Report Advanced Programming for HPC

Ha Ngoc Linh - M21.ICT006

January 8, 2023

1 Formula

1.1 RGB to HSV

$$H = \begin{cases} 0^\circ & \Delta \ = 0 \\ 60^\circ \times \frac{G-B}{\Delta} mod 6 & max \ = R \\ 60^\circ \times \frac{B-R}{\Delta} + 2 & max \ = G \\ 60^\circ \times \frac{R-G}{\Delta} + 4 & max \ = B \end{cases} \tag{1}$$

$$S = \begin{cases} 0 & \text{max} = 0 \\ \frac{\Delta}{max} & max \ \neq 0 \end{cases} \tag{2}$$

$$V = max \tag{3}$$

1.2 Kuwahara filter

1. Get V value in HSV space of the image.
2. Define window size (region size) (ω).
3. In each region:
   - Calculate the summary value of all pixels in the region to get the mean value
   $$mean = \frac{\sum r(x,y)}{\omega \times \omega} \tag{4}$$
   - Calculate the standard deviation of the region i
   $$std_i = \sqrt{\frac{\sum((V(x,y) - mean)^2)}{\omega \times \omega}} \tag{5}$$
4. Find the minimum value in 4 standard deviation values (4 regions)
5. Assign mean(R,G,B) value of this region as new color
$$R = avg_R = \frac{\sum(R(x,y)}{\omega \times \omega} \tag{6}$$
$$B = avg_B = \frac{\sum(B(x,y)}{\omega \times \omega} \tag{7}$$
$$G = avg_G = \frac{\sum(G(x,y)}{\omega \times \omega} \tag{8}$$

2 Kuwahara filter in CPU

2.1 Transfer RGB to HSV in CPU

To transfer RGB to HSV, I follow the formula (1), (2), (3) on each pixel of the image:

```
height, width = src.shape[0], src.shape[1]
for tidx in range(height):
    for tidy in range(width):
```

I calculate H, S, V values of each pixel by following this code:

```
b = src[tidx, tidy, 0] / 255
g = src[tidx, tidy, 1] / 255
r = src[tidx, tidy, 2] / 255
cmax = max(r, g, b)
cmin = min(r, g, b)
delta = cmax - cmin
if delta == 0:
    h = 0
elif cmax == r:
    h = ((((g - b) / delta) % 6) * 60)  % 360
elif cmax == g:
    h = ((((b - r) / delta) + 2) * 60) % 360
elif cmax == b:
    h = ((((r - g) / delta) + 4) * 60) % 360
if cmax == 0:
    s = 0
else:
    s = delta / cmax
v = cmax

dst[tidx, tidy, 0] = h % 360
dst[tidx, tidy, 1] = s * 100
dst[tidx, tidy, 2] = v * 100
```

2.2 Kuwahara filter function in CPU

Following each step defined in section 1.2

- To get V value

  ```
  v = hsv[:, :, 2]
  ```

- window_size will be passed into the Kuwahara filter function with integer type value
- On each region, the standard deviation value will be calculated by following code

  ```
  # first region  rx1 belongs to [tidx - window_size, tidx] and ry1 belongs to
  sum1 = np.float32(0)
  for rx1 in range(tidx - window_size, tidx):
      for ry1 in range(tidy - window_size, tidy):
          if rx1 >= 0 and ry1 >= 0 and rx1 < height and ry1 < width:
              sum1 += v[rx1, ry1]
  # find standard deviation of first region
  mean1 = sum1 / ((window_size) * (window_size))
  sum1 = np.float32(0)
  for rx1 in range(tidx - window_size, tidx):
      for ry1 in range(tidy - window_size, tidy):
          if rx1 >= 0 and ry1 >= 0 and rx1 < height and ry1 < width:
              sum1 += (v[rx1, ry1] - mean1) ** 2
  std1 = math.sqrt(sum1 / ((window_size) * (window_size)))
  ```

The calculation is the same for 3 regions left by looping all the pixels of the image

```
height, width = src.shape[0], src.shape[1]
for tidx in range(height):
    for tidy in range(width):
```

- finding the minimum standard deviation value

```
min_std = min(std1, std2, std3, std4)
```

