**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Систем Автоматизированного Проектирования**

отчет

**по преддипломной практике**

Тема: **Сравнительное исследование алгоритмов**

**разбиения изображения на мазки**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент гр. 2301 |  |  |  | Мезенцев Р.С. |
|  |  | *подпись* |  |  |
| Руководитель практики | к.т.н., доцент |  |  | Михалков В.А |
|  | *(Уч. степень, уч. звание)* | *подпись* |  |  |
| Согласовано  Руководитель ВКР | д.т.н., профессор |  |  | Сольницев Р.И. |
|  | *(Уч. степень, уч. звание)* | *подпись* |  |  |

Санкт-Петербург

2018

**ЗАДАНИЕ**

**на ПРЕДДИПЛОМНУЮ практику**

|  |  |  |
| --- | --- | --- |
| Студент Мезенцев Р.С. | | |
| Группа 2301 | | |
| Тема практики: Сравнительное исследование алгоритмов разбиения изображения на мазки | | |
| Задание на практику:   1. Проектирование архитектуры программных компонентов. 2. Программная реализация алгоритмов. 3. Тестирование. 4. Сравнение работы алгоритмов. 5. Оформление отчета по проделанной работе | | |
| Сроки прохождения практики: 12.02.2018 – 21.05.2018 | | |
| Дата сдачи отчета: 21.05.2018 | | |
| Дата защиты отчета: 21.05.2018 | | |
|  | | |
| Студент |  | Мезенцев Р.С. |
| Руководитель ВКР |  | Сольницев Р.И. |
| Руководитель практики |  | Михалков В.А. |

**ГРАФИК выполнения заданий**

**ПРЕДДИПЛОМНОЙ ПРАКТИКИ**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент Мезенцев Р.С. | | | | | |
| Группа 2301 | | | | | |
| Тема практики: Сравнительное исследование алгоритмов разбиения изображения на мазки | | | | | |
| № п/п | Наименование работ | | | Срок выполнения |
| 1 | Разработка задания на практику | | | 12.02 – 16.02 |
| 2 | Проектирование архитектуры программных компонентов | | | 17.02 – 28.02 |
| 3 | Программная реализация алгоритмов | | | 01.03 – 21.04 |
| 4 | Тестирование | | | 22.04 – 07.05 |
| 5 | Сравнение работы алгоритмов | | | 08.05 – 15.05 |
| 6 | Оформление отчета по преддипломной практике | | | 16.05 – 21.05 |
| Студент | |  | Мезенцев Р.С. | | |
| Руководитель практики | |  | Михалков В.А. | | |
| Согласовано  Руководитель ВКР | |  | Сольницев Р.И. | | |

**АННОТАЦИЯ**

Работа посвящена компонентам САПР полиграфической системы на основе аддитивной технологии (ПСОАТ).

Основой работы является решение одной из задач процесса автоматизации системы машинной живописи – разработка программного обеспечения для управления роботом-манипулятором с целью нанесения изображения на поверхность кистью. Для этого разрабатываются и тестируются два алгоритма, после чего выявляются преимущества и недостатки каждого.

**SUMMARY**

The work is dedicated to the CAD component-based additive technology printing system (AT PS).

The basis of the work is to solve one of the problems of automating the process of painting machine systems - development of software for the control of a robot manipulator with a view to drawing the image on the surface of the brush. For this, two algorithms are developed and tested, after which the advantages and disadvantages of each are determined.

ОГЛАВЛЕНИЕ

[Введение 7](#_Toc514620954)

[1. Проектирование архитектуры программных компонентов 9](#_Toc514620955)

[2. Программная реализация алгоритмов 11](#_Toc514620956)

[3. Тестирование 20](#_Toc514620957)

[4. Сравнение работы алгоритмов 24](#_Toc514620958)

[Заключение 27](#_Toc514620959)

[Список использованных источников 28](#_Toc514620960)

**Определения, обозначения и сокращения**

ПО – Программное обеспечение.

GUI (GraphicalUserInterface) – графический интерфейс пользователя.

VS (VisualStudio) – это интегрированная среда разработки.

