UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI **GRADUATE SCHOOL**

USTH-M21-012
Information and Communication Technology

Title:

Advanced programming for HPC Part A

Final Project Report

by

NGO THE TU
M21.ICT.012

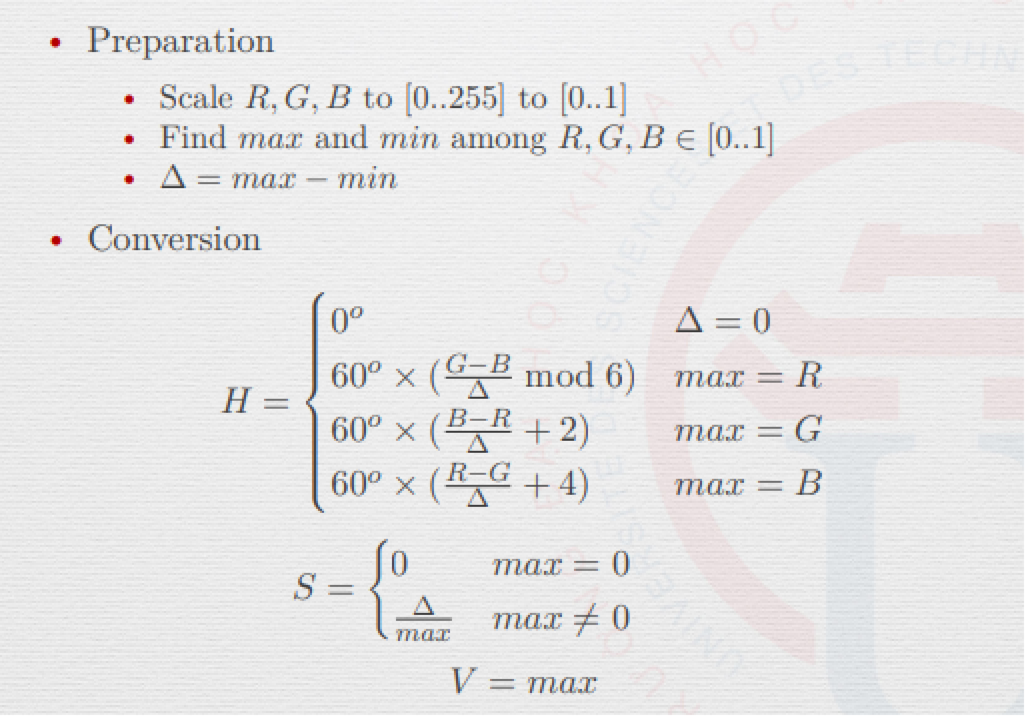
Hanoi, January 2023

1. Summary

The final project required implementing the Kuwahara filter into the project. The Kuwahara filter is a non-linear smoothing filter used for adaptive noise reduction in image processing. In order to successfully reduce noise and blur edges, linear low-pass filters make up the majority of picture smoothing filters. However, the Kuwahara filter can blur the image while maintaining sharp edges.

2. Formula

2.1 : RGB to HSV



- Preparation
  - Scale R, G, B to $[0..255]$ to $[0..1]$
  - Find max and min among $R, G, B \in [0..1]$
  - $\Delta = max - min$
- Conversion

$$H = \begin{cases} 0^o & \Delta = 0 \\ 60^o \times (\frac{G-B}{\Delta} \bmod 6) & max = R \\ 60^o \times (\frac{B-R}{\Delta} + 2) & max = G \\ 60^o \times (\frac{R-G}{\Delta} + 4) & max = B \end{cases}$$

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

$$V = max$$

3.Kuwahara filter in CPU

3.1 : Convert RGB to HSV in CPU

Follow the formula , i write function convert RGB to HSV

```
def rgb_to_hsv_cpu(src):
    height, width = src.shape[0], src.shape[1]
    dst_hsv = np.zeros((height, width, 3), dtype=np.uint8)
    for tidx in range(height):
        for tidy in range(width):
            r = src[tidx, tidy, 2]/255
            g = src[tidx, tidy, 1]/255
            b = src[tidx, tidy, 0]/255

            c_max = max(r, g, b)
            c_min = min(r, g, b)
            df = c_max - c_min
            if df == 0:
                h = 0
            elif c_max == r:
                h = ((((g - b) / df) % 6) * 60)  % 360
            elif c_max == g:
                h = ((((b - r) / df) + 2) * 60) % 360
            elif c_max == b:
                h = ((((r - g) / df) + 4) * 60) % 360
            if c_max == 0:
                s = 0
            else:
                s = df / c_max
            v = c_max

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

3.2 : Kuwahara in CPU

- First, i get V value :

  ```
  v_hsv = hostOutput1[:,:,2]
  ```

- size will be passed into the Kuwahara filter function with integer type value
- The following code will be used to calculate the standard deviation value for each region.

