**«Санкт-Петербургский государственный электротехнический университет**

**«ЛЭТИ» им. В.И.Ульянова (Ленина)»**

**(СПбГЭТУ “ЛЭТИ”)**

|  |  |  |
| --- | --- | --- |
| **Направление** | 09.03.01 - Информатика и вычислительная техника | |
| **Программа** | Компьютерные технологии инжиниринга | |
| **Факультет** | Компьютерных технологий и информатики | |
| **Кафедра** | Систем автоматизированного проектирования | |
| *К защите допустить* |  | |
| Зав. кафедрой |  | Рыжов Н.Г. |

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**МАГИСТРА**

**Тема: КОМПОНЕНТЫ САПР полиграфической системы аддитивной печати**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент |  |  |  | Мезенцев Р.С. |
|  |  | *подпись* |  |  |
| Руководитель | ассистент |  |  | Каримов А.И. |
|  | *(Уч. степень, уч. звание)* | *подпись* |  |  |
| Консультанты | к.т.н., доцент |  |  | Иванов А.Н. |
|  | *(Уч. степень, уч. звание)* | *подпись* |  |  |

Санкт-Петербург

2018

**ЗАДАНИЕ**

**на выпускную квалификационную работу**

|  |  |
| --- | --- |
|  | Утверждаю |
|  | Зав. кафедрой САПР |
|  | Рыжов Н.Г. |
|  | «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_ г. |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Студент | Мезенцев Р.С. | | | |  | Группа | 2301 | |
| Тема работы: | | | | | | | | |
| Место выполнения ВКР: кафедра САПР, СПбГЭТУ | | | | | | | | |
| Исходные данные (технические требования): | | | | | | | | |
| Перечень отчетных материалов: пояснительная записка, иллюстративный материал | | | | | | | | |
| Дополнительные разделы: | | | | | | | | |
|  | | | | | | | | |
| Дата выдачи задания | | | Дата представления ВКР к защите | | | | | |
| «01»\_ 2018 г. | | | «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_ г. | | | | | |
|  | | |  | | | | | |
| Студент | |  | | Мезенцев Р.С. | | | |
| Руководитель | |  | | Каримов А.И. | | | |
|  | |  | |  | | | |

**календарный план выполнения**

**выпускной квалификационной работы**

|  |  |
| --- | --- |
|  | Утверждаю |
|  | Зав. кафедрой САПР |
|  | Рыжов Н.Г. |
|  | «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_ г. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | Мезенцев Р.С. |  | Группа | 2301 |
| Тема работы: КОМПОНЕНТЫ САПР ПОЛИГРАФИЧЕСКОЙ СИСТЕМЫ АДДИТИВНОЙ ПЕЧАТИ | | | | |

|  |  |  |
| --- | --- | --- |
| № п/п | Наименование работ | Срок выполнения |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |
| 6 |  |  |

|  |  |  |
| --- | --- | --- |
| Студент |  | Мезенцев Р.С. |
| Руководитель |  | КаримовА.И. |

**РЕФЕРАТ**

**АННОТАЦИЯ**

**ABSTRACT**

СОДЕРЖАНИЕ

[Определения, обозначения и сокращения 7](#_Toc511683936)

[Введение 7](#_Toc511683937)

[1. Обзор существующих технологий и решений 7](#_Toc511683938)

[1.1. Обзор решения 1 7](#_Toc511683939)

[1.2. Обзор решения 2 7](#_Toc511683940)

[1.3. Обзор решения 3 7](#_Toc511683941)

[Выводы 7](#_Toc511683942)

[2. Теоретическое описание алгоритмов 7](#_Toc511683943)

[2.4. Описание алгоритмов 1 7](#_Toc511683944)

[2.5. Описание алгоритмов 2 7](#_Toc511683945)

[2.6. Описание алгоритмов 3 7](#_Toc511683946)

[Выводы 7](#_Toc511683947)

[3. Реализация программного решения 7](#_Toc511683948)

[3.1. Выбор инструментов разработки 7](#_Toc511683949)

[3.2. Проектирование архитектуры 7](#_Toc511683950)

[3.3. Разработка 7](#_Toc511683951)

[3.4. Тестирование 7](#_Toc511683952)

[3.5. Отладка 7](#_Toc511683953)

[4. Составление бизнес-плана по коммерциализации результатов нир магистранта 7](#_Toc511683954)

[4.1. Концепция экономического обоснования разработки технического продукта 7](#_Toc511683955)

[4.2. Расчёт полных затрат при разработке программного модуля 7](#_Toc511683956)

[Заключение 7](#_Toc511683957)

[Список литературы 7](#_Toc511683958)

[ПРИЛОЖЕНИЕ А 8](#_Toc511683959)

* **Определения, обозначения и сокращения**
* **Введение**

Если говорить о живописи как об искусстве, то введение машины в процесс творчества может рассматриваться как нежелательное действие, снижающее художественную ценность произведений. Однако для целей реставрации, копирования, создания монументальных и интерьерных росписей, оформления предметов быта и др., ручной труд может быть успешно автоматизирован с использованием роботов-манипуляторов и специальных станков для смешения красок. Это повлечет за собой снижение стоимости этих работ, позволит использовать их более широко и, в итоге, будет способствовать повышению эстетической привлекательности современной урбанистической среды.

Система может использоваться при изготовлении картин и их репродукций, при декоративном оформлении технических объектов и интерьеров, в качестве аттрактивного экспоната выставок и развлекательных мероприятий, для научно-исследовательских задач в области искусственного интеллекта. Особенно ценной она может оказаться при реставрации картин, позволяя свести к минимуму искажения, вносимые специалистом. Кроме этого, машинная живопись могла бы заменить другие виды полиграфии там, где не требуется высокое разрешение, но необходим толстый, прочный, устойчивый к выцветанию красочный слой (баннеры, фасадные росписи и др.).

* **Обзор существующих технологий и решений**
* **Обзор решения eDavid**

Робот eDavid создан командой разработчиков из Констацского университета (Германия). Изначально робот разрабатывался для сварки деталей автомобилей, но в конечном итоге нашел своё применение в живописи. Робот снабжен камерой, сенсорами и набором кистей (см. рисунок 1)

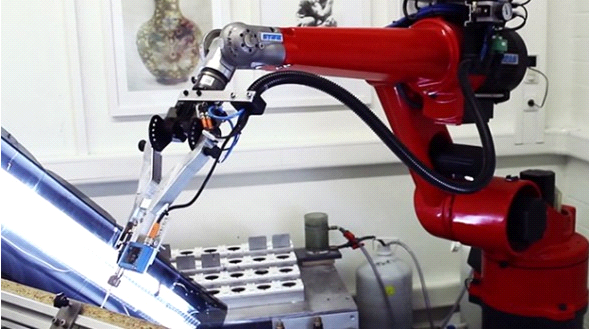


Рисунок 1 – Робот eDavid

Робот способен рисовать достаточно сложные композиционные картины, применяя при этом различные художественные направления. У робота 24-цветная палитра, робот работает только с акриловыми красками. Для создания картин робот использует набор из пяти кистей. Процесс рисования начинается с фотографирования оригинала. После этого робот анализирует фото, холст, размеры, выбирает подходящий цвет. На следующем этапе робот начинает наносить мазки, при этом применяется механизм обратной связи – имитация действия человека-художника. Каждые несколько минут робот делает фотографию своей работы и сравнивает её с оригиналом, после чего продолжает работу, либо корректирует последний шаг. Также во время оценки робот анализирует толщину, положение и цвет мазков, в результате чего выбирает кисть и соответствующий цвет из имеющейся палитры. Пример работы робота показан на рисунке 2.

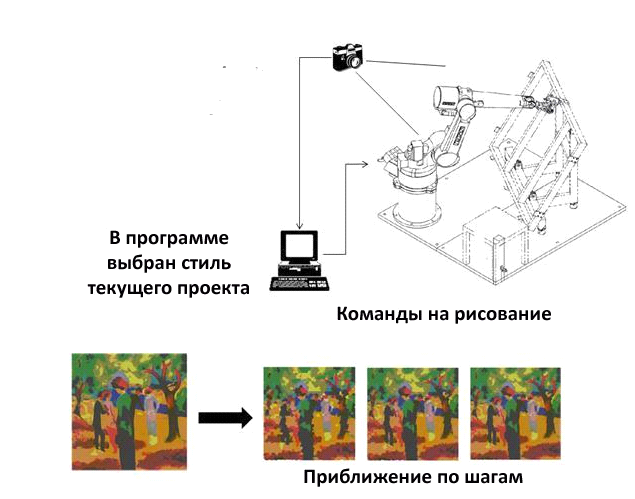


Рисунок 2 – Пример работы робота eDavid

Закончив работу, робот подписывает картину своим именем в нижнем правом углу. Примеры полученных результатов показаны на рисунке 3.



Рисунок 3 – Результаты работы робота eDavid

Преимуществами данного робота являются наличие механизма обратной связи, поддержка множества техник рисования, возможность работы с нескольким числом кисточек. Недостатками являются работа только с акриловыми красками, ограниченная палитра цветов, длительность рисования (до 10 часов).

* **Обзор решения bitPaintr**

Автором этого робота является голландский художник Пиндар Ван Армар. Изначально у робота должны были быть ассистенты, но в текущей пятой версии, построенной спустя 10 лет разработки, робот рисует картины самостоятельно (см. рисунок 4).



Рисунок 4 – Внешний вид робота bitPaintr

Ключевой особенностью этого проекта является применение специализированного искусственного интеллекта для создания портретов. Особенности используемых алгоритмов не раскрываются, но известно, что имеются настройки сложности и выбора стиля, которых всего 5 штук. Для начала рисования необходимо загрузить изображение и указать необходимые настройки. Процесс рисования занимает от 5 минут до 24 часов, при этом робот имеет возможность рисовать акриловыми и масляными красками (см рисунок 5).



Рисунок 5 – Результат работы робота bitPaintr

Максимальный размер картины составляет 30 на 42 дюйма. Проект изначально имел коммерческое направление применения, для чего автор разместил объявление на Kickstarter. В результате сбора средств была собрана сумма в 5665 долларов, целью было собрать 4800 долларов. Преимуществами этого робота является использование систем искусственного интеллекта, детализация полученных рисунков. Недостатками является время рисования картин, а также небольшое число стилей.

* **Обзор решения AARON**

Робот был создан в 1972 году Гарольдом Коэном – международно-известным художником, профессором Калифорнийского университета в Сан-Диего. Первые варианты робота рисовали черным и белым цветом, в последующих решениях применялся крупномасштабный струйный принтер на холсте.



Рисунок 6 – Внешний вид робота AARON

Разработка ПО изначально велась на C, но в начале 1990-ых годов ПО разрабатывалось на LISP. Стили рисования изображений закодированы, робот не имеет возможности самостоятельно изучать их. Программное обеспечение имеет циклы обратной связи, которые позволяют принимать решение относительно дальнейших действий. На самом верхнем уровне робот принимает решение, которое имеет отношение к композиции и организации изображения. На самом низком уровне он вычисляет инструкции для создания форм и линий, которые создают рисунок.



Рисунок 6 – Результат работы робота AARON

Автор робота утверждает, что творческая часть AARON является программным обеспечением независимо от того, какое устройство нанесения краски он присоединяет к нему, и при этом избегает называть AARON роботом.

Преимуществами робота является уникальный подход к рисованию – рисунки робота уникальны, никогда не повторяются за счет алгоритма с миллионами комбинаций. Недостатками являются отсутствие стилей для рисования, невозможность выбора исходного изображения для репродукции, использование принтера для рисования картины, а также сильная привязка к алгоритмам разработки.

* **Выводы**

Проведя анализ разработанных ранее решений, было установлено:

* Практически все проекты скрывают свою внутреннюю логику работы программных компонентов;
* Проекты сильно отличаются друг от друга как по возможностям, так и по характеристикам;
* Ни один и рассмотренных проектов не способен работать полностью в автоматическом режиме.
* **Теоретическое описание алгоритмов**
* **Описание алгоритмов фильтрации**

Главной задачей алгоритмов фильтрации является улучшение качества изображения. В области обработки изображений чаще всего встречаются гауссов и импульсный шумы, а также их комбинации (см. рисунок 6).

