**Документация к лабораторной работе №1**

**"Исследование цветовых моделей и создание конвертера цветов"**

**1. Цель работы**

Разработка интерактивного приложения для изучения цветовых моделей HSV, XYZ, LAB и преобразований между ними с визуализацией результатов.

**2. Выполненные задачи**

**2.1. Исследование цветовых моделей**

- HSV (Hue, Saturation, Value) - цилиндрическая модель, интуитивно понятная для пользователей

- XYZ - модель, основанная на исследованиях МКО (Международной комиссии по освещению)

- LAB - перцепционно-равномерная модель, учитывающая особенности человеческого зрения

**2.2. Реализация преобразований между моделями**

Создана цепочка преобразований: HSV ↔ XYZ ↔ LAB через промежуточное преобразование в RGB.

**2.3. Разработка пользовательского интерфейса**

- Визуальный предпросмотр цвета

- Три группы элементов управления для каждой модели

- Слайдеры и поля ввода для точной настройки

- Кнопка выбора цвета из палитры

**3. Техническая реализация**

**3.1. Архитектура приложения**

Приложение разработано на C++ с использованием фреймворка Qt, что обеспечивает кроссплатформенность и богатый набор UI-компонентов.

Основные классы:

- MainWindow - главное окно приложения

- ColorConverter - класс для преобразований между цветовыми моделями

**3.2. Алгоритмы преобразований**

1. HSV → RGB (Преобразование через геометрическую модель цилиндра):

double c = (v / 100.0) \* (s / 100.0);

double x = c \* (1 - std::abs(std::fmod(h / 60.0, 2) - 1));

double m = (v / 100.0) - c;

2. RGB → XYZ (Линеаризация и матричное преобразование):

auto linearize = [](double c) {

return (c > 0.04045) ? std::pow((c + 0.055) / 1.055, 2.4) : c / 12.92;

};

3. XYZ → LAB (Нормализация относительно белой точки D65 и нелинейное преобразование):

double fx = labF(x / D65.x());

double l = 116 \* fy - 16;

double a = 500 \* (fx - fy);

**3.3. Особенности реализации интерфейса**

Синхронизация элементов управления:

- Слайдеры и поля ввода связаны двунаправленными соединениями

- Механизм блокировки обновлений предотвращает рекурсивные вызовы

- Автоматический пересчет всех моделей при изменении любой компоненты

Обработка граничных условий:

void checkColorBounds(const QColor& color)

{

bool hasClipping = (color.red() == 255 || color.red() == 0 ||

color.green() == 255 || color.green() == 0 ||

color.blue() == 255 || color.blue() == 0);

}

**4. Сложности и решения**

**4.1. Проблема: Разные диапазоны значений в моделях**

- HSV: H(0-360), S(0-100), V(0-100)

- XYZ: X(0-100), Y(0-100), Z(0-100)

- LAB: L(0-100), A(-128-127), B(-128-127)

Решение: Использование QSlider с разными диапазонами и масштабированием значений.

**4.2. Проблема: Обеспечение точности преобразований**

Решение:

- Использование QDoubleSpinBox с настраиваемой точностью

- Тщательная проверка граничных условий

- Обработка особых случаев (деление на ноль, отрицательные значения)

**4.3. Проблема: Циклические обновления интерфейса**

Решение: Введение флага m\_updating для предотвращения рекурсии:

void MainWindow::updateAllViews(int sourceModel)

{

if (m\_updating) return;

m\_updating = true;

...

m\_updating = false;

}

**5. Функциональные возможности**

**5.1. Способы задания цвета**

1. Поля точного ввода - для каждого параметра трех моделей

2. Ползунки - для плавного изменения значений

3. Палитра цветов- через стандартный диалог Qt

**5.2. Визуализация**

- Предпросмотр текущего цвета

- Синхронизированное отображение во всех моделях

- Визуальная индикация выхода за допустимые диапазоны

**5.3. Обработка ошибок**

- Предупреждения о некорректных значениях

- Автоматическая корректировка выходящих за границы значений

- Информативные сообщения об ошибках преобразований

**6. Результаты работы**

**6.1. Достигнутые цели**

- Реализованы точные преобразования между HSV, XYZ, LAB

- Создан удобный интерактивный интерфейс

- Обеспечены три способа задания цвета

- Реализован автоматический пересчет между моделями

- Добавлена обработка граничных условий и ошибок

**6.2. Особенности реализации**

- Использование современного C++ и Qt 6

- Модульная архитектура с разделением логики и интерфейса

- Поддержка высокого DPI и различных тем оформления

- Кроссплатформенная совместимость

**7. Требования:**

* Qt 6.0 или выше
* Компилятор C++ с поддержкой C++17
* CMake 3.16 или выше

**8. Совместимость с операционными системами**

**8.1. Поддержка версий Windows**

Приложение было протестировано и успешно работает на следующих версиях операционной системы Windows:

- Windows 10 (рекомендуемая версия)

- Windows 11

- Windows 8.1

- Windows 8

- Windows 7 (с установленным обновлением Platform Update)

- Windows XP (требуется установка дополнительных компонентов .NET Framework)

**8.2. Системные требования**

- Минимум 512 МБ оперативной памяти

- 50 МБ свободного места на диске

- Разрешение экрана не менее 1024×768 пикселей

**9. Известные проблемы и ограничения**

**9.1. Погрешности преобразования при использовании палитры цветов**

В ходе тестирования выявлена следующая проблема: при выборе цвета через встроенную палитру Qt (QColorDialog) наблюдаются значительные погрешности при преобразовании между цветовыми моделями.

Проявление проблемы:

- После выбора цвета в палитре значения в полях ввода HSV, XYZ и LAB могут незначительно отличаться от ожидаемых

- Обратные преобразования демонстрируют накопление погрешности

- Наиболее заметно на насыщенных и темных цветах

Причины:

- Многократные преобразования между цветовыми пространствами (RGB → HSV → XYZ → LAB и обратно)

- Округления в промежуточных вычислениях

- Особенности реализации алгоритмов преобразования в Qt

Временное решение: Для минимизации погрешностей рекомендуется:

1. Использовать прямые поля ввода для точной настройки цвета

2. Ограничить использование палитры для приблизительного выбора цвета

3. После выбора цвета в палитре выполнять тонкую настройку через числовые поля

**10. Заключение**

В ходе лабораторной работы успешно разработано приложение для изучения цветовых моделей и преобразований между ними. Приложение демонстрирует корректную работу алгоритмов преобразования, предоставляет удобный интерфейс для взаимодействия и обрабатывает различные граничные случаи.

Полученный опыт включает:

- Глубокое понимание цветовых моделей и их математических основ

- Навыки работы с графическим интерфейсом на Qt

- Решение проблем синхронизации данных в реальном времени

- Обработку ошибок и граничных условий в численных алгоритмах