Санкт-Петербургский государственный университет

Группа 23.Б12-мм

Линейный фильтр изображений.

Куклин Илья Николаевич

Студенческий проект

Преподаватель:
ассистент кафедры теоретической и прикладной механики, Г.А.Нестерчук

Санкт-Петербург
2024

Оглавление

Введение

Мой пэт-проект представляет из себя линейный фильтр изображения, который работает с изображением, переведенным в двумерный массив и после применяющий к этому массиву три разных фильтра, изменяя само изображение, но не изменяя его размер. Не смотря на то, что фильтры для фотографий довольно распространенная практика, применяемая в всевозможных соц.сетях, не все знают как это работает на практике.

Ещё в середине XX века обработка изображений была по большей части аналоговой и выполнялась оптическими устройствами. Подобные оптические методы до сих пор важны, в таких областях как, например, голография. Тем не менее, с резким ростом производительности компьютеров, эти методы всё в большей мере вытеснялись методами цифровой обработки изображений. Методы цифровой обработки изображений обычно являются более точными, надёжными, гибкими и простыми в реализации, нежели аналоговые методы. В цифровой обработке изображений широко применяется специализированное оборудование, такое как процессоры с конвейерной обработкой инструкций и многопроцессорные системы. В особенной мере это касается систем обработки видео. Обработка изображений выполняется также с помощью программных средств компьютерной математики, например, MATLAB, Mathcad, Maple, Mathematica и др. Для этого в них используются как базовые средства, так и пакеты расширения Image Processing.

Большинство методов обработки одномерных сигналов (например, медианный фильтр) применимо и к двухмерным сигналам, которыми являются изображения. Некоторые из этих одномерных методов значительно усложняются с переходом к двухмерному сигналу. Обработка изображений вносит сюда несколько новых понятий, таких как связность и ротационная инвариантность, которые имеют смысл только для двухмерных сигналов. В обработке сигналов широко используются преобразование Фурье, а также вейвлет-преобразование и фильтр Габора. Обработку изображений разделяют на обработку в пространственной

области (преобразование яркости, гамма коррекция и т. д.) и частотной (преобразование Фурье, и т. д.). Преобразование Фурье дискретной функции (изображения) пространственных координат является периодическим по пространственным частотам с периодом 2pi.

Еще один вариант фильтрации иозображений, это матричный фильтр. Он использует матрицу коэффициентов для изменения цвета или яркости каждого пикселя изображения в зависимости от его окружения. Применяя матрицу фильтра к изображению, можно достичь различных эффектов, таких как увеличение резкости, сглаживание, детектор границ и многие другие.

Процесс применения матричного фильтра состоит из следующих шагов:

Проход по каждому пикселю изображения. Для каждого пикселя формируется окрестность заданного размера (обычно 3x3 или 5x5 пикселей) вокруг него. Для каждого пикселя в окрестности умножаются его значения на соответствующие значения из матрицы фильтра. Результаты умножений суммируются и записываются в значение центрального пикселя. Повторение процесса для всех пикселей изображения. Таким образом, матричный фильтр позволяет выполнять различные операции по обработке изображений, изменяя их внешний вид и улучшая качество. (В моем проекте можно наблюдать вариант использования матричного фильтра в качестве размытия. Который рассматривает матрицу 5x5 для пикселей.)

При работе с фильтрами, кто-то мог представлять себе просто перекрашивание всех пикселей, или работа с их цветами и насыщенностью как в графическом редакторе, по типу Paint. Однако, математика вновь всему голова. Ниже представлена программа, работающая с изображениями формата BMP, которая переводит видимое изображение в двумерный массив, применяет к каждому элементу массива определенное правило и затем переводя массив в уже изменённое изображение.

1. Постановка задачи

Моя программа должна брать изображение, формата BMP, из папки с этой программой и после создавать сразу три изображения.

1. Чтение изображения из общей папки с программой. Перевод изображения в двумерный массив, элементы которого, это пиксели в изображении.

2. Программа применяет три функции (фильтра) к полученному массиву.

3. Программа должна переводить двумерный массив после каждого примененного фильтра, что в сумме даёт три разных вариации одного изображения.