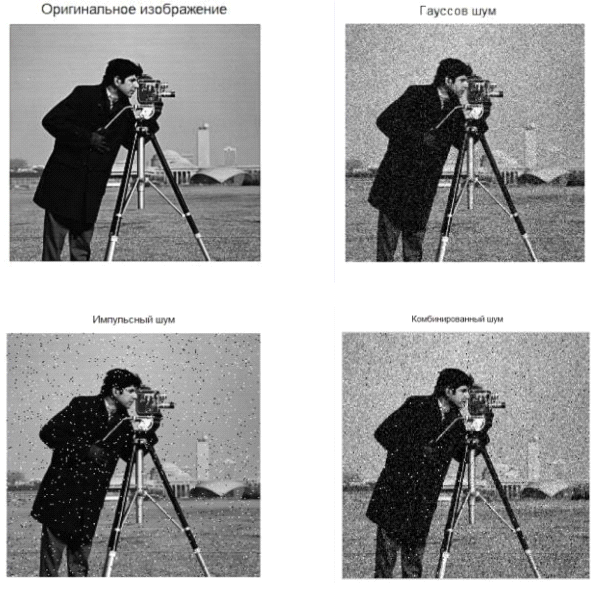


Рисунок 1.1 – Виды шума

На текущий момент не существует универсальных фильтров, которые подавляют все виды шумов. Самые распространённые методы удаления шумов:

* Сглаживающие фильтры;
* Медианные фильтры;
* Ранжирующие фильтры;

Кроме этого, фильтры классифицируются на линейные и нелинейные. Для подавления гауссова шума могут использоваться оба типа. Линейный фильтр состоит из ядра фильтра, который может быть задан на растровом изображении. Фильтрация производится операцией дискретной свертки (взвешенного суммирования). К сожалению, применение линейных фильтров не дает возможность получить приемлемый вариант решения в широком спектре задач. Нужно принимать во внимание характер процессов кодирования, который как правило, является нелинейным, также, как и процессы передачи и восприятия информации, например, датчиков информации, зрительной системы человека и т.п.

Определение единой теории нелинейной фильтрации невозможно, в отличие от теории линейных методов фильтрации. Любой из перечисленных классов имеет свои преимущества и свою область применения. К примеру, известно, что лучшие результаты для сохранения перепадов оттенков, локальных пиков яркости, и других границ на изображениях может дать применение медианного фильтра.

Медианный фильтр использует нелинейный способ подавления шума. Основу медианного фильтра составляет апертура фильтра, которая состоит из нечетного числа отсчетов. Принцип работы заключается в следующем:

* Определяется медианное значение элементов последовательности, т.е. находится средний по порядку член ряда, который образуется путем упорядочивания последовательности;
* Значения, попадающие в маску, заменяются полученным медианным значением.

Медианный фильтр позволяет решать и задачи импульсного шумоподавления, и задачи подавления аддитивного шума, однако в случае подавления импульсного шума он является наиболее эффективным (см. рисунок 6).



Рисунок 6 – Результат работы медианного фильтра

Также его очень важной особенностью является сохранение перепадов яркости (см. рисунок 6).

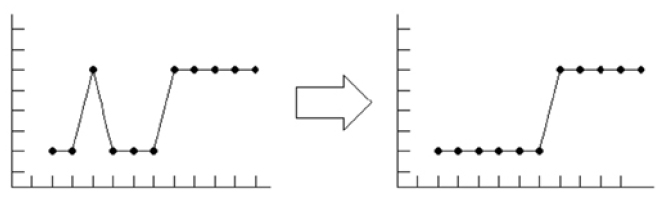


Рисунок 6 – Сохранение перепадов

Ранжирующий фильтр, также, как и медианный, использует маску, причем маска не включает центральный пиксель:

TODO: сдел. формулой

Отображение элементов показано на рисунке 6.

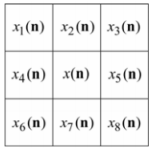


Рисунок 6 – Элементы маски

Последовательность w(n) можно упорядочить в последовательность:

 (2.15) TODO: сдел. формулой

Где порядок элементов можно определить как:

(2.15) TODO: сдел. формулой

А ранжирующее среднее можно определить как

TODO: сдел. формулой

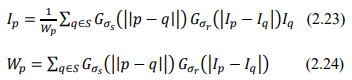
Исходя из всего этого, можно заметить, что значение ранжирующего среднего похоже на определение медианы для маски 3\*3, однако в последовательность w(n) не включено значение центрального пикселя w(n). Соответственно, размер маски также может быть произвольным, как и в медианном фильтре. Пример работы ранжирующего фильтра показан на рисунке 6.



Рисунок 6 – Результат работы ранжирующего фильтра

а – исходное изображение; b – результат работы фильтра.

Для удаления шумов с изображения с сохранением четких границ можно применить билатеральный фильтр. Сохранению границ способствует измерение веса в соответствии со значениями расстояний на плоскости исходного изображения, а также на его оси. Используя функцию Гаусса Go в виде зависимости и вычисляя исходное изображение I черно-белым, можно определить результат фильтра Ip для всех пикселей p:

TODO: сдел. формулой

Где S - множество всех позиций пикселей на изображении;

R - количество цветов;

 - позиция центрального пикселя;

Wp - сумма весов для нормализации значений;

q - позиция соседнего пикселя;

Go, Gr - функции Гаусса с параметрами os и or;

 - интенсивности пикселей p и q в диапазоне [0,1];

os означает число соседних пикселей, которые учитываются при обработке одного пикселя в области S, а or описывает потерю в весе соседних пикселей с разными интенсивностями.

На рисунке 6 показан пример работы билатерального фильтра.



Рисунок 6 – Результат работы билатерального фильтра

* **Описание алгоритмов вычисления векторов градиентов**

Градиент изображения используется для реализации автоматического измерения координат точек снимков, выделение границ объектов на изображении, а также отделения фона изображения от самих объектов. Самыми известными и наиболее используемыми являются операторы Собеля, Робертса и Превитта. Эти алгоритмы базируются на преобразовании изображения с помощью скользящей маски – определенной группе пикселей, которые представляют собой квадратную матрицу (см. рисунок 6).

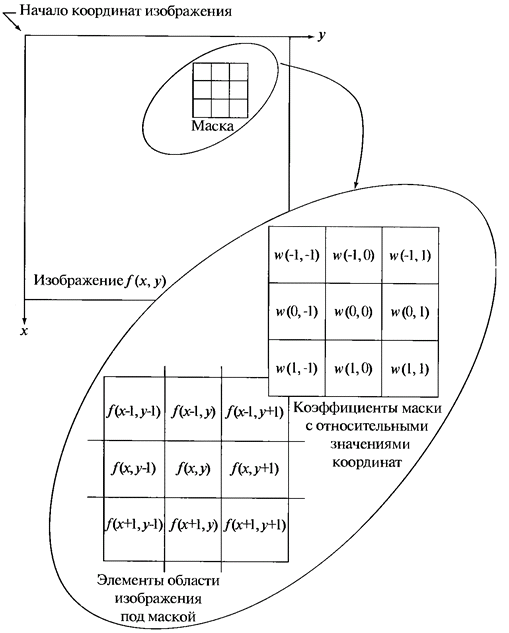


Рисунок 6 **–** Схема пространственной фильтрации

Алгоритм основан на обычном перемещении маски фильтра от одной точки к другой точке изображения; в каждой точке (x,y) отклик фильтра вычисляется с помощью предварительно заданных связей. Для линейной пространственной фильтрации отклик определяется суммой произведения коэффициентов фильтра на значения пикселей маски фильтра. Для маски 3х3 элемента, показанной на рисунке 6, отклик ***R*** линейной фильтрации в точке (x,y) изображения составит:

(1.1)  
что, как видно, является суммой произведений коэффициентов маски на значения пикселей непосредственно под маской. Также можно заметить, что коэффициент w(0,0) находится при значении f(x,y), указывая тем самым, что маска центрирована в точке (x,y). При обнаружении перепадов яркости применяются дискретные аналоги производных первого и второго порядка.  
 Первая производная одномерной функции f(x) определяется как разность значений соседних элементов:

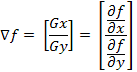
(1.2)

Здесь попользуется запись в виде частной производной, благодаря чему сохраняются те же обозначения в случае с двумя переменными ***f(x,y)***, где необходимо иметь дело с частными производными по двум пространственным осям. Применение частной производной не изменяет существа рассмотрения.

Аналогично, вторая производная находится разностью соседних значений первой производной:

(1.3)

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

(1.4)

Направление вектора градиента совпадает с направлением максимальной скорости изменения функции f в точке (x,y) [2].

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

(1.5)

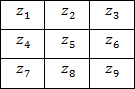
Эта величина является значением максимальной скорости изменения функции f в точке (x,y), где максимум достигается в направлении вектора ∇f. Величину ∇f также часто называют градиентом.

Направление вектора градиента является очень важной характеристикой. Обозначим α(x,y) угол между направлением вектора ∇f в точке (x,y) и осью x. Как известно из математического анализа,

(1.6)

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

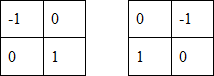
Рассмотрим оператор Робертса. Пусть область 3х3 (см. рис. 2), содержит значения яркости в области некоторого участка изображения.

  
Рисунок 2. Окрестность 3х3 внутри изображения

Одним из способов нахождения первых частных производных в точке image является применение следующего перекрестного градиентного оператора Робертса [2]:

image(1.7)  
и  
image (1.8)

Полученные производные можно реализовать путем обработки изображения с помощью оператора, приведенного на рисунке 3, используя описанную выше процедуру фильтрации.

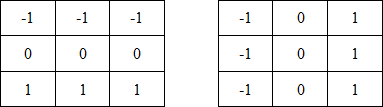
  
Рисунок 3. Примеры масок оператора Робертса

Использование масок размерами 2х2 не очень удобно, потому что у них отсутствует центральный элемент, что сильно отражается на результате выполнения фильтрации. Но этот недостаток создает другое, полезное свойство этого алгоритма – высокую скорость обработки изображения.

Еще один оператор – Превитта, так же, как и оператор Робертса, работает с областью изображения, показанной на рисунке 2, однако использование масок задается другими выражениями [2]:

image(1.9)  
и  
image (1.10)

В приведенных выше формулах разность между суммами по верхней и нижней строкам области является приближенным значением производной по оси x, а разность между суммами по первому и последнему столбцам этой области – производной по оси y. Для реализации этих формул применяется оператор, описываемый масками, приведенными на рисунке 4, который называется оператором Превитта.

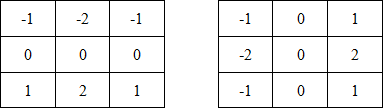
  
Рисунок 4. Маски оператора Превитта

Оператор Собеля тоже использует такую маску, отображенную на рисунке 2. Он очень похож на оператор Превитта, отличие заключается в применении весового коэффициента 2 для средних элементов [2]:

image(1.11)  
и  
image (1.12)

Это увеличенное значение необходимо для уменьшения эффекта сглаживания за счет увеличения весов средних точек.

Примеры масок, используемых оператором Собеля, показаны 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)

Таким образом, были рассмотрены алгоритмы вычисления градиентов на изображении.

