Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский университет ИТМО»
(Университет ИТМО)

Факультет программной инженерии и компьютерной техники

Параллельные вычисления

Лабораторная работа №1

Вариант 2 -Усредняющий фильтр Гаусса

Преподаватель: Жданов Андрей Дмитриевич

Выполнили: Демьяновский Савелий Игоревич,
Юров Максим Алексеевич,
P4115

Санкт-Петербург
2024

Содержание

Цель работы

Научиться реализовывать один из простых алгоритмов обработки изображений.

Текст программы

Программа состоит из нескольких файлов:
main.py:

```
if __name__ == '__main__':
    image_path = 'image.jpg'
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    kernel_size = 19
    sigma = 3

    plt.figure(figsize=(10, 10))

    plt.subplot2grid( shape: (2, 2),  loc: (0, 0), colspan=2)
    show_image(plt, image,  title: 'Original image.')

    plt.subplot2grid( shape: (2, 2),  loc: (1, 0))
    cv_image, cv_duration = with_open_cv(image, kernel_size, sigma)
    show_image(plt, cv_image,  title: 'With open CV.', cv_duration)

    plt.subplot2grid( shape: (2, 2),  loc: (1, 1))
    alg_image, alg_duration = without_open_cv(image, kernel_size, sigma)

    show_image(plt, alg_image,  title: 'Custom alg.', alg_duration)

    plt.tight_layout()
    plt.show()
```

В этом файле получаем изображение, делаем его черно-белым, обрабатываем его встроенной в openCV функцией усредняющего фильтра Гаусса и нашим самописным

алгоритмом, который делает тоже самое. Далее выводим оригинал и 2 обработанных изображения.

Результат работы:

Original image.

With open CV. duration: 0.0009019374847412109

Custom alg. duration: 1.8122191429138184

without_open_cv.py:

```python
def gaussian_kernel(size, sigma):  1 usage
    kernel = np.zeros( shape: (size, size), dtype=float)
    center = size // 2
    sum_val = 0.0

    for i in range(size):
        for j in range(size):...

    kernel /= sum_val

    return kernel


def apply_gaussian_blur(image, kernel):  1 usage
    img_height, img_width = image.shape
    blurred_image = np.zeros_like(image, dtype=float)
    kernel_size = kernel.shape[0]
    pad = kernel_size // 2

    padded_image = np.pad(image, ((pad, pad), (pad, pad)), mode='constant', constant_values=0)

    for i in range(img_height):
        for j in range(img_width):
            region = padded_image[i:i + kernel_size, j:j + kernel_size]
            blurred_image[i, j] = np.sum(region * kernel)

    return blurred_image[pad:pad + img_height, pad:pad + img_width]


def without_open_cv(image, kernel_size, sigma, only_duration=False):  4 usages
    start_time = time.time()
    res_image = apply_gaussian_blur(image, gaussian_kernel(kernel_size, sigma))
    duration = time.time() - start_time

    if only_duration:
        return duration

    return res_image, duration
```

Здесь реализован наш алгоритм усредняющего фильтра Гаусса, в *gaussian_kernel* создаём ядро по заданному размеру и сигме, в *apply_gaussian_blur* обрабатываем массив картинки по этому с помощью этого ядра, а в *without_open_cv* запускаем все функции, измеряем их время работы и отдаём на выход обработанную картинку и длительность работы алгоритма. Также тут есть работа с флагом *only_duration,* это нужно для работы benchmark.py, об этом будет далее.

with_open_cv.py:

```python
def with_open_cv(image, kernel_size, sigma, only_duration=False):
    start_time = time.time()

    res_image = cv.GaussianBlur(src=image,
                                ksize=(kernel_size, kernel_size),
                                borderType=cv.BORDER_DEFAULT,
                                sigmaX=sigma,
                                sigmaY=sigma)

    duration = time.time() - start_time

    if only_duration:
        return duration

    return res_image, duration
```

