**Министерство науки и высшего образования Российской Федерации**
**Федеральное государственное бюджетное образовательное учреждение высшего образования**
**«Московский государственный технический университет имени Н.Э. Баумана**
**(национальный исследовательский университет)»**
**(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Теоретическая информатика и компьютерные технологии

# РАСЧЁТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

## к курсовой работе по дисциплине «Алгоритмы компьютерной графики»

## на тему «Коррекция латеральных хроматических аберраций на фотографиях»

Студент ____ИУ9-51_____ _____ А.Д.Сорова
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ _А.В.Брагин__
(Подпись, дата) (И.О.Фамилия)

Москва, 2019

# АННОТАЦИЯ

В рамках курсовой работы были рассмотрены методы исправления хроматических аберраций на изображениях, их преимущества и недостатки.

Цель работы - реализация программного кода, считывающего изображение, определяющего степень его аберрации и исправляющего его.

В конечном итоге написана программа, получающая на вход изображение и выдающая непрерывные функции от координат пикселей - полиномы - для последующего исправления хроматической аберрации.

Работа состоит из 30 страниц, содержит 7 рисунков, 10 источников и 3 приложения.

Содержание

# Введение

С помощью зрительного аппарата человек имеет возможность получать изображения реального мира. Подобно этому устройству на основе линз были созданы такие вспомогательные приборы, как микроскопы, телескопы, фотокамеры, но они не являются совершенными. Вследствие свойства преломления волн разной длины на различные углы, на получаемых изображениях появляются артефакты, так называемые аберрации. Искажения, обусловленные дисперсией света, называются хроматическими аберрациями.

С целью их исправления в оптических приборах комбинируются линзы из неодинаковых по дисперсии света сортов оптического стекла. Объектив, в котором исправлена хроматическая аберрация для лучей света двух различных длин волн, называется ахроматическим, для трёх - апохроматическим. Однако в данной работе речь пойдёт о программном исправлении хроматических аберраций.

# 1. СВЯЗАННЫЕ ОПРЕДЕЛЕНИЯ

## 1.1 Аберрации. Типы

Под аберрацией понимается ошибка или погрешность изображения, формируемого объективом на матрице фотоаппарата. Аберрации делятся на два типа: хроматические (цветовые) и монохроматические. Монохроматические подразделяются на сферическую аберрацию, кому, дисторсию, кривизну поля и астигматизм, а хроматические аберрации представляются двумя типами - продольные (хроматизм положения) и боковые (хроматизм увеличения) [1].

Хроматизм положения.

Продольные, или осевые, хроматические аберрации можно называть продольным хроматизмом или хроматизмом положения. Когда свет проходит через линзу или систему линз (оптическую систему), он разлагается на лучи разного цвета в зависимости от длины волны излучения соответствующего цвета, что, как известно, называется дисперсией света. Показатель преломления синих лучей больше, чем красных, поэтому точка фокуса синих лучей будет расположена ближе к линзе, затем сфокусируется зеленый, и лишь потом - красный. Таким образом, изображения объекта в лучах разного цвета будут располагаться на разных расстояниях от линзы (Рисунок 1). Разность фокусных расстояний излучения синего и красного цветов называется хроматизмом положения или хроматической разностью положения.

Хроматизм увеличения.

Причина появления хроматизма увеличения (или латеральных хроматических аберраций) та же, что и для хроматизма положения:

дисперсия света. При наличии этой аберрации изображения внеосевой точки, образованные оптической системой в лучах волн различной длины, располагаются на различных расстояниях от главной оптической оси, вследствие чего изображения одного и того же предмета в лучах разного цвета имеют несколько различный размер (Рисунок 2).

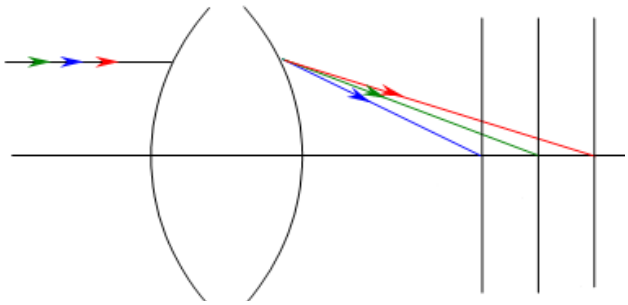

***Рисунок 1*** *- Хроматизм положения.*

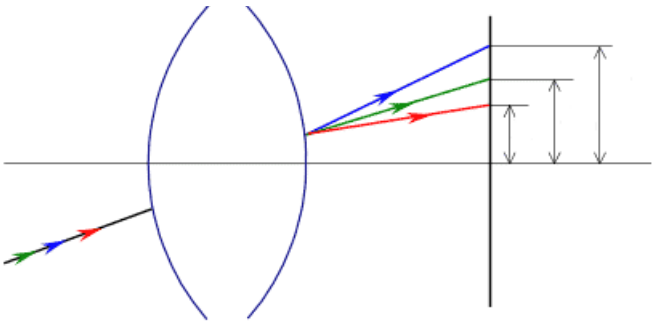

***Рисунок 2*** *- Хроматизм увеличения.*

### 1.2 Формат изображения PGM

PGM расшифровывается как Portable Gray Map - полутоновый формат хранения изображения. На 1 пиксель приходится 8 или 16 бит, в отличие от форматов PBM (Portable BitMap) и PPM (Portable PixMap), в которых на 1 пиксель приходится 1 бит и 24 (по 8 бит на каждую компоненту RGB) соответственно [2].