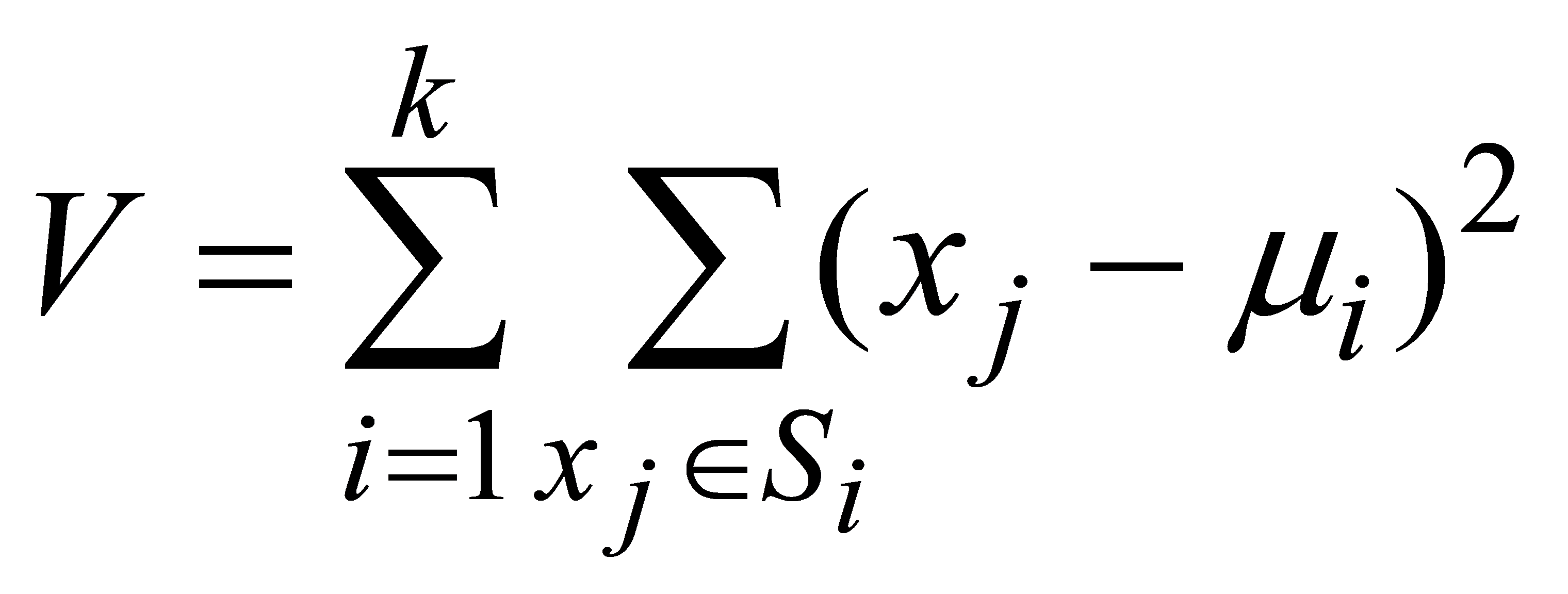
* **Описание алгоритмов кластеризации**

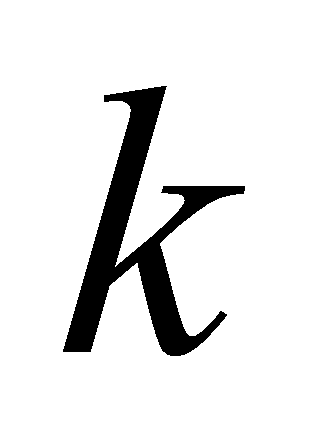
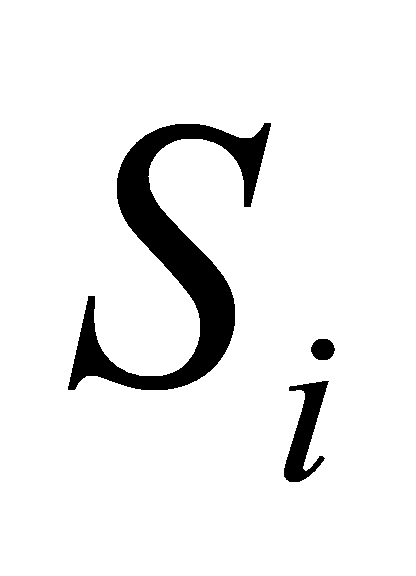
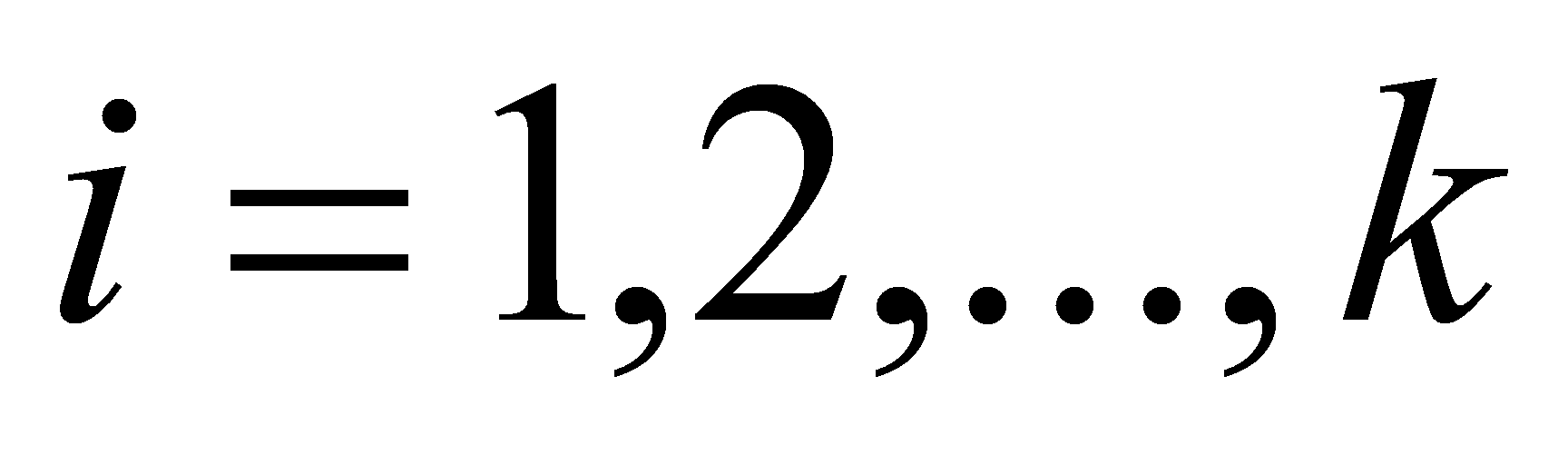
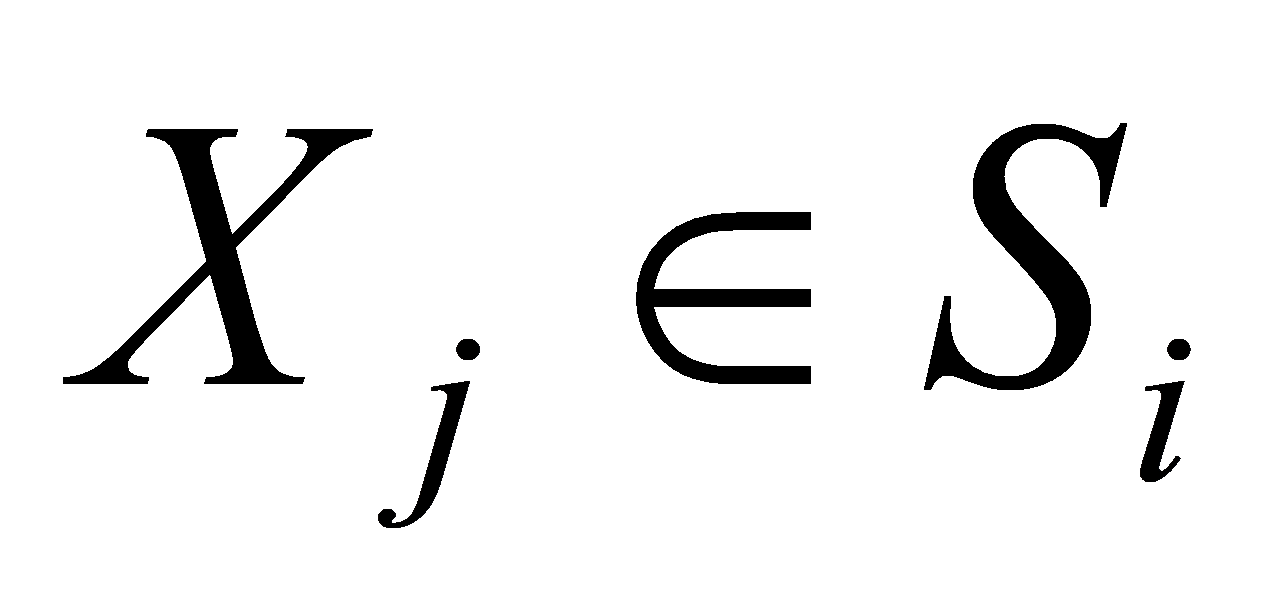
Кластеризация ­**–** задача разбиения множества объектов на группы (кластеры) по степени схожести друг на друга. Для того, чтобы сравнить два объекта, нужно определить критерий, с помощью которого можно сравнить эти объекты. Очень часто таким критерием является расстояние между этими объектами. Существует множество мер расстояния:

* Евклидово расстояние – самое распространенное расстояние. Это расстояние представляет собой геометрическое расстояние в многомерном пространстве;
* Квадрат евклидова расстояния – позволяет придать большие веса для отдаленных друг от друга объектов;
* Расстояние Чебышева – максимум модуля разности компонент векторов;
* Расстояние городских кварталов (манхэттенское расстояние) – среднее разностей по всем координатам. Эта мера расстояния приводит к результатам, похожим на расстояния Евклида, однако для нее влияние отдельных больших разностей уменьшается;

Выбор критерия схожести осуществляется исследователем, при этом стоит учитывать, что в зависимости от выбранной меры результаты кластеризации будут отличатся.

K-means – очень популярный метод кластеризации, особенностью которого является минимизация суммарного квадратичного отклонения точек кластеров от центров этих кластеров:

, (1)

где - число кластеров, - полученные кластеры, и - центры масс векторов.

Алгоритм разбивает множество элементов векторного пространства на известное заранее число кластеров. В основе алгоритма лежит итеративный пересчет центров масс каждого кластера, полученного на предыдущем шаге, после чего векторы разбиваются на кластеры вновь с учетом близости центров масс по выбранной метрике (см рисунок 7).

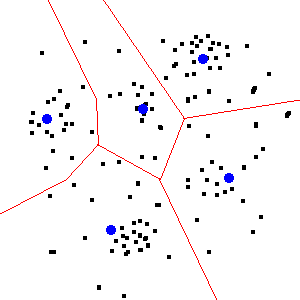


Рисунок 6 **–** Пример результата работы k-means

Работа алгоритма заканчивается, когда в результате итерации не изменяется ни один центр масс. Алгоритм имеет следующие недостатки:

* Чувствительность к шуму существенно искажает результаты кластеризации;
* Отсутствие гарантии достижения глобального минимума выбранной метрики.
* Полученный результат зависит от выбора исходных центров масс;
* Количество кластеров необходимо определить до начала работы алгоритма;

k-means++ – улучшенная версия предыдущего алгоритма. Его отличительной особенностью является поиск лучших начальных значений центров масс кластеров, что повышает стабильность результатов.

Шаги алгоритма:

* Выбрать случайным образом центр масс из набора исходных векторов.
* Для всего набора исходных векторов найти квадрат расстояния до ближайшего центра масс (только из тех, которые уже выбраны) dx².
* Выбрать из этих векторов следующий центр масс такой, чтобы вероятность выбора точки была пропорциональной вычисленному для неё квадрату расстояния. Нужно также учитывать, чтобы центр масс не совпадал с одним из уже выбранных в качестве центров масс векторов.
* Выполнять шаги 2 и 3 до тех пор, пока не будут найдены все центры масс.

Затем выполняется обычный алгоритм K-Means.

Другим вариантом решения проблемы нестабильности k-means является использование алгоритма c-means. Его особенностью является использование вероятности принадлежности объекта к кластеру. В остальном этот алгоритм имеет те же проблемы, как и k-means.

* **Описание алгоритмов сегментации**

Сегментация изображений – разбиение изображения на области (сегменты), которые являются однородными по некоторому признаку. Целью сегментации является упрощение представления изображения, в результате чего его становится проще анализировать.

Алгоритмы сегментации делятся на группы:

* иерархическое разбиение (все фрагменты одного уровня разбиваются на несколько сегментов).
* Сегментация на неперекрывающиеся области;
* Сегментация на перекрывающиеся области;

Если учесть, что разные области одного и того же объекта могут существенно отличаться друг от друга по признакам, имя общим только семантику, то можно сделать вывод, что задача сегментации схожа на задачу распознавания.

Часто алгоритмы классифицируют следующим образом:

* Пороговые алгоритмы;
* Граничные алгоритмы;
* Алгоритмы наращивания областей;
* алгоритмы сегментации на основе кластеризации.

Когда нужно распознать изображение, объекты внутри которого имеют большие перепады по яркости, стоит применять пороговые алгоритмы. Например, для сегментации пороговым алгоритмом необходимо получить из полутонового изображения бинарное, для этого нужно установить пороговое значение, оптимальное значение которого можно получить, проведя анализ гистограмм. После этого функция изображения отображает элементы изображения в значение 1, если уровень яркости выше порогового, в ином случае выставляет в значение 0 (см. рисунок 8).

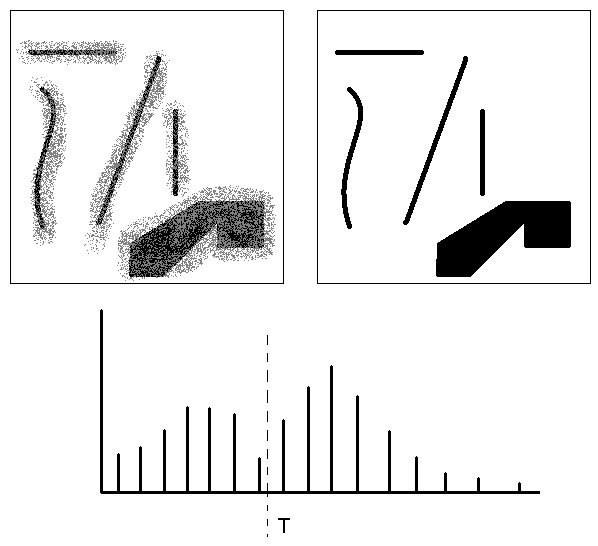


Рисунок 8 **–** Пример сегментации изображения с использованием порога T

Если на изображении отсутствует шум и другие возмущения, то изменение яркости происходит на границах выделяемой области. Эту модель можно упростить до единой размерности, после чего распределение яркости будет однозначно совпадать с формой контура границы. Стоит отметить, что распределение яркости – первая производная от контура границ.