```
for i in range(height):
    for j in range(width):

Sizes=(((i-size,i),(j-size,j)),((i,i+size),(j-size,j)),((i-size,i),(j,j+size)),((i,i+size),(j,j+size)))

sum1 = 0
totalsum = 0
for i in range(sizes[0][0]):
    for j in range(sizes[0][1]):
        sum1 = sum1 + (v[i,j])
        totalsum = totalsum + (v[i,j] * v[i,j])
        resultdev1 = math.sqrt(abs(totalsum / (size*2) - (sum1 / (size *2)) *2))
```

- By looping through all of the pixels in the image, the calculation is the same for the 3 regions that are left.

- Finding the minimum standard deviation value

  ```
  Min = min(resultdev1,resultdev2,resultdev3,resultdev4)
  ```

4. Kuwahara filter in GPU

4.1 : Convert RGB to HSV in GPU

Follow the formula , i write function convert RGB to HSV

```
@cuda.jit(device=True)
def rgb_to_hsv_gpu(src, dst):
  tidx = cuda.threadIdx.x + cuda.blockIdx.x * cuda.blockDim.x
```

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

r = src[tidx, tidy, 2]/255
g = src[tidx, tidy, 1]/255
b = src[tidx, tidy, 0]/255

tidMax = max(r, g, b)
tidMin = min(r, g, b)

df = tidMax - tidMin

if df == 0:
    dst[tidx, tidy, :] = 0
elif r == tidMax:
    dst[tidx, tidy, 0] = ((((g - b) / df) % 6) * 60) % 360
elif g == tidMax:
    dst[tidx, tidy, 0] = ((((b - r) / df) + 2) * 60) % 360
elif b == tidMax:
    dst[tidx, tidy, 0] = ((((r - g) / df) + 4) * 60) % 360

if tidMax == 0:
    dst[tidx, tidy, 1] = 0
else:
    dst[tidx, tidy, 1] = (df/tidMax) * 100

dst[tidx, tidy, 2] = tidMax*100
```

4.2 : Kuwahara in GPU

The Kuwahara filtering code for GPU processing is quite similar to the code for CPU processing employing the same formulas ((4), (5), (6), (7), and (8). Obtaining the V value in HSV space from the image is the first step. This process is identical to how a computer works. By utilizing cuda rather than loop, we have:

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

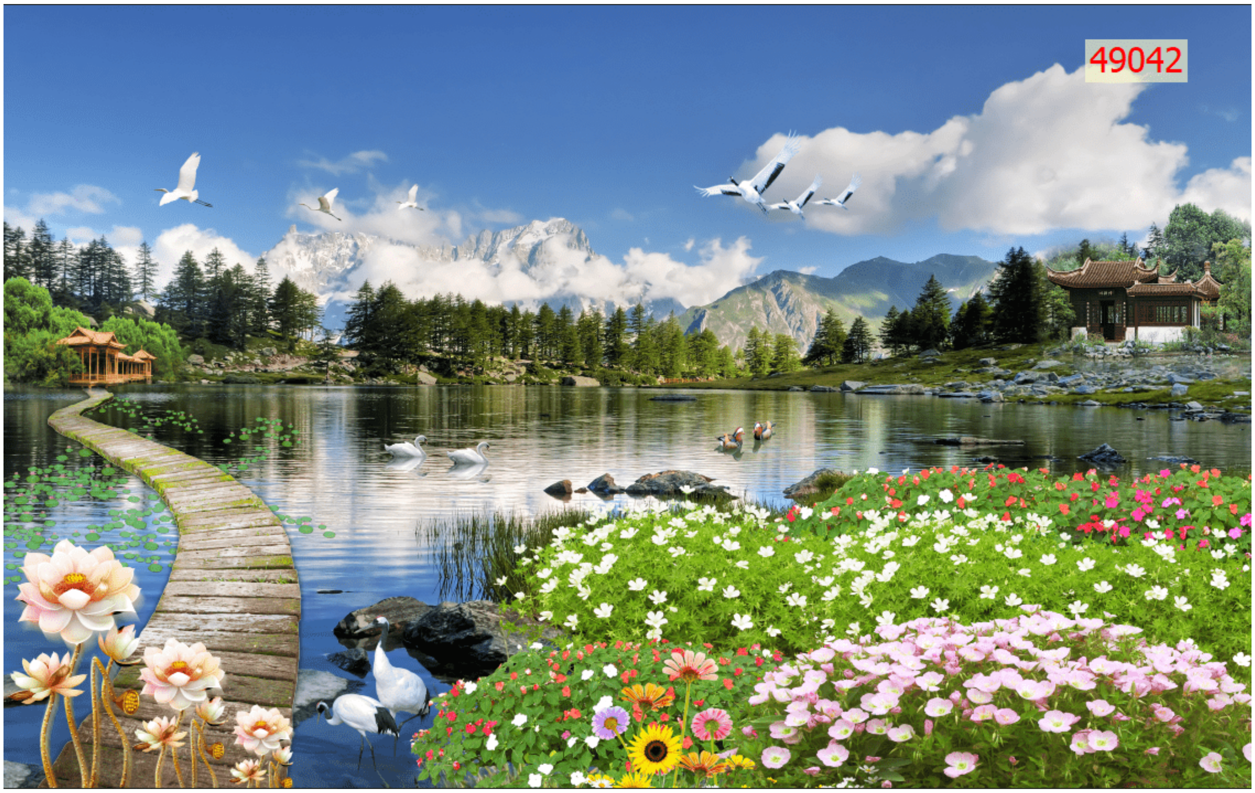


Figure 1 : original picture.

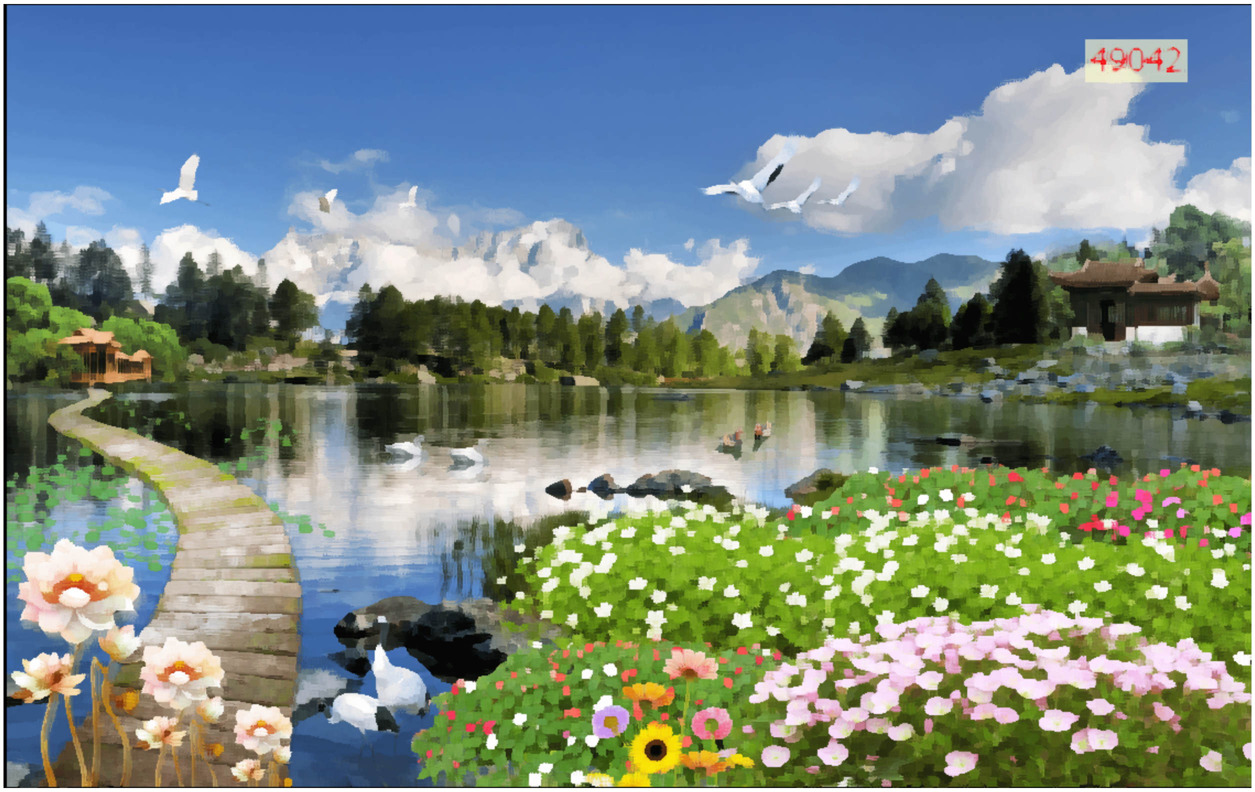


Figure 2 : picture_kuwahara_GPU