Любое PGM - изображение содержит последовательно:

1) Два символа - P2 или P5. P2 означает, что интенсивность каждого пикселя представлена ASCII кодом, а в P5 обычным (двоичным) числом.
2) Произвольное количество пробельных символов (пробел, табуляция, возврат каретки, переход на новую строку).
3) Десятичное число в виде ASCII символа, означает ширину изображения (wid).
4) Произвольное количество пробельных символов.
5) Десятичное число в виде ASCII символа, означает длину изображения (len).
6) Произвольное количество пробельных символов.
7) Десятичное число в виде ASCII символа - максимальное значение серого (maxval) для пикселей (больше 0, меньше 256 - 8 бит на пиксель, либо меньше 65536 - 16 бит на пиксель).
8) Один пробельный символ (обычно переход на новую строку).
9) Массив из len строк по wid столбцов, элементами являются двоичные числа в интервале от 0 до maxval (0 - чёрный цвет, maxval - белый).

Строки, начинающиеся с символа ‘#’, считаются комментариями. В файле типа P2 не должно быть строк длины более 70 символов.

### 1.2 Фильтр Байера.

Фильтр Байера - это матрица цветных фильтров, накладываемая поверх фотоматрицы. Каждый фотоприемник накрывается светофильтром одного из цветов RGB и воспринимает 1/3 цветовой информации участка изображения, а 2/3 отсекается фильтром [3].

Подсчитывается количество попавших фотонов в каждую ячейку фотоматрицы, что и является интенсивностью цвета, соответствующего цвету фильтра (Рисунок 3). Для получения остальных цветовых компонентов используются значения из соседних ячеек (дебайеризация).

В классическом фильтре Байера применяются светофильтры трёх основных цветов в определённом порядке порядке (Рисунок 4). Заметим, что фотодиодов зелёного цвета в каждой ячейке в два раза больше, чем фотодиодов других цветов, так как человеческий глаз более чувствителен к зелёному цвету, чем к красному и синему вместе взятым.

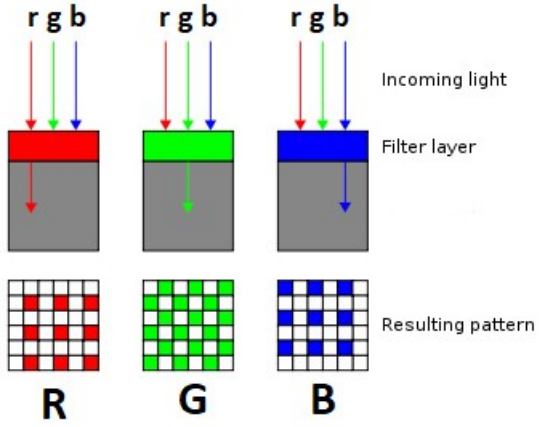


***Рисунок 3*** *- Действие светофильтров*

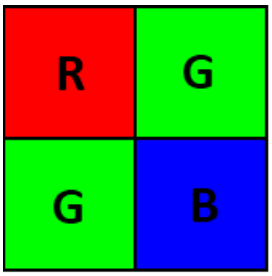


***Рисунок 4*** *- Фрагмент фильтра Байера*

# 2. ОБЗОР СУЩЕСТВУЮЩИХ АЛГОРИТМОВ ИСПРАВЛЕНИЯ АБЕРРАЦИЙ

В статье [4] предложены два метода исправления хроматичесикх аберраций.

Первый способ связан со смещением плоскости изображения (вперёд для синих лучей, назад для красных), для чего нужна дополнительная информация, например, фокальное расстояние объектива, расположение центра аберрации (зачастую не совпадает с центром фотографии). Это предполагает работу с аппаратурой, что является дорогим удовольствием, поэтому предлагается другой способ - устранение аберраций при цифровой обработке изображений.

Находятся границы объектов на изображении. Границы являются переходными областями, где компоненты RGB колеблются в определённых интервалах. Так как наибольшую ответственность за яркость цвета несёт зелёная компонента (она преобладает в фильтре Байера), относительно неё для каждого пикселя вычисляются разности R - G и B - G. Переходную зону составляют те пиксели, у которых данные разности входят в интервал между соответствующими разностями пикселей на концах переходной области. Алгоритм поиска переходной области запускается отдельно по горизонтали и по вертикали.

У каждого пикселя фиксируем одну координату. Он характеризуется значениями R, G, B и значениями градиента для каждого цвета. В переходной зоне интенсивность хотя бы одной компоненты RGB меняется, так как там происходит изменение цвета. Зелёный цвет находится в середине спектра, поэтому в основном все

аберрации считаются относительно красной и синей компонент, чтобы требовались меньшие по значениям трансформации. Пиксель считается принадлежащим области, если хотя бы одно значение градиента больше граничного заданного значения (порога).

Начинаем с первого пикселя. Первый пиксель, у которого значение "зелёного" градиента превышает пороговое значение или равно ему, становится затравочным. Относительно него ищем границы переходной области (затравочный пиксель заведомо ей принадлежит). Берём пиксель после (или до) затравочного. Выбираем наибольшее из всех значений градиента, причём чтобы направление роста интенсивности данного цвета совпадало с направлением роста интенсивности зелёной компоненты затравочного пикселя.