Эти алгоритмы обладают существенным недостатком: они могут давать хорошие результаты при сегментации простых изображений, однако они не дают хороших результатов при наличии шумов и других помех. Кроме этого, при наличии больших интервалов между экстремумами, поиск минимума будет затруднительным.

Для корректного применения этих алгоритмов нужно избегать смещения при определении порога, используя ужесточение контроля одинаковости распределения в светлых и темных областях гистограммы яркости. Также необходимо разбивать изображение на малые части, в результате чего гистограмма яркости будет иметь ярко выраженные экстремумы, но также нужно следить, чтобы эти части были достаточно большими, чтобы была возможность оценить местоположение экстремумов и описать окрестность.

Если на изображении есть связность внутри разных сегментов, то целесообразно применять алгоритмы наращивания областей. Благодаря их использованию происходит группировка в однородные области граничных элементов с близкими уровнями яркости. Наиболее распространенными алгоритмами поиска сегментов являются алгоритмы расщепления и слияния областей, алгоритмы водоразделов (см. рисунок 9), алгоритмы цетроидного связывания.

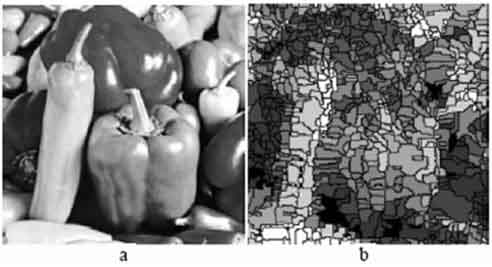


Рисунок 9 – Пример работы алгоритма водоразделов

а – исходное изображение; b – сегментированное изображение.

Наиболее простыми способами наращивания областей является применение совокупности свойств разных групп точек для определения элементарных областей., и уже на их основе строятся более сложные варианты алгоритмов.

При использовании центроидного связывания на основе информации об объекте выбираются стартовые точки, которым присваиваются разные метки, при этом точки, которые имеют одинаковые метки, группируются в наборы. Этот алгоритм также пригоден для сегментации простых изображений, для сложных изображений нужно выбирать точки по итерациям, на каждой итерации необходимо рассматривать набор точек на принадлежность из соседей к данному множеству. Точки, которые добавлены в множество на предыдущих итерациях, не рассматриваются. В результате, проводится анализ всех всех множеств, точки, которые были добавлены к множеству на текущей итерации, называются фронтом, а объединение фронтов называется волной, поэтому такие алгоритмы называются волновыми.

Алгоритм слияния и расщепления делит множество точек с помощью деления определенным образом изображения на квадратные области, которые в дальнейшем проверяются на однородность яркости. Если квадратная область не попадает под эти условия, то её заменяют четырьмя меньшими областями, а подходящие по условию четыре квадратные области могут быть объедены в одну.



Рисунок 10 – Пример работы алгоритмов сегментации

а – исходное изображение; б – метод слияния, в – метод слияния/разбиения.

Недостатками алгоритма является выделение общих фрагментов, которые не отражают информацию об изменениях яркости внутри области и возможных границах. Преимуществами данного алгоритма являются хорошие результаты при использовании их на зашумленных изображениях.

Другим способом сегментации являются граничные методы, особенностью которых является поиск точек, которые лежат на границах областей. Главной характеристикой для поиска границ является яркость, однако можно использовать и другие характеристики, такие как градиент и текстура.

Границей обозначаются сильные изменения яркости на изображении. Такие переходы могут быть пошаговыми, в случае резких изменений, или линейными, в случае резких изменений яркости, но с возвращением значением на место через короткие промежутки. Последние ситуации достаточно редки благодаря сглаживанию на изображении, поэтому если рассматривать изменения в пределах конечного отрезка, то пошаговые изменения идентифицируются как линейно возрастающие, а линейные представляются треугольниками. Возможны случаи, когда удобно определять границу по обоим признакам одновременно.

Фрагмент границы соответствует координатам границы и направлению границы, которое можно определить градиентным углом. Контур формируется из набора границ или математических кривых, которые моделируют этот набор. Есть алгоритмы объединения границ, которые создают упорядоченные группы границ, используя для этого локальную информацию для поиска принадлежности пикселя границе. Также существуют алгоритмы поиска границ, которые используют глобальную информацию.

Эти алгоритмы, как правило, включают в себя фильтрацию, усиление, выделение, определение местоположения и направления. Существует множество алгоритмов, таких как операторы Собеля, Робертса, Кэнни, Превитта.

Данные алгоритмы можно разбить на две группы: алгоритмы сравнения с эталоном и градиентно-дифференциальные алгоритмы. Алгоритмы обеих групп проверяют, в каком случае изменение градиента яркости становится большим, чтобы определить, что здесь существует граница.

Также существуют алгоритмы сегментации, основанные на кластеризации (примеры алгоритмов кластеризации были описаны в предыдущей главе). Их недостатками являются вероятность сбоя в случае некорректного определения параметров модели по отношению к сегментируемым данным, а также в случае, когда модель содержит неполный набор характеристик сегментов. Кроме этого, алгоритмы могут допустить ошибку, в случае если сегменты имеют разную форму, плотность, или размер. Существуют различные варианты работы алгоритмов. Одни измеряют сходство двух кластеров с использованием динамической модели, другие объединяют кластеры при наличии условия внутренней связанности и схожести классов между собой. Также существуют алгоритмы, использующие жесткую фильтрацию, используют нечеткую логику, иерархическую кластеризацию, и другие алгоритмы.

Можно сделать вывод, что каждый из алгоритмов имеет свою область пременения, которая зависит от входных данных, различных требований для распознавания, вычислительных затрат и времени на принятие решений.

**Описание градиентного алгоритма разбиения изображения в карту мазков**

В выполненной бакалаврской работе был написан алгоритм преобразования изображения в карту мазков на основе кластеризации. В результате его использования были обнаружены недостатки:

* С увеличением разрешения исходных изображений время обработки существенно увеличивается;
* Соседние мазки с резким перепадом значения яркости искажают границу между сегментами;
* Длина мазков сильно зависит от размеров сегментов: чем больше сегмент, тем больше возможная длина мазка.

Таким образом, возникла потребность в доработках, с целью устранить недостатки текущего решения.

Алогритм, устраняющий пересчиcленные проблемы:

1. Фильтрация исходного изображения;
2. Вычисление градиентов;
3. Создание суперпикселей (Сегментация);
4. Формирование мазков;
5. Сохранение мазков в формате HPGL.

Первым этапом выступает фильтрация изображения, она позволяет снизить уровень шума на изображении, а также сокращает сложность анализа изображения.

На втором этапе вычисляются градиенты каждого пикселя, что позволяет определить границы объектов и направление формирования мазков.

Третьим этапом выступает сегментация изображения на суперпиксели. Этот этап необходим для того, чтобы сформировать элементарные объекты на изображении, которые впоследствии будут сформированы в мазки. Благодаря такому подходу повышается быстродействие работы алгоритма, поскольку нет необходимости вычислять характеристики для каждого пикселя, оперируя суперпикселями. Также на этом этапе вычисляются характеристики каждого суперпикселя: среднее значения цвета в заданных форматах, среднее значение величины градиента и его направление.

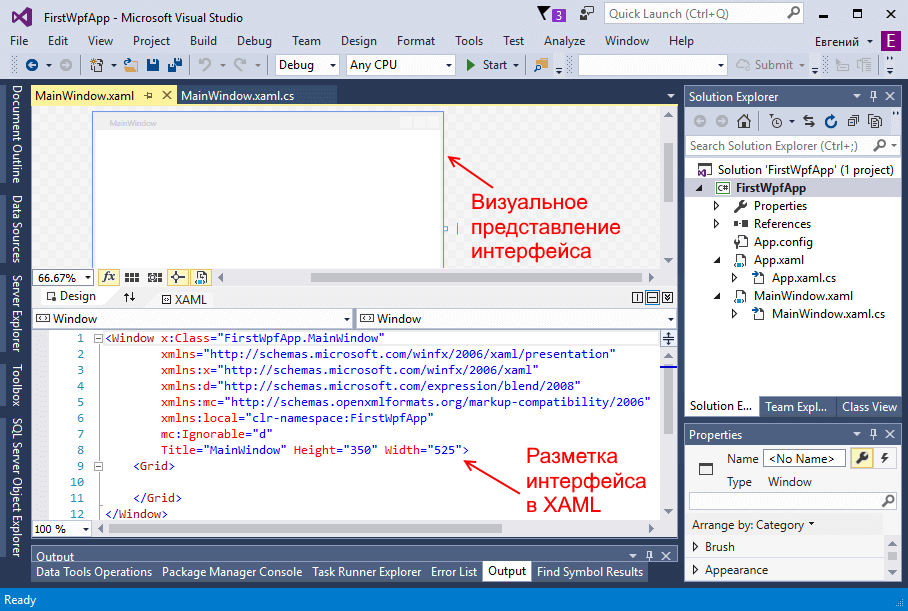
Мазки формируются исходя из заданных параметров: максимально возможной длины и погрешности по цвету. Суперпиксели формируют мазки согласно заданным параметрам из полученных на предыдущем этапе суперикселям, с учетом их характеристик. Этот этап требует вычислительных затрат больше, чем любой другой этап алгоритма.

Последним этапом полученные мазки сохраняются в формате HPGL - стандартный язык для всех плоттеров. Благодаря использованию этого языка, появляется возможность работы с любым устройством, которое поддерживает его, таким образом формируется регламент взаимодействия и пропадает зависимость от компонентов разрабатываемого приложения. Другими словами, результат работы программы может использовать любое устройство, работающее по стандарту HP-GL/2.

* **Выводы**
* **Выбор инструментов разработки**

Клиентская часть будет выполнена с применением технологии WPF (Windows Presentation Foundation) - системы для построения клиентских приложений операционной системы Windows, её отличительными чертами являются визуально привлекательные возможности взаимодействия с пользователями, а также наличие широкого спектра возможностей по созданию интерактивных настольных приложений. В основе этой технологии лежит векторная система визуализации, которая не зависит от разрешения устройства вывода, а также создана с учетом всех возможностей современного графического оборудования. Графической технологией, лежащей в основе WPF, является DirectX, в отличии от технологии Windows Forms, в которой используется GDI/GDI+. Таким образом, производительность WPF выше, за счет использования аппаратного ускорения графики. Также важной особенностью является использования языка декларативной разметки XAML: он дает возможность создавать насыщенный графический интерфейс за счет использования декларативного объявления интерфейса, или используя код на языке C#, либо совмещать и то и другое.

Пример WPF приложения, написанного в среде Visual Studio, показан на рисунке 13.

Рисунок 13 – Приложение в WPF

Другими важными преимуществами являются возможность использования в приложениях двухмерной и трехмерной графики, богатый набор встроенных в систему элементов управления, возможность создавать свои элементы, возможность создания анимации, привязки данных, стили, темы, шаблоны, и многое другое. Также преимуществом является поддержка множества систем семейства Windows - от Windows XP до Windows 10.

Архитектура WPF схематически показана на рисунке 12.

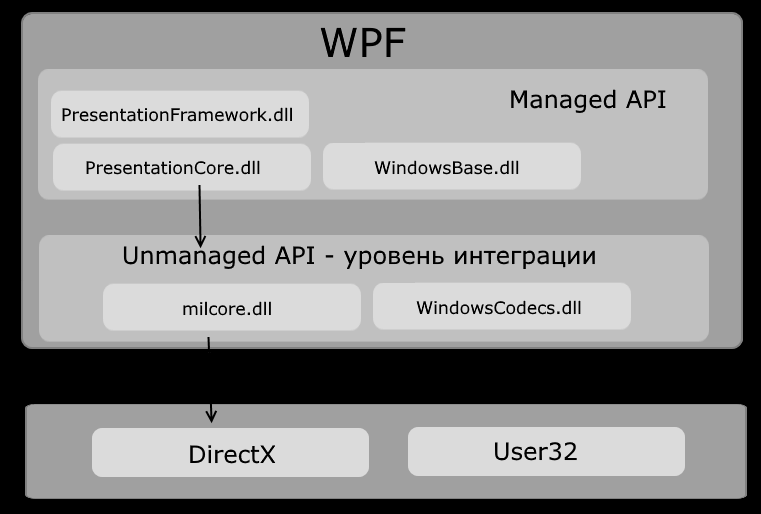


Рисунок 12 – Архитектура WPF

Из рисунка видно, что WPF разбивается на два уровня: уровень интеграции с DirectX (unmanaged API) и уровень, работающей под управлением общеязыковой среды выполнения .NET - CLR. Этот API описывает функциональность работы платформы WPF, и включает в себя следующие компоненты:

1. PresentationFramework.dll: включает основные реализации элементов управления и компонентов, которые применяются для построения графического интерфейса;
2. PresentationCore.dll: включает базовые типы для PresentationFramework.dll;
3. WindowsBase.dll: включает вспомогательные классы, которые могут использоваться вне платформы WPF;

Компоненты, которые служат для интеграции с DirectX:

1. milcore.dll: обеспечивает интергацию компонентов WPF с DirectX;
2. WindowsCodecs.dll: обеспечивает низкоуровневую поддержку для изображений в WPF.