2. Описание кода

2.1. Структура Pixel

```
struct Pixel
{
    unsigned char r, g, b;
};
```

2.2. Чтение изображения. Перевод в двумерный массив.

```
struct Pixel** readBMP(const char* filename, unsigned int* width, unsigned
↪  int* height)
{
    //Здесь создается указатель на файл file, инициализируемый значением
    ↪  0x0 (равное NULL), а так же открываем файл в режиме чтения
    ↪  бинарного файла.
    FILE* file = 0x0;
    fopen_s(&file, filename, ''rb'');
    if (!file)
    {
        fprintf(stderr, ''Невозможно прочитать файл %s\n'', filename);
        return NULL;
    }

    //Создается массив header из 54 элементов типа unsigned char, куда
    ↪  будут считываться первые 54 байта информации из файла BMP (это
    ↪  заголовок BMP).
    unsigned char header[54];

    //Считываем первые 54 байта из файла в массив header, чтобы получить
    ↪  информацию из заголовка BMP.
    fread(header, sizeof(unsigned char), 54, file);

    //Сохраняем ширину и высоту изображения, которая находится в 18-м и
    ↪  22-м байте заголовка (4 байта типа unsigned int).
    *width = *(unsigned int*)&header[18];
    *height = *(unsigned int*)&header[22];
```

```
21
22      //Выделяется память под указатель на указатель image, который будет
        ↪  представлять изображение.
23      struct Pixel** image = (struct Pixel**)malloc(*height * sizeof(struct
        ↪  Pixel*));
24      for (unsigned int i = 0; i < *height; i++)
25      {
26          image[i] = (struct Pixel*)malloc(*width * sizeof(struct Pixel));
27      }
28
29      //Вычисляется необходимое дополнение (padding) для каждой строки
        ↪  изображения, так как длина строки в BMP-файле должна быть кратна
        ↪  4.
30      unsigned int padding = (*width * 3) % 4;
31      if (padding != 0)
32      {
33          padding = 4 - padding;
34      }
35      for (unsigned int y = 0; y < *height; y++)
36      {
37          for (unsigned int x = 0; x < *width; x++)
38          {
39              fread(&image[*height - 1 - y][x], sizeof(struct Pixel), 1,
                ↪  file);
40          }
41          fseek(file, padding, SEEK_CUR);
42      }
43
44      //Закрываем файл и возвращаем указатель на двумерный массив image,
        ↪  содержащий все пиксели изображения.
45      fclose(file);
46
47      return image;
48  }
```

2.3. Запись двумерного массива обратно в изображение.

```
void writeBMP(const char* filename, unsigned int width, unsigned int height, struct Pixel** image)
{
    FILE* file = 0x0;
    fopen_s(&file, filename, ''wb'');
    if (!file)
    {
        fprintf(stderr, ''Невозможно записать файл %s\n'', filename);
        return;
    }

    // Создается массив header из 54 байт, начальные значения которого представляют стандартный заголовок для файла BMP.
    unsigned char header[54] =
    {
            0x42, 0x4D,             // Буквы 'BM' (тип файла)
            0, 0, 0, 0,             // Размер файла (пока неизвестен)
            0, 0,                   // Зарезервированные байты
            0, 0,                   // Зарезервированные байты
            54, 0, 0, 0,            // Смещение данных пикселей в файле
            40, 0, 0, 0,            // Размер заголовка BITMAPINFOHEADER
            width & 0xFF, width >> 8 & 0xFF, width >> 16 & 0xFF, width >> 24 & 0xFF, // Ширина изображения
            height & 0xFF, height >> 8 & 0xFF, height >> 16 & 0xFF, height >> 24 & 0xFF, // Высота изображения
            1, 0,                   // Количество цветовых плоскостей (1)
            24, 0,                  // Битов на пиксель (24 бита)
            0, 0, 0, 0,             // Тип сжатия (нет сжатия)
            0, 0, 0, 0,             // Размер данных изображения (пока неизвестен)
            0, 0, 0, 0,             // Горизонтальное разрешение пикселей в метрах (пока неизвестно)
            0, 0, 0, 0,             // Вертикальное разрешение пикселей в метрах (пока неизвестно)
            0, 0, 0, 0,             // Количество цветов в палитре (пока не используется)
```

```
29              0, 0, 0, 0                  // Количество важных цветов (пока не
                ↪ используется)
30
31      };
32
33      //Вычисляем padding (дополнение до кратности 4) и общий размер данных
        ↪ изображения.
34      unsigned int padding = (width * 3) % 4;
35
36      if (padding != 0)
37      {
38          padding = 4 - padding;
39      }
40
41      //Обновляем значения в заголовке файла BMP для размера файла и размера
        ↪ данных.
42      unsigned int dataSize = (width * 3 + padding) * height;
43
44      *(unsigned int*)&header[2] = 54 + dataSize;
45      *(unsigned int*)&header[34] = dataSize;
46
47      //Записываем заголовок файла в начало файла.
48      fwrite(header, sizeof(unsigned char), 54, file);
49
50      //Записываем данные пикселей изображения в файл с учетом padding для
        ↪ каждой строки.
51      for (unsigned int y = 0; y < height; y++)
52      {
53          for (unsigned int x = 0; x < width; x++)
54          {
55              fwrite(&image[height - 1 - y][x], sizeof(struct Pixel), 1,
                    ↪ file);
56          }
57
58          unsigned char pad = 0;
59
60          for (unsigned int p = 0; p < padding; p++)
61          {
62              fwrite(&pad, sizeof(unsigned char), 1, file);
63          }
```

```
    }

    fclose(file);
}
```