WPF (Windows Presentation Foundation) – система для построения клиентских приложений Windows.

SOLID (single responsibility, open-closed, Liskov substitution, interface segregation и dependency inversion) – основные принципы ООП и проектирования, соблюдение которых повышают вероятность создания программистом системы, которую будет легко поддерживать и расширять в течении долгого времени.

GradientTracer – алгоритм разбиения изображения в карту мазков, в основе которого лежит нахождения градиентов на изображении.

SmearTracer – алгоритм разбиения изображения в карту мазков, который состоит из алгоритмов кластеризации, сегментации, а также других алгоритмов обработки изображений.

# Введение

Работа посвящена разработке программы разбиения изображения на колористические примитивы для полиграфической системы на основе аддитивной технологии (ПСОАТ). Основной проблемой является необходимость применения эвристического алгоритма для автоматизации технологии живописного процесса ПСОАТ.

Основной идеей при разработке программы служит реализация двух алгоритмов разбиения изображения на мазки. Первый подход подразумевает выполнение четырех ключевых этапов:

* Фильтрации;
* Кластеризации;
* Контурной сегментации;
* Аппроксимации и векторизации.

Второй подход состоит из четырех ключевых этапов:

* Фильтрации;
* Нахождения градиентов;
* Создания суперпикселей;
* Формирования мазков.

Объект исследования – полиграфическая система на основе аддитивной технологии.

Предмет исследования – преобразование изображения в карту колористических примитивов (мазков) для ПСОАТ.

Целью научно-исследовательской работы является исследование компонентов САПР полиграфической системы на основе аддитивной технологии.

Из цели вытекает ряд задач:

* Проектирование архитектуры программных компонентов;
* Программная реализация алгоритмов;
* Тестирование.
* Сравнение работы алгоритмов.
* оформление отчетных материалов по проекту, написание пояснительной записки.

# Проектирование архитектуры программных компонентов

При проектировании архитектуры была разработана высокоуровневая структура, показанная на рисунке 1.1.

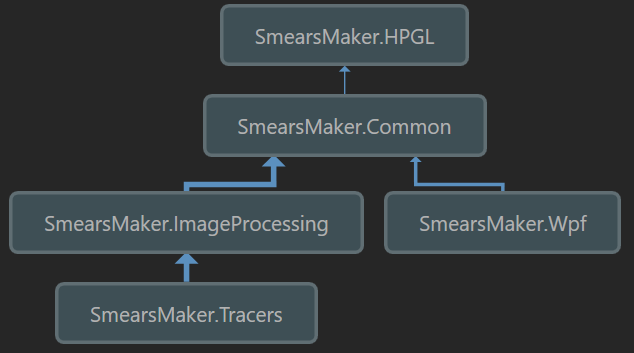


Рисунок 1.1 – Архитектура приложения

Она описывает основные компоненты системы, а также их взаимодействие друг с другом. При разработке архитектуры был проведен анализ потребностей пользователей, а также были учтены принципы SOLID [1]. Из рисунка видно, что система будет состоять из пяти компонентов, связанных между собой. Рассмотрим каждый из компонентов по отдельности.

SmearsMaker.Common - компонент, который содержит в себе интерфейсы приложения и базовые классы, которые формируют основную инфраструктуру будущего приложения.

SmearsMaker.HPGL - компонент, включающий в себя типы для работы с языком HPGL [2]. Эти типы позволяют читать из файлов информацию, а также записывать в них сформированную карту мазков.

SmearsMaker.ImageProcessing - компонент, включающий типы для работы с изображениями. Типы реализуют алгоритмы фильтрации, кластеризации, операторы поиска градиентов [3]. Кроме этого, компонент включает типы для сегментации и формирования карты мазков.

SmearsMaker.Tracers - компонент, который содержит в себе различные реализации алгоритмов формирования изображения в карту мазков.

SmearsMaker.Wpf - компонент, реализующий пользовательский интерфейс, содержит в себе типы, формирующие GUI, а также точку входа в приложение.

Таким образом, были сформированы различные компоненты системы, удовлетворяющие различным принципам проектирования и разработана архитектура будущего приложения.