Рассматриваем случай, когда выбранное значение больше порогового или равно ему. Это значит, что пиксель принадлежит переходной области. Пока выполняется данное условие, двигаемся дальше. Следующую переходную область начинаем искать с координаты правой границы последней переходной области +1.

Теперь, когда найдены переходные, потенциально опасные, области, исправляем хроматические аберрации в этих областях. Для каждой переходной области выполняется следующее: если у какого-то пикселя разности R - G, B - G выходят за установленные границы, то значения RGB - компонент пикселей заменяются так, чтобы разность стала максимально/минимально возможной (в соответствии со значениями разностей пикселей на концах переходных областей).

В статьях [5] и [6] предлагается исправлять хроматические аберрации так, как если бы их исправлял обычный пользователь, только программно. Сначала говорится о ранее использованных методах.

1) При создании снимка делается не один кадр, а три. Для каждого из трёх основных цветов с разным фокусом надо подобрать правильное значение приближения и наклона камеры, чтобы в дальнейшем корректно объединить кадры. Сложность состоит в том, чтобы сделать эти три разных снимка примерно в одно и то же время, так как объект перед камерой может измениться (особенно когда снимаются сцены действий).

2) Вычисляются значения приближений и поворотов для каждого цвета (насколько они смещены относительно "идеального" изображения) и выражаются в виде кубических сплайнов, с помощью которых находится такое искажение изображения, при котором исправляются аберрации. Но когда выдержка фотографии слишком велика или, наоборот, мала, этот метод не применим.

Теперь рассмотрим основной метод, предложенный в статье. В диафрагму фотоаппарата (непрозрачную преграду, ограничивающую количество поступающего освещения) вставляется пластина с тремя цветовыми фильтрами, которые пропускают только один цвет RGB. Таким образом можно получить три разных слоя одного и того же изображения, правда, немного сдвинутых. Предложенный метод состоит в том, чтобы находить ключевые точки на двух слоях и сопоставлять их с соответствующими точками на зафиксированном третьем слое с помощью метрики L. Строим матрицу A, состоящую из цветовых компонент каждого пикселя (матрица размера 3x3).

Метрика L определяется как отношение произведения собственных значений матрицы A к произведению диагональных элементов этой матрицы, возведённых в квадрат. Значение данной метрики колеблется от 0 до 1.

Берётся начальный пиксель и выбирается конкретный параметр. В окружении начального пикселя на разных слоях выбираются точки с отступом справа, снизу и слева, отступ по значению равен выбранному параметру (выбрали три пикселя). Такой выбор пикселей связан со смещением цветных каналов изображения вследствие использования фильтра. Задача состоит в том, чтобы выбрать такой параметр, с которым бы значение метрики L становилось минимальным, так как чем значение метрики L больше, тем меньше коллинеарность пикселей в данной области (ограниченной параметром).

Итак, как же выбрать ключевые точки? Нужно брать такие, в которых метрика принимает большие значения, но также чтобы была возможность найти смещение слоёв, позволяющее исправить аберрацию. Хорошим выбором является поиск таких пикселей в точках с высоким значением градиента, особенно если рядом находится граница некоторого объекта на изображении. Таким образом, аберрации исправляются посредством всевозможных изменений размера и смещения двух слоёв относительно третьего. Но так как мы знаем, что несоответствие слоёв не должно быть очень большим (используются качественные линзы), имеем право ограничить поиск подходящих трансформаций.

Мы можем найти пиксель с высоким значением метрики L, который не будет давать нам возможность корректно сопоставить

слои изображения. В таком случае нужно просто больше не рассматривать данный пиксель. То есть можно ограничить значение метрики и искать пиксели, не превышающие его.

Чтобы найти корректную трансформацию, нам надо знать всего две ключевые точки, так что есть и другой способ их поиска. Можно всего лишь найти два пикселя с наименьшим значением метрики L и сопоставить слои так, чтобы они совпадал на итоговом изображении. Однако лучше искать больше "хороших" пикселей, так как любые два из них могут давать корректную трансформацию только для какой-то области, а не для целого изображения.

# 3. АЛГОРИТМ ИСПРАВЛЕНИЯ ХРОМАТИЧЕСКИХ АБЕРРАЦИЙ

## 3.1 Входные данные

Сначала на вход алгоритму подаётся одно тестовое изображение. Оно должно представлять собой изображение множества чёрных дисков на светлом фоне в формате PGM, любого размера. После выполнения замеров алгоритм в состоянии работать с любыми другими изображениями, сделанными с помощью камеры с теми же настройками, так как хроматические аберрации будут исправляться на всех изображениях аналогичным образом.

## 3.2 Поставновка задачи

По тестовому изображению алгоритм должен высчитывать коэффициенты полиномов от координат пикселей. Данные полиномы применяются к координатам R и B каналов поданного изображения с хроматической аберрацией. Аберрация исправляется.

## 3.3 Выходные данные

На выходе получаем три изображения в формате PGM: отдельно каждый цветовой канал исправленного изображения.

## 3.4 Описание алгоритма

Для написания программы был использован алгоритм исправления латеральных хроматических аберраций, предложенный в статье [7].

