Лабораторная работа 5

АВП, БГУИР 01/04/2023

1 Введение

1.1 Представление цифровых изображений

Изображение размера NxM представляется в виде матрицы кодирующей интенсивности пикселей $\boldsymbol{I}_{NxM}$. В общем случае изображение может иметь несколько интенсивностей соотвествующих одному пикселю. Так, одноканальное(1). изображение зачастую используется для черно-белых изображений, трех-компонентное(2) для цветных, где r - интенсивность красной компоненты, g - зеленой, b - голубой и четырех-компонентное(3) для цветного с параметром прозрачности.

$$\boldsymbol{I}(x, y) = G \tag{1}$$

$$\boldsymbol{I}(x, y) = \begin{pmatrix} r & g & b \end{pmatrix} \tag{2}$$

$$\boldsymbol{I}(x, y) = \begin{pmatrix} \alpha & r & g & b \end{pmatrix} \tag{3}$$

В памяти же это может быть представленно как сплошной непрерывный массив, описывающий всю матрицу целиком(см. рис. 1), или же как массив указателей на отдельные строки матрицы. Эта лабораторная работа подразумевает первый способ представления. В качестве типа данных зачастую используется $uint8\_t$ кодирующий интенсивность в диапазоне $0 - 255$, однако может $float$, $uint32\_t$ и $bool$.

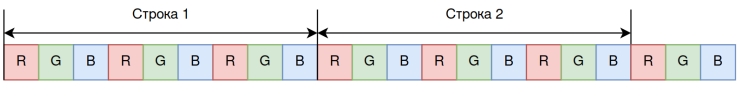

Рис. 1: Представление изображения в памяти

У этого представления есть минус. В идеале, адреса, к которым обращаются, должны быть выравнены на определенную границу, степень двойки. Поскольку каждый пиксель состоит из 3 компонент, адреса последующих пикселей идут с шагом кратным 3, а адреса последующей строки начинаются с невыравненного адреса, что сдвигает все пиксели в строке. Чтобы решить эту проблему, в конец каждой строки можно добавить немного неиспользуемого пространства(padding), которое гарантирует выравнивание следующей(см. рис. 2). Для этого CUDA имеет в своем API ряд функций: $cudaMallocPitch$ - для выделения памяти с выравниванием по строкам, $cudaMemcpy2D$ для копирования из сплошного участка памяти в участок с padding'ами и наоборот.

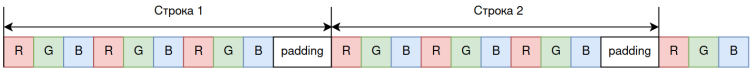

Рис. 2: Представление изображения в памяти с выравниванием

2 Операции над изображениями

2.1 Поэлементные преобразования

Простейшим видом операции над изображениями является поэлементное преобразование. В таком случае результат преобразования зависит только от одного пикселя - входного(4).

$$\boldsymbol{I_o}(x, y) = f(\boldsymbol{I}(x, y)) \tag{4}$$

Также в общем виде могут использоваться несколько входных изображений(5).

$$\boldsymbol{I_o}(x, y) = f(\boldsymbol{I_1}(x, y), \boldsymbol{I_2}(x, y)) \tag{5}$$

В данной ЛР будет рассмотрено пороговое преобразование для обнаружение пикселей определнного цвета $\boldsymbol{C} = \begin{pmatrix} r_0 & g_0 & b_0 \end{pmatrix}$. Данное преобразование принимает трех-цветное изображение и генерирует одноканальное изображение типа *bool*. Для этого нужно сравнить разницу(6) между входным пикселей и целевым $\boldsymbol{C}$, и максимально допустимой разницей *threshold*.

$$\boldsymbol{I_o}(x, y) = (||\boldsymbol{I}(x, y)) - \boldsymbol{C}|| \leq threshold) \tag{6}$$

2.2 Преобразование Хафа

Преобразование Хафа(Hough Transform) используется для обнаружения паттернов, которые можно описать с помощью аналитического метода(с помощью формул). Рассмотрим преобразование на примере поиска линий на изображении. На вход подается матрица описывающая точки интереса, например, после порогового преобразования. Линия описываются уравнением $y = kx + b$. x и y - переменные или же координаты пикселя на изображении, а k и b - постоянные описывающие данную линию. Для изображения существует конечное число линий, которые можно провести на изображении, а значит и конечное количество параметров. Линию также можно описать в полярной системе координат используя длину нормали и ее угол(см. 3).

$$\rho(\theta) = x * cos(\theta) + y * sin(\theta) \tag{7}$$

Таким образом получается конечное значение радиусов(от 0 до M+N) и конечное количество возможных углов, например от $-180°$ до $180°$. Можно перебрать все возможные варианты для целевых точек и сохранить в массив аккумулятор, какие прямые можно провести сквозь них. Таким образом точки лежащие на одной прямой создадут больше откликов для пары (ρ, θ). После этого по массиву ищется максимальные значение, а их местоположение будет описывать конкретную прямую. Псевдокод для реализации представлен далее.

