ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

УТВЕРЖДАЮ
Академический руководитель
образовательной программы «Программная инженерия»,
профессор департамента программной инженерии, канд. техн. наук

_____ В.В. Шилов
«___» _____ 2018 г.

Выпускная квалификационная работа

на тему «Android 2D графический редактор с «жидкой» палитрой»

по направлению подготовки 09.03.04 «Программная инженерия»

Научный руководитель
доцент департамента
программной инженерии
факультета компьютерных наук
кандидат технических наук

Р. З. Ахметсафина

_____
Подпись, Дата

Выполнил
студент группы БПИ143
4 курса бакалавриата
образовательной программы
«Программная инженерия»

А. С. Варгулёв

_____
Подпись, Дата

Москва 2018

АННОТАЦИЯ

Современные графические редакторы предоставляют огромный инструментарий для создания художественных изображений, однако пока они способны симулировать далеко не все возможности, доступные в реальных рисовании и живописи. Хотя графические редакторы уже успешно имитируют большое количество различных инструментов рисования, они всё ещё не предоставляют такого контроля над цветом, как физические палитры, в том числе возможность создавать пользовательские многоцветные градиенты. Несмотря на то, что редкие экспериментальные решения существуют, ни одно из них не поддерживает набирающий популярность Android – операционную систему, наиболее распространённую среди планшетных компьютеров, которые по сути являются идеальным устройством для цифровых рисования и живописи. Данная работа в основном сосредоточена на имитации «жидкой» палитры и смешивании цветов.

Целью данной работы является разработка Android-приложения для рисования, симулирующего реальную палитру для смешивания цветов и адаптированного к характеристикам современных планшетных компьютеров. Ожидается, что оно улучшит результаты работы художников.

Программа реализована на языке Java с использованием графического интерфейса OpenGL ES 2.0.

Данная работа содержит 39 страниц, 3 главы, 13 рисунков и 19 источников.

Ключевые слова: графические пользовательские интерфейсы, изоповерхности, java, android, opengl.

ABSTRACT

As advanced as they are nowadays, modern graphics editors are yet to simulate all the options available in real-life painting and drawing. While successfully imitating many different painting tools, they still lack the control over colour provided by physical palettes, including the ability to create custom multi-colour gradients. Although rare experimental solutions do exist, none of them targets the increasingly popular Android – a mainstream OS among tablet PCs, that are a basically ideal device for digital drawing and painting. The study mainly focuses on liquid palette imitation and colour mixing.

The aim of this work is developing an Android drawing application that simulates an actual color mixing palette and is adapted to hardware characteristics of modern tablet PCs. Presumably, it will improve the artists' drawing experience.

The programme is implemented in Java programming language with use of OpenGL ES 2.0 graphical interface.

This paper contains 39 pages, 3 chapters, 13 illustrations and 19 references.

Keywords: graphical user interfaces, isosurfaces, java, android, opengl.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

- **GPU** – процессор видеокарты.
- **GUI** – графический пользовательский интерфейс (Graphical User Interface)
- **HSB** – цветовая модель, в которой координатами цвета являются тон (Hue), насыщенность (Saturation) и яркость (Brightness).
- **HSL** – цветовая модель, в которой координатами цвета являются тон (Hue), насыщенность (Saturation) и светлота (Lightness).
- **HSV** – см. HSB.
- **RGB** – цветовая модель, основанная на смешении красного (Red), зелёного (Green) и синего (Blue) цветов.
- **RYB** – цветовая модель, основанная на смешении красного (Red), жёлтого (Yellow) и синего (Blue) цветов.
- **Аддитивный синтез цветов** – метод синтеза цвета, основанный на сложении цветов излучающих объектов.
- **Аддитивно-усреднённый синтез цветов** – метод синтеза цвета, основанный на сложении с усреднением цветов излучающих объектов.
- **Базовый цвет** – (в рамках данной работы) один из смешиваемых цветов при смешении.
- **Градиент** – плавный переход от одного цвета к другому.
- **Диффузное отражение** – отражение света, падающего на поверхность, под углом, отличным от угла падения.
- **Субтрактивный синтез цветов** – метод синтеза цвета, основанный на вычитании спектральных составляющих цветов из белого.
- **Шейдер** – программа, исполняемая GPU.

ВВЕДЕНИЕ

С тех пор, как в начале семидесятых появилось первое ПО для работы с графикой [1], набор доступных художнику инструментов непрерывно развивается и пополняется. Одним из главных направлений разработки при программировании универсальных и предназначенных для рисования графических редакторов является симуляция реальных физических инструментов. В основном это кисти [2] и фильтры [3], имитирующие поведение определённых красок, карандашей, различные техники рисования, и таким образом позволяющие создавать правдоподобные цифровые рисунки.

Однако даже самое современное ПО едва ли может в полной мере имитировать процесс реального рисования – он слишком сложен. Это в том числе относится и к смешиванию цветов, которое является одним из ключевых элементов в реальной живописи, однако в графических редакторах заменено на практичные, но ограниченные селекторы цветов. Несмотря на то, что селекторы позволяют с высокой точностью выбрать желаемый цвет, они не предоставляют возможности выбрать плавный настраиваемый переход от одного цвета к другому – градиент – для взятия оттенков из него. Это вынуждает некоторых художников вручную создавать импровизированные палитры, что довольно просто сделать для двух цветов, но намного сложнее для трёх и более.

Недавние эксперименты по созданию цифровой палитры для смешивания цветов [4] показали, что она может послужить внушительным инструментом, совмещая в себе ощущение и возможности физического смешивания цветов с искуственными скоростью, плавностью и точностью. Кроме того, возможность ведения истории изменений делает изменения недеструктивными, открывая огромные возможности по обработке изображения, например перекраску всего рисунка в новую гамму при помощи изменения цвета всего лишь одной небольшой части палитры.

Несмотря на успех исследования и эксперимента, на данный момент существует всего одна реализация такой палитры. Более того, она представляет собой исключительно экспериментальное приложение для проведения исследования, реализованное только для платформы iOS.

Исследование рынка [5] показывает, что ОС Android на планшетных компьютерах примерно вдвое более популярна, чем iOS. В связи с этим, **целью** данной работы является разработка графического редактора для Android, снабжённого «жидкой» палитрой для смешивания цветов.

Хотя общая концепция палитры и всего приложения основывается на работе Adobe Research [4], графический интерфейс приложения ориентирован на средние характеристики современных планшетных компьютеров на платформе Android. Главным образом учитываются размеры дисплея и совместимых с сенсорным экраном таких устройств стилусов. Предполагается, что это приложение позволит профессиональным художникам и художникам-любителям создавать цифровые рисунки, затрачивая меньше времени на подбор оттенков для создания плавного перехода между цветами.

Для достижения поставленной цели был выделен следующий набор **задач**:

1. Изучение существующих средств виртуального смешивания цветов.
2. Определение функциональных требований к программе.
3. Определение требований к графическому интерфейсу программы в соответствии с характеристиками предполагаемого оборудования.
4. Разработка алгоритма построения и раскраски пятна на палитре.
5. Разработка архитектуры приложения.
6. Реализация алгоритма построения и раскраски пятна на палитре.
7. Реализация инструментов работы с рисунком.
8. Разработка технической документации.

Работа состоит из трёх глав. В первой главе приводится обзор существующих решений в сфере виртуального смешивания цветов. Во второй главе описываются решения в области графического интерфейса программы и ключевые алгоритмы её работы. В третьей главе приводятся особенности реализации программы. В заключении работы содержится анализ полученных результатов и перспектив дальнейшего развития программы. Также работа сопровождается приложением в виде технической документации.

ГЛАВА 1. ОБЗОР ИСТОЧНИКОВ И СУЩЕСТВУЮЩИХ РЕШЕНИЙ

Данная глава содержит обзор существующих подходов и решений в области создания виртуальной палитры и смешивания цветов.

1.1. Подходы к построению палитры

Можно выделить три основных подхода к построению палитры – симуляция реальной физической палитры, создание градиентной палитры и комбинированный (гибридный) подход. Каждый подход имеет свои преимущества и недостатки.

1.1.1. Симуляция физической палитры

Одним из направлений разработки в области смешивания цветов является симуляция поведения реальной физической палитры. Для данного подхода характерны относительно высокая сложность реализации и наследование как преимуществ реальной палитры, так и недостатков.

Высокая сложность реализации обусловлена в первую очередь необходимостью симуляции нетривиальной физики процесса смешивания. Кроме того, реальные краски основываются на химически разнородных пигментах, что приводит к несистематичным результатам отражения света и, таким образом, определения результирующего цвета при смешивании. Для получения реалистичных переходов требуется либо применять данные о физических характеристиках больших наборов отдельных красок в совокупности с аппроксимацией поведения цвета при смешивании (например, при помощи теории Кубелки-Мунка [14]), либо проводить обучение системы на результатах пользовательского выбора желаемого перехода между цветами.