Имеется тестовое изображение с дисками на белом фоне, представляющими собой закрашенные чёрные круги фиксированного радиуса. Учитывая то, что камера может находиться не строго перпендикулярно к плоскости изображения, диски считаем эллипсами.

Возьмём точку из области конкретного диска с координатами (x, y). Тогда они связаны с координатами (x1, x2), соответствующими этой точке на идеальном изображении I (если бы камера была строго перпендикулярна к плоскости изображения), следующим отношением:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \lambda 1 * \cos\theta & -\lambda 2 * \sin\theta & 0 \\ \lambda 1 * \sin\theta & \lambda 2 * \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x1 - tu \\ x2 - tv \\ 1 \end{pmatrix},$$

где (tu, tv) - координаты центра диска, λ1, λ2 - параметры растяжения вдоль большой и малой осей эллипса соответственно, θ - угол между большой осью и осью x.

Предполагается, что уровень яркости L (интенсивности) изображения линейно возрастает от центра диска к его краям (светлеет). Чтобы минимизировать сумму разностей интенсивности цвета между пикселем диска на нашем изображении и соответствующим пикселем на изображении I в квадрате, используется алгоритм Левенберга-Марквардта [8]:

$$\min \sum_{x1,y1} (L(x1, y1) - L(x, y))^2 \to 0$$

Цель алгоритма - поиск направления наискорейшего спуска $\vec{p}$ функции в точке. Оно определяется из системы:

$$\left[J^{\mathrm{T}}(x_{\mathrm{k}})J(x_{\mathrm{k}}) + \lambda_{\mathrm{k}} I\right] p_{\mathrm{k}} = -J^{\mathrm{T}}(x_{\mathrm{k}}) f(x_{\mathrm{k}}),$$

где $\lambda_k$ - конкретная неотрицательная константа, своя для каждого шага (параметр регулизации), J - матрица Якоби (матрица первых производных) в точке x , I – единичная матрица.

Иногда I заменяют на $\mathrm{diag}(J^TJ)$ для улучшения сходимости.

Следующим образом определяется $x_{k+1}$ : $x_{k+1} = x_k + p_k$

Параметр $\lambda_k$ можно устанавливать исходя из отношения между фактическими изменениями функции, достигнутыми в результате пробных шагов, и ожидаемыми величинами этих изменений при интерполяции. Если значение этого отношения больше нуля, значит, мы хорошо интерполируем функцию, иначе нужно увеличить параметр $\lambda_k$. Начальное значение $\lambda_k$ задаётся как $t*\max\{a_{ij}\}$, где $a_{ij}$ - элементы матрицы $J^TJ$. Рекомендуется обозначать t за $10^{-3}$.

Итак, для каждого цветового канала изображения определяются расположения дисков и их центров - ключевых точек. Центры дисков в каналах R и B ставятся в пару к ближайшему центру диска в канале G. Основной выбрана зелёная компонента, так как именно она преобладает в фильтре Байера.

Идея состоит в том, чтобы для различных каналов составить полиномы от двух переменных, координат центров дисков, переводящие данные каналы в изображение, лишённое хроматической аберрации. Степенью полиномов выбрано число 11, так как в таком случае искажения исправляются, но при более высокой степени увеличение точности становится незначительным.

Ключевые точки представлены в пиксельных координатах $(x_f, y_f)$, где f - один из трёх цветов: (r) для красного, (g) для зелёного, (b) для синего. Несоответствие между R (или B) и G каналом исправляется за счёт выбора коэффициентов полиномов $p_{fx}$, $p_{fy}$, наилучшим образом аппроксимирующих уравнения:

$$x_{gi} = p_{fx}(x_{fi}, y_{fi})$$

$$y_{gi} = p_{fy}(x_{fi}, y_{fi}),$$

где f = r или f = b, i является индексом ключевой точки.

Полином $p_{fx}$ степени m с коэффициентами $\{p_{x0}, p_{x1}, \ldots, p_{x(m+1)(m+2)/2}\}$ имеет в случае m = 11 (m+1)(m+2)/2 = 78 неизвестных для каждого цветового канала. Он может быть расписан в виде:

$$x_g = p_{x0}x^m_f + p_{x1}x^{m-1}_f y_f + p_{x2}x^{m-2}_f y^2_f + p_{xm}y^m_f + p_{xm+1}x^{m-1}_f + \ldots + p_{xm+2}x^{m-2}_f y_f + p_{x2m}y^{m-1}_f + p_{x(m+1)(m+2)/2-3}x_f + \ldots + p_{x(m+1)(m+2)/2-2}y_f + p_{x(m+1)(m+2)/2-1}.$$

Чтобы данная система была решаема, количество неизвестных не должно превосходить количество ключевых точек, то есть на тестовом изображении должно быть более 156 чёрных дисков. Коэффициенты полиномов высчитываются таким образом, чтобы минимизировалась разница между значениями левой и правой частей уравнений написанной выше системы:

$$\sum_{i=1}^{K} (p_{fx}(x_{fi}, y_{fi}) - x_{gi})^2 + (p_{fy}(x_{fi,}\ y_{fi}) - y_{gi})^2 \to 0\ ,$$

где K - количество ключевых точек. Нахождение коэффициентов полиномов (координаты вектора p) сводится к решению системы линейных уравнений A*p=v, где A - матрица, составленная из координат центров дисков отдельно для каждого канала изображения.

