


Рисунок 8 – Исходное изображение

2. Размытие в соответствии с радиусом кисти. В MATLAB [3] за это отвечает код

H = fspecial('disk',bs2);

img = imfilter(img,H,'replicate');

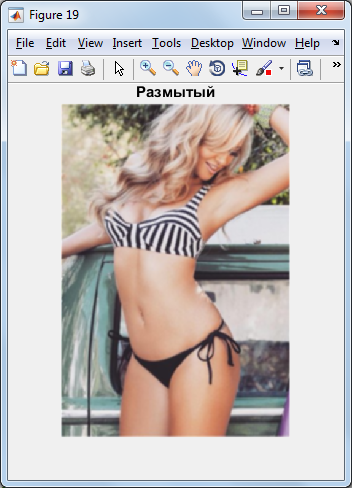


Рисунок 9 – Размытое изображение

3. Нахождение направления, перпендикулярного градиенту яркости изображения в каждой точке. В MATLAB за это отвечает код

[~,Gdir] = imgradient(imggray);

Gdir = (Gdir)\*pi/180;

При визуализации, если черный соответствует углу –pi, белый — углу pi, а промежуточные углы – промежуточными оттенками серого, то получится так:

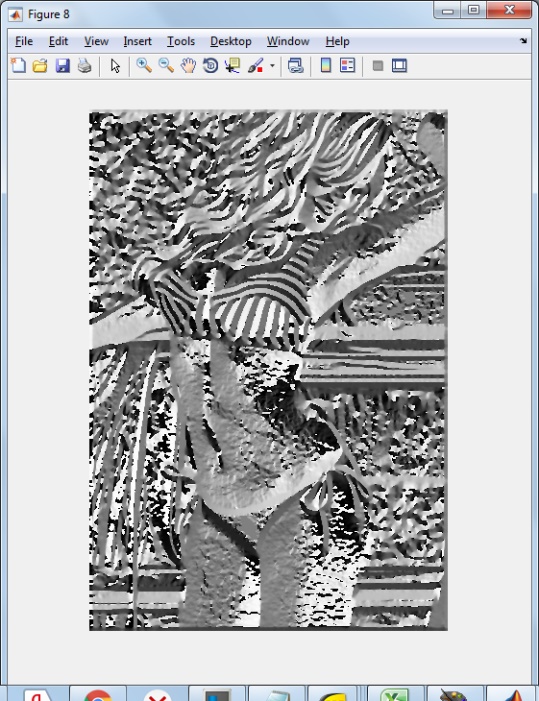


Рисунок 10 – Карта градиентов

4. Грубая проработка. Идем по холсту с заданным шагом, и если в текущей точке погрешность цвета между холстом и изображением больше tol, то:

4.1. Начинаем мазок. Его цвет равен среднему цвету по пикселям, попавшим в кружок, соответствующий следу кисти.

4.2. Проверка условия: мазок меньше максимальной длины и цветовая дистанция между следующим сегментом мазка и текущей областью исходного изображения не превышает допустимое значение tol.

Если оно выполнено, следующая точка выбирается в соответствии с направлением перпендикуляра градиента Gdir.

Код MATLAB

if(nPts > 1) %если точек в мазке больше одной, то определим

%направление предыдущего сегмента мазка

dX = pX - pxs.Xs(end-1);

dY = pY - pxs.Ys(end-1);

if dX ~= 0

al0 = atan(dY/dX);

if dY > 0 && dX < 0

al0 = pi + al0;

end

if dY < 0 && dX < 0

al0 = al0 - pi;

end

else

if dY > 0

al0 = pi/2;

else

al0 = -pi/2;

end

end

else

al0 = Gdir(pX,pY);

end

%al0 - предыдущее направление мазка

%выберем новое так, чтобы не было излома мазка

if abs(al0 - Gdir( pX,pY)) > pi/2

alpha = Gdir(pX,pY) - pi;

else

alpha = Gdir(pX,pY);

end

Длина нового сегмента выбирается либо исходя из радиуса кисти, например, 2R, либо случайным образом с помощью функции rand (задается в настройках).