# Программная реализация алгоритмов

Ключевым компонентом системы является компонент SmearsMaker.Common.dll. Он включает в себя следующие типы:

ITracer – интерфейс, определяющий функционал работы алгоритма создания карты мазков;

TracerBase – абстрактный класс, определяющий базовое поведение алгоритма создания карты мазков, является общим классом для всех алгоритмов трассировки;

ImageView ­– класс, который включает в себя результат работы алгоритма: изображение и имя операции;

ImageSetting – класс, который описывает настройку для алгоритма трассировки, включает имя настройки и значение;

IProgress – интерфейс, описывающий функционал, определяющий этап выполнения алгоритма. При использовании класса TracerBase используется внутренняя реализация ProgressImpl.

Схема, описывающая зависимости перечисленных типов, показана на рисунке 2.1.

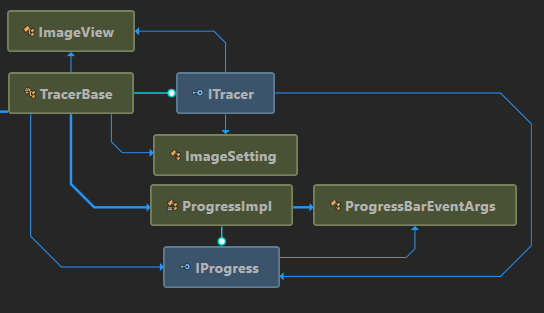


Рисунок 2.1 – Типы, входящие в компонент SmearsMaker.Common

Также в этот компонент входят следующие типы:

ImageModel – класс, определяющий модель для работы алгоритма трассировки. Включает с себя основные характеристики загруженного изображения: высоту, ширину, коллекцию пикселей, а также ссылку на загруженное изображение;

PointCollection – коллекция точек Point, позволяющая добавлять, удалять, копировать точки, а также содержит методы для добавления слоев для новых точек. Данные хранятся во внутренней коллекции ObservableCollection, позволяющей следить за любыми изменениями над данными;

ObservableCollectionExtention – статический класс, расширяющий поведение класса ObservableCollection. Позволяет выполнять обход коллекции в цикле Foreach [4];

Point – класс, задающий поведение точки, содержит позицию пикселя на изображении, а также коллекцию пикселей, которые входят в состав всех слоев, используемых в модели. Также содержит типы добавления новых слоев, однако эти методы недоступны для вызова из типа Point: добавление новых слоев происходит через коллекцию PointCollection;

PixelCollection – коллекция, внутри которой используется словарь, содержащий пиксели, ключом является имя слоя, которому принадлежит пиксель;

Pixel – класс, который содержит данные пикселя на изображении.

BaseShape – абстрактный класс, описывающий базовое поведение любого объекта на изображении.

Зависимость между перечисленными типами показана на рисунке 2.2.

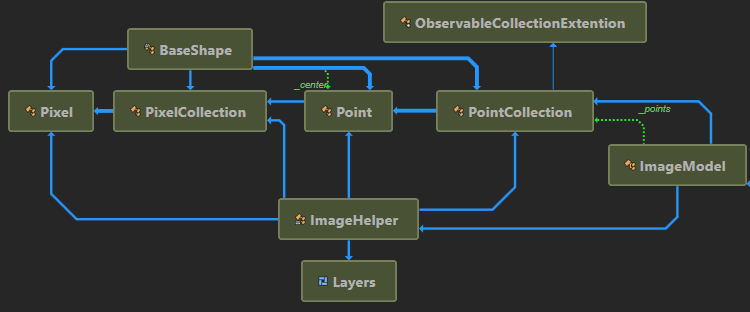


Рисунок 2.2 – Типы, входящие в компонент SmearsMaker.Common

Следующая сборка, которую мы рассмотрим, называется SmearsMaker.HPGL. Она предназначена для работы с языком HPGL и содержит следующие типы:

IPltReader – интерфейс, задающий функционал для чтения Plt файла;

IPltReaderImpl – реализация интерфейса IPltReader;

IPltCreator – интерфейс, задающий функционал для создания Plt файла;

Painter – внутренный вспомогательный класс, задачей которого явлется формирование изображения из геометрических фигур;

PltHelper – вспомогательный статический класс, содержит методы для работы с изображениями.