- Calculate the mean of RGB space color of the region as new color

```
# assign the mean of the region with minimum standard deviation to the pixel
avg_r = np.float32(0)
avg_g = np.float32(0)
avg_b = np.float32(0)

if min_std == std1:
    for rx1 in range(tidx - window_size, tidx):
        for ry1 in range(tidy - window_size, tidy):
            if rx1 >= 0 and ry1 >= 0 and rx1 < height and ry1 < width:
                avg_r += src[rx1, ry1, 2]
                avg_g += src[rx1, ry1, 1]
                avg_b += src[rx1, ry1, 0]
if min_std == std2:
    for rx2 in range(tidx, tidx + window_size):
        for ry2 in range(tidy - window_size, tidy):
            if rx2 >= 0 and ry2 >= 0 and rx2 < height and ry2 < width:
                avg_r += src[rx2, ry2, 2]
                avg_g += src[rx2, ry2, 1]
                avg_b += src[rx2, ry2, 0]

if min_std == std3:
    for rx3 in range(tidx - window_size, tidx):
        for ry3 in range(tidy, tidy + window_size):
            if rx3 >= 0 and ry3 >= 0 and rx3 < height and ry3 < width:
                avg_r += src[rx3, ry3, 2]
                avg_g += src[rx3, ry3, 1]
                avg_b += src[rx3, ry3, 0]

if min_std == std4:
    for rx4 in range(tidx, tidx + window_size):
        for ry4 in range(tidy, tidy + window_size):
            if rx4 >= 0 and ry4 >= 0 and rx4 < height and ry4 < width:
                avg_r += src[rx4, ry4, 2]
                avg_g += src[rx4, ry4, 1]
                avg_b += src[rx4, ry4, 0]

avg_r = avg_r / (window_size * window_size)
avg_g = avg_g / (window_size * window_size)
avg_b = avg_b / (window_size * window_size)

dst[tidx, tidy, 2] = avg_r
dst[tidx, tidy, 1] = avg_g
dst[tidx, tidy, 0] = avg_b
```

3 Kuwahara filter in GPU

3.1 RGB to HSV in GPU

With the same formula (1), (2), (3) but using GPU Instead of using for loop, cuda will be used for processing

```
tidx = cuda.threadIdx.x + cuda.blockIdx.x * cuda.blockDim.x
tidy = cuda.threadIdx.y + cuda.blockIdx.y * cuda.blockDim.y
```

The formula is the same as the CPU function so in GPU, there is no much different in this part

```
b = src[tidx, tidy, 0] / 255
g = src[tidx, tidy, 1] / 255
r = src[tidx, tidy, 2] / 255
cmax = max(r, g, b)
cmin = min(r, g, b)
delta = cmax - cmin
if delta == 0:
    h = 0
elif cmax == r:
    h = ((((g - b) / delta) % 6) * 60)  % 360
elif cmax == g:
    h = ((((b - r) / delta) + 2) * 60) % 360
elif cmax == b:
    h = ((((r - g) / delta) + 4) * 60) % 360
if cmax == 0:
    s = 0
else:
    s = delta / cmax
v = cmax

dst[tidx, tidy, 0] = h % 360
dst[tidx, tidy, 1] = s * 100
dst[tidx, tidy, 2] = v * 100
```

3.2 Kuwahara filter in GPU without shared memory

With the same formula (4), (5), (6), (7), (8) of using-CPU function, the code for processing filtering Kuwahara with GPU is not much different from using CPU The first step is getting V value from the image in HSV space. This step is not different from the CPU function Instead of using loop, by using cuda, we have:

```
tidx = cuda.threadIdx.x + cuda.blockIdx.x * cuda.blockDim.x
tidy = cuda.threadIdx.y + cuda.blockIdx.y * cuda.blockDim.y
```

The process on each region is not different from the CPU function. (The code is the same as the section 2.2)