К преимуществам реальной палитры в данном случае относятся её интерактивность, привычность использования и интуитивность процесса получения желаемого результата.

К недостаткам реальной палитры в данном случае относятся деструктивность изменений (ведение истории изменений лишь частично это исправляет), неравномерность цветовых переходов, затраты времени на ручное смешивание цветов и сложность точного подбора цветов для получения нужного оттенка.

Всё это делает палитры-симуляторы непрактичными в любой ситуации, кроме случаев, когда симуляция реальной палитры является самоцелью.

Данный подход применяется в приложениях IntuPaint [6], TangiPaint [7] и Paper for iPad [15]. Ни одно из этих приложений не ведёт историю изменений палитры, таким образом, любые вносимые в неё изменения деструктивны.

1.1.2. Градиентные палитры

Другим подходом является создание палитры на основе градиентов. В данном подходе симуляция поведения краски как таковой отбрасывается с целью упрощения расчётов и облегчения работы с палитрой. Подход заключается в создании палитры в виде набора градиентов между парами цветов.

К преимуществам данного подхода можно отнести высокие контроль и точность работы, а также недеструктивность изменений – любой цвет легко заменить, поскольку он остаётся представлен отдельным объектом на протяжение всей работы с палитрой.

К недостаткам данного подхода относится ограниченность результатов – получение результатов смешивания трёх и более цветов проблематично (если вообще возможно), а отображение смеси цветов не цельное – по сути, одномерное (рис. 1) – что затрудняет точный выбор желаемого оттенка.

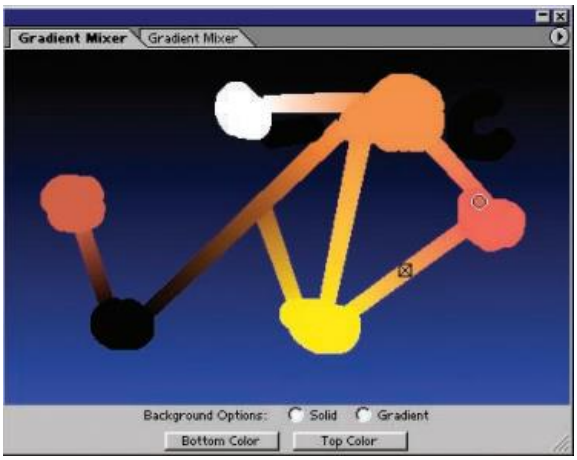


Рисунок 1 – пример градиентной палитры [8].

При таком подходе у пользователя как правило отсутствует какое-либо ощущение работы с реальной палитрой в силу отсутствия визуального сходства с ней.

Данный набор характеристик делает градиентные палитры довольно практичной альтернативой импровизированным палитрам, однако по сути просто дублирует их функционал и принципы построения.

Данный подход применяется в приложении из статьи «Interactive Color Palette Tools» [8]. В силу ручного нанесения основных цветов редактирование существующей части палитры по ходу использования в данном приложении затруднительно.

1.1.3. Гибридный подход

Два описанных выше подхода можно объединить в один, чтобы совместить преимущества реальной палитры и цифровых градиентов.

В случае гибридного подхода палитра представляет собой интерактивное пятно – результат упрощённого «слияния» нескольких капель краски.

Для привычности использования не требуется полноценная симуляция физики жидкости, в данном случае достаточно обеспечить объединение капель при сближении и геометрическую G^1-непрерывность (см. п. 1.4) контура пятна.

Использование многоцветного двумерного градиента позволяет обеспечить высокую точность и плавность при работе с цветом. Кроме того, такой градиент задаётся и настраивается относительно небольшим числом параметров, что облегчает работу с палитрой и позволяет относительно просто реализовать её интерактивность и историю изменений, что в свою очередь положительно сказывается на недеструктивности. Также отпадает необходимость в ручном смешивании, что не только гарантирует плавность переходов, но и сильно экономит время пользователя при работе с палитрой.

Основным недостатком данного подхода является то, что большинство стандартных цветовых пространств дают всё менее реалистичный (близкий к поведению реальных красок) результирующий цвет при смешивании с увеличением числа базовых цветов. Эта проблема может быть решена применением более сложных алгоритмов получения результирующего цвета (в т.ч. с применением теории Кубелки-Мунка [14]).

Единственная существующая на данный момент реализация данного подхода – Playful Palette [4]. Данное приложение является закрытым и эксклюзивным для iOS. Кроме того, его графический интерфейс ориентирован на использвание стилуса, например, работа с колесом использованных цветов (рис. 2) при большом количестве цветов затруднительна без применения стилуса, в то время как большинство современных планшетных компьютеров им не укомплектованы.

Рисунок 2 – фрагмент скриншота приложения Playful Palette с палитрой и колесом использованных цветов.

1.2. Цветовые пространства и алгоритмы смешивания

Для всех трёх подходов одним из ключевых элементов вычислений является получение результирующего цвета при смешении нескольких «базовых». Как правило, данная задача сводится к получению цвета из набора цветов с коэффициентами – весами – дающими в сумме единицу. Как уже упоминалось в п. 1.1, данная операция нетривиальна. Есть несколько основных способов её реализации.

1.2.1. Среднее арифметическое взвешенное

Одним из наиболее очевидных решений является расчёт среднего арифметического взвешенного всех цветов из набора. Однако для этого требуется выбрать представление цветов в виде чисел.

Выбор цветового пространства RGB для представления цветов является наиболее простым и быстрым с точки зрения вычислений решением – из любого другого цветового пространства финальный цвет в конечном счёте придётся переводить в RGB, поскольку дисплей отображает цвета именно в таком формате. Однако результаты, получаемые при RGB-смешивании путём усреднения значений красного, зелёного и синего каналов, далеки от таковых при работе с реальной краской – так, при смешении синего с жёлтым получается серый цвет, а не зелёный. Это и общее «загрязнение» цвета при RGB-смешивании связаны с тем, что в данном случае речь идёт об аддитивно-усреднённом синтезе цвета, в то время как реальные краски ведут себя как нечто среднее между системами с субтрактивным синтезом цвета и аддитивно-усреднённым синтезом цвета [16] (рис. 3).

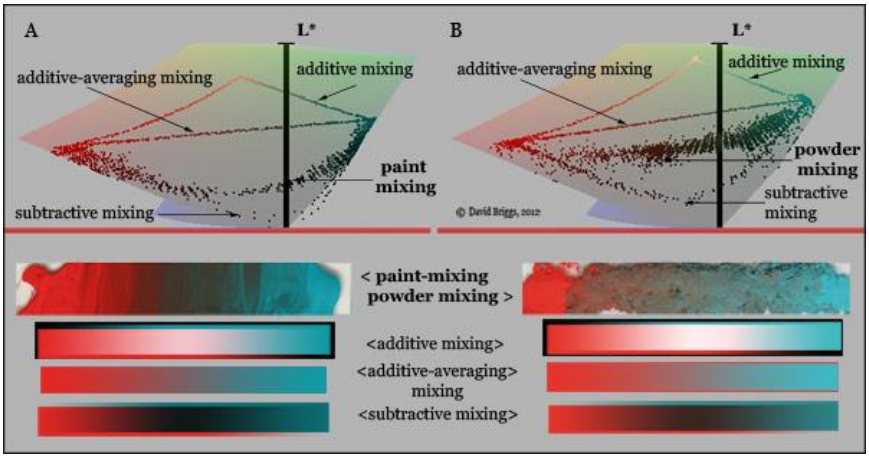


Рисунок 3 – поведение краски при смешивании в сравнении с различными системами синтеза цвета [17].

Несмотря на большую привычность и наглядность понятий, используемых для описания цвета в HSB/HSL и кажущуюся логичность плавного равномерного перехода по всем трём параметрам этого пространства, в реальности их применение приводит к ряду проблем, связанных с цикличностью параметра Hue, что делает такой подход неприменимым для трёх и более цветов.

Предположительно, наилучший результат при использовании среднего арифметического взвешенного достигается путём усреднения значений, полученных в результате вычислений с применением аддитивно-усреднённого и субтрактивного синтезов.

1.2.2. Теория Кубелки-Мунка

В качестве альтернативы среднего арифметического взвешенного для цветов в стандартных цветовых пространствах можно использовать среднее арифметическое взвешенное для значений векторов физических коэффициентов теории Кубелки-Мунка, описывающей поведение реальных красок при смешивании [14].

Основным минусом такого решения являются сложность имплементации и необходимость опираться на большой набор заранее рассчитанных величин.

Теория Кубелки-Мунка описывает связь диффузного отражения с такими параметрами материала, как рассеивание и поглощение, в случае достаточно толстого слоя материала для того, чтобы дальнейшее увеличение его толщины не влияло на результат. Применение данной теории позволяет на основе измеренных характеристик реальных

красок предсказывать цвет результата их смешения исходя из его предполагаемых характеристик.