Зависимости между типами продемонстрированы на рисунке 2.3.

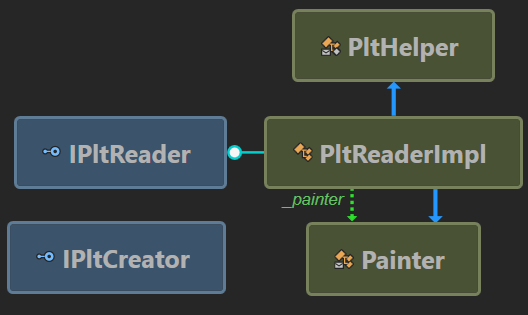


Рисунок 2.3 – Типы, входящие в компонент SmearsMaker.Common

Сборка SmearsMaker.ImageProcessing содержит реализации различных алгоритмов обработки изображений. Она содержит следующие типы:

IFilter – интерфейс, задающий функциональность фильтра;

MedianFilter – реализация интерфейса IFilter, описывает поведения медианного фильтра;

IDetector – ­­­­ интерфейс, который задает функциональность оператора;

Sobel­ ­– реализация интерфейса IDetector, описывает поведения оператора Собеля;

IClusterizer – интерфейс, задающий функциональность алгоритма кластеризации;

Kmeans ­– абстрактный класс, реализующий интерфейс IClusterizer, описывает базовое поведение алгоритма Kmeans, позволяет реализовывать множество алгоритмов, основанных на KMeans;

KMeansClassic – класс, является реализацией алгоритма Kmeans;

Cluster – класс, описывающий поведения кластера;

ISplitter – интерфейс, задающий функциональность алгоритма разбиения любой фигуры на сегменты;

IStrokesBuilder – интерфейс, задающий функциональность алгоритма формирования мазков;

BrushStroke – класс, описывающий поведения мазка.

Перечисленные типы показаны на рисунке 2.4.

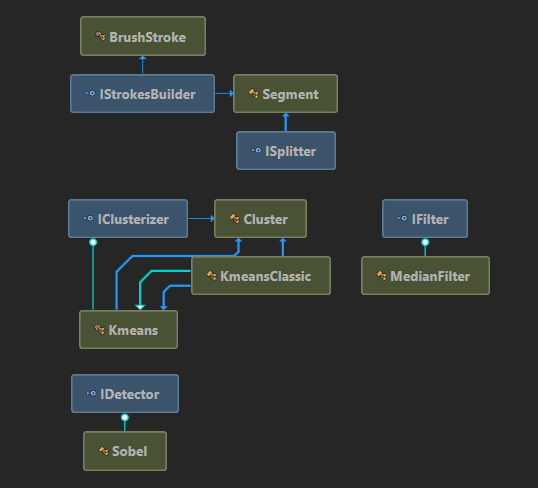


Рисунок 2.4 – Типы, входящие в компонент SmearsMaker.ImageProcessing

Сборка SmearsMaker.Tracers является самой большой, поэтому рассмотрим её по частям. Сборка содержит множество типов, которые формируют логику работы различных вариантов разбиения изображения на мазки. Типы, которые являются вспомогательными, и используются всеми реализациями алгоритмов формирования карты мазков:

Utils – статический вспомогательный класс, который содержит методы для вычисления различных расстояний, а также включает методы, генерирующие различные данные;

BrushStrokeImpl – реализация мазков;

ImageHelper – статический вспомогательный класс, который содержит методы для работы с изображениями;

PltHelper – статический вспомогательный класс, который содержит методы, помогающие сформировать plt из различных типов;

BaseShapeExtentions – статический класс, расширяющий функциональность класса BaseShape;

BrushStrokeExtension – статический класс, расширяющий функциональность класса BrushStroke;

IServicesFactory­ – интерфейс, который определяет, какие алгоритмы будут использоваться;

Типы, которые необходимы для работы алгоритма SmearTracer:

Smear – класс, который содержит ссылки на мазок, сегмент и кластер;

STracer – класс, реализующий алгоритм SmearTracer;

SmearFactory – реализация интерфейса IServicesFactory, которая создает необходимые типы для выполнения алгоритма;

StlImageSettings – класс, который описывает, какие настройки могут использоваться в алгоритме SmearTracer.