3.3 Kuwahara filter in GPU with shared memory

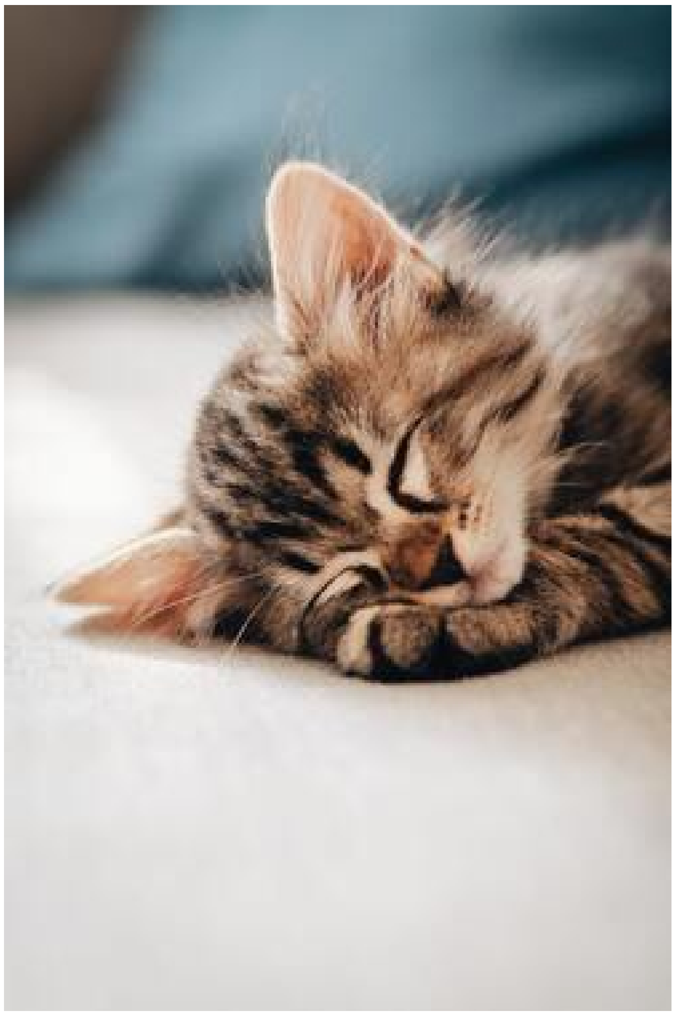

Figure 1: original image

4 Comparing Kuwahara function between using CPU and GPU

Using the same image to apply Kuwahara filter in both CPU and GPU:

Apply Kuwahara filter with CPU function: (the timer is added to get the running time)

```
# start timer
start = time.time()
filtered = kuwahara_filter_hsv(root, v, 8)
# end timer
end = time.time()
print("Time taken for kuwahara filter in CPU: ", end - start)
```

Apply Kuwahara filter with GPU function: (the timer is added to get the running time)

```
blockSize = (16, 16)
gridSize = (math.ceil(gpu_img.shape[0] / blockSize[0]), math.ceil(gpu_img.shape[1] / bloc
output_cuda_image_data = cuda.device_array(np.shape(gpu_img), np.uint8)
# start timer
start = time.time()
kuwahara_filter_hsv[gridSize, blockSize](gpu_img, output_cuda_image_data, v, 8)
# end timer
end = time.time()
print("Time taken for kuwahara in GPU: ", end - start)
```

In both function, the window_size was be assigned by 8

The result of 2 functions :

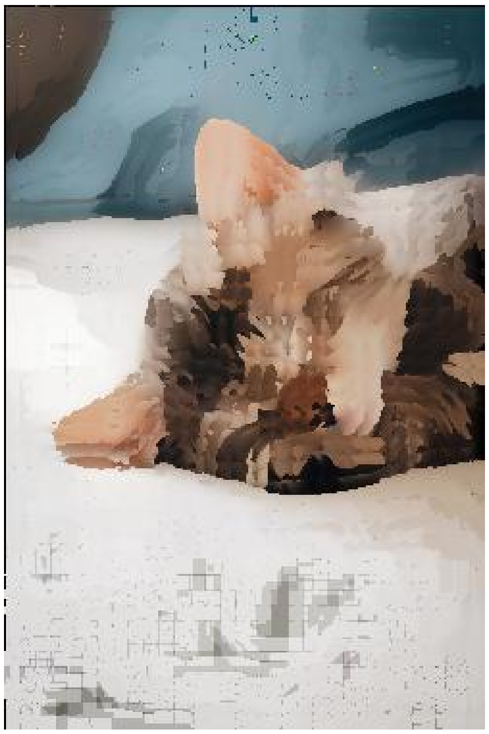

Figure 2: Kuwahara filter in CPU

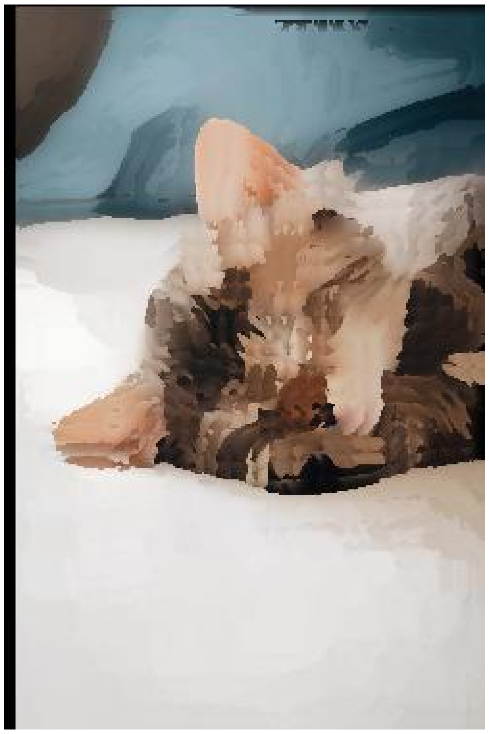

Figure 3: Kuwahara filter in GPU

There is a little bit difference between 2 output images. The reason may come from variable type and how I detect the position of each region

The running time of each functions are:

- CPU: 87.74110865592957s
- GPU: 1.6664512157440186s

The running time is affected by the size of the image (in this case, the image size is 250*377 and the file size is 14,486 bytes). In this test, I run the on google Colab free version. But it is clear to see that in GPU, the performance is much better than CPU