Теперь, когда коэффициенты полиномов определены, на вход алгоритму можно подавать любое изображение, сделанное с теми же настройками камеры.

# 4. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Программа написана на языке Java. В программу передаётся три параметра (в командную строку). Первый — путь к тестовому изображению в формате PGM (чёрные диски на белом фоне), а второй и третий — пути к файлам для записи полиномов, исправляющих хроматические аберрации для определённых настроек камеры.

Для чтения изображения используются классы java.io.File и java.io.FileInputStream [9]. Подавая экземпляр первого класса в конструктор второго, мы имеем возможность считывать указанный в пути файл побайтово. Узнав размеры изображения, мы составляем матрицу, в которую записываем интенсивность каждого пикселя. Если данное PGM изображение типа P2, то предварительно вычитаем из каждого считанного числа ASCII код 0 (48). Таким образом мы создали экземпляр класса Image. Описанная часть кода представлена в Приложении А.

Далее идёт разделение на три цветовых канала, создаются три экзепляра класса Image. Изображения каналов R и B в два раза меньше в сравнении с исходным изображением вследствие расположения светофильтров матрицы Байера. Интенсивности R и B пикселей переносятся в экземляры класса Image соответствующих каналов. Пиксели G канала сохраняют своё положение относительно исходного изображения, но значение пустующих пикселей вычисляется как среднее арифметическое соседних (четырёх, прилегающих рёбрами) пикселей.

Переходим к нахождению ключевых точек — центров дисков, представленных классом CCStats (хранятся координаты, два радиуса (по малой и большой осям эллипса), площадь диска и длина «окружности» (эллипса), то есть количество пикселей на краях области). Отдельно для каждого канала F:

1) Проводим бинаризацию изображения. Высчитываем максимальное и минимальное значения интенсивности пикселей канала, берём среднее значение mid. Создаём экземпляр класса Image, считая все его писели белыми (изначально заполняем матрицу значениями 255). Значение каждого пикселя p в F сравниваем с mid, если p<mid, то пиксель считаем чёрным (устанавливаем значение 0). На выходе получаем бинаризированное изображение. Данная часть программного кода представлена в Приложении Б.

2) Для каждого пикселя бинаризированного изображения, представленного классом Pixel (хранятся координаты), считаем количество смежных чёрных пикселей. Если затравочный пиксель чёрный, то кладём его в массив mas, в котором храним все чёрные смежные пиксели, смежные пиксели (любого цвета) уходят в стек, затравочный окрашиваем в белый. Пока стек не пуст, проделываем это с каждым пикселем из стека (начинаем с проверки цвета). Эта часть программы представлена в Приложении В.

3) Если количество чёрных пикселей вокруг затравочного больше

180,  то мы предположительно нашли чёрный диск. Подаём mas в функцию, определяющую координаты центра области (средние значения координат x (аналогично для y) среди пикселей из mas), два радиуса (по большой и малой осям, как среднее арифметическое максимального и минимального значений x (аналогично для y)), длину окружности (количество пикселей из mas, у которых есть хотя бы один белый сосед). Считается качественная характеристика эллипса comp — отношение площади нашего эллипса (размер массива mas) к площади $L^2/4\pi$ «идеального» круга (с гладкой границей), где L — длина окружности.

4) Если меньший из радиусов области не меньше восьми пикселей, а comp находится в пределах от 0.7 до 1.3, то наконец считаем данную область чёрный диском, а его центр — ключевой точкой.

Затем проверяется количество найденных ключевых точек на разных каналах изображения. Если они не равны дру другу, то программа прерывается.

Следующий шаг - сопоставление ключевых точек на разных слоях изображения. Фиксируется одна ключевая точка в канале G. В радиусе пяти пикселей ищется ближайшая ключевая точка к фиксированной. Для каждого из планов R и B чёрные диски вырезаются в отдельные изображения (экземляры класса Image). Для каждого такого изображения с помощью алгоритма Левенберга-Марквардта отдельно для планов R и B высчитывается другие координаты центров дисков, уже в системе координат «идеального» изображения (используется аффиное преобразование координат), а

также новые радиусы (вдоль большой и малой осей эллипса), угол поворота относительно оси x на «идеальном» изображении. Значения радиусов умножаются на два (изначально планы R и B были в два раза меньше). Для плана G проделывается то же самое, только радиусы остаются того же размера.

Теперь составляются полиномы степени 11. Решается система линейных уравнений, но для начала матрица A, составленная из координат центров дисков отдельно для каждого канала изображения, нормализуется (для упрощения вычислений). То есть суммируются квадраты координат x и y, из суммы берётся квадрат, и каждый элемент матрицы делится на получившееся число. С помощью экземпляра класса java.io.FileWriter в два отдельных файла по указанным путям записываются коэффициенты полиномов вместе с соответствуюими степенями x и y.

# 5. ТЕСТИРОВАНИЕ

В качестве тестового изображения было выбрано изображение с 26*37 = 962 чёрными дисками на белом в фоне. В ходе работы программа обнаружила 962 ключевых точки на каждом из каналов изображения, что соответствует действительности. Коэффициенты полиномов был успешно выведены в файлы. На обработку тестового изображения потребовалось около 7 минут. Результат на Рисунке 5.