2.4. Фильтр: Повышение яркости.

```
//Функция increaseBrightness, которая принимает ширину и высоту изображения, двумерный массив пикселей image и значение яркости value, на которое нужно увеличить яркость каждого пикселя.
void increaseBrightness(unsigned int width, unsigned int height, struct Pixel** image, unsigned char value)
{
    //Двойной цикл for, который проходит по всем пикселям изображения по строкам (y) и столбцам (x).Увеличиваем значение каждого канала (r,g,b) пикселя на значение value, но если результат превышает 255, то оставляем значение равным 255 (максимальное значение для канала).
    for (unsigned int y = 0; y < height; y++)
    {
        for (unsigned int x = 0; x < width; x++)
        {
            image[y][x].r = image[y][x].r + value > 255 ? 255 : image[y][x].r + value;
            image[y][x].g = image[y][x].g + value > 255 ? 255 : image[y][x].g + value;
            image[y][x].b = image[y][x].b + value > 255 ? 255 : image[y][x].b + value;
        }
    }
}
```

2.5. Фильтр: Черно-белое изображение.

```
// функция применения черно-белого фильтра к изображению
void applyGrayscaleFilter(unsigned int width, unsigned int height, struct Pixel** image)
{
    for (unsigned int y = 0; y < height; y++)
```

```
    {
        for (unsigned int x = 0; x < width; x++)
        {
            image[y][x] = convertToGrayscale(image[y][x]);
        }
    }
}
struct Pixel convertToGrayscale(struct Pixel pixel)
{
    struct Pixel grayscale_pixel;

    //Вычисление среднего значения каналов (r,g,b) цветов текущего
    ↪ пикселя, чтобы получить значение оттенка серого.
    unsigned char average = (pixel.r + pixel.g + pixel.b) / 3;

    grayscale_pixel.r = average;
    grayscale_pixel.g = average;
    grayscale_pixel.b = average;

    return grayscale_pixel;
}
```