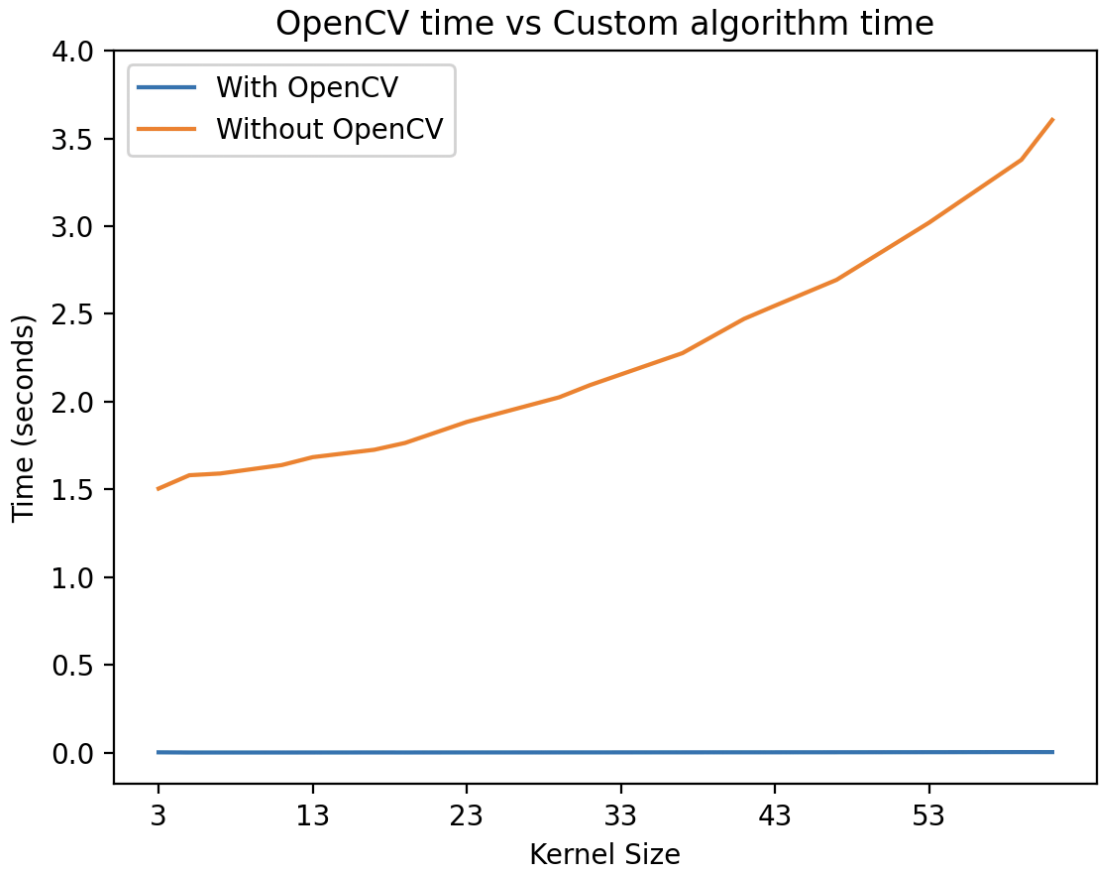
Здесь работает все также, как и в **without_open_cv.py,** но алгоритм обработки взят из библиотеки openCV.

benchmark.py:

```python
def start_benchmark():  1 usage
    image_path = 'image.jpg'
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    sigma = 4
    k_size = 61

    primes = [x for x in range(1, k_size + 1) if is_prime(x) and x % 2 != 0]

    times_with_cv = []
    times_without_cv = []

    for kernel_size in primes:
        time_cv = with_open_cv(image, kernel_size, sigma,  only_duration: True)
        time_no_cv = without_open_cv(image, kernel_size, sigma,  only_duration: True)

        times_with_cv.append(time_cv)
        times_without_cv.append(time_no_cv)

    plt.plot( *args: primes, times_with_cv, label='With OpenCV')
    plt.plot( *args: primes, times_without_cv, label='Without OpenCV')
    plt.xlabel('Kernel Size')
    plt.ylabel('Time (seconds)')
    plt.title('OpenCV time vs Custom algorithm time')
    plt.legend()

    plt.xticks(np.arange(min(primes), max(primes) + 1, 10))
    plt.yticks(np.arange(0, max(times_without_cv + times_with_cv) + 0.5, 0.5))

    plt.show()


if __name__ == '__main__':
    start_benchmark()
```

В этом файле мы создаём массив простых нечетных чисел - это будут размеры ядер для сравнительного теста нашего алгоритма и библиотечного. Мы просто запускаем все те же функции, описанные выше, но с флагом *only_duratuon,* он говорит функциям отдавать только длительность работы алгоритма, без картинки.

Далее мы просто строим графики зависимости размера ядра и времени для двух алгоритмах, результат ниже:



Результаты работы

По графику из **benchmark.py** видно, что наш алгоритм гораздо менее эффективен, нежели библиотечный, он в целом квадратичный, что логично: мы смотрим на k^2 ячеек каждый раз при сглаживании пикселя, где k-размер ядра.
Библиотечная функция гораздо быстрее за счет того, что под капотом огромное количество оптимизаций - от того, что код библиотеки написан не на Python, а на C++ и параллелится, до в целом более алгоритмически эффективной обработки.

Вывод

В результате выполнения работы мы реализовали один из простых алгоритмов обработки изображений и сравнили его с готовым библиотечным вариантом.

Источники

https://habr.com/ru/articles/324070/