Типы, которые необходимы для работы алгоритма GradientTracer:

GTracer – абстрактный класс, описывающий алгоритм GradientTracer;

HexagonGTracer – вариант алгоритма GTracer, с использованием шестиугольных суперпикселей;

HexagonTracerFactory – реализация интерфейса IServicesFactory, которая создает необходимые типы для выполнения алгоритма;

SquadGTracer – вариант алгоритма GTracer, с использованием квадратных суперпикселей;

SquadTracerFactory – реализация интерфейса IServicesFactory, которая создает необходимые типы для выполнения алгоритма;

GtImageSettings – класс, который описывает, какие настройки могут использоваться в алгоритме GradientTracer.

Зависимости меду перечисленными типами показаны на рисунках 2.5-2.7;

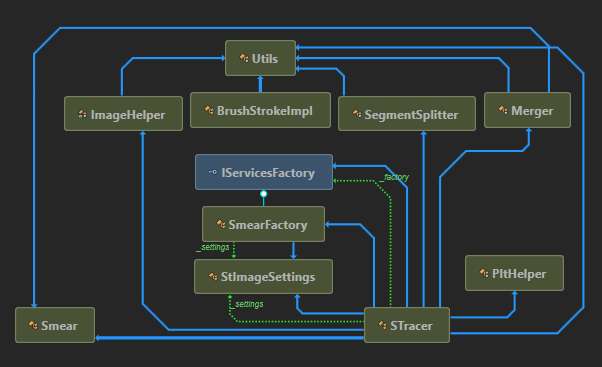


Рисунок 2.5 – Типы, входящие в компонент SmearsMaker.Tracers

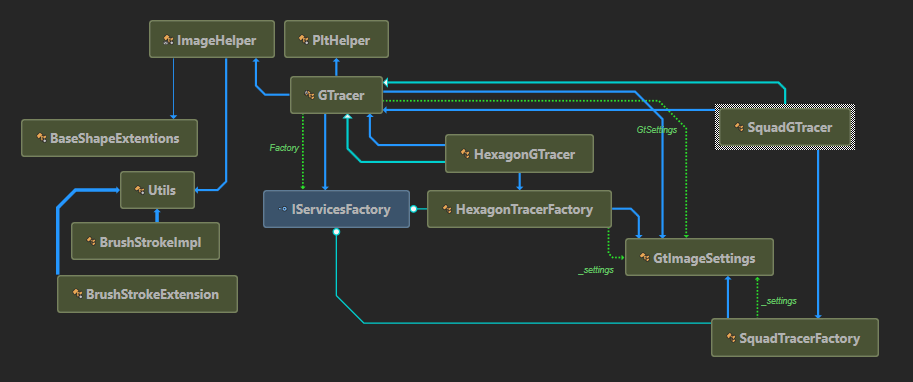


Рисунок 2.6 – Типы, входящие в компонент SmearsMaker.Tracers

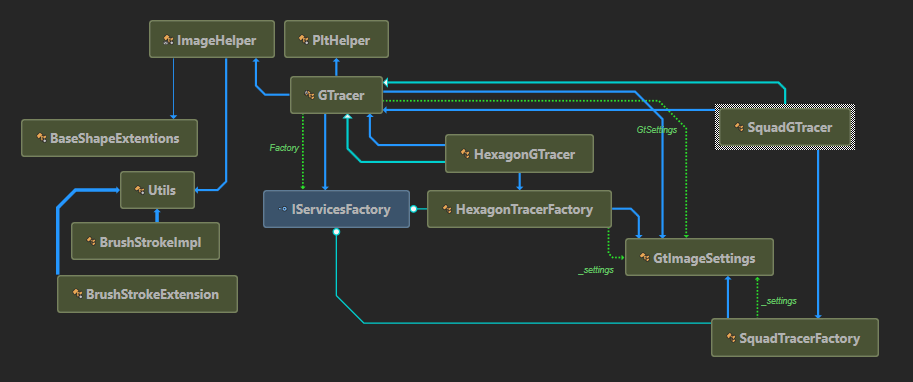


Рисунок 2.7 – Типы, входящие в компонент SmearsMaker.Tracers

Также в сборке находятся типы, содержащие логику формирования мазков и разбиения изображения или его части на суперпиксели:

GradientStrokesBuilder – класс, реализующий интерфейс IStrokesBuilder. Содержит логику формирования мазков из частей изображения с использованием градиента;

SimpleStrokesBuilder – класс, реализующий интерфейс IStrokesBuilder. Содержит логику формирования мазков из частей изображения без учета градиента;

PairStrokesBuilder – класс, реализующий интерфейс IStrokesBuilder. Содержит альтернативную логику формирования мазков из частей изображения без учета градиента;

SuperPixelSplitter – класс, реализующий интерфейс ISplitter. Содержит логику разбиения изображения или его части на квадратные суперпиксели;

HexagonSplitter – класс, реализующий интерфейс ISplitter. Содержит логику разбиения изображения или его части на шестиугольные суперпиксели.

Перечисленные типы не имеют зависимостей (см. рисунок 2.8).

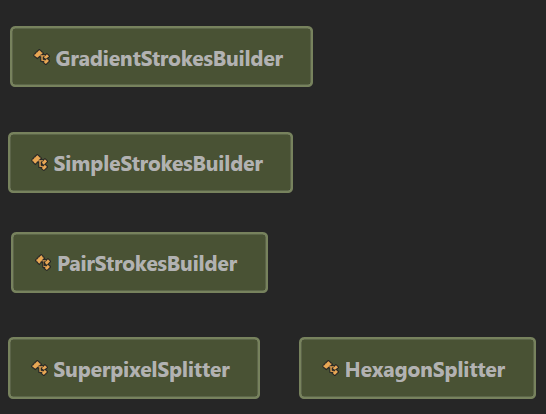


Рисунок 2.8 – Типы, входящие в компонент SmearsMaker.Tracers

Последней сборкой в приложении является SmearsMaker.Wpf, которая отвечает за пользовательский интерфейс. Типы, находящиеся в сборке:

Settings, Resources, App – типы, которые формируются при создании приложения WPF;

App.сonfig – конфигурационный файл, который содержит в себе описание сборок с различными реализациями алгоритмов разбиения изображения. ПО умолчанию содержит ссылку на компонент SmearsMaker.Tracers;

MainWindow – главное окно приложения, содержит меню с кнопками, окно вывода изображения, а также кнопку запуска алгоритма;

SettingsWindow – окно приложения, в котором выводятся настройки выбранного алгоритма;

ApplicationViewModel – класс, описывающий логику работы пользовательского приложения, представляет собой прослойку между разметкой окна и бизнес логикой приложения;

Command – класс, реализующий интерфейс ICommand, который позволяет выполнять связывание событий в WPF [5];

FileHelper – вспомогательный статический класс, содержащий методы для работы с файлами.

Зависимости меду перечисленными типами показаны на рисунке 2.9.

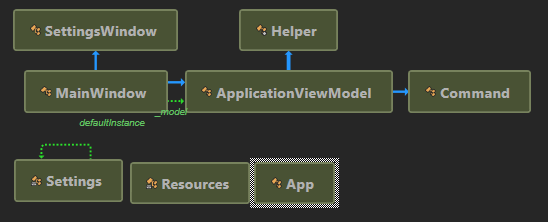


Рисунок 2.9 – Типы, входящие в компонент SmearsMaker.Wpf

В данном разделе была описана программная реализация двух алгоритмов формирования карты мазков: SmearTracer и GradientTracer в двух реализациях: с использованием квадратных и шестиугольных суперикселей. Также была описана реализация клиентского пользовательского интерфейса с использованием технологии WPF.