1.2.3. Обучение системы на основе статистических данных

Вместо того, чтобы опираться на физические показатели отдельных цветов, можно использовать алгоритмы машинного обучения на основе данных, составляемых человеком. Такой подход применяется в Paper for iPad [15] – для обучения системы было составлено 100 пар цветов, для каждой из которых каждому человеку из группы опрашиваемых людей предлагалось выбрать один из нескольких переходов между цветами, кажущийся ему наиболее естественным. При смешивании цветов программа строит переход по кривой внутри цветового пространства, схожей с таковой для наиболее близкой пары.

Главным недостатком такого решения являются большие трудозатраты на имплементацию, однако его результаты близки к тому, что пользователь по статистике воспринимает, как естественное.

1.3. Обоснование необходимости разработки программного решения

Исходя из преимуществ и недостатков описанных подходов можно заключить, что гибридный подход является наиболее подходящим для приложений, в которых не ставится цель точно симулировать реальный процесс рисования. Однако в рамках этого подхода на данный момент существует всего одно приложение, ограниченное не самой популярной среди планшетных компьютеров ОС. Кроме того, оно плохо подходит для работы при отсутствии стилуса. Также следует учесть высокую популярность графических редакторов для планшетных компьютеров с ОС Android, что демонстрирует их актуальность.

Таким образом, создание аналога данной программы для ОС Android с графическим интерфейсом, адаптированным под работу без стилуса на в среднем меньших устройствах [9] можно считать перспективным.

1.4. Используемый алгоритм рисования пятна

В единственном существующем приложении, применяющем гибридный подход – Playful Palette – для рисования и раскраски пятна используется неравенство, основанное на работе Дж. Ф. Блинна о металлических [12].

В данной работе также применяется алгоритм, основанный на металлических, однако он отличается от использованного в Playful Palette.

Алгоритм и понятие меташара подробно описываются в главе 2 данной работы.

Данный алгоритм был выбран в силу большого числа преимуществ перед также рассмотренными алгоритмами на основе дуг окружностей (с использованием алгоритма Брезенхема), лемнискат и совмещения дуг окружностей с кривыми Безье, а также в силу соответствия всем главным требованиям.

К числу главных требований к алгоритму относятся геометрическая G^1-непрерывность контура получаемой фигуры и возможность разделения пятна на отдельные пятна путём достаточного удаления капель друг от друга (кривая называется G^1-непрерывной, если вдоль этой кривой единичный вектор её касательной изменяется непрерывно).

Основным преимуществом данного алгоритма является независимость расчётов, производимых для каждого пикселя, что обеспечивает высокий уровень параллелизма. Распараллеливание алгоритма, применяющегося для большого количества отдельных пикселей, является ключевым фактором повышения его быстродействия, которое необходимо для возможности работы с палитрой в реальном времени. Также алгоритм позволяет использовать полученные в ходе его вычислений данные для того, чтобы одновременно с принадлежностью каждого пикселя к пятну определить его результирующий цвет и выполнить сглаживание (anti-aliasing) пятна на этапе определенения его формы.

При смешивании цветов используется один из трёх алгоритмов смешивания цветов на выбор пользователя. Подробно алгоритмы описаны в главе 2 данной работы.

1.5. Используемые средства реализации алгоритма рисования пятна и графического интерфейса программы

В силу того, что вычисления алгоритма можно проводить для каждого пикселя независимо, для максимизации параллелизма алгоритма и быстродействия программы было решено реализовывать алгоритм в виде фрагментного шейдера на языке GLSL, отображая его средствами библиотеки OpenGL SE версии 2.0. При этом для минимизации затрат на разработку архитектуры и общего графического интерфейса программы для большинства прочих компонентов программы OpenGL SE было решено не применять, Вместо этого используя встроенные возможности Android API.

Таким образом, OpenGL SE применяется только в отдельных компонентах (палитра) посредством использования класса GLSurfaceView.

Реализующий алгоритм фрагментный шейдер также выполняет подсветку границ отдельных капель в некоторых режимах работы с палитрой и отображение курсора выбора цвета. Для повышения быстродействия все алгоритмы в нём реализованы без использования операции взятия квадратного корня.

1.6. Используемые решения в области графического пользовательского интерфейса

Исходя из целевой ОС, предполагаемый планшетный компьютер, на котором будет работать программа, обладает следующими характеристиками [9]:

- 21-22 см дисплей (среднее значение ~21.67 см / 8.27”);
- соотношение сторон дисплея 5:8;
- ёмкостный сенсорный экран;
- отсутствие стилуса.

В силу меньших по сравнению с iOS-планшетами размеров (~21.67 см против ~24.87 см), отсутствия стилуса, а также ёмкостного сенсорного экрана, требующего большую площадь нажатия [10], элементы интерфейса должны быть достаточно крупными для того, чтобы пользователь мог без проблем работать при помощи пальцев.

В силу более вытянутого дисплея (5:8 против 3:4), чем у iOS-планшетов, имеет смысл располагать большую часть элементов управления вдоль одной или обеих узких сторон экрана.

Таким образом, решено было отказаться от реализации панели использованных цветов в виде колеса, как это было в Playful Palette (рис. 1), вместо этого реализовав её в виде прокручивающейся ленты.

Кроме того, было принято решение снабдить полями, показывающими цвет, на который пользователь в данный момент навёл курсор, палитры и селекторы цветов (поскольку курсор и выбранный цвет будут скрыты пальцем пользователя).

1.7. Функциональные требования к программе

Программа должна обеспечивать возможность выполнения перечисленных ниже функций:

1) Работа с рисунком:
   – нанесение мазков инструментом «кисть»;
   – нанесение мазков инструментом «ластик»;

– изменение цвета фона (холста);
– прокрутка и масштабирование холста;

2) Работа с файлами:
   – создание нового рисунка;
   – сохранение результатов работы в виде изображения;
   – открытие изображений для использования их в качестве холста;
3) Инструменты рисования:
   – инструмент «кисть» настраиваемых размера, цвета и прозрачности для нанесения мазков;
   – инструмент «ластик» настраиваемого размера для удаления частей мазков с холста;
   – инструмент для выбора цвета с помощью селектора цвета;
4) Работа с историей изменений:
   – ведение истории изменений рисунка как набора мазков;
   – отмена внесённых изменений;
   – повторное внесение недавно отменённых изменений;
5) Работа с палитрой:
   – добавление новых капель на палитру с выбором цвета при помощи селектора цвета;
   – удаление капель с палитры;
   – изменение цвета существующих капель при помощи селектора цвета;
   – изменение размера существующих капель;
   – изменение положения существующих капель;
   – взятие цвета для рисования с палитры;
   – обозначение границ капель при работе с палитрой;
6) Работа с панелью использованных цветов:
   – автоматическое добавление использованных в рисовании цветов на панель;
   – прокрутка панели;
   – взятие цвета для рисования с панели.

Программа должна поддерживать работу с изображениями в формате JPEG.

В данной главе были рассмотрены существующие подходы к построению палитры и смешиванию цветов, обоснованы и сформулированы требования к программе, а также выбраны основные подходы, алгоритмы и средства их реализации.

ГЛАВА 2. АЛГОРИТМЫ СОЗДАНИЯ ЖИДКОЙ ПАЛИТРЫ И GUI-РЕШЕНИЯ

Данная глава содержит алгоритм построения пятна на палитре и описание решений в области графического пользовательского интерфейса.

2.1. Алгоритм построения пятна

Для построения пятна в программе используется видоизменённое неравенство меташара, основанное на работе Дж. Блинна [12].

2.1.1. Меташары

Ключевым понятием алгоритма является меташар. Меташар в общем случае является n-мерной изоповерхностью – поверхностью, проходящей через все точки с одинаковым значением какой-то величины – определённой n-мерной функцией и пороговым значением. Меташары обладают свойством геометрической G^1-непрерывности, также известным, как математическая гладкость, и конечности.

Меташары могут быть использованы для создания сложных гладких объектов в пространстве, моделирования молекул и электронных зарядов, пузырей, капель и других объектов и явлений.

На практике чаще всего используются трёхмерные меташары, которые обычно задаются следующим неравенством (или его более быстрыми в вычислении аналогами):

$$\sum_{i=1}^{k} \frac{1}{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2} \geq F_0,$$

где i – индекс одного из центров меташара, x_i, y_i и z_i – координаты данного центра, k – число центров, F_0 – пороговое значение.

Важным аргументом в пользу использования меташаров в двумерной графике заключается в том, что для построения достаточно проверить истинность неравенства для каждого пикселя, что позволяет в силу независимости таких проверок распараллелить процесс, в том числе с использованием GPU, а это резко повышает быстродействие.

2.1.2. Определение принадлежности точки к пятну

Двумерный меташар может быть задан следующим неравенством по аналогии с приведённым выше:

$$\sum_{i=1}^{k} \frac{1}{(x-x_i)^2+(y-y_i)^2} \geq F_0$$

Заметим, что при $F_0 = 1$ изоповерхность – в данном случае, кривая на плоскости – будет проходить вблизи k окружностей радиуса 1 с центрами (x_i, y_i), при этом «переходя» от одной окружности к другой в местах, где окружности распложены близко друг к другу, визуально соединяя их в единое пятно при достаточной кучности.