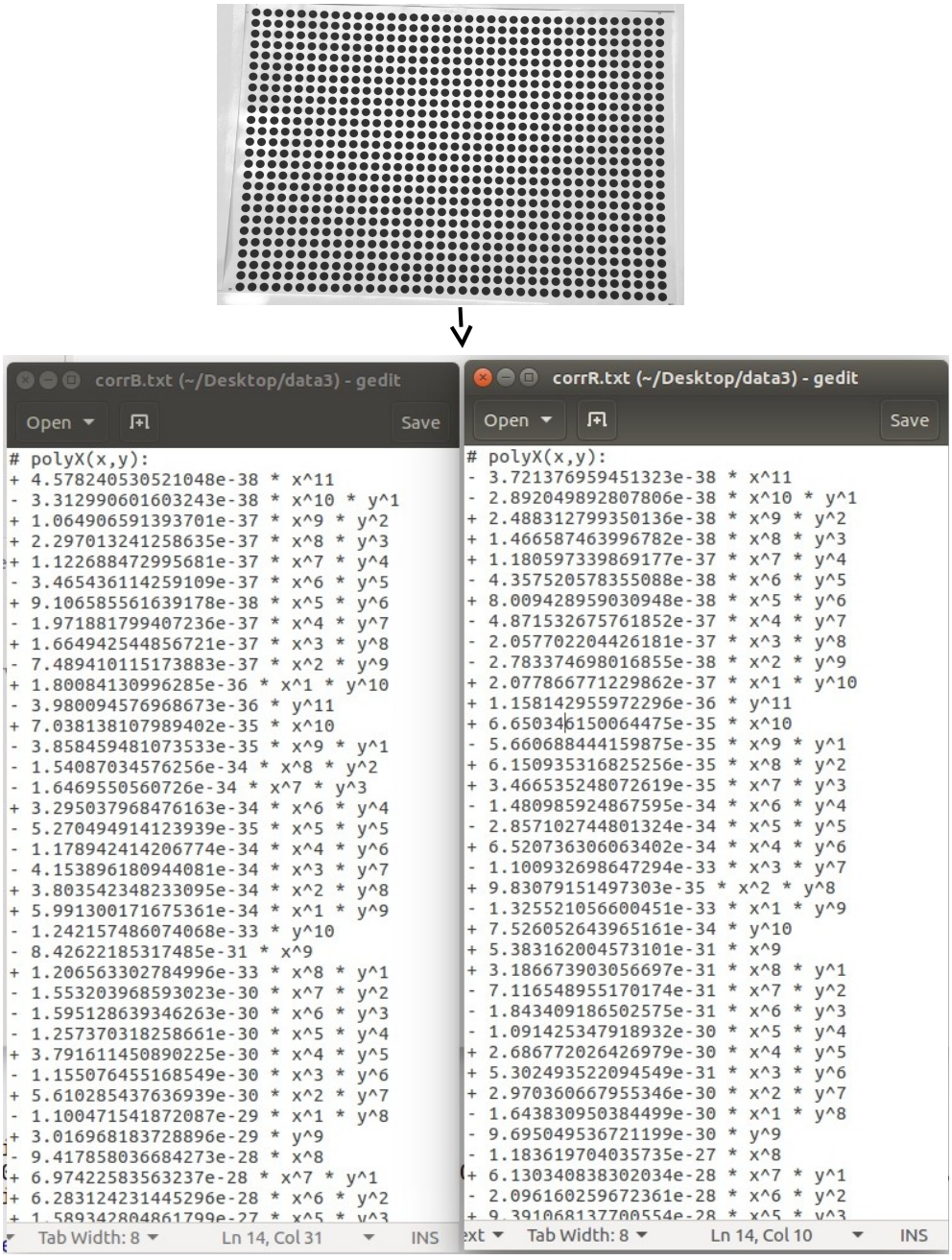


***Рисунок 5*** *- Коэффициенты полиномов*

Пример работы алгоритма при подаче ему другого изображения, сделанного с теми же настройками камеры, что и тестовое изображение, представлен на Рисунке 6.