# Тестирование

Для тестирования будем использовать два исходных изображения: фотографию “Лена” размером 512 на 512 пикселей (см. рис. 6) и фотографию “Лимоны” размером 275 на 183 пикселя (см. рис. 3.1).



Рисунок 3.1 – Исходное изображение “Лена”



Рисунок 3.2 – Исходное изображение ”Лимоны”

Сначала запустим алгоритм SmearTracer, настройки для алгоритма показаны на рисунках 3.3-3.4.

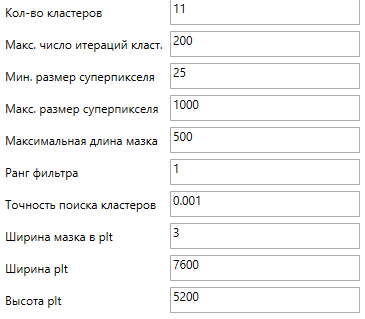


Рисунок 3.3 – Окно настроек для изображения ”Лимоны”

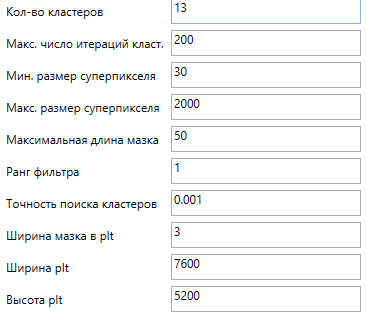


Рисунок 3.4 – Окно настроек для изображения ”Лена”

Полученные карты мазков показаны на рисунках 3.5-3.6.



Рисунок 3.5 – результат алгоритма SmearTracer для изображения ”Лимоны”

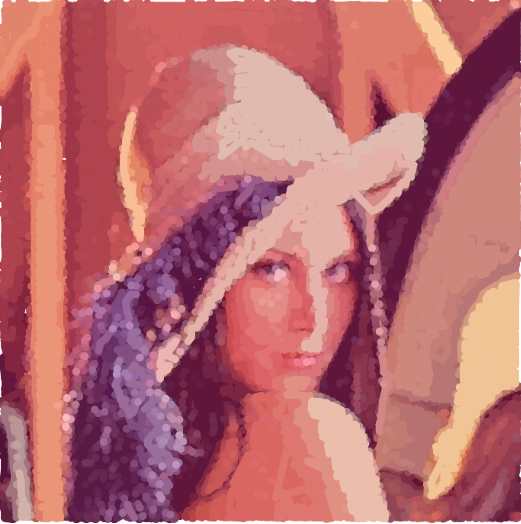


Рисунок 3.6 – результат алгоритма SmearTracer для изображения ”Лена”

Запустим алгоритм GradientTracer, настройки для алгоритма показаны на рисунках 3.7-3.8.

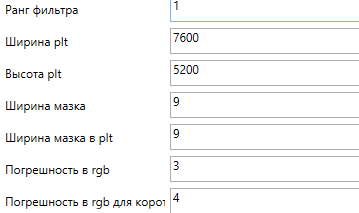


Рисунок 3.7 – Окно настроек для изображения ”Лимоны”

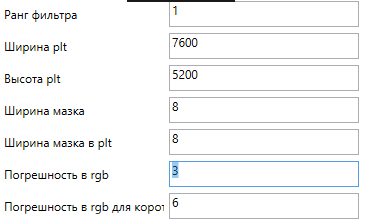


Рисунок 3.8 – Окно настроек для изображения ”Лена”

Полученные карты мазков показаны на рисунках 3.9-3.10.



Рисунок 3.9 – результат алгоритма GradientTracer для изображения ”Лимоны”



Рисунок 3.10 – результат алгоритма GradientTracer для изображения ”Лена”

Таким образом, было проведено тестирование разработанных алгоритмов на паре тестовых изображений.

# Сравнение работы алгоритмов

В ходе работы алгоритма SmearTracer на изображении ”Лимоны” было сформировано 2187 мазков за 17 секунд, а на изображении ”Лена” 3392 мазка за 24 секунды. На рисунках 3.11 и 3.12 показана разница между исходным изображением и изображением, разбитым на мазки.