Еще ниже находятся компоненты операционной системы и DirectX, которые осуществляют визуализацию компонентов приложения.

В остальном, для разработки будут использоваться те же технологии, что и в бакалаврской работе: компоненты приложения будут разбиты на библиотеки классов dll, разработка компонентов будет производиться в среде Visual Studio.

* **Проектирование архитектуры**

В данной главе будет рассмотрена программная реализация пользовательского приложения с использованием двух алгоритмов формирования карты мазков: SmearTracer и GradientTracer. При проектировании архитектуры была разработана высокоуровневая структура, показанная на рисунке 14.

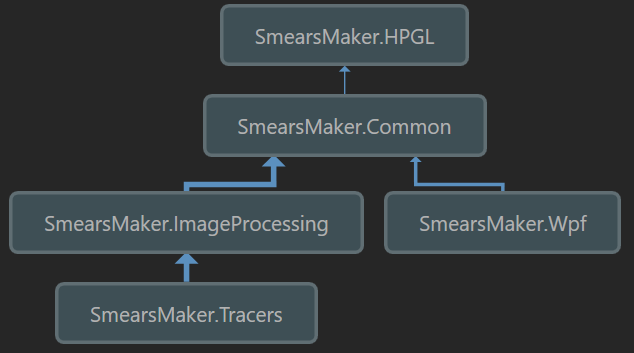


Рисунок 14 – Архитектура приложения

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. 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. 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. 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.

* **Разработка**

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

1. ITracer – интерфейс, определяющий функционал работы алгоритма создания карты мазков;
2. TracerBase – абстрактный класс, определяющий базовое поведение алгоритма создания карты мазков, является общим классом для всех алгоритмов трассировки;
3. ImageView ­– класс, который включает в себя результат работы алгоритма: изображение и имя операции;
4. ImageSetting – класс, который описывает настройку для алгоритма трассировки, включает имя настройки и значение;
5. IProgress – интерфейс, описывающий функционал, определяющий этап выполнения алгоритма. При использовании класса TracerBase используется внутренняя реализация ProgressImpl.

Схема, описывающая зависимости перечисленных типов, показана на рисунке 15.

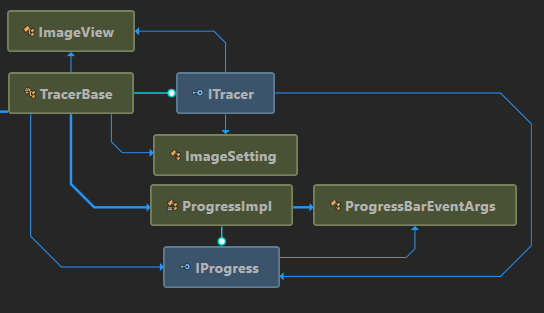


Рисунок 15 – Типы, входящие в компонент SmearsMaker.Common

Также в этот компонент входят следующие типы:

1. 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;
2. 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;
3. 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;
5. PixelCollection – коллекция, внутри которой используется словарь, содержащий пиксели, ключом является имя слоя, которому принадлежит пиксель;
6. Pixel – класс, который содержит данные пикселя на изображении.
7. BaseShape – абстрактный класс, описывающий базовое поведение любого объекта на изображении.

Зависимость между перечисленными типами показана на рисунке 16.



Рисунок 16 – 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:

1. IPltReader – интерфейс, задающий функционал для чтения Plt файла;
2. IPltReaderImpl – реализация интерфейса IPltReader;
3. IPltCreator – интерфейс, задающий функционал для создания Plt файла;
4. Painter – внутренный вспомогательный класс, задачей которого явлется формирование изображения из геометрических фигур;
5. PltHelper – вспомогательный статический класс, содержит методы для работы с изображениями.

Зависимости между типами продемонстрированы на рисунке 16.

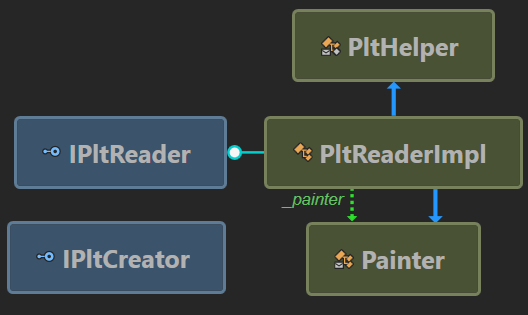


Рисунок 16 – Типы, входящие в компонент SmearsMaker.Common

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

1. IFilter – интерфейс, задающий функциональность фильтра;
2. MedianFilter – реализация интерфейса IFilter, описывает поведения медианного фильтра;
3. IDetector – ­­­­ интерфейс, который задает функциональность оператора;
4. Sobel­ ­– реализация интерфейса IDetector, описывает поведения оператора Собеля;
5. IClusterizer – интерфейс, задающий функциональность алгоритма кластеризации;
6. Kmeans ­– абстрактный класс, реализующий интерфейс IClusterizer, описывает базовое поведение алгоритма Kmeans, позволяет реализовывать множество алгоритмов, основанных на KMeans;
7. KMeansClassic – класс, является реализацией алгоритма Kmeans;
8. Cluster – класс, описывающий поведения кластера;
9. ISplitter – интерфейс, задающий функциональность алгоритма разбиения любой фигуры на сегменты;
10. IStrokesBuilder – интерфейс, задающий функциональность алгоритма формирования мазков;
11. BrushStroke – класс, описывающий поведения мазка.

Перечисленные типы показаны на рисунке 17.

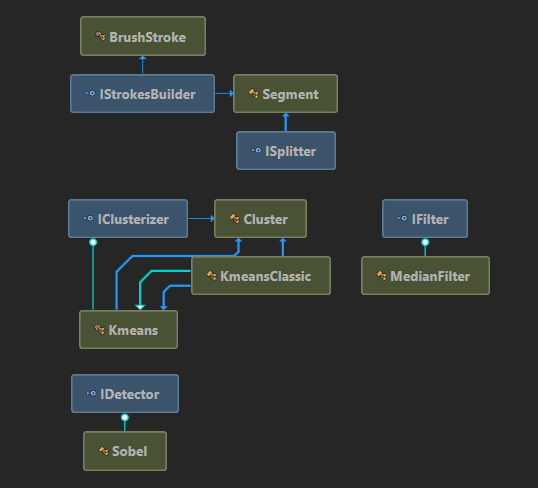


Рисунок 17 – Типы, входящие в компонент SmearsMaker.ImageProcessing

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:

1. Utils – статический вспомогательный класс, который содержит методы для вычисления различных расстояний, а также включает методы, генерирующие различные данные;
2. BrushStrokeImpl – реализация мазков;
3. ImageHelper – статический вспомогательный класс, который содержит методы для работы с изображениями;
4. PltHelper – статический вспомогательный класс, который содержит методы, помогающие сформировать plt из различных типов;
5. BaseShapeExtentions – статический класс, расширяющий функциональность класса BaseShape;
6. BrushStrokeExtension – статический класс, расширяющий функциональность класса BrushStroke;
7. IServicesFactory­ – интерфейс, который определяет, какие алгоритмы будут использоваться;

Типы, которые необходимы для работы алгоритма SmearTracer:

1. Smear – класс, который содержит ссылки на мазок, сегмент и кластер;
2. STracer – класс, реализующий алгоритм SmearTracer;
3. SmearFactory – реализация интерфейса IServicesFactory, которая создает необходимые типы для выполнения алгоритма;
4. StlImageSettings – класс, который описывает, какие настройки могут использоваться в алгоритме SmearTracer.

Типы, которые необходимы для работы алгоритма GradientTracer:

1. GTracer – абстрактный класс, описывающий алгоритм GradientTracer;
2. HexagonGTracer – вариант алгоритма GTracer, с использованием шестиугольных суперпикселей;
3. HexagonTracerFactory – реализация интерфейса IServicesFactory, которая создает необходимые типы для выполнения алгоритма;
4. SquadGTracer – вариант алгоритма GTracer, с использованием квадратных суперпикселей;
5. SquadTracerFactory – реализация интерфейса IServicesFactory, которая создает необходимые типы для выполнения алгоритма;
6. GtImageSettings – класс, который описывает, какие настройки могут использоваться в алгоритме GradientTracer.

Зависимости меду перечисленными типами показаны на рисунках 18-20;

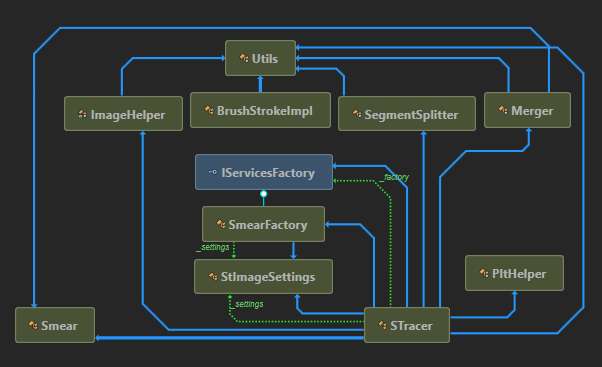
Рисунок 18 – Типы, входящие в компонент SmearsMaker.Tracers

Рисунок 19 – Типы, входящие в компонент SmearsMaker.Tracers

Рисунок 20 – Типы, входящие в компонент SmearsMaker.Tracers

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

1. GradientStrokesBuilder – класс, реализующий интерфейс IStrokesBuilder. Содержит логику формирования мазков из частей изображения с использованием градиента;
2. SimpleStrokesBuilder – класс, реализующий интерфейс IStrokesBuilder. Содержит логику формирования мазков из частей изображения без учета градиента;
3. PairStrokesBuilder – класс, реализующий интерфейс IStrokesBuilder. Содержит альтернативную логику формирования мазков из частей изображения без учета градиента;
4. SuperPixelSplitter – класс, реализующий интерфейс ISplitter. Содержит логику разбиения изображения или его части на квадратные суперпиксели;
5. HexagonSplitter – класс, реализующий интерфейс ISplitter. Содержит логику разбиения изображения или его части на шестиугольные суперпиксели.

Перечисленные типы не имеют зависимостей (см. рисунок 21).

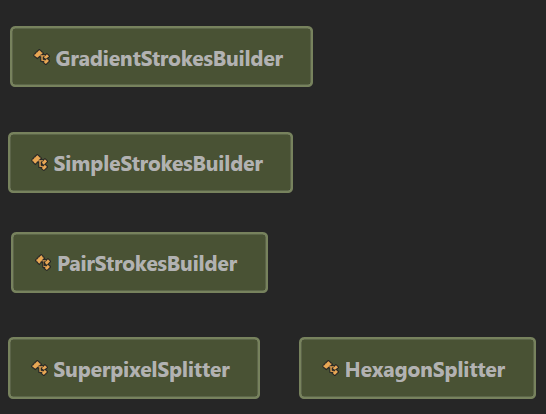


Рисунок 21 – Типы, входящие в компонент SmearsMaker.Tracers

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

1. Settings, Resources, App – типы, которые формируются при создании приложения WPF;
2. App.сonfig – конфигурационный файл, который содержит в себе описание сборок с различными реализациями алгоритмов разбиения изображения. ПО умолчанию содержит ссылку на компонент SmearsMaker.Tracers;
3. MainWindow – главное окно приложения, содержит меню с кнопками, окно вывода изображения, а также кнопку запуска алгоритма;
4. SettingsWindow – окно приложения, в котором выводятся настройки выбранного алгоритма;
5. ApplicationViewModel – класс, описывающий логику работы пользовательского приложения, представляет собой прослойку между разметкой окна и бизнес логикой приложения;
6. Command – класс, реализующий интерфейс ICommand, который позволяет выполнять связывание событий в WPF;
7. FileHelper – вспомогательный статический класс, содержащий методы для работы с файлами.

Зависимости меду перечисленными типами показаны на рисунке 22.

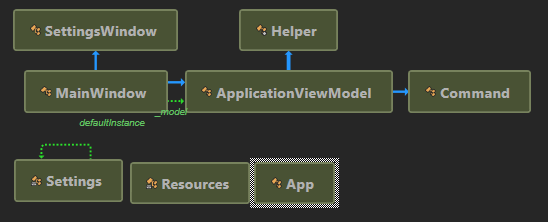


Рисунок 22 – Типы, входящие в компонент SmearsMaker.Wpf

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.