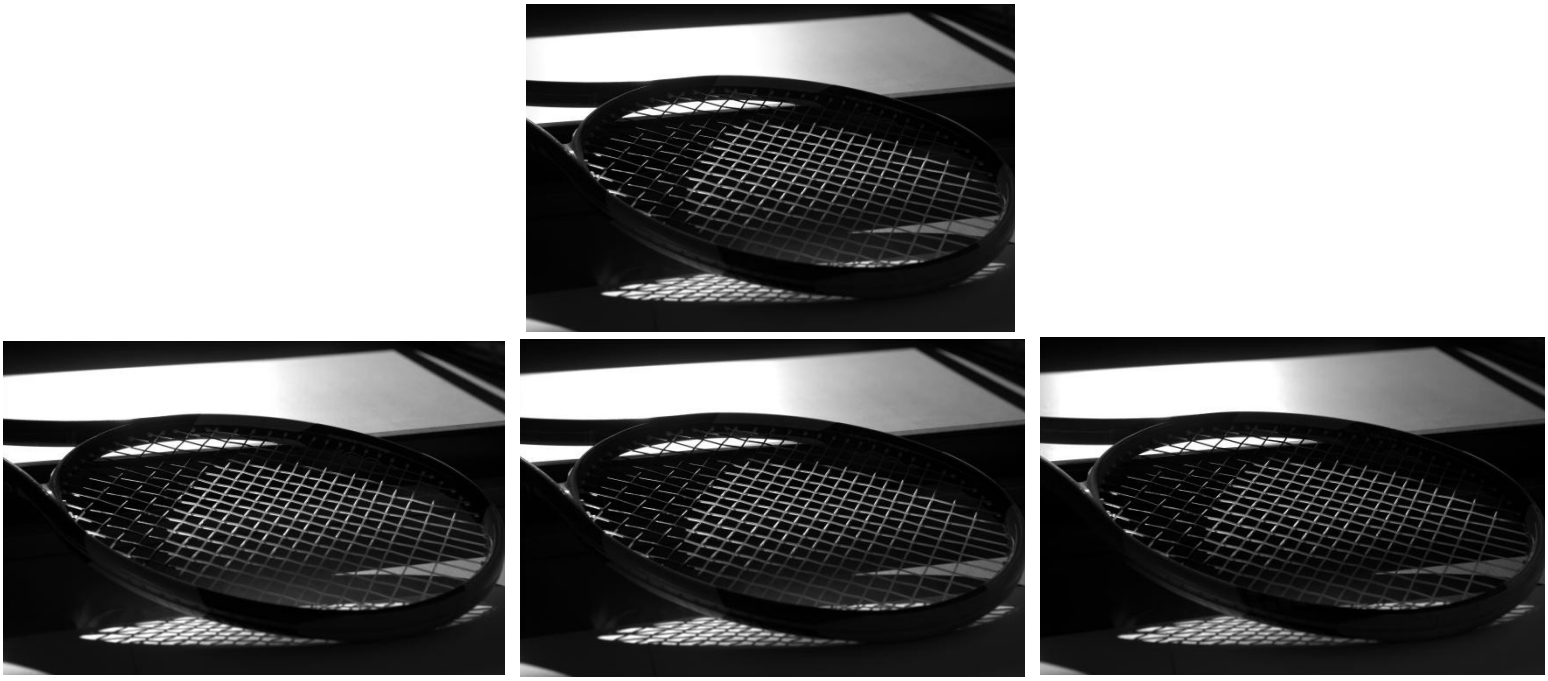

***Рисунок 6*** *- Результат работы программы: разложение на три канала без хроматических аберраций*

Конечно, разница почти не заметна. Но соединив эти три канала в одно изображение, получим результат на Рисунке 7.

***Рисунок 7*** *- Объединение трёх каналов*

Также в качестве тестового изображения было использовано изображение с искусственно созданной аберрацией. В программе Adobe After Effects [10] изображение было разделено по каналам RGB, красный план был смещён на два пикселя влево, синий на два пикселя вправо, затем каналы снова были соединены. На данном тестовом изображении программа обнаружила неравное количество ключевых точек на разных планах, поэтому прекратила работу.

Возможно, проблема состоит в том, что искажение на два пикселя для современных фотокамер - это слишком большая погрешность, а алгоритм был разработан с учётом того, что такая большая неточность невозможна.

## ЗАКЛЮЧЕНИЕ

В рамках курсового проекта был написан программный код, считывающий изображение и записывающий в файлы коэффициенты полиномов от координат пикселей для последующего исправления хроматической аберрации.

Были рассмотрены различные алгоритмы исправления искажений на изображениях, их возможная реализация.

Дальнейшее развитие проекта может быть направлено на увеличение скорости работы программы, расширение множества форматов входных изображений.

## Список литературы

1) 1998 - 2018 ПиКО - Кафедра Прикладной и Компьютерной Оптики. Учебное пособие по курсу “Основы оптики”[Электронный ресурс]. Режим доступа: http://aco.ifmo.ru/el_books/basics_optics/. Дата обращения к ресурсу: 22.09.2018

2) PGM Format Specification [Электронный ресурс]. Режим доступа: http://netpbm.sourceforge.net/doc/pgm.html. Дата обращения к ресурсу: 13.11.2018

3) CAMBRIDGE in Color. A learning community for photographers [Электронный ресурс]. Режим доступа: https://www.cambridgeincolour.com/ru/tutorials-ru/camera-sensors.htm. Дата обращения к ресурсу: 26.11.2018

4) Soon-Wook Chung, Byoung-Kwang Kim, Woo-Jin Song. Removing chromatic aberration by digital image Processing. Optical Engineering 49(6), June 2010.

5) Benjamin T. Cecchetto. 2009. CPSC 525 Correction of Chromatic Aberration from a Single Image Using Keypoints.

6) Bando, Y., Chen, B., and Nishita, T. 2008. Extracting depth and matte using a color-filtered aperture. In International Conference on Computer Graphics and Interactive Techniques, ACM New York, NY, USA.