Новая точка проверяется на попадание в границы изображения. Если она располагается в кадре и попадает в точку, где погрешность текущего цвета холста больше tol (он либо пуст, либо закрашен в этой точке неудачным мазком на предыдущей итерации), то:

Подсчитаем, сколько пикселей потенциально нового сегмента попали на области, ранее закрашенные ядрами мазков. Для этого используется canvas2. На данной итерации перекрытие minOverlap должно быть не более 20% (число настраивается)

Код MATLAB:

%параметрическая линия

N = max(abs(pX - nX), abs(pY - nY));

if N == 0

N = 1;

end

for t = 0:1/N:1

xo = round(nX + (pX - nX)\*t);

yo = round(nY + (pY - nY)\*t);

for Xl = round(xo - bs2):round(xo + bs2)

for Yl = round(yo - bs2):round(yo + bs2)

if Xl > 0 && Xl <= m && Yl > 0 && Yl <= n

if canvas(Xl,Yl,1) >= col(1) %если цвет пикс. холста светлее

r2 = (double(Xl) - xo)^2 + (double(Yl) - yo)^2;

if r2 < bsQuad

colSum = colSum + imglab(Xl,Yl,:);

ncol = ncol + 1; %число пикселей в мазке

if(canvas2(Xl,Yl) ~= 0)

overlap = overlap + 1;

end

end

end

end

end

end

end

avrcol = colSum./ncol;

ovf = minOverlap;

overlap = overlap/ncol;

if ncol == 1

overlap = ovf;

end

Здесь же учитываем, что более темные мазки перекрывают более светлые (особенность работы робота). Следует предусмотреть возможность обращение к таблице порядка нанесения цветов (поскольку данных по машине еще нет, то таблицы тоже нет).

Если перекрытие меньше minOverlap, то сегмент принимается:

* Рисуется новый сегмент на карте мазков неким случайным цветом (не черным). Его радиус равен радиусу ядра.
* Рисуется новый сегмент на холсте цветом мазка. Радиус равен R. Учитываем, что более темные мазки перекрывают более светлые.

Важно: мазок должен содержать минимум две точки (один отрезок). Иначе он не считается мазком и не добавляется в итоговый список мазков.

4.3 Этап заканчивается, когда весь холст пройден. В итоге получается

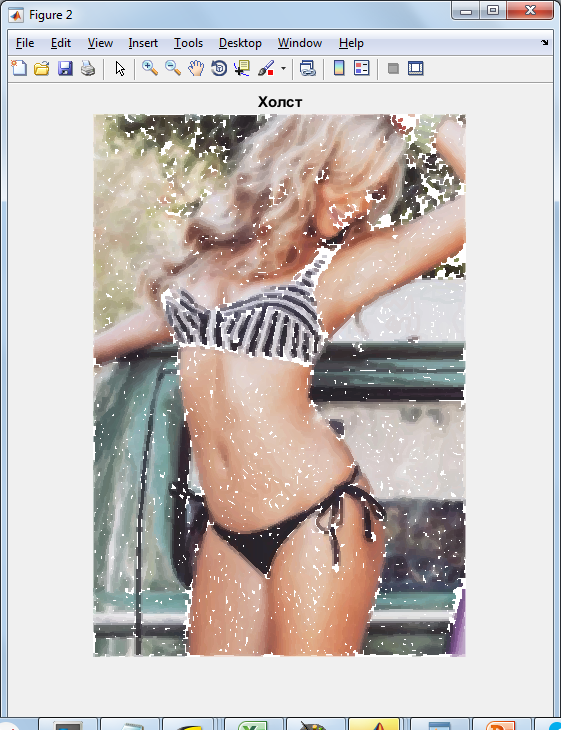
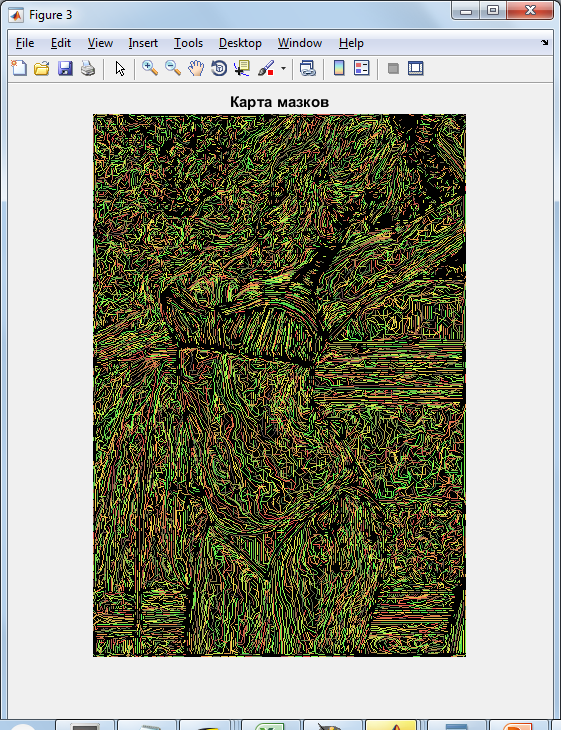