* **Описание разработанного программного решения**

Для начала работы нужно запустить приложение SmearsMaker.Wpf.exe. Запустив приложение, нужно открыть файл, нажав на вкладку Menu, а затем на кнопку Open. Результат загрузки фотографии показан на рисунке 8.



Рисунок 8 – загруженное изображение “Лена”

Следующим шагом нужно выбрать алгоритм. Для этого нужно перейти во вкладку Algorithms, а затем выбрать нужный алгоритм. Алгоритмы загружаются при открытии приложения при помощи рефлексии, причем ссылки на компоненты с реализованными алгоритмами описываются в файле SmearsMaker.Wpf.config. Таким образом, можно реализовать любое число алгоритмов и привязать их к программе путем изменения всего лишь одного конфигурационного файла.

После этого можно изменить настройки алгоритма, для этого нужно перейти на вкладку Edit и нажать кнопку Settings (см. рисунок 8). Настройки применяются нажатием кнопки OK. Значения по умолчанию формируются непосредственно в коде приложения, и при любом новом открытии фотографии или выбора алгоритма из значения вычисляются заново.

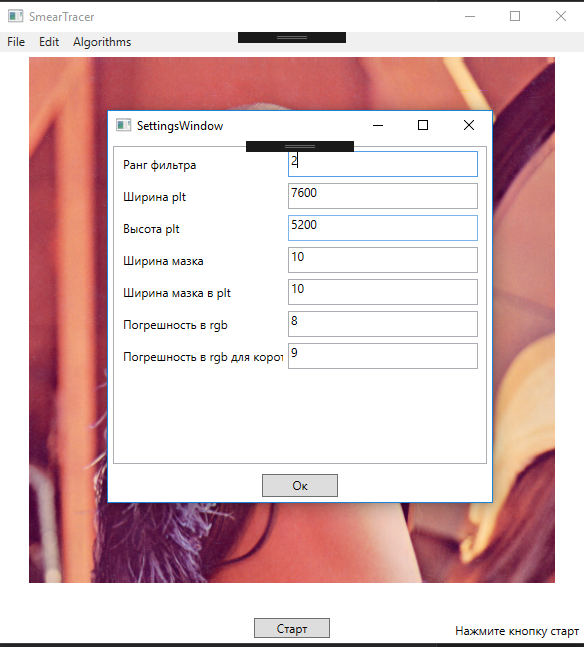


Рисунок 8 – Окно с настройками

Для запуска алгоритма необходимо нажать кнопку Старт, а затем подождать окончания работы, дождавшись статуса “Готово”. Статус выводится в нижней правой части окна, он отображает текущий этап выполнения алгоритма, либо выводит подсказки пользователю. Значения статуса формируются разработчиком в коде приложения. Результат окончания работы алгоритма показан на рисунке 9.

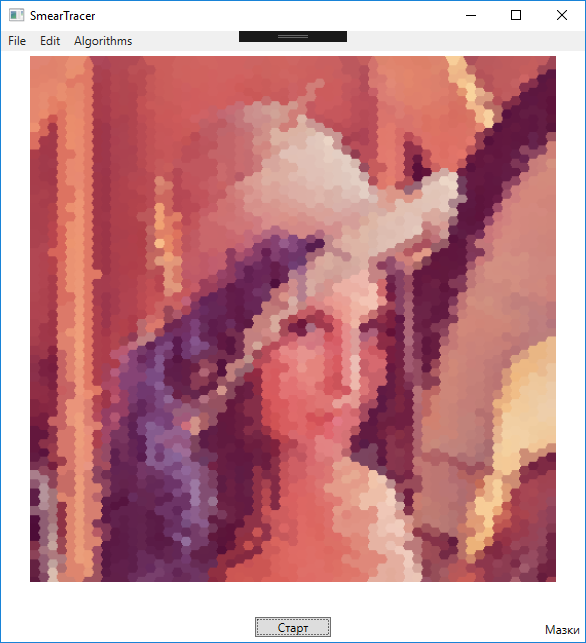


Рисунок 9 – Результат работы программы

После этого можно сохранить результаты работы программы в файл, для сохранения файла plt нужно перейти на вкладку File и нажать кнопку Save plt, после чего указать путь и имя сохраняемого файла, и нажать кнопку Save. Также в программе реализована возможность просматривать полученные plt файлы, для этого необходимо перейти во вкладку File и нажать на кнопку Open plt. Результат открытия plt файла показан на рисунке 10.

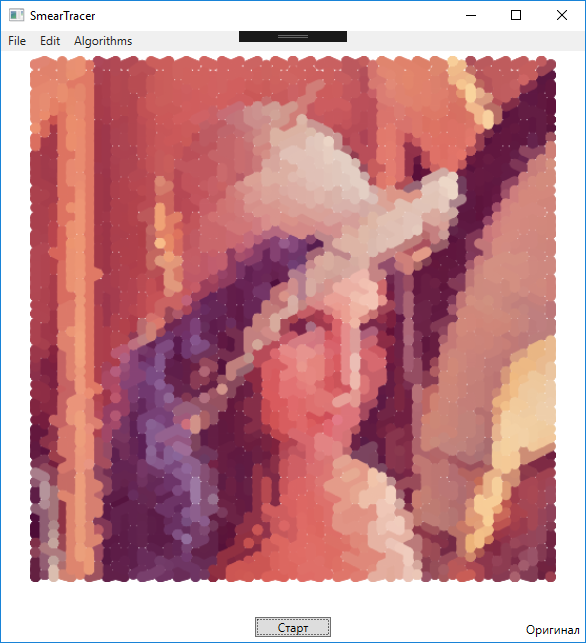


Рисунок 10 – Загруженный plt файл

Для сохранения изображений, полученных в ходе работы алгоритма, нужно перейти во вкладку File и нажать кнопку Save images, а после этого указать папку и нажать кнопку Save. Если необходимо скопировать изображение, представленное на главном окне, то достаточно перейти во вкладку Edit и нажать кнопку Copy.

Еще одним важным моментов являются файлы логов приложения, которые записываются в папку logs. Логи записываются в два файла: info.log и error.log. Первый файл содержит информацию о скорости работы различных этапов алгоритмов, а также другую детальную информацию. Пример сформированного файла показан на рисунке 11. Во второй файл записывается информация о всех ошибках, возникших в хоте работы программы. Существует также возможность изменить правила формирования файлов с логами, изменив файл Nlog.config.

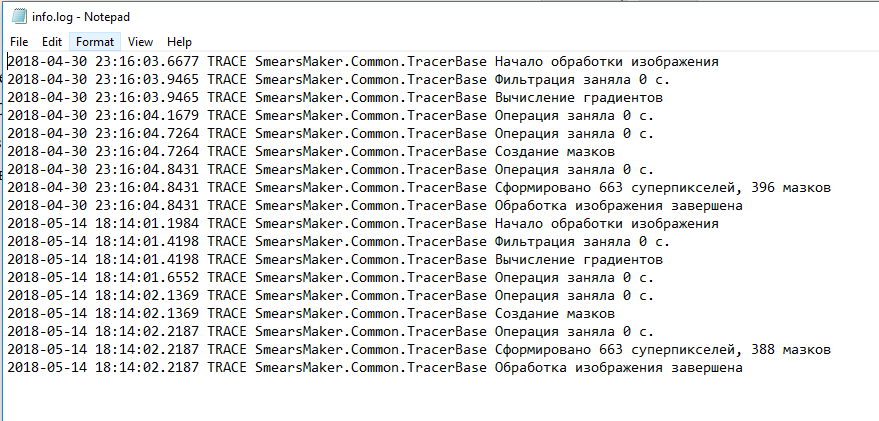


Рисунок 11 – Содержимое файла логов info.log

Таким образом, была описана работа с разработанным программным решением.

* **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. 7).



Рисунок 6 – Исходное изображение “Лена”



Рисунок 7 – Исходное изображение ”Лимоны”

Сначала запустим алгоритм SmearTracer, настройки для алгоритма показаны на рисунках 8-9.



Рисунок 8 – Окно настроек для изображения ”Лимоны”

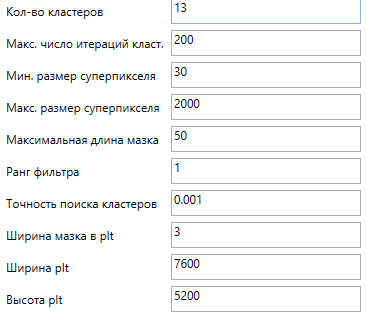


Рисунок 9 – Окно настроек для изображения ”Лена”

Полученные карты мазков показаны на рисунках 10-11.



Рисунок 10 – результат алгоритма SmearTracer для изображения ”Лимоны”

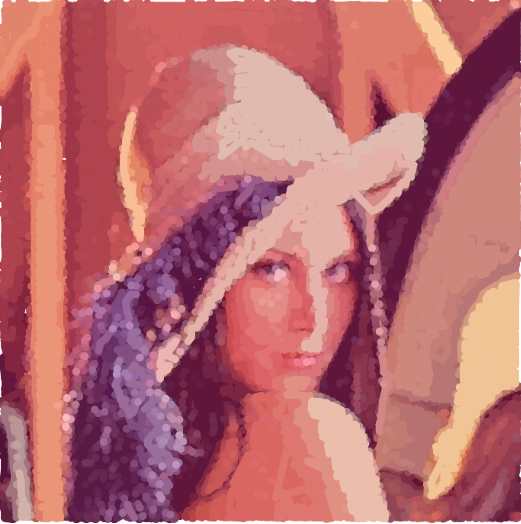


Рисунок 11 – результат алгоритма SmearTracer для изображения ”Лена”

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 12 и 13 показана разница между исходным изображением и изображением, разбитым на мазки.



Рисунок 12 – разница изображений ”Лимоны”



Рисунок 13 – разница изображений ”Лена”

Запустим алгоритм GradientTracer, настройки для алгоритма показаны на рисунках 14-15.

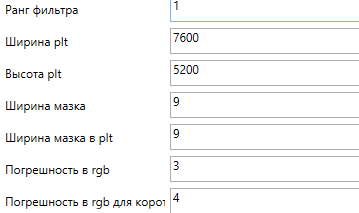


Рисунок 14 – Окно настроек для изображения ”Лимоны”

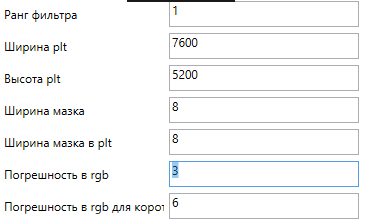


Рисунок 15 – Окно настроек для изображения ”Лена”

Полученные карты мазков показаны на рисунках 16-17.



Рисунок 16 – результат алгоритма GradientTracer для изображения ”Лимоны”

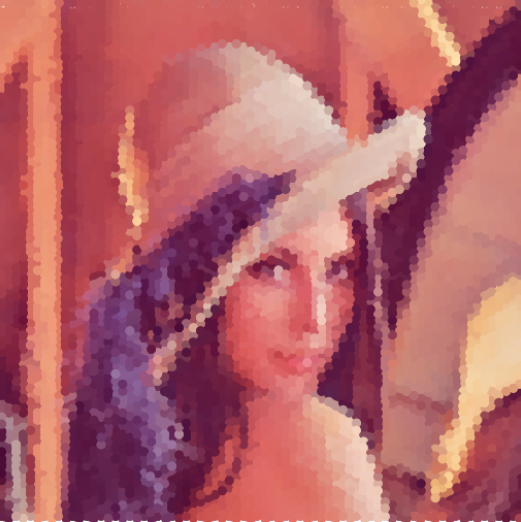


Рисунок 17 – результат алгоритма GradientTracer для изображения ”Лена”

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 18 и 19 показана разница между исходным изображением и изображением, разбитым на мазки.

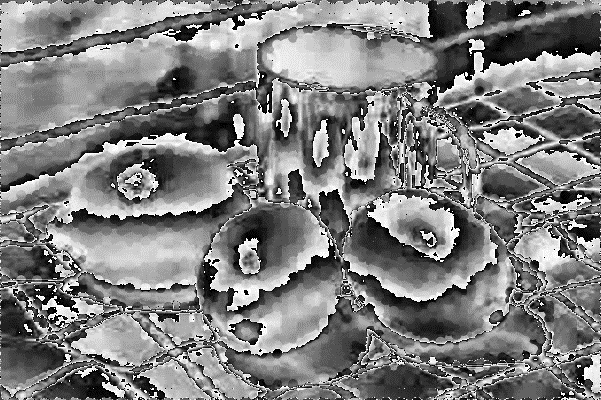


Рисунок 18 – разница изображений ”Лимоны”

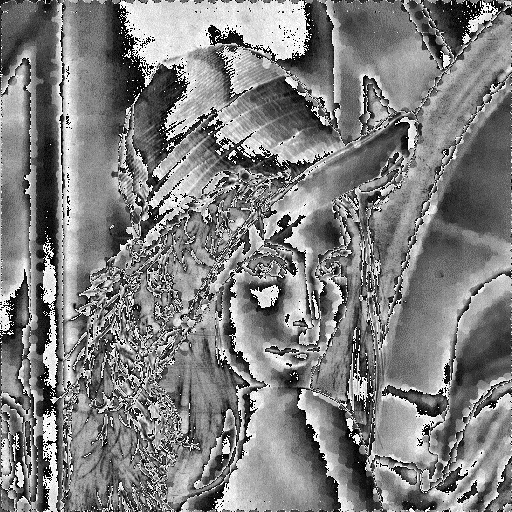


Рисунок 19 – разница изображений ”Лена”

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.