7) Victoria Rudakova, Pascal Monasse. Precise correction of lateral chromatic aberration in images. R. Klette, M. Rivera, and S. Satoh. PSIVT, Oct 2013, Guanajuato, Mexico. Springer, 8333, pp.12-22, 2013, Lecture Notes in Computer Science.<hal-00858703>

8) Алгоритм Левенберга — Марквардта для нелинейного метода наименьших квадратов и его реализация на Python. [Электронный

ресурс]. Режим доступа : https://habr.com/ru/post/308626/. Дата обращения к ресурсу: 25.03.2019

9) Java™ Platform, Standard Edition 7 API Specification. [Электронный ресурс]. Режим доступа: https://docs.oracle.com/javase/7/docs/api/. Дата обращения к ресурсу: 14.03.2019

10) ADOBE AFTER EFFECTS CC. [Электронный ресурс]. Режим доступа: https://www.adobe.com/products/aftereffects.html. Дата обращения к ресурсу: 16.03.2019

## Приложение А. Считывание изображения в формате PGM.

В листинге А1 представлен метод класса Image, отвечающий за считывание изображения в формате PGM.

### Листинг А1

```
public static Image read_pgm(String name) throws IOException {
   boolean bin = false;
   int xsize, ysize, depth, x,y;
   Image im;
   File file = new File(name);
   FileInputStream reader = new FileInputStream(file);
   int c;
   if (reader.read() != 'P')
      throw new Error("not a PGM file!");
   if ((c=reader.read()) == '2')
      bin = false;
   else if ( c == '5')
      bin = true;
   else
      throw new Error("not a PGM file!");
   skip_whites_and_comments(reader);
   xsize = get_num(reader, -1);          /* X size */
   int ch = skip_whites_and_comments(reader);
   ysize = get_num(reader, ch);          /* Y size */
   ch = skip_whites_and_comments(reader);
   depth = get_num(reader, ch);
   if(depth==0)
         throw new Error("Warning: depth=0, probably invalid PGM file\n");
   im = new Image(xsize, ysize, 0);
   for(y=0;y<ysize;y++)
      for(x=0;x<xsize;x++) {
         im.data[x + y * xsize] = bin ? (double) reader.read()
                : (double) get_num(reader, -1);
      }
   return im;
}
```

## Приложение Б. Бинаризация изображения.

В листинге Б1 представлен метод главного класса (Main), реализующий бинаризацию изображения. На вход подаются пороговые значения: пиксели с интенсивностью меньшей этого значения считаются чёрными.

### Листинг Б1

```
public static void binarization(Image imgbiR, Image imgbiG, Image imgbiB, Image
imgR, Image imgG, Image imgB, double threR, double threG, double threB) {
   int wiRB = imgR.xsize;
   int heRB = imgR.ysize;
   int wiG = imgG.xsize;
   int heG = imgG.ysize;
   double red, blue, green;
   for (int i = 0; i < wiRB; i++) {
      for (int j = 0; j < heRB; j++) {
         red = imgR.data[i + j * wiRB];
         if (red <= threR) {
            imgbiR.data[i + j * wiRB] = 0;
         }
         blue = imgB.data[i + j * wiRB];
         if (blue <= threB)
                        imgbiB.data[i + j * wiRB] = 0;
         if (wiG == wiRB && heG == heRB) {
            green = imgG.data[i+j*wiRB];
            if (green <= threG)
                              imgbiG.data[i+j*wiRB] = 0;
         } else {
            green = imgG.data[i*2+1+j*2*imgbiG.xsize];
            if (green <= threG)
                              imgbiG.data[i*2+1+j*2*imgbiG.xsize] = 0;
            green = imgG.data[i*2+j*2*imgbiG.xsize];
            if (green <= threG)
                              imgbiG.data[i*2+j*2*imgbiG.xsize] = 0;
            green = imgG.data[i*2+(j*2+1)*imgbiG.xsize];
            if (green <= threG)
                              imgbiG.data[i*2+(j*2+1)*imgbiG.xsize] = 0;
            green = imgG.data[i*2+1+(j*2+1)*imgbiG.xsize];
            if (green <= threG)
                              imgbiG.data[i*2+1+(j*2+1)*imgbiG.xsize] = 0;
         }
      }
   }
}
```

**Приложение В. Определение чёрной области вокург пикселя.**

В листинге В1 представлен метод класса CCStats (координаты центра диска, его радиусы, площадь, длина окружности), реализующий поиск всех чёрных пикселей вокруг данного.

**Листинг В1**

```
public static int extract_cc_(Pixel p, ArrayList<Pixel> cc, Image img) {
   Stack<Pixel> s= new Stack<>();
   double color; int px, py;
   if (p.x >= 0 && p.x < img.xsize && p.y >= 0 && p.y < img.ysize) {
      color = img.data[p.x + p.y * img.xsize];
      px=p.x;py=p.y;
   } else
      return 0;
   while (color == 0 || !s.empty()) {
      if (color == 0) {
         cc.add(p);
         color=255;
         img.data[p.x + p.y * img.xsize]=255;
            s.push(new Pixel(p.x + 1, p.y));
            s.push(new Pixel(p.x - 1, p.y));
            s.push(new Pixel(p.x, p.y+1));
            s.push(new Pixel(p.x, p.y-1));
            px=p.x;py=p.y;
      }
      if (!s.empty()) {
         p = s.pop();
         if (p.x >= 0 && p.x < img.xsize && p.y >= 0 && p.y < img.ysize)
            color = img.data[p.x + p.y * img.xsize];
         else {
            color = 255;
            img.data[px + py * img.xsize]=255;
         }
      }
   }
   return cc.size();
}
```