Рисунок 11 – Карта мазков

5. Закраска пробелов.

*Текущая реализация*:

Идем по холсту с шагом в 1 пиксель, и если в текущей точке погрешность цвета между холстом и изображением больше tol, то:

5.1. Повторение тех же действий, что и в пп.4.1-4.2, но:

а) новый сегмент пробуется несколько раз в случайном направлении со случайной длиной (градиент не используется), число итераций (попыток) настраивается параметром MaxIters. Выбирается сегмент с наименьшей средней погрешностью.

б) Коэффициент перекрытия на данном этапе maxOverlap равен 90% (настраивается).

5.2. В итоге получается (новые мазки голубыми оттенками).

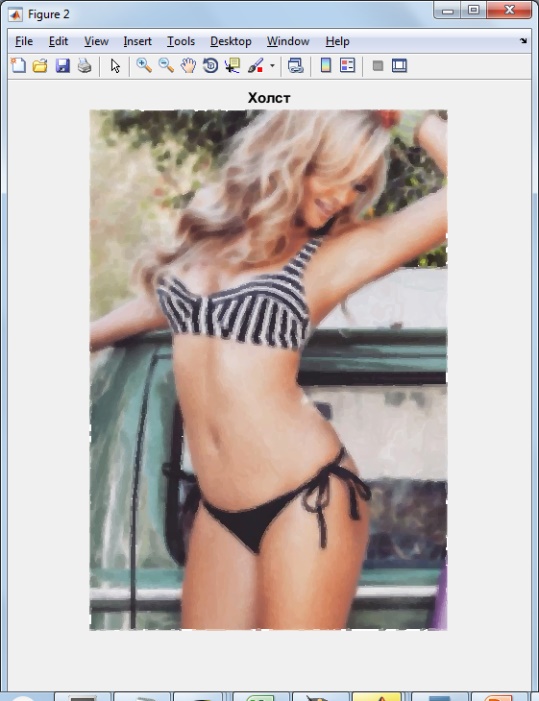
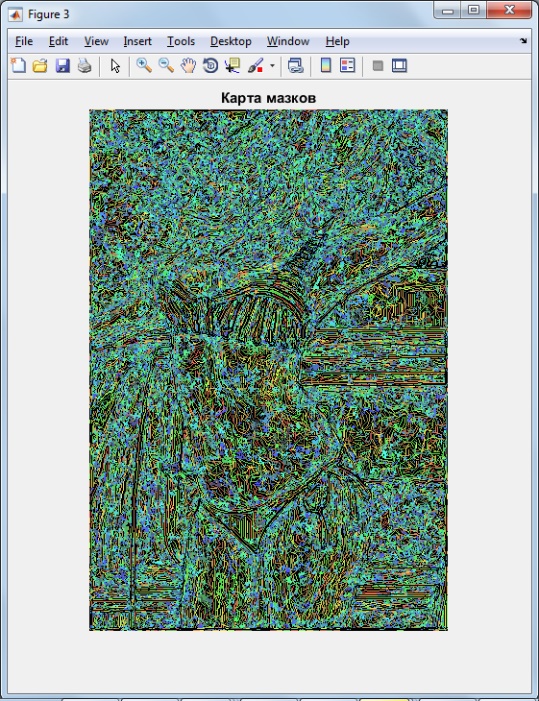