Это позволяет использовать данное неравенство для построения меташара, имитирующего слияние капель жидкости (напр., краски) равного радиуса. Несложная модификация неравенства позволяет также выставлять радиусы отдельных капель:

$$\sum_{i=1}^{k} \frac{{r_i}^2}{(x - x_i)^2 + (y - y_i)^2} \geq 1, \quad (1)$$

где r_i – радиус i-й капли.

На рис. 4 приведён пример построения пятна с 4 центрами по данной формуле в онлайн-сервисе Desmos [13]. Также на нём изображены окружности с теми же центрами и радиусами.

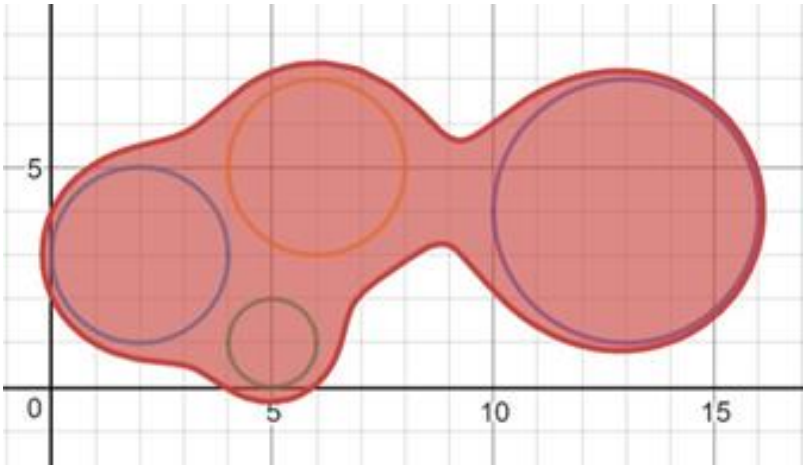


Рисунок 4 – пятно с 4 центрами.

Данное неравенство позволяет также реализовать сглаживание (антиалиасинг), задавая прозрачность пикселя в соответствии с тем, насколько значение левой части неравенства близко к единице – чем меньше значение, тем прозрачнее пиксель.

В программе для рисования пятна реализовано именно такое неравенство.

2.1.3. Определение результирующего цвета точки

В каждой точке – т.е. для каждого пикселя – для расчёта результирующего цвета рассчитываются веса базовых цветов, после чего они смешиваются при помощи одного из трёх алгоритмов на выбор пользователя, соответствующих трём режимам смешивания цветов на палитре в программе – «RGB», «Advanced RGB», «Advanced RYB» (рис. 5).

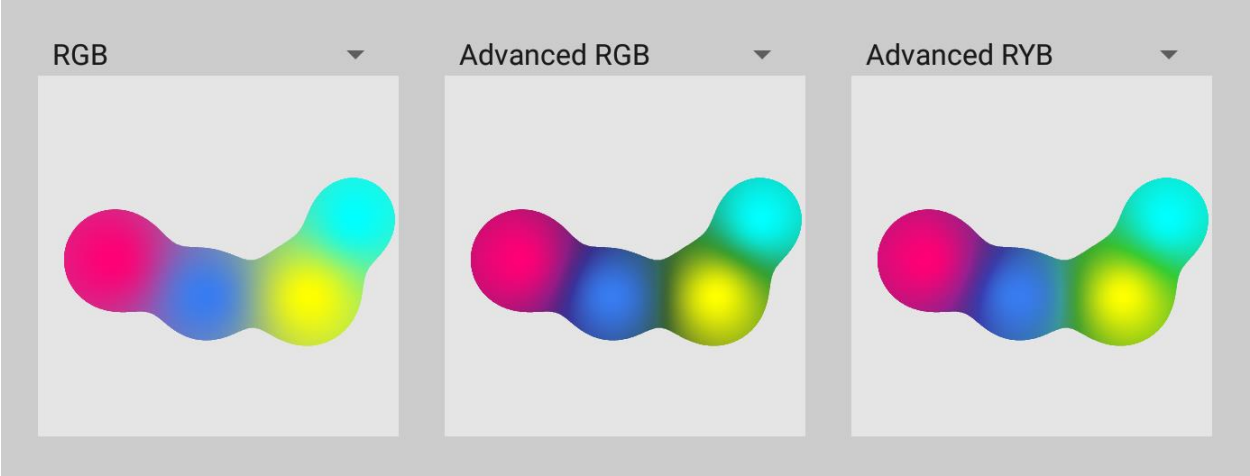


Рисунок 5 – различные режимы смешивания цветов.

После расчёта всех весов веса нормируются – делятся на их сумму – затем смешивание происходит попарно: для первых двух цветов применяется алгоритм смешивания с предварительным расчётом весов внутри пары, результирующий цвет получает вес, равный суммарному весу смешанных цветов. Затем этот цвет смешивается аналогично со следующим, и процесс повторяется до тех пор, пока все базовые цвета не будут смешаны в один результирующий. Это позволяет разрабатывать и реализовывать алгоритмы смешивания для двух цветов, а не для произвольного числа цветов.

2.1.3.1. Определение весов базовых цветов

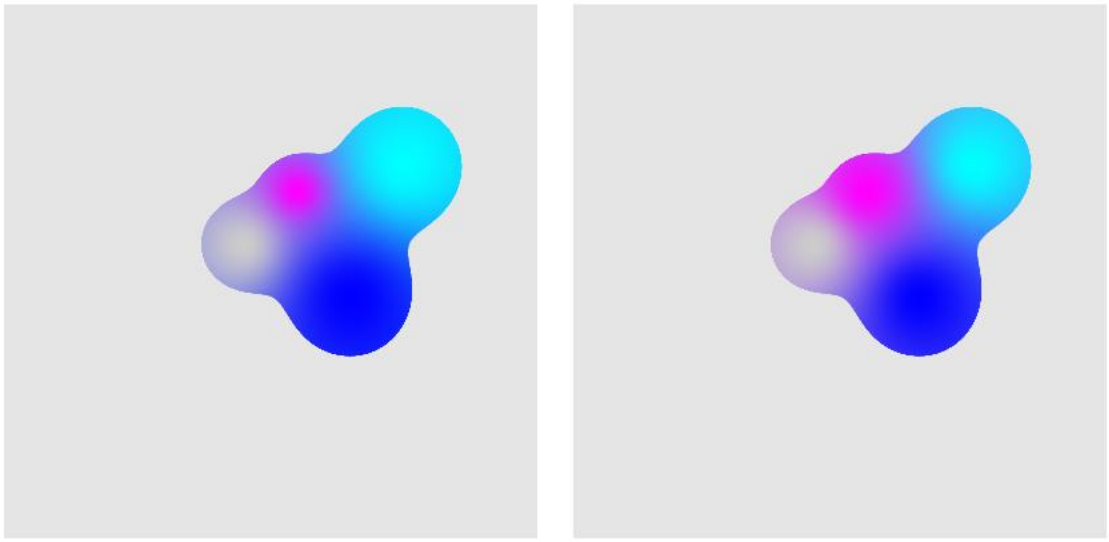
При использовании элементов суммы неравенства (1) в качестве весов для цветов область видимого смешивания получается недостаточно большой для удобного использования. Кроме того, по краям пятна происходит загрязнение усреднённым цветом, так как веса базовых цветов всех капель становятся слишком близки друг к другу. Это также отрицательно сказывается на правдоподобности поведения палитры и удобстве использования. Последний эффект особо заметен на сравнительно небольших каплях.

Эмпирически была получена модификация формулы веса, в достаточной мере компенсирующая приведённые выше недостатки:

$$wt_i = \frac{(r_i + c)^2}{(x - x_i)^2 + (y - y_i)^2},$$

где c – константа, зависящая от размера палитры и, соответственно, диапазона используемых радиусов капель.

Для обеспечения возможности простой настройки размеров палитры в коде программы, вычисления производятся в системе координат, в которой точке (0, 0) соответствует левый нижний угол квадратной палитры, а точке (1, 1) – правый верхний угол. В таких условиях было подобрано оптимальное значение константы $c = 5$ (рис. 6).



а) б)

Рисунок 6 – палитра: а) с использованием неизменённых весов; б) с использованием изменённых весов с $c = 5$.

2.1.3.2. Алгоритм режима смешивания «RGB»

Режим смешивания «RGB» – наиболее быстрый в расчётах и простой в имплементации режим, реализованный в программе.

Для результирующего цвета значения r, g и b (красного, зелёного и синего канала соответственно) рассчитываются как среднее арифметическое взвешенное для пар соответствующих значений базовых цветов.