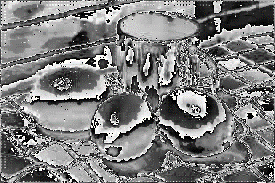


Рисунок 3.11 – Разница изображений ”Лимоны”



Рисунок 3.12 – Разница изображений ”Лена”

В ходе работы алгоритма GradientTracer на изображении ”Лимоны” было сформировано 2145 мазков за 3 секунды, а на изображении ”Лена” 3352 мазка за 6 секунд. На рисунках 3.13 и 3.14 показана разница между исходным изображением и изображением, разбитым на мазки.

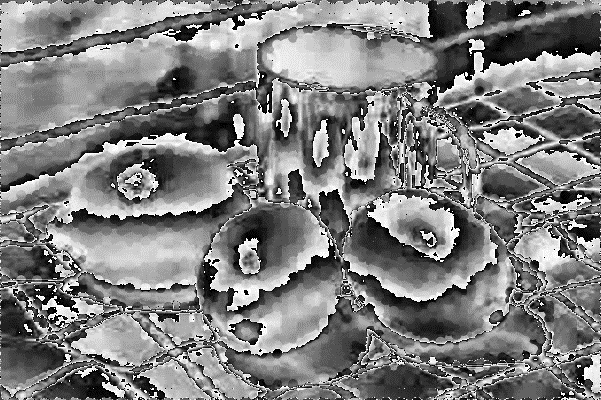


Рисунок 3.13 – разница изображений ”Лимоны”

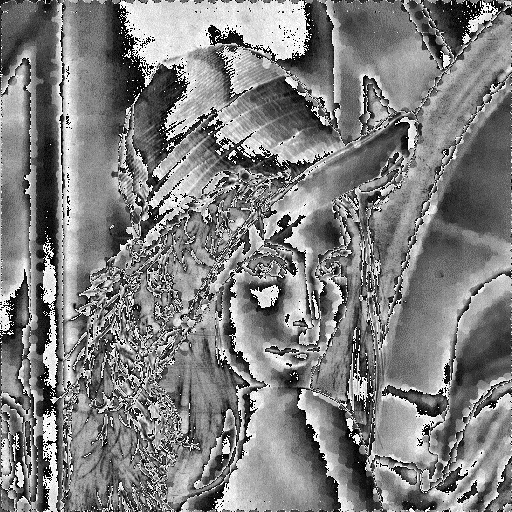


Рисунок 3.14 – разница изображений ”Лена”

Исходя из полученных результатов, можно сделать вывод, что алгоритм GradientTracer быстрее алгоритма SmearTracer в 4 раза. Кроме того, разница между исходным изображением и полученной картой мазков при использовании алгоритма SmearTracer имеет более прерывистую структуру, чем разница изображений, полученная алгоритмом GradientTracer, из чего следует вывод, что алгоритм GradientTracer формирует более точную карту мазков, чем SmearTracer.

# Заключение

В ходе выполнения преддипломной практики было реализовано и протестировано полноценное программное решение, содержащее два алгоритма разбиения изображения на колористические примитивы, с учетом конструктивных особенностей ПСОАТ. Помимо этого, был проведен сравнительный анализ работы алгоритмов, в результате чего были выявлены преимущества и недостатки алгоритмов.

# Список использованных источников

1. SOLID // Хабр URL: https://habr.com/post/348286/ (дата обращения: 10.04.18).
2. HP-GL // Wikipedia URL: https://en.wikipedia.org/wiki/HP-GL (дата обращения: 10.04.18).
3. Методы нахождения границ изображения // Хабр URL: https://habr.com/post/128753/ (дата обращения: 10.04.18).
4. How to make a Visual C# class usable in a foreach statement // Microsoft Support URL: https://support.microsoft.com/en-us/help/322022/how-to-make-a-visual-c-class-usable-in-a-foreach-statement (дата обращения: 10.04.18).
5. ICommand Interface // msdn URL: https://msdn.microsoft.com/en-us/library/system.windows.input.icommand(v=vs.110).aspx (дата обращения: 10.04.18).