Рисунок 12 – Оптимизированная карта мазков

Если после этапа 5 остается много незакрашенных пикселей, можно повторить его несколько раз.

*Недостаток данного подхода*: получается много мелких мазков, их число значительно превосходит число мазков, полученных на этапе 4.

*Как можно*: разбить изображение, условно, на небольшие фрагменты (фреймы) — это могут быть квадраты или суперпиксели. Мазки сортируются по группам, принадлежащим фреймам (если мазок попадает сразу в несколько фреймов, его части рассматриваются независимо, точки перехода из фрейма в фрейм «замораживаются»). С помощью генетического или другого эвристического алгоритма проводится оптимизация:

а) на каждой итерации точки мазков немного передвигаются каким-либо образом (например, так, чтобы закрасить маленькие пробелы и увеличить большие). Возможно введение фактора случайности.

б) если пробелы оказываются рядом с концами мазков, то мазки удлиняются

в) если пробелы оказываются между мазками, то они закрашиваются новыми мазками.

Целевая функция тем меньше, чем меньше:

* средняя погрешность по фрейму
* количество новых мазков

6. Кластеризация цветов. Объединяем цвета в кластеры, усредняем цвета по кластерам. В итоге получаем картинку с заданным количеством цветов (например, 50).



Рисунок 13 – Результат

7. Сохранение PLT-файла, используется текстовый язык HP-GL/2.

Если без размытия, то с теми же настройками получится:

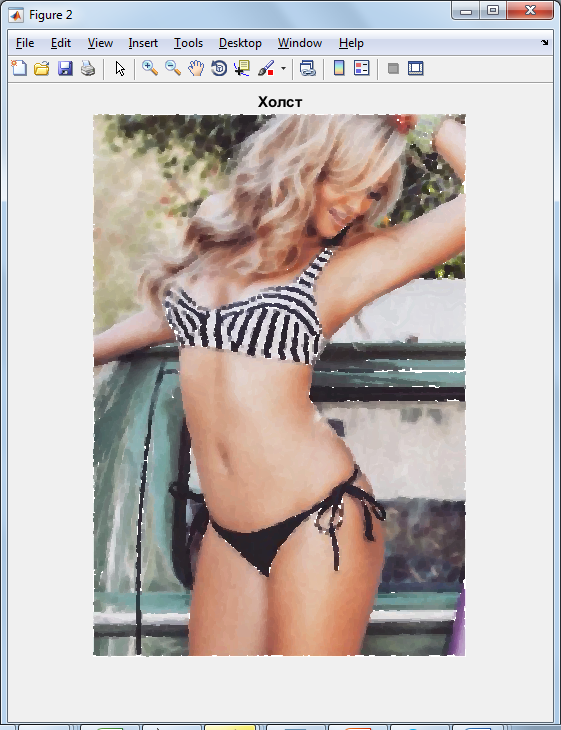


Рисунок 14 – результат без размытия

# Реализация алгоритма на платформе .Net

Вся основная работа велась в среде Visual Studio 2017. Для реализации пользовательского интерфейса были применены паттерны MVVM [4], для бизнес-логики применялись принципы SOLID [5]. В результате был получен интерфейс, показанный на рисунке 15.



Рисунок 15 – Пользовательский интерфейс

Пример окна настроек для градиентного метода показан на рисунке 16. При загрузке картинки параметры окна настраиваются автоматически.