Среднее арифметическое взвешенное n элементов определяется как:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i p_i}{\sum_{i=1}^{n} p_i},$$

где x_i – i-й элемент, а p_i – его вес.

2.1.3.3. Алгоритм режима смешивания «Advanced RGB»

При смешивании реальные краски ведут себя ближе всего к чему-то среднему между субтрактивным и аддитивно-усреднённым синтезом цветов [17]. Данный режим смешивания грубо имитирует подобное поведение – поведение реальных красок сильно зависит от конкретных цветов и конкретных пигментов, т.е. по сути в определённой мере беспорядочно.

Сначала базовые цвета, подаваемые на вход в формате RGB с диапазоном значений [0; 1] для каждого канала, переводятся в формат HSB с диапазоном значений [0; 1] для каждой из трёх характеристик H $(Hue), S$ $(Saturation), B$ $(Brightness)$ [18].

Затем производится переход от полярной системы координат с $2\pi H$ в качестве угла и B в качестве радиуса при константном S к декартовой, в которой вычисляются координаты предварительного цвета А – субтрактивной составляющей результирующего цвета – как среднее арифметическое взвешенное координат базовых цветов. Координаты переводятся обратно в полярную систему координат, давая значения Hue и $Brightness$ цвета А. Значение $Saturation$ вычисляется как среднее арифметическое взвешенное таковых у базовых цветов. После этого предварительный цвет А переводится в формат RGB.

Предварительный цвет Б – аддитивно-усреднённая компонента результирующего цвета – рассчитывается в RGB. Его значения r, g и b (красного, зелёного и синего канала соответственно) рассчитываются как среднее арифметическое взвешенное для пар соответствующих значений базовых цветов.

Результирующий цвет получается путём смешивания предварительных цветов в RGB (по аналогии с расчётом предварительного цвета Б), при этом в качестве весов используются значения 0.6 и 0.4 соответственно, подобранные на основании данных о поведении реальной краски [18].

2.1.3.4. Алгоритм режима смешивания «Advanced RYB»

Как и алгоритм режима смешивания «Advanced RGB», данный режим имитирует гибрид субтрактивного и аддитивно-усреднённого синтеза цветов. Он также имитирует

работу в цветовом пространстве RYB, исторически закрепившемся среди художников несмотря на несоответствие современным представлениям о восприятии цвета человеком. Кроме того, алгоритм данного режима даёт более правдоподобный результат для некоторых пар цветов.

Сначала базовые цвета, подаваемые на вход в формате RGB с диапазоном значений [0; 1] для каждого канала, переводятся в формат HSB с диапазоном значений [0; 1] для каждой из трёх характеристик $H\ (Hue), S\ (Saturation), B\ (Brightness)$ [18].

После этого производится нелинейное преобразование цветов по Hue, заданное следующей формулой:

$$H_{RYB} = \begin{cases} 2H, & H \leq \frac{1}{6}; \\ \frac{2}{3}H + \frac{2}{9}, & \frac{1}{6} < H \leq \frac{2}{3}; \\ H, & H > \frac{2}{3} \end{cases}$$

Данное преобразование устанавливает на шкале Hue жёлтый в точку $\frac{1}{3}$ – до преобразования данному значению соответствует зелёный – «заменяя» цветовое пространство на RYB.

Затем производится переход от полярной системы координат с $2\pi H$ в качестве угла и B в качестве радиуса при константном S к декартовой, в которой вычисляются координаты предварительного цвета А – субтрактивной составляющей результирующего цвета – как среднее арифметическое взвешенное координат базовых цветов. Координаты переводятся обратно в полярную систему координат, давая значения Hue и $Brightness$ цвета А. Значение $Saturation$ вычисляется как среднее арифметическое взвешенное таковых у базовых цветов. После этого производится обратное преобразование (от цветового пространства RYB к цветовому пространству RGB), а затем предварительный цвет А переводится из формата HSB в формат RGB.

Предварительный цвет Б – аддитивно-усреднённая компонента результирующего цвета – рассчитывается в RGB. Его значения r, g и b (красного, зелёного и синего канала соответственно) рассчитываются как среднее арифметическое взвешенное для пар соответствующих значений базовых цветов.

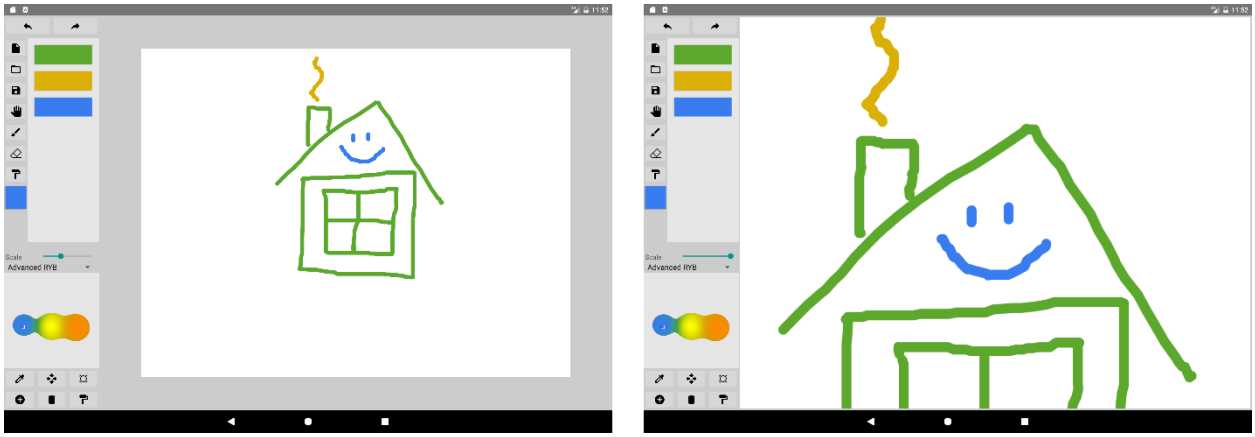
Результирующий цвет получается путём смешивания предварительных цветов в RGB (по аналогии с расчётом предварительного цвета Б), при этом в качестве весов используются значения 0.6 и 0.4 соответственно, подобранные на основании данных о поведении реальной краски [18].

2.2. Решения в области графического пользовательского интерфейса

Среди требований к программному продукту значится ряд требований к графическому пользовательскому интерфейсу, обусловленных предположительными характеристиками устройства и условиями использования – например, возможностью как наличия, так и отсутствия стилуса. В данном разделе приводятся конкретные решения в области графического пользовательского интерфейса, разработанные для максимального соответствия требованиям и максимального удобства эксплуатации программы.

2.2.1. Холст

Прямоугольный холст (рис. 7) по умолчанию создаётся такого размера, чтобы занимать всё не занятое панелью инструментов место. Это позволяет работать с ним целиком без необходимости прокрутки при масштабе 100%, который в ходе тестирования проявил себя как оптимальный.



а) б)

Рисунок 7 – скриншоты приложения с масштабом холста: а) менее 100%; б) более 100%.

Холст может иметь произвольные пропорции и за счёт этого при открытии изображения его пропорции подстраиваются под любое изображение. При этом из соображений производительности и исходя из диапазонов толщины кисти и ластика слишком большое изображение будет уменьшено до размеров, сопоставимых по площади с размерами холста по умолчанию с сохранением исходных пропорций.

Диапазон масштаба холста составляет от 10% до 210%. Данный диапазон также проявил себя как оптимальный как с точки зрения использования (с учётом возможных пропорций открываемого изображения), так и с точки зрения нагрузки на устройство

исходя из особенностей реализации функции масштабирования холста. Масштабирование выполняется одним пальцем при помощи слайдера инструмента «Pan», чтобы обеспечить возможность изменять масштаб при помощи стилуса.

Цвет холста может быть изменён в любой момент при помощи инструмента заливки холста. При этом открываемое изображение не заменяет собой цвет холста, и при помощи ластика его части можно стереть.

При рисовании на холсте применяется сглаживание.

2.2.2. Панель инструментов

Панель инструментов (рис. 8) состоит из кнопок отмены и повторения действия – операций с историей изменений, кнопок создания нового, открытия существующего и сохранения текущего изображения, кнопок переключения между инструментами, кнопки заливки холста, поля текущего цвета, набора слайдеров для настройки инструментов, состав которого зависит от текущего инструмента, панели использованных цветов, палитры, выпадающего списка для переключения режима смешивания цветов на палитре и кнопок операций с палитрой.