Для GPU реализации в качестве алгоритма поиска максимума можно использовать модифицированный Parallel Reduce для двухмерного(или трехмерного случая). В случае поиска окружности(8) на изображении поиск происходит для параметров x_0 и y_0 описывающих

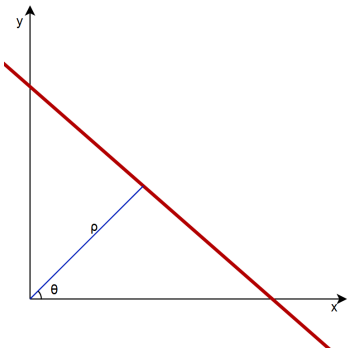

Рис. 3: Задание прямой в полярной системе координат

Algorithm 1 Псевдокод преобразования Хафа для CPU

Accumulator $\leftarrow \mathbf{0}_{N_\rho x N_\theta}$ ▷ Изначально пустая матрица для всех возможных комбинаций ρ и θ
for `x` $\subset$ `[0; N)` do
  for `y` $\subset$ `[0; M)` do
    if $\boldsymbol{I}(x, y) \neq true$ then ▷ Не интересующие пиксели пропускаются
      continue
    end if
    for $\theta \subset$ `[-180; 180)` do
      $\rho = x * cos(\frac{\theta}{180} * \pi) + y * sin(\frac{\theta}{180} * \pi)$
      $Accumulator[\rho][180 + \theta] += 1$
    end for
  end for
end for
$(Best\rho, Best\theta) \leftarrow argmax Accumulator$

центр окружности и R_0 - радиуса, т.е. для 3 компонентов.

$$(x - x_0)^2 + (y - y_0)^2 = R_0^2 \tag{8}$$

2.3 Афинные и эластичные преобразования

Данный тип преобразований прямым образом не оперерирует над интенсивностью пикселей, он оперерует их местоположением(9). Для этого задается функция соответствия. Функция соответствия определяет, где пиксель с координатами (x, y) находился в старом изображении.

$$\boldsymbol{I_o}(x, y) = \boldsymbol{I}(f(x, y)) \tag{9}$$

2.3.1 Афинное преобразование

Простейшее преобразование сдвигающее пиксель на dx вправо или влево по X, на dy вверх или вниз по Y или одновременно:

$$f(x, y) = \{x + dx, y + dy\} \tag{10}$$

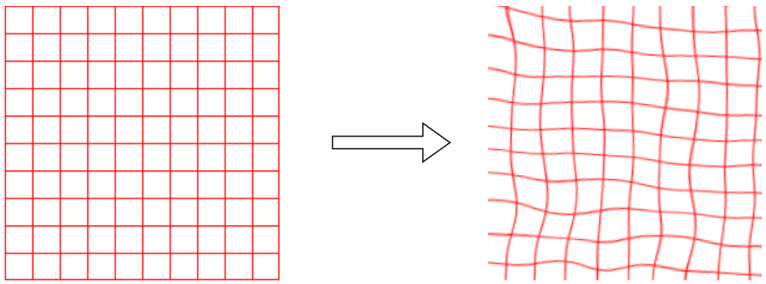

Рис. 4: Пример деформирующего преобразования

2.3.2 Масштабирующее преобразование

Чтобы увеличить или уменьшить размер используется масштабирующее преобразование, где k_x - коэффициент масштабирования по горизонтали, k_y - коэффициент масштабирования по вертикали:

$$f(x, y) = \{x * k_x, y * k_y\} \tag{11}$$

2.3.3 Преобразование вращения вокруг центра координат

Данное преобразование вращает координату пикселя вокруг оси координат на угол θ.

$$x\prime = x * cos(\theta) - y * sin(\theta)$$
$$y\prime = y * cos(\theta) + x * sin(\theta)$$

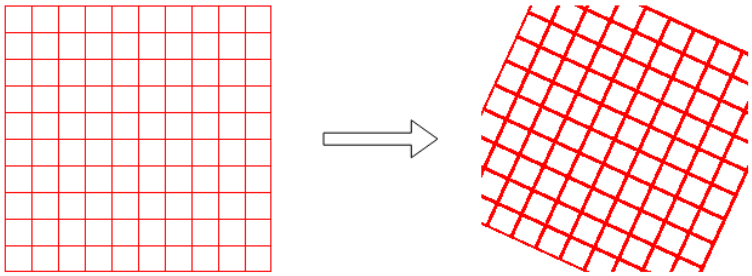

Рис. 5: Пример вращающего преобразования

2.3.4 Преобразование рыбьего глаза

Данное преобразование убирает или добавляет оптические искажения от линзы на основании параметра Ω. Для этого сперва начало координат переносится в центр изображения и нормируется таким образом, что координаты из диапозона $[0; N)$ стали $[-1; 1]$:

$$x_0 = (x - \frac{N+1}{2})) * \frac{2}{N}$$
$$y_0 = (y - \frac{M+1}{2})) * \frac{2}{M}$$