* **Составление бизнес-плана по коммерциализации результатов нир магистранта**
* **Концепция экономического обоснования разработки технического продукта**

Продукт представляет из себя программно-аппаратный комплекс и предназначен для создания, воспроизведения и реставрации художественных изображений, имитируя способ смешения красок на палитре и их нанесения кистью. Система состоит из робота-манипулятора, обеспечивающего наложение краски на основу, и автоматической палитры – специального устройства для смешения красок и подачи смеси нужного цвета на рабочий инструмент робота.

Основными потребителями являются компании, связанные с выполнением художественных работ оформительского, рекламного и шрифтового характера. Целью проекта является автоматизация технологии машинной живописи, снижение стоимости живописных, реставрационных и оформительских работ.

* **Расчёт полных затрат при разработке программного модуля**

Потребность в инвестициях:130528.6764 руб.

Эффективность реализации проекта:

* Стоимость одной репродукции: 20 т.р.
* Число продаж услуги в первом году, после реализации: 50

Эффективность проекта – 3 302 055 руб.

Срок окупаемости инвестиций составляет 8.5 месяцев

Стенд для отработки системы машинной живописи представлен на рис.6.

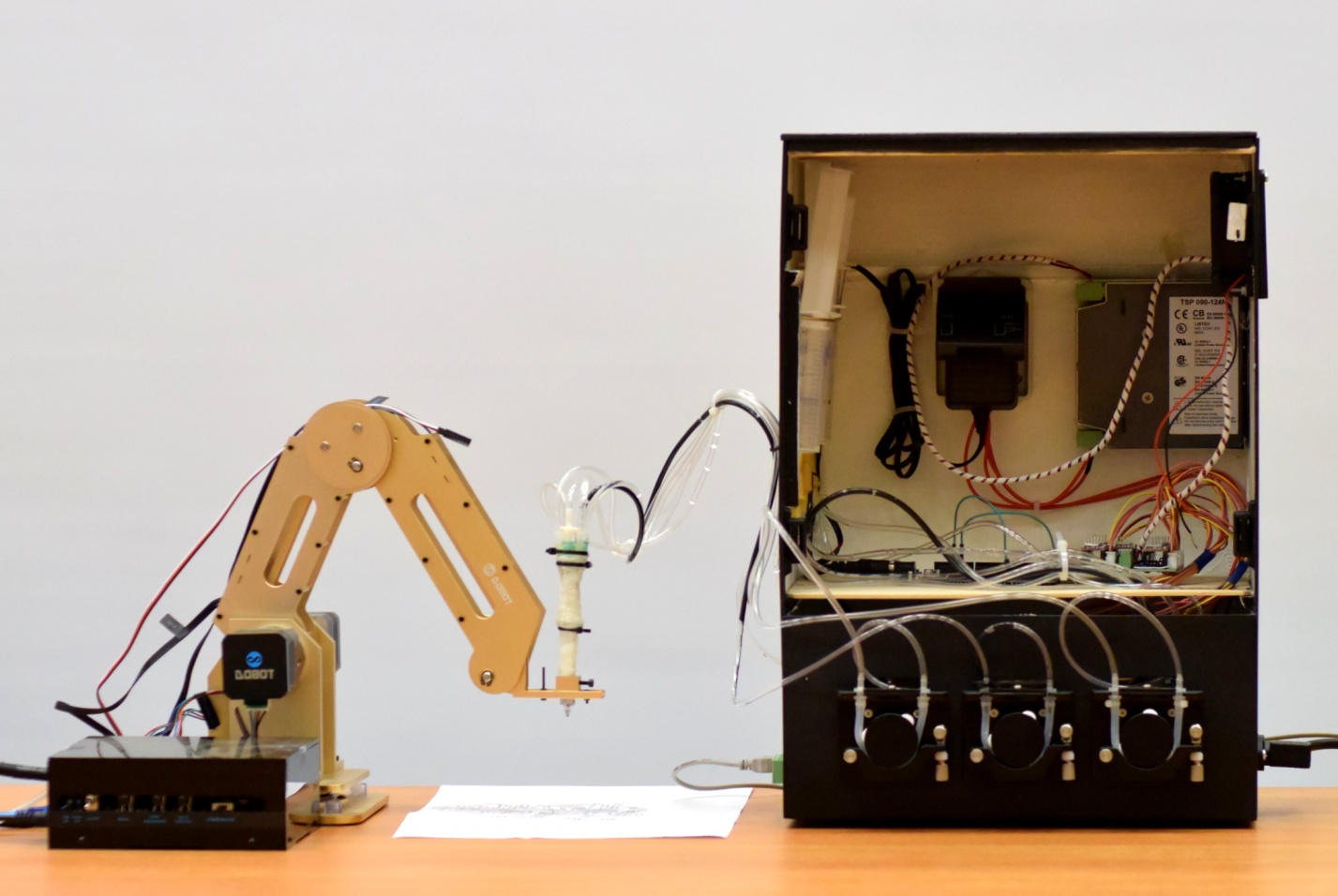


Рис. 6 – Система машинной живописи

Автоматическая палитра представляет собой устройство в отдельном корпусе, включающее в себя блок перистальтических насосов, блок из пяти емкостей с красками и одной емкости с водой для промывания смесителя, блоки питания, смеситель, установленный на роботе-манипуляторе и подключенный с помощью гибких полимерных трубок к автоматической палитре. Блок перистальтических насосов в составе автоматической палитры работает от источника питания с напряжением 24 В. Шесть насосов Kamoer KCS-B, снабженные драйверами Pololu a4988, подключены к системе управления на основе микроконтроллера ATMega 2560. Управление автоматической палитрой осуществляется c персонального компьютера по USB-интерфейсу.

Робот-манипулятор Dobot снабжен собственным контроллером. Точность позиционирования до 0,2 мм позволяет осуществлять прецизионное нанесение краски на основу. Размер рабочего поля ограничен геометрическими размерами робота и составляет 150х150 мм.

В таблице 4.1 изложены обязанности сотрудников.

*Таблица 4.1 - Обязанности сотрудников*

|  |  |  |
| --- | --- | --- |
| № | Этапы и содержание выполняемых работ | Исполнитель |
| 1 | Разработка ТЗ | Руководитель |
| 2 | Анализ ТЗ | Инженер |
| 3 | Изучение предметной области | Инженер |
| 4 | Консультация по работе | Руководитель |
| 5 | Разработка алгоритмов | Инженер |
| 6 | Написание библиотек | Инженер |
| 7 | Разработка ядра программы | Инженер |
| 8 | Разработка пользовательского интерфейса | Инженер |
| 9 | Разработка управляющего ПО | Инженер |
| 10 | Интеграция управляющего ПО в состав ПСОАТ | Инженер |
| 11 | Тестирование программы | Инженер |
| 12 | Оценка полноты решения и выявление недочетов | Руководитель |
| 13 | Исправление ошибок | Инженер |
| 14 | Оформление пояснительной записки | Инженер |

Разработка продукта длилась четыре месяца. Работы, которые велись во время разработки продукта и их сроки, представлены в таблице 4.2.

*Таблица 4.2 – Календарный план выполнения работ*

|  |  |  |
| --- | --- | --- |
| № п/п | Наименование работ | Срок выполнения, дней |
| 1 | Анализ ТЗ | 5 |
| 2 | Изучение предметной области | 4 |
| 3 | Разработка алгоритмов | 11 |
| 4 | Разработка библиотек | 12 |
| 5 | Разработка управляющего ПО | 9 |
| 6 | Интеграция управляющего ПО в ПСОАТ | 2 |
| 7 | Тестирование системы | 10 |
| 8 | Отладка продукта | 25 |
| 9 | Оформление инструкции пользователя | 7 |
| 10 | Выпуск конечной версии продукта | 2 |

Любой продукт нуждается в стабильной поддержке со стороны разработчиков. Далее планируется выделять некоторое рабочее время на поддержку системы и на её рекламу.

В штате небольшой компании останется два человека: разработчик и руководитель. Задача разработчика – выпуск продукции, поддержка системы, исправление ошибок, постепенный выпуск новых версий ПО с исправленными ошибками и мелкими дополнениями. Задача руководителя – консультация разработчика, продвижение программного продукта, общение с клиентами, заключение договоров с клиентами. Месячная заработная плата Руководителя—58800 руб., инженера – 17000 руб. Дневная заработная плата определяется как частное месячной заработной платы и количества рабочих дней в месяце (21-рабочий день).

Перечень работ по разработке системы и трудоемкость их выполнения приведены в таблице 5.1.

*Таблица 5.1 – Данные о длительности выполнения работы*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Этапы и содержание выполняемых работ | Исполнитель | Длительн. работ, дн. | Трудоемк.  t0, ч-дн. | Ставка,  руб./дн. |
| 1 | Разработка ТЗ | Руководитель | 2 | 2 | 2800 |
| 2 | Анализ ТЗ | Студент | 1 | 1 | 809,524 |
| 3 | Изучение предметной области | Студент | 2 | 2 | 809,524 |
| 4 | Консультация по работе | Руководитель | 4 | 4 | 2800 |
| 5 | Разработка алгоритмов | Студент | 5 | 5 | 809,524 |
| 6 | Написание библиотек | Студент | 6 | 6 | 809,524 |
| 7 | Разработка ядра программы | Студент | 2 | 2 | 809,524 |
| 8 | Разработка пользовательского интерфейса | Студент | 14 | 14 | 809,524 |
| 9 | Разработка управляющего ПО | Студент | 15 | 15 | 809,524 |
| 10 | Интеграция управляющего ПО в состав ПСОАТ | Студент | 5 | 5 | 809,524 |
| 11 | Тестирование программы | Студент | 2 | 2 | 809,524 |
| 12 | Оценка полноты решения и выявление недочетов | Руководитель | 2 | 2 | 2800 |
| 13 | Исправление ошибок | Студент | 2 | 2 | 809,524 |
| 14 | Оформление пояснительной записки | Студент | 10 | 10 | 809,524 |
| **Итого** | | | 72 | 72 | 74209,5 |

Данные о трудоемкости выполняемых работ и ставки каждого из исполнителей позволяют определить расходы на заработную плату и отчисления на страховые взносы на обязательное социальное, пенсионное и медицинское страхование.

*Зосн* – расходы на основную заработную плату исполнителей(руб.); *k*—количество исполнителей; *Ti*—время, затраченное исполнителем на проведение исследования(дни); *Ci* —ставка *i*-го исполнителя (руб./день).

*Здоп* – расходы на дополнительную заработную плату исполнителей(руб.); *Ндоп* - норматив дополнительной заработной платы (14%).

*Зсоц* – расходы на социальные нужды с заработной платы(руб.); *Нсоц* – норматив отчислений на страховые взносы на обязательное социальное, пенсионное и медицинское страхование.

Все данные о расходах на заработную плату и отчисления на страховые взносы на обязательное социальное, пенсионное и медицинское страхование представлены в таблице 5.2.

*Таблица 5.2 – Данные о зарплате и социальных отчислениях*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Наименование должности | Ставка,  руб./дн. | Время работы, T, дн. | Затраты на осн. зарплату, руб. | Расходы на доп. зарплату,  руб. |
| Инженер | 809,524 | 64 | 51809,5 | 7253,26 |
| Руководитель | 2800 | 8 | 22400 | 3136 |
| Итого: | | | 74209,5 | 10389,26 |
| Норматив отчислений на социальные нужды | | | 30% | |
| Отчисления на страховые взносы на обязательное социальное, пенсионное и медицинское страхование | | | 25379,628 | |

Затраты, связанные с приобретением необходимых материалов:

*Зм* – затраты на материалы(руб.); *Gl* – норма расхода материала за единицу продукции(шт.); *Цi* – цена приобретения единицы материала (руб./ шт.); *Нт.з.* – норма транспортно-заготовительных расходов (15%).

Данные о количестве, стоимости каждого из них, общих затратах на их покупку приведён в таблице 5.3.