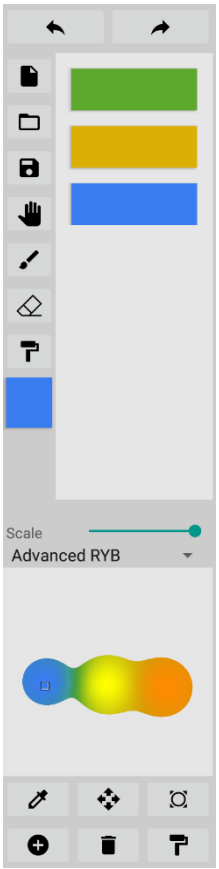


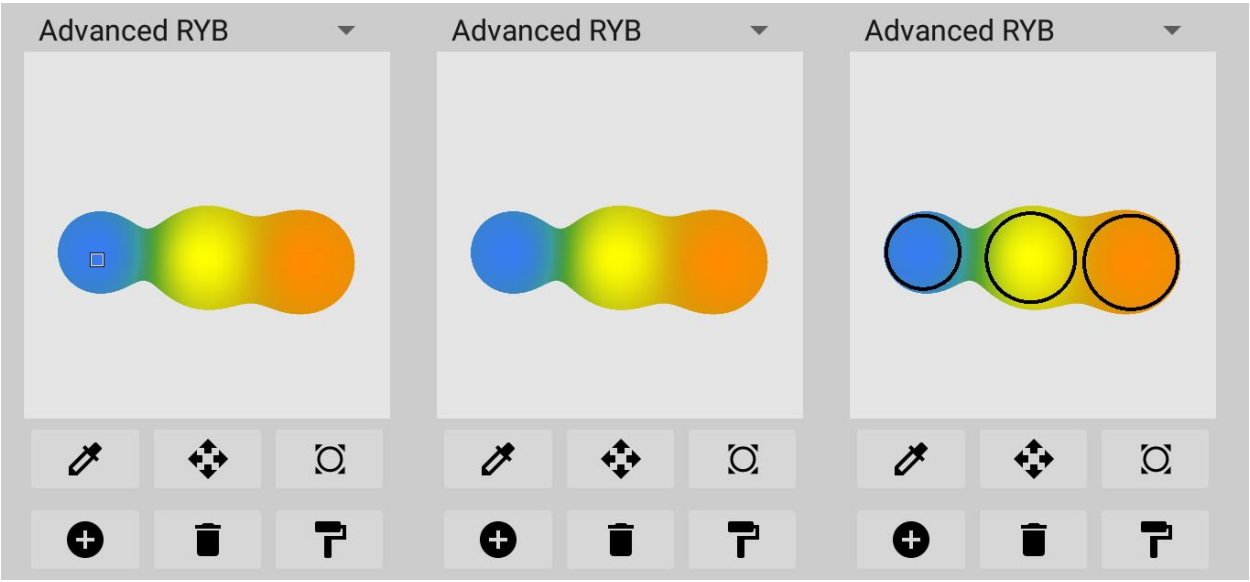
Рисунок 8 – панель инструментов.

Поле текущего цвета в реальном времени отображает выбранный или ещё выбираемый на палитре цвет для рисования, что позволяет видеть скрытый пальцем или стилусом цвет на палитре, на который сейчас наведён её курсор. Оно также работает как кнопка для открытия диалога выбора цвета при помощи селектора цвета.

В силу того, что для Android-планшетов характерно соотношение сторон экрана 5:8, панель инструментов зафиксирована по левому узкому краю экрана, а само приложение зафиксировано в горизонтальной ориентации.

2.2.3. Палитра

Палитра имеет не только три режима смешивания цветов, но и шесть режимов взаимодействия, и выглядит по-разному в зависимости от режима (рис. 9). Переключение режимов производится при помощи кнопок под палитрой, переключение режимов смешивания цветов производится при помощи выпадающего списка над палитрой.



а) б) в)

Рисунок 9 – палитра в режиме: а) выбора цвета; б) добавления капли; в) перемещения капли.

В режиме выбора цвета она оснащена квадратным курсором, при помощи передвижения которого по палитре выбирается цвет. Курсор состоит из двух

концентрических квадратных контуров – чёрного и белого – что позволяет отчётливо видеть его на любом цвете.

В режиме добавления капли курсор не отображается.

В режимах перемещения капли, изменения размеров капли, удаления капли и перекрашивания капли курсор также не отображается, но отображаются чёрным цветом границы отдельных капель для облегчения попадания по ним. При этом при нажатии на общую область нескольких капель нажатие происходит на самую маленькую из них, чтобы каплю нельзя было целиком закрыть одной каплей большего размера.

В режиме изменения размера капли установка нового размера производится путём нажатия на произвольную точку капли и перетаскивания её на желаемое место, при этом центр капли положения не меняет. Таким образом, для выполнения данного действия достаточно одного пальца, а значит его можно было выполнять стилусом.

2.2.4. Панель использованных цветов

Панель использованных цветов (рис. 10) занимает всё доступное по вертикали место между кнопками взаимодействия с историей изменений и слайдерами настройки инструментов. В ней реализована прокрутка по вертикали, а также поддержка перетаскивания объектов. Для «взятия» объекта необходимо зажать его - выполнить т.н. «долгое» нажатие. Затем объект можно поместить на другое место внутри панели или удалить, «отпустив» его вне панели.

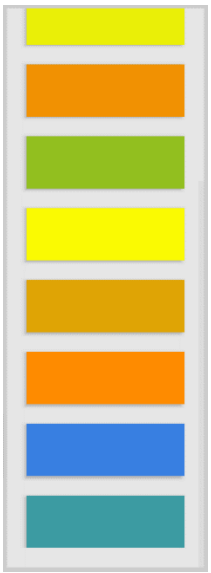


Рисунок 10 – панель использованных цветов с большим количеством цветов, чем её размеры позволяют одновременно отобразить, прокрученная вниз.

Повторное использование уже имеющегося на панели цвета не приводит к добавлению на панель дубликата, даже если цвет был заново выбран вне неё. При добавлении нового цвета панель при необходимости автоматически прокручивается вниз.

Выбор цвета для рисования с панели осуществляется простым нажатием на него.

Наличие тени у цветов на панели позволяет видеть на ней даже цвет, совпадающий с цветом её фона.

2.2.5. Диалог селектора цвета

Диалог селектора цвета (рис. 11) вызывается в нескольких случаях, а именно при добавлении капли, перекраске капли, заливке холста и ручном выборе цвета для рисования.

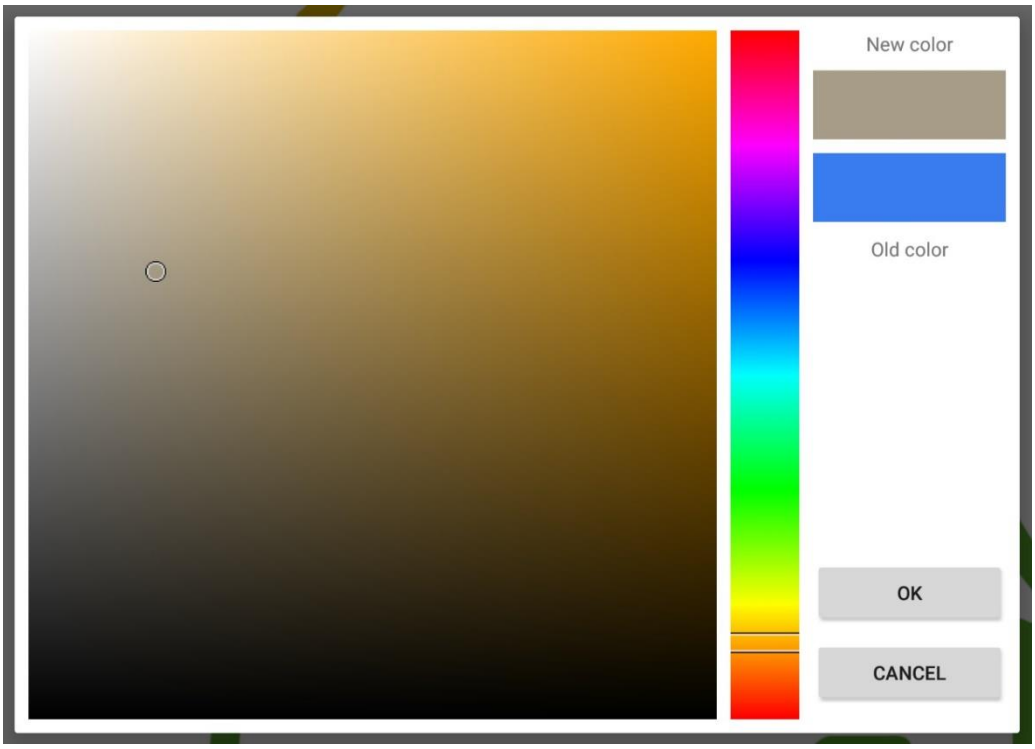


Рисунок 11 – диалог селектора цвета.

Выбор цвета выполняется при помощи двух курсоров. Оба курсора состоят из чёрного и белого контуров, а потому чётко видны на любом цвете.

Диалог также оснащён полями старого и нового цветов. В поле старого цвета отображается установленный на данный момент цвет в случае выбора цвета для рисования или добавления капли, нынешний цвет капли в случае перекраски капли, и нынешний цвет холста при заливке холста. В поле нового цвета отображается выбранный или выбираемый

цвет, что позволяет видеть в режиме реального времени выбираемый цвет, который может быть скрыт пальцем или стилусом. Соседство этих полей позволяет наглядно сравнивать старый и новый цвета.