Далее для каждого пикселя считается расстояние до центра и угол:

$$r = \sqrt{x_0^2 + y_0^2}$$
$$\theta = atan2(y_0, x_0)$$

Далее высчитывается коэффициент для каждого пикселя:

$$scale = min(\frac{1}{|cos(\theta) + \epsilon|}, \frac{1}{|sin(\theta) + \epsilon|}) \tag{12}$$

Далее высчитывается новый радиус для каждого пикселя:

$$r\prime = min(scale, 1) * r^{\Omega} \tag{13}$$

И наконец новые координаты высчитываются по формуле

$$x\prime = \frac{N}{2} * r\prime * cos(\theta) + \frac{N+1}{2}$$
$$y\prime = \frac{M}{2} * r\prime * sin(\theta) + \frac{M+1}{2}$$

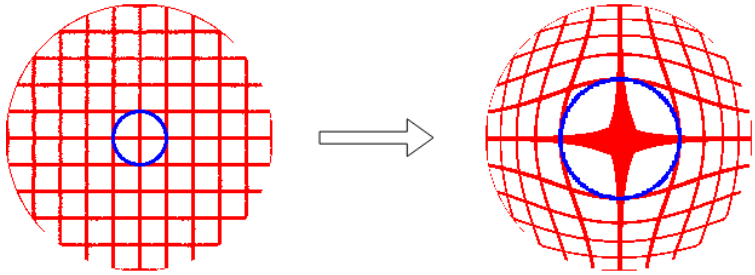

Рис. 6: Преобразование 'рыбий глаз'

2.4 Методы интерполяции

При применении подобных преобразований могут возникать дробные значения координат. Поскольку цифровое изображение представляет из себя матрицу с дискретными целыми индексами, то возникает вопрос обработки подобных случаев. Например, какой цвет должен иметь пиксель с координатой $(0.8; 0.3)$? На этот вопрос отвечают алгоритмы интерполяции.

2.4.1 Интерполяция до ближайшего соседа(NEAREST)

Исходя из названия понятно, что берется цвет ближайшего соседа. В данном случае это будет $(0; 1)$ - т.е. зеленый.

$$\boldsymbol{I_o}(x, y) = \boldsymbol{I}([x\prime], [y\prime]) \tag{14}$$

2.4.2 Билинейная интерполяция(BILINEAR)

При данном виде интерполяции подразумевается линейная зависимость цвета от координаты. Поскольку изображение представляет из себя двумерную функцию интерполяция проводится в два этапа:

- Используя алгоритм интерполяции определяется цвет вспомогательных пикселей, например, по вертикали(см. 9)
- Алгоритм применяется уже для вспомогательных пикселей

Алгоритм не зависит от направления и может применяться в произвольном порядке. Из-за этого его можно сделать не зависимым от координат(вместо t будет подставлен и x, и y). t_0 и t_1 - ближайшие известные пиксели, $\boldsymbol{C_0}$ и $\boldsymbol{C_1}$ - их цвета соответственно.

$$\alpha = \frac{t_1 - t}{t_1 - t_0} \tag{15}$$

$$\boldsymbol{C} = \boldsymbol{C_0} * \alpha + (1 - \alpha) * \boldsymbol{C_1} \tag{16}$$

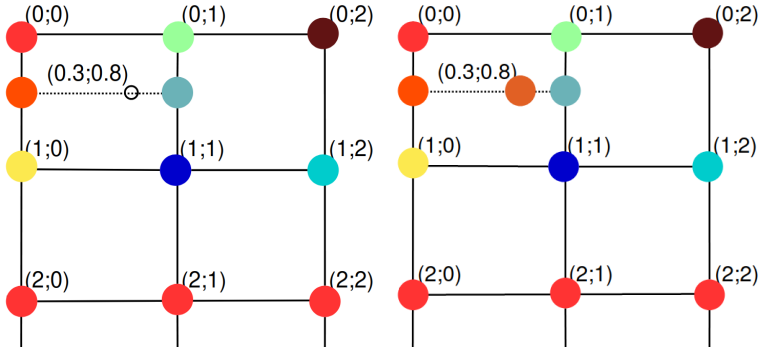

Рис. 7: Билинейная интерполяция

В случае, если пиксель выходит за пределы изображения, то берется либо один из крайних пикселей, либо черный(или белый) цвет.