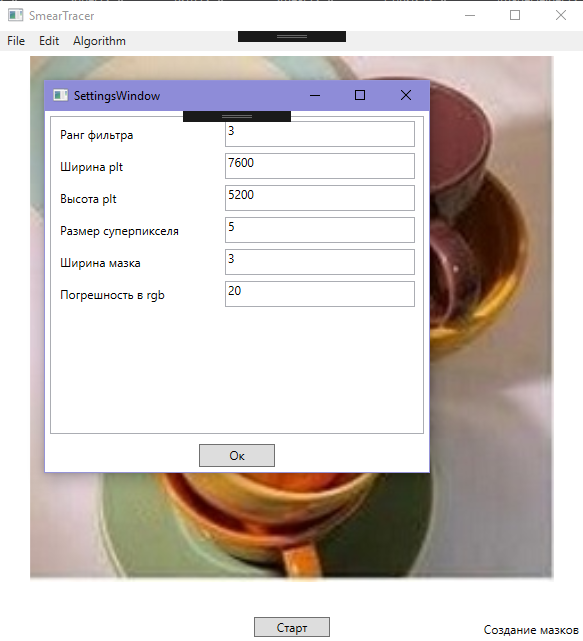


Рисунок 16 – Окно настроек

Также была добавлена возможность сохранять результаты в виде plt- файла непосредственно из меню главного окна, а также копировать результаты в виде изображения в буфер обмена, либо сохранять в файл.

Еще одной важной доработкой стало создание отзывчивого главного окна, для реализации этой функции все вычисления были перенесены в фоновый поток, в основном потоке выполнялась работа только пользовательского интерфейса. Благодаря этой доработке появилась возможность выводить статусы операций на главное окно приложения, что дает возможность отслеживать, на какой стадии выполняется работа.

Версия алгоритма на текущей стадии работы алгоритма GradientTracer выглядит следующим образом:

1. Загрузка изображения и настроек
2. Сглаживание изображения
3. Построение карты градиентов (используется оператор Собеля)
4. Разбиение изображения на суперпиксели
5. Создание мазков.
6. Вывод результатов в виде изображений.

Создание мазков реализовано следующим образом:

1. Разбиваем суперпиксели на пары в соответствии с направлениями градиентов их центров, а также с учетом погрешности, указанной в настройках.
2. Итерационно объединяем мазки с учетом их расположения на изображении, т.е. учитываем расстояние между ними.

# Тестирование

Для тестирования применялся набор изображений, используемых в предыдущих работах. Пример результата работы алгоритмов показан на рисунках 17-20.



Рисунок 17 – Результат работы алгоритма SmearTracer



Рисунок 18 – Результат работы алгоритма GradientTracer



Рисунок 19 – PLT файл алгоритма SmearTracer



Рисунок 20 – PLT файл алгоритма GradientTracer

Как видно из рисунков, разница в алгоритмах с точки зрения точности работы не существенна, из чего можно сделать вывод, что требуются дальнейшие доработки и улучшения.

# Заключение

Таким образом в ходе работы был доработан алгоритм SmearTracer, а также был реализован алгоритм GradientTracer. Был улучшен пользовательский интерфейс. Был проведен сравнительный анализ обоих алгоритмов, выявлены преимущества и недостатки каждого их них.

# Список литературы

1. Оператор Собеля // Википедия URL: https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80\_%D0%A1%D0%BE%D0%B1%D0%B5%D0%BB%D1%8F (дата обращения: 26.12.17). Уоллс К. Spring в действии. – Litres, 2017.
2. О градиенте изображения // Хабрахабр URL: https://habrahabr.ru/post/114452/ (дата обращения: 26.12.17).
3. MATLAB // mathworks URL: https://www.mathworks.com/ (дата обращения: 26.12.17).
4. Паттерн MVVM // metanit URL: https://metanit.com/sharp/wpf/22.1.php (дата обращения: 26.12.17).
5. SOLID (объектно-ориентированное программирование) // wikipedia URL: https://ru.wikipedia.org/wiki/SOLID\_(объектно-ориентированное\_программирование) (дата обращения: 26.12.17).