Приведённые решения в области графического пользовательского интерфейса делают максимально удобным и интуитивным использование приложения как при помощи стилуса, так и при помощи пальцев. Любое действие можно выполнить одним пальцем, а значит и стилусом. Кроме того, при выборе цвета выбираемый цвет всегда отображается в отдельном поле, что позволяет видеть цвет, который мог загородить палец или стилус.

В данной главе были описаны используемые в программе алгоритмы, а также обоснованы и описаны решения в области графического пользовательского интерфейса программы.

ГЛАВА 3. ТЕХНОЛОГИЧЕСКИЕ РЕШЕНИЯ

Данная глава содержит описание программной реализации алгоритмов и решений, описанных в главе 2 данной работы.

3.1. Технологии и инструменты разработки

Для разработки приложения использовалась интегрированная среда разработки Android Studio 2.2.3.

При разработке алгоритма рисования пятна использовался онлайн-сервис для построения графиков функций «Desmos» [13].

При создании палитры использовались средства подмножества графического интерфейса OpenGL для встраиваемых систем OpenGL ES 2.0.

3.2. Общая архитектура приложения

Основой приложения является единственный Android Activity – MainActivity (рис. 12). Его макет расположен в отдельном XML-файле activity_main.xml.

Одним из ключевых элементов программы является PaintingView – элемент, в котором производится рисование. При обмене данными с MainActivity для получения информации о текущем инструменте используется перечисление Tool.

С PaintingView связан ещё и класс UsedPanelView – панель использованных цветов как элемент.

История изменений также ведётся в PaintingView с применением классов Stroke и EraserStroke. История изменений подробнее описывается в п. 3.6.

Палитра как элемент графического пользовательского интерфейса представлена классом Palette, который управляет вспомогательными классами PaletteRenderer и PaletteCanvas. Класс Blob используется для представления данных об отдельных каплях. Сами алгоритмы рисования пятна реализованы в шейдерах на языке GLSL.

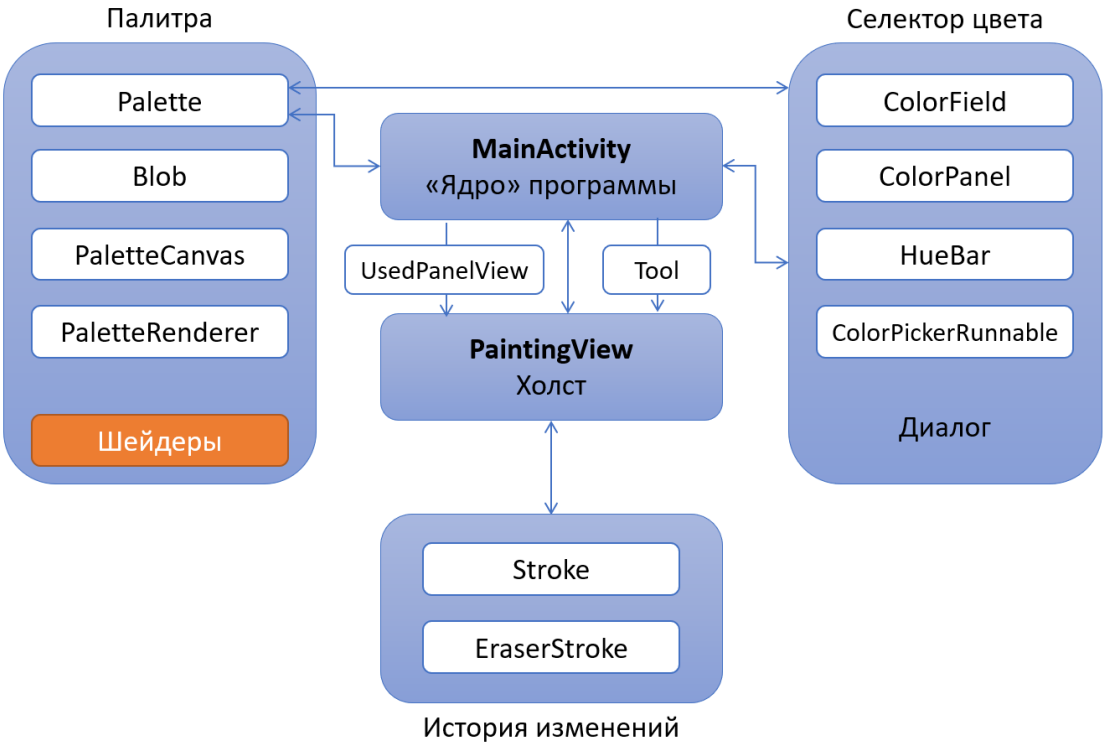


Рисунок 12 – общая схема архитектуры приложения.

3.3. Особенности имплементации палитры

Основным классом палитры является класс Palette. Он унаследован от класса GLSurfaceView, используемого при реализации графики при помощи OpenGL ES.

При инициализации палитры используется класс PaletteRenderer, который в том числе позволяет загружать шейдеры при помощи своего статического метода loadShader().

Рисование палитры происходит в классе PaletteCanvas, который используется для подготовки и передачи данных в шейдеры paletteCanvas.vert и paletteCanvas.frag.

Для хранения данных об отдельных каплях используются объекты класса Blob. Объекты данного класса хранят информацию о радиусе, координатах и цвете капель. Также данный класс реализует сравнение капель по размеру.

3.3.1. Фрагментный шейдер

paletteCanvas.frag – пиксельный (фрагментный) шейдер, реализующий построение пятна. Он принимает данные до 13 капель – именно такое количество показало себя как достаточное и оптимальное – и дополнительные данные, в т.ч. режим палитры и режим смешивания цветов.

Попарное смешивание цветов реализовано в методах addColorRGB(), addColorAdvancedRGB() и addColorAdvancedRYB(). Алгоритмы, реализованные в этих методах, описаны в главе 2. Ниже приведён код метода addColorAdvancedRYB():

```
vec3 addColorAdvancedRYB(vec3 oldc, vec3 newc, float newwt)
{
    float oldwt = 1. - newwt;
    float aver = 0.4;

    vec3 hsb1 = rgb2hsv(oldc);
    vec3 hsb2 = rgb2hsv(newc);
    hsb1[0] = rgb2ryb(hsb1[0]);
    hsb2[0] = rgb2ryb(hsb2[0]);

    float x1 = hsb1[2] * cos(hsb1[0] * 2. * M_PI);
    float y1 = hsb1[2] * sin(hsb1[0] * 2. * M_PI);
    float x2 = hsb2[2] * cos(hsb2[0] * 2. * M_PI);
    float y2 = hsb2[2] * sin(hsb2[0] * 2. * M_PI);

    float x = oldwt * x1 + newwt * x2;
    float y = oldwt * y1 + newwt * y2;

    float angle;

    if (x > 0.)
    {
        if (y >= 0.)
            angle = atan(y / x);
        else
            angle =  atan(y / x) + 2. * M_PI;
    }
    else if (x < 0.)
        angle =  atan(y / x) + M_PI;
    else
    {
        if (y > 0.)
            angle = M_PI * 0.5;
        else if (y < 0.)
            angle = M_PI * 1.5;
        else
            angle = 0.;
    }

    if (hsb1[0] - hsb2[0] > 0.5)
        hsb1[0] -= 1.;
    else if (hsb2[0] - hsb1[0] > 0.5)
        hsb2[0] -= 1.;

    float huSUBT = ryb2rgb(0.5 * angle / M_PI);
    float saSUBT = oldwt * hsb1[1] + newwt * hsb2[1];
    float brSUBT = sqrt(x * x + y * y);

    float rAVER = oldwt * oldc.r + newwt * newc.r;
    float gAVER = oldwt * oldc.g + newwt * newc.g;
    float bAVER = oldwt * oldc.b + newwt * newc.b;

    vec3 rgb = hsv2rgb(vec3(huSUBT, saSUBT, brSUBT));
```

```
    float r = rAVER * aver + rgb.r * (1. - aver);
    float g = gAVER * aver + rgb.g * (1. - aver);
    float b = bAVER * aver + rgb.b * (1. - aver);

    return vec3(r, g, b);
}
```

В основном методе каждая новая капля «сливается» с предыдущей, давая предварительное значение результирующего цвета и предварительное значение левой части неравенства (1), определяющего принадлежность пикселя к пятну. Ниже приведён код обработки одной из капель:

```
if (0 < n)
{
    weight = wt(x, y, blobsX[0], blobsY[0], blobsR[0]);
    cweight = cwt(x, y, blobsX[0], blobsY[0], blobsR[0]);
    sum += weight;
    colorSum += cweight;
    tempColor = addColor(tempColor, vec3(blobsColorR[0], blobsColorG[0],
blobsColorB[0]), cweight / colorSum);
    if (inCircle(x, y, blobsX[0], blobsY[0], blobsR[0]))
        edge = true;
}
```

При переборе капель цикл не используется в силу ряда особенностей и ограничений компилятора GLSL.