Рис. 8: Сравнение алгоритмов интерполяции. Слева направо: оригинал, NEAREST, BILINEAR

3 Задание

3.1 Условие

Для выполнения потребуется наличие видеокарты. Если в бригаде есть видеокарта NVIDIA - работа выполняется на ней с помощью технологии CUDA. Для видеокарт от AMD, Intel, Apple - OpenCL. Язык выполнения - C/C++.

1. Установить библиотеку OpenCV(stb_image , или другую обеспечивающую чтение и запись изображений).
2. Согласно варианту реализовать алгоритм восстановления изображения:

   (a) Используя пороговое преобразования обнаружить маркер цвета $\boldsymbol{C}$ согласно варианту.
   (b) Используя преобразования Хафа определить параметры маркера;
   (c) Используя выданное преобразование восстановить изображение.

   Реализация - на GPU. Для отладки приветствуется реализация на CPU. Разрешается использовать OpenCV или другие вспомогательные библиотеки(и языки).

3.2 Варианты

1. Маркер - зеленая($\boldsymbol{C} = \begin{pmatrix} 0 & 255 & 0 \end{pmatrix}$) линия. Линия отклонена на $[-\pi; \pi]$ радиан от горизонта. После обратного преобразования она должна проходит строго горизонтально и делить изображение на две равные части. Метод интерполяции - LINEAR.

2. Маркер - красная($\boldsymbol{C} = \begin{pmatrix} 255 & 0 & 0 \end{pmatrix}$) окружность, центр которого совпадает с центром изображения. Преобразование - рыбий глаз. После обратного преобразования должна получиться радиус окружность должен быть равен 10% от меньшей стороны изображения. Метод интерполяции - LINEAR.

3. Маркер - желтый ($\boldsymbol{C} = \begin{pmatrix} 255 & 255 & 0 \end{pmatrix}$) прямоугольник со сторонами параллельными сторонам изображения, центр которого может не совпадать с центром изображения и соотношением сторон 2к1(ширина к высоте), полученный после вращения, масштабирования и сдвига. В результате обратного преобразования должен получиться повернутый на 30° квадрат с длиной стороны в 10% от меньшей стороны изображения. Метод интерполяции - NEAREST.

4. Маркер - синяя($\boldsymbol{C} = \begin{pmatrix} 0 & 0 & 255 \end{pmatrix}$) линия. Линия отклонена на $[-\pi; \pi]$ радиан от вертикали. После обратного преобразования она должна проходит строго вертикально и делить изображение на две равные части. Метод интерполяции - LINEAR.

5. Маркер - окружность цвета $\boldsymbol{C} = \begin{pmatrix} 255 & 0 & 255 \end{pmatrix}$, центр которого совпадает с центром изображения. Преобразование - рыбий глаз. После обратного преобразования должна получиться радиус окружность должен быть равен 10% от меньшей стороны изображения. Метод интерполяции - LINEAR.

6. Маркер - прямоугольник цвета $\boldsymbol{C} = \begin{pmatrix} 0 & 255 & 255 \end{pmatrix}$ со сторонами параллельными сторонам изображения, центр которого может не совпадать с центром изображения и соотношением сторон 2к3(ширина к высоте), полученный после вращения, масштабирования и сдвига. В результате обратного преобразования должен получиться повернутый на 55° квадрат с длиной стороны в 25% от меньшей стороны изображения. Метод интерполяции - NEAREST.

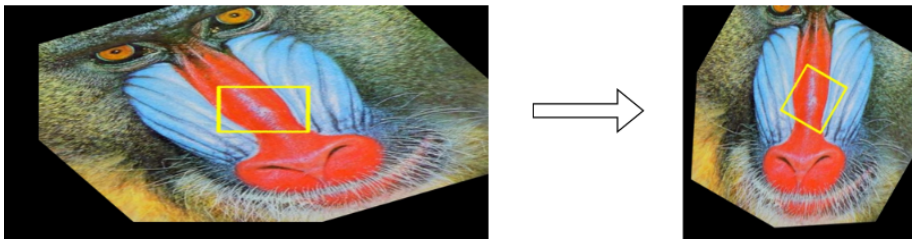

Рис. 9: Входное и выходное изображение для варианта 3

3.3 Требования и допущения

- Использование pitched памяти;
- Использование shared памяти для всех видов интерполяции;
- Запрещено использовать текстурную память, шейдеры;
- Преобразования, за исключением масштабирующего, не меняет размер изображения;
- Масштабирующее преобразование `меняет размер` изображения;
- Коэффициент масштабирование k для каждой оси $\frac{1}{3} \leq k \leq 3$;
- Не более $2^{32} - 1$ пикселей в изображении(для преобразования Хафа координату пикселя можно закодировать вместе со значением аккумулятора в $uint64\_t$).