2.6. Фильтр: Размытие.

```
void applyLinearFilter(unsigned int width, unsigned int height, struct
↪ Pixel** image) {
    // Копия картинки в памяти
    struct Pixel** filtered_image = (struct Pixel**)malloc(height *
    ↪ sizeof(struct Pixel*));
    for (unsigned int i = 0; i < height; i++)
    {
        filtered_image[i] = (struct Pixel*)malloc(width * sizeof(struct
        ↪ Pixel));
    }

    // Сама фильтрация изображения
    for (unsigned int y = 0; y < height; y++)
    {
        for (unsigned int x = 0; x < width; x++)
        {
```

```
14              // Усреднение как критериальная функция с увеличенным радиусом
15              unsigned int sum_r = 0;
16              unsigned int sum_g = 0;
17              unsigned int sum_b = 0;
18              unsigned int count = 0;
19
20              for (int dy = -5; dy <= 5; dy++)
21              {
22                  for (int dx = -5; dx <= 5; dx++)
23                  {
24                      int nx = x + dx;
25                      int ny = y + dy;
26
27                      if (nx >= 0 && nx < width && ny >= 0 && ny < height)
28                      {
29                          sum_r += image[ny][nx].r;
30                          sum_g += image[ny][nx].g;
31                          sum_b += image[ny][nx].b;
32                          count++;
33                      }
34                  }
35              }
36              // Усредняем и пишем в отфильтрованное изображение
37              filtered_image[y][x].r = sum_r / count;
38              filtered_image[y][x].g = sum_g / count;
39              filtered_image[y][x].b = sum_b / count;
40          }
41      }
42
43
44      // отфильтрованное -> исходное
45      for (unsigned int i = 0; i < height; i++)
46      {
47          for (unsigned int j = 0; j < width; j++)
48          {
49              image[i][j] = filtered_image[i][j];
50          }
51      }
52
53      // Чистим память: отфильтрованное изображение уже не нужно
```

```
    for (unsigned int i = 0; i < height; i++)
    {
        free(filtered_image[i]);
    }
    free(filtered_image);
}
```

2.7. Вызов фильтров в main

```
int main(int argc, char* argv[])
{

    unsigned int width, height;

    //Вызывается функция readBMP, которая считывает изображение из файла
    ↪ ''input.bmp''.
    struct Pixel** image = readBMP(''input.bmp'', &width, &height);
    if (!image)
    {
        return 1;
    }

    //устанавливаем значение, на которое мы увеличиваем яркость.
    unsigned char v = 100;
    increaseBrightness(width, height, image, v);

    //Записывается обработанное изображение в файл ''output1.bmp''.
    writeBMP(''output1.bmp'', width, height, image);

    //Освобождаем память.
    for (unsigned int i = 0; i < height; i++)
    {
        free(image[i]);
    }
    free(image);

    image = readBMP(''input.bmp'', &width, &height);
    applyGrayscaleFilter(width, height, image);
    writeBMP(''output2.bmp'', width, height, image);

```

```
31      for (unsigned int i = 0; i < height; i++)
32      {
33          free(image[i]);
34      }
35      free(image);
36
37      image = readBMP(''input.bmp'', &width, &height);
38
39      //Вызов функции осуществляется несколько раз, для более эффектного
        ↪  размытия.
40      applyLinearFilter(width, height, image);
41      applyLinearFilter(width, height, image);
42      applyLinearFilter(width, height, image);
43      applyLinearFilter(width, height, image);
44      writeBMP(''output3.bmp'', width, height, image);
45
46      return 0;
47  }
```

3. Примеры

Рис. 1: Исходное изображение

Рис. 2: Резльтат Фильтра повышения яркости.

Рис. 3: Результат черно-белого фильтра

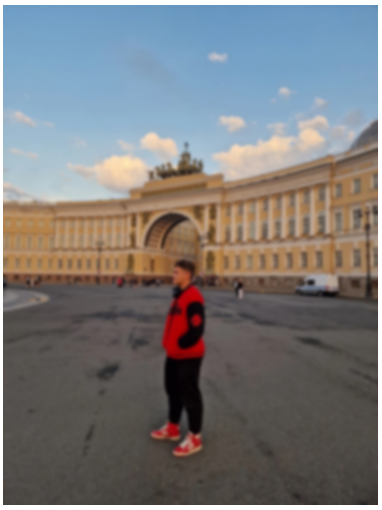

Рис. 4: Результат фильтра размытие

Заключение

Пусть мой проект не являлся чем-то особо уникальным, однако делать его было особо интересно. Раньше, как проходила работа изменения изображения, меня особо не интересовало, ибо важным был лишь результат фильтрации фото. Мой проект является альтернативным взглядом обычного пользователя на столь обыденную функцию телефонов, социальных сетей и графических редакторах.