В конце производится проверка ряда условий – режим палитры, положение курсора, принадлежность пикселя пятну и др. – и установка окончательного цвета пикселя.

Курсор и границы отдельных капель также рисуются в данном шейдере.

Использование пиксельного шейдера позволяет применять аппаратное ускорение вычислений и обеспечить быстродействие палитры, от которой требуется работать в режиме реального времени.

3.4. Особенности имплементации холста

Основным классом холста является класс PaintingView, унаследованный от View.

Для рисования на холсте используется класс android.graphics.Path, в который помещаются данные о перемещении пальца (или стилуса) по экрану. Ниже приведён фрагмент кода onTouchEvent(), в котором происходит обработка событий касания элемента:

```
switch (event.getAction())
{
    case MotionEvent.ACTION_DOWN:
        path.moveTo(x, y);
        break;
    case MotionEvent.ACTION_MOVE:
        path.lineTo(x, y);
        break;
    case MotionEvent.ACTION_UP:
```

```
            path.lineTo(x, y);
            if (((MainActivity)getContext()).getTool() == Tool.BRUSH)
                present.push(new Stroke(path, paint));
            else if (((MainActivity)getContext()).getTool() == Tool.ERASER)
                present.push(new EraserStroke(path, paint, this));
            future.clear();
            path.reset();
            if (((MainActivity)getContext()).getTool() == Tool.BRUSH)
                usedPanelView.addColor(color);
            break;
        default:
            return false;
}
```

Как видно из приведённого выше кода, данные о полученных мазках инкапсулируются в объекты класса Stroke и (или) его наследника EraserStroke, которые являются основными элементами истории изменений (см. раздел 3.6).

Ниже приведён фрагмент кода метода onDraw(), в котором производится рисование мазков:

```
canvas.drawBitmap(bitmap, 0, 0, canvasPaint);
canvas.drawColor(backgroundColor);

if (backgroundBitmap != null)
    canvas.drawBitmap(backgroundBitmap, 0, 0, canvasPaint);

for (Stroke elem : present)
    elem.draw(canvas);
if (((MainActivity)getContext()).getTool() == Tool.BRUSH)
    new Stroke(path, paint).draw(canvas);
else if (((MainActivity)getContext()).getTool() == Tool.ERASER)
    new EraserStroke(path, paint, this).draw(canvas);
```

В этом коде также происходит отрисовка открытого изображения, если оно имеется.

3.5. Имплементация истории изменений

История изменений реализована по поведенческому паттерну проектирования Command [19]. Два стека хранят «настоящее» и «будущее» соответственно, при отрисовке холста рисуются все мазки, хранящиеся в настоящем, что видно из кода выше.

При отрисовке для объекта-мазка вызывается его метод draw(). Ниже приведён код класса Stroke, представляющего мазок кистью:

```
package com.asvarg.liquipaint.strokes;

import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Path;

public class Stroke
{
    protected Path path;
```

```
    protected Paint paint;

    public Stroke(Path path, Paint paint)
    {
        this.path = new Path(path);
        this.paint = new Paint(paint);
    }

    public void draw(Canvas canvas)
    {
        canvas.drawPath(path, paint);
    }
}
```

Наследование класса Stroke и переопределение метода draw() позволяют добавлять в историю другие мазки или более сложные действия. Ниже приведён переопределённый метод draw() из класса-наследника EraserStroke, реализующего мазок ластика:

```
public void draw(Canvas canvas)
{
    int alpha = paint.getAlpha();
    paint.setColor(paintingView.getBackgroundColor());
    paint.setAlpha(alpha);
    canvas.drawPath(path, paint);
}
```

Имплементация приведённых выше частей программы ориентирована на возможность ввода новых функций, что крайне важно для графических редакторов, которые имеют практически бесконечный выбор возможных дополнительных функций, которые можно ввести.

Реализация всего заявленного в требованиях к программе функционала означает, что наибольшим ограничением качества результатов применения мной данной программы в ходе тестирования является мой навык рисования (рис. 13).

Рисунок 13 – программа в действии

ЗАКЛЮЧЕНИЕ

В рамках данной работы был разработан графический редактор для OS Android, поддерживающий весь набор основных функций, необходимых для рисования, и снабжённый «жидкой» палитрой, что делает гибридные палитры доступными широчайшему рынку пользователей Android-планшетов.

Имплементация ряда частей приложения располагает к дальнейшему совершенствованию и дополнению продукта.

Было изучено большинство, если не все, существующих на сегодняшний день подходов и решений в сфере организации палитры для смешивания цветов, а также рынок устройств и их характеристики. На основе полученных данных было разработано приложение, построенное на наиболее современных и перспективных подходах и адаптированное под актуальные устройства и условия эксплуатации.

Был разработан и реализован алгоритм рисования пятна, поддерживающий попиксельное распараллеливание.

В ходе работы были дополнительно разработаны алгоритмы смешивания цветов, приближенные к поведению реальных красок, которые могут послужить отправной точкой в дальнейшей разработке направления симуляции поведения физических красок.

К основным возможным направлениям дальнейшей разработки можно отнести развитие алгоритмов реалистичного смешивания цветов, пополнение функционала программы как стандартными для графических редакторов инструментами, так и специфичными возможностями, которые открывает использование палитры – например, режим перекраски изображения [4], а также дальнейшее совершенствование графического пользовательского интерфейса программы на основе обратной связи пользователей.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. A.R. Smith. Digital paint systems: an anecdotal and historical overview // IEEE annals of the history of computing. – 2001. – 23(2). – С. 4-30.
2. 25 high resolution marker pen brushes. [Электронный ресурс] URL: http://www.premiumpixels.com/freebies/25-high-resolution-marker-pen-brushes/ (Дата обращения: 12.02.2018, режим доступа: свободный).
3. Use the oil paint filter. [Электронный ресурс] URL: https://helpx.adobe.com/photoshop/using/oil-paint-filter.html (Дата обращения: 12.02.2018, режим доступа: свободный).
4. M. Shugrina, J. Lu, S. Diverdi. Playful Palette: an interactive parametric color mixer for artists // ACM transactions on graphics. – 2017. – 36(4). – 61.
5. Tablet operating systems' market share worldwide from 2013 to 2020. [Электронный ресурс] URL: https://www.statista.com/statistics/272446/global-market-share-held-by-tablet-operating-systems/ (Дата обращения: 12.02.2018, режим доступа: свободный).
6. P. Vandoren, T. Van Laerhoven, L. Claesen, J. Taelman, C. Raymaekers, F. Van Reeth. IntuPaint: bridging the gap between physical and digital painting // IEEE international workshop on horizontal interactive human computer system (tabletop). – 2008.
7. A.M. Blatner, J.A. Ferwerda, B.A. Darling, R.J. Bailey. TangiPaint: a tangible digital painting system // Color imaging conference. – Jan. 2011.
8. B.J. Meier, A.M. Spalter, D.B. Karelitz. Interactive color palette tools // IEEE computer graphics and applications. – 2004. – 24(3).
9. Screen sizes. [Электронный ресурс] URL: http://screensiz.es/tablet, (Дата обращения: 13.02.2018, режим доступа: свободный).
10. L.K. Baxter. Capacitive sensors: design and applications. – 1997. – С. 138-139.
11. J. Daintith, E. Wright. A dictionary of computing. 7th ed. – 2016.
12. J.F Blinn. A generalization of algebraic surface drawing // ACM transactions on graphics. – 1982. – 1(3). – С. 235-256.
13. Desmos. [Электронный ресурс] URL: https://www.desmos.com/calculator, (Дата обращения: 15.02.2018, режим доступа: свободный).
14. J.H. Nobbs. Kubelka – Munk theory and the prediction of reflectance. – 1985.
15. The magical tech behind Paper for iPad’s color-mixing perfection. [Электронный ресурс] URL: https://www.fastcompany.com/3002676/magical-tech-behind-paper-ipads-color-mixing-perfection (Дата обращения: 12.02.2018, режим доступа: свободный).

16. Additive-averaging color mixing. [Электронный ресурс] URL: http://www.huevaluechroma.com/044.php (Дата обращения: 12.02.2018, режим доступа: свободный).
17. Colour mixing in paints. [Электронный ресурс] URL: http://www.huevaluechroma.com/061.php (Дата обращения: 12.02.2018, режим доступа: свободный).
18. HSV (цветовая модель). [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/HSV_(%D1%86%D0%B2%D0%B5%D1%82%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C) (Дата обращения: 13.02.2018, режим доступа: свободный).
19. Паттерн Command (команда). [Электронный ресурс] URL: http://cpp-reference.ru/patterns/behavioral-patterns/command/ (Дата обращения: 15.03.2018, режим доступа: свободный).