*Таблица 5.3 – Стоимость приобретенных материалов*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Материал | Тип | Количество | Цена за единицу, руб. | Сумма, руб. |
| Холст | для печати продукции | 1 | 200 | 200 |
| Специальные краски | Для печати продукции | 1 | 650 | 650 |
| Кисточки | Набор | 1 | 100 | 100 |
| Съемный носитель | USB-флеш-накопитель | 1 | 500 | 500 |
| Итого | | | | 1450 |
| Норма транспортно-заготовительных расходов | | | | 217,5 |
| Затраты на материалы | | | | 1667,5 |

Сторонней организацией, оказывающей услуги при выполнении работы являлся поставщик интернет-услуг. Ежемесячная плата за интернет составляет 500 руб. В день оплата составляет 16,67 руб. Во время выполнения проекта количество дней пользования услугами составляло 65 дней. Таким образом, затраты, связанные с оказанием сторонними организациями услуг за вычетом НДС, составляют *Зс.у*. = 16,67∙65-0,18∙16,67∙65 = 888,511 руб. В процессе выполнения проекта используются оборудования: ЭВМ и ПСОАТ. Основной затратой, связанной с эксплуатацией оборудования является затрата на электроэнергию.

,

*Зэо* – затраты на содержание и эксплуатацию оборудования (руб.);

*Сiмч* – расчетная себестоимость одного машино-часа работы оборудования на *i*-й технологической операции (руб./м-ч);

*tiм* – количество машино-часов, затрачиваемых на выполнение *i*-й технологической операции (м-ч).

Расчет стоимости потребления электроэнергии за час определяется как произведение мощности оборудования (кВт) на размер тарифа (2,69 руб./кВт∙ч). ЭВМ имеет мощность 0,33 кВт, принтер - 0,4 кВт. Отсюда следует, что:

*Сэвмiмч* = 0,33∙2,69 = 0,8877 руб.

*Спiмч* = 0,4∙2,69 = 1,076 руб.

Время эксплуатации оборудования, затраты на содержание и эксплуатацию оборудования на каждом этапе работы представлены в таблице 5.4.

*Таблица 5.4 – Затраты на содержание и эксплуатацию оборудования*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Этапы и содержание выполняемых работ | Время работы оборудования, ч. | | Зэо, руб. | |
| ЭВМ | ПСОАТ | ЭВМ | ПСОАТ |
| 1 | Разработка ТЗ | 12 | - | 10,6524 | 0 |
| 2 | Анализ ТЗ | 7 | - | 6,2139 | 0 |
| 3 | Изучение предметной области | 6 | - | 5,3262 | 0 |
| 4 | Консультация по работе | - | - | 0 | 0 |
| 5 | Разработка алгоритмов | 20 | - | 17,754 | 0 |
| 6 | Написание библиотек | 18 | - | 15,9786 | 0 |
| 7 | Разработка ядра программы | 8 | - | 7,1016 | 0 |
| 8 | Разработка пользовательского интерфейса | 42 | - | 37,2834 | 0 |
| 9 | Разработка управляющего ПО | 45 | - | 39,9465 | 0 |
| 10 | Интеграция управляющего ПО в состав ПСОАТ | 16 | 2 | 14,2032 | 2,152 |
| 11 | Тестирование программы | 7 | 1 | 6,2139 | 1,076 |
| 12 | Оценка полноты решения и выявление недочетов | 4 | - | 3,5508 | 0 |
| 13 | Исправление ошибок | 5 | - | 4,4385 | 0 |
| 14 | Оформление пояснительной записки | 32 | - | 28,4064 | 0 |
| Итого: | | 222 | 3 | 197,0694 | 3,228 |

Общая Зэо = 200,2974 руб.

Амортизационные отчисления по основному средству i за год определяются как:

*Ai*– амортизационные отчисления за год по *i*-му основному средству (руб.); *Цпнi*– первоначальная стоимость *i*-го основного средства (руб.); *Hai* - годовая норма амортизации *i*-го основного средства (%).

Величина амортизационных отчислений по *i*-му основному средству, используемому студентом при работе над ВКР (при условии, что время выполнения измеряется в месяцах), определяется по формуле:

где *Aiвкр1* - амортизационные отчисления по *i*-му основному средству, используемому студентом в работе над ВКР (руб.); *Tiвкр1* - время, в течение которого студент использует *i*-ое основное средство (мес.).

Среднее количество рабочих дней в году 21, средняя продолжительность рабочего дня 8 ч. Тогда амортизационные отчисления по *i*-му основному средству, используемому студентом при работе над ВКР (при условии, что время выполнения измеряется в часах):

где *Aiвкр2*– амортизационные отчисления по *i*-му основному средству, используемому студентом в работе над ВКР (руб.) за период выполнения работы; *Tiвкр2* – время, в течение которого студент использует *i*-ое основное средство (часов). Данные об амортизационных отчислениях приведены в таблице 5.5.

*Таблица 5.5 – Амортизационные отчисления*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Наименование оборудования | Кол-во, ед. | Первоначальная стоимость, руб. | Время работы, ч. | Ai, руб. | AiВКР2, руб. |
| Lenovo U410 | 1 | 21090 | 222 | 6959,7 | 766,4 |
| Контроллер робота-манипулятора | 1 | 4300 | 3 | 1419 | 2,1 |
| Итого |  | 25390 | 225 | 8378,7 | 768,5 |

Накладные расходы определяются как:

где *kпр* – процент на прочие затраты (15%).

Таким образом, *Рпр* = 0,15 \* 113503,1964 = 17025,48 руб.

Для расчета совокупной величины затрат, связанных с проведением исследования, все данные помещены в таблицу 5.6.

*Таблица 5.6 – Капитальные затраты*

|  |  |  |
| --- | --- | --- |
| № | Статья затрат | Сумма(руб.) |
| 1 | Основная заработная плата | 74209,5 |
| 2 | Дополнительная заработная плата | 10389,26 |
| 3 | Страховые взносы | 25379,628 |
| 4 | Затраты на материалы | 1667,5 |
| 5 | Услуги сторонних организаций | 888,511 |
| 6 | Затраты на содержание и эксплуатацию | 200,2974 |
| 7 | Амортизационные отчисления | 768,5 |
| 8 | Накладные расходы | 17025,48 |
| Итого затрат: | | 130528,6764 |

Затраты на оформление ИП и авторского права составляют:

Инвестиционные затраты составляют:148428,68 руб.

Финансирование данного проекта планируется за счет кредита в размере 300000 рублей на 2 года. Процентная ставка составляет 14%. В месяц необходимо выплачивать 14250 рублей. Затраты на рекламу: 15000 рублей в месяц. Так как продукт нуждается в поддержке, планируется, что программист будет заниматься поддержкой и разработкой обновлений 2 дня в неделю по 8 часов, то есть 9 дней в месяц. Руководитель будет консультировать сотрудника раз в неделю, то есть 4 дня в месяц. Посчитаем заработную плату, дополнительную заработную плату, отчисления на социальные нужды и социальные взносы за 1 месяц:

руб.

Все данные о расходах на заработную плату и отчисления на страховые взносы на обязательное социальное, пенсионное и медицинское страхование представлены в таблице 5.7.

*Таблица 5.7. – Данные о зарплате и социальных отчислениях за месяц*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Наименование должности | Ставка,  руб./дн. | Время работы, T, дн. | Затраты на осн. зарплату, руб. | Расходы на доп. зарплату,  руб. |
| Инженер |  | 9 | 7285,716 | 1020 |
| Руководитель | 2800 | 4 | 11200 | 1568 |
| Итого: | | | 18485,7 | 2588 |
| Норматив отчислений на социальные нужды | | | 30% | |
| Отчисления на страховые взносы на обязательное социальное, пенсионное и медицинское страхование | | | 6322.11 | |

Программист, работая в месяц 72 часа, будет использовать ПСОАТ примерно 1.5 часа в месяц, а ЭВМ примерно 70 часов.

Затраты на содержание и эксплуатацию оборудования:

Также нужно учитывать затраты на холст и затраты на краску для рисования. холст будет стоить 250 рублей в месяц, краска - 400. Итого 650 руб. Затраты, связанные с оказанием сторонними организациями услугами, составляют 500 руб. (плата за интернет в месяц). Амортизационные отчисления в год составляют 8378,7 руб., а в месяц – 698,225 руб.

Накладные расходы в месяц:

Данные о расходах за месяц (без учета кредита) представлены в таблице 5.8.

*Таблица 5.8 - Текущие расходы в месяц и в год*

|  |  |  |  |
| --- | --- | --- | --- |
| № | Статья затрат | Сумма(руб.) /мес. | Сумма(руб.)/год. |
| 1 | Основная заработная плата | 18485,7 | 221828,4 |
| 2 | Дополнительная заработная плата | 2588 | 31056 |
| 3 | Страховые взносы | 6322.11 | 75865,32 |
| 4 | Затраты на материалы | 650 | 7800 |
| 5 | Услуги сторонних организаций | 500 | 6000 |
| 6 | Затраты на содержание и эксплуатацию | 63,753 | 765,036 |
| 7 | Амортизационные отчисления | 698,225 | 8378,7 |
| 8 | Накладные расходы | 4396,1682 | 52754,0184 |
| 9 | Затраты на рекламу | 15000 | 180000 |
| Итого затрат: | | 48703,9562 | 584447,4744 |

Проанализировав рынок составим следующий план продаж на ближайшие 5 лет. В первый год, в силу того, что мы не будем сильно прорекламированы, планируется продать 50 работ. Далее, став более популярными можно распространить 75 и 110 во 2 и 3 годы. В 4 году продажи достигнут пика, а в пятом продажа продукта снизится в силу того, что часть рынка будет заполнена. В зависимости от объема продаж была рассчитана выручка за каждый год. Данные плана продаж за ближайшие 5 лет сведены в таблицу 1.

*Таблица 5.9 – План продаж*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Год | 1 | 2 | 3 | 4 | 5 |
| Количество проданных экземпляров | 50 | 75 | 110 | 125 | 90 |
| Выручка | 1000000 | 1500000 | 2200000 | 2500000 | 1800000 |

Согласно полученным данным выручка в первый год составляет - 1000000руб., во второй – 1500000 руб., в третий – 2200000 руб., в четвёртый – 2500000 руб., в пятый – 1800000 руб. Рассчитаем прибыль за каждый год, который рассчитывается по формуле:

где *П*-прибыль, *Зт*-затраты, *Н*-налоги.

Объект налогообложения – доход (по упрощенной системе). Ставка налога – 6%. Данные о прибыли за каждый год представлены в таблице 5.10:

*Таблица 5.10 - Планируемая прибыль за ближайшие 5 лет*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Год | 1 | 2 | 3 | 4 | 5 |
| Выручка | 1000000 | 1500000 | 2200000 | 2500000 | 1800000 |
| Затраты | 584447,4744 | 584447,4744 | 584447,4744 | 584447,4744 | 584447,4744 |
| Налог | 60000 | 90000 | 132000 | 150000 | 108000 |
| Кредит | 171000 | 171000 | 0 | 0 | 0 |
| Прибыль | 184552.5256 | 654552.5256 | 1483552.5256 | 1765552.5256 | 1107552.5256 |

По данным из этой таблицы видно, что хорошую прибыль мы начнем получать, начиная с третьего года продаж. Это связано с тем, что в первые 2 года нам необходимо погасить кредит, который был взят нами на финансирование проекта. Как только кредиты закончатся - прибыль резко увеличится.

Построим график изменения прибыли по годам:

Рисунок 6 – График изменения прибыли

Данные о движении денежных средств представлены в таблице 5.11.

Таблица 5.11 – План движения денежных средств

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Показатель** | **0** | **1-й год** | **2-й год** | **3-й год** | **4-й год** | **5-й год** |
| Поступление средств | 300 000 | 1000000 | 1500000 | 2200000 | 2500000 | 1800000 |
| Выбытие средств | 130528.6764 | 815447.4744 | 674447.4744 | 716447.4744 | 734447.4744 | 476447.4744 |
| Чистый денежный поток | 169471.3236 | 184552.5256 | 825552.5256 | 1483552.5256 | 1765552.5256 | 1323552.5256 |

Чистая текущая стоимость проекта:

руб.

Показатель рентабельности инвестиций:

*ROI* > 100% – это говорит о том, что инвестиции окупились. Доход превышает первоначальные вложения более, чем в 7 раз.

Период окупаемости инвестиций:

*DPP* = 8,5 месяцев

Таким образом, из расчетов экономической эффективности инвестиционного проекта получены следующие результаты, приведенные в таблице 5.12.

Таблица 5.12 – Показатели экономической эффективности проекта

|  |  |
| --- | --- |
| **Показатели** | **Значение** |
| Чистая текущая стоимость проекта *NPV*, руб. | 3302055.18 |
| Показатель рентабельности инвестиций *ROI* | 7.35 |
| Период окупаемости *DPP,* мес. | 8,5 |

Исходя из рассчитанных выше показателей, можно сделать вывод о том, что проект является эффективным.

* **Заключение**
* **Список литературы**
* **ПРИЛОЖЕНИЕ А**

Исходный код программного модуля