Project: Kuwahara Filter Implementation

Ta Quang Minh - M23.ICT.009

November 11, 2024

Abstract

This report discusses the implementation and optimization of the Kuwahara filter using both CPU and GPU with shared and non-shared memory approaches . Performance measurements are analyzed to enhance the efficiency of the filter.

1 Introduction

In the field of image processing, the Kuwahara filter is a smoothing technique that is often used to preserve edges while reducing noise within homogeneous regions of an image.

Unlike linear filters that may blur edges, the Kuwahara filter works by dividing a local window into sub-regions, analyzing the variance in each region, and selecting the mean value from the region with the lowest variance to replace the center pixel.

The Kuwahara filter is particularly suitable for applications in "fine-art transformation," where it creates an effect like using brush strokes and make the image a feeling similar to traditional painting.

2 Implementation outline

To implement the Kuwahara filter for colored image, we conduct the following steps:

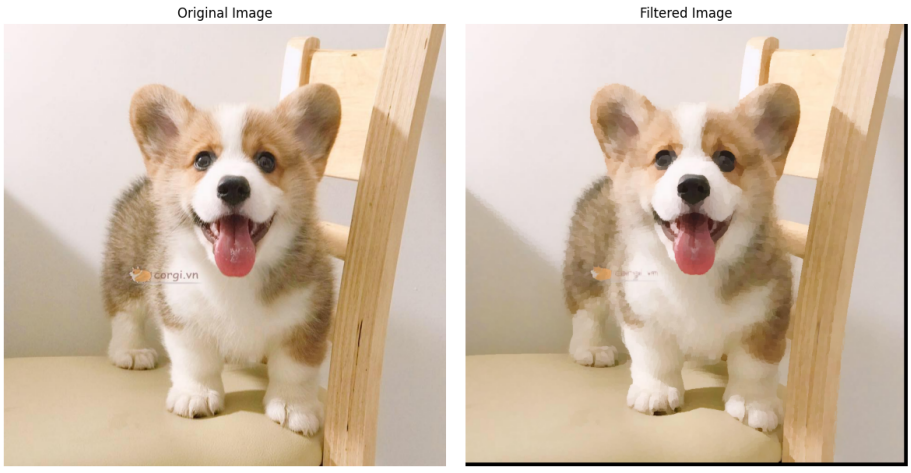
- We first reuse the previous labwork code to transform the image from rgb to hsv format. Only the brightness (v) value of the image are needed
- For pixels in the edges, we use black padding to deal with them
- We extract 4 relevant quadrant for each pixelwith the kernel size as parameter. The quadrant with the lowest variance in brightness will be determined
- We finally calculate the mean level of red, green and blue of the corresponding quadrant in rgb format and assign that value for the pixel.

3 Implementation details

- **Kuwahara Filter without CPU**
  - Each pixel of the original image is iterated one by one by the for loop
  - A list is used to keep the value of the variance of the 4 quadrant in brightness stat in hsv format, alongside with the quadrant coordinate.
  - When the quadrant with lowest variance is determined, rgb value is updated by the mean value in that quadrant.
  - We utilized numpy to implement the operations.
- **Kuwahara Filter without Shared Memory**
  - This time many pixels will be processed at the same time. Each thread will corresponding with 1 pixel in the original image.
  - The calculation is similar to with the CPU version, all carried out on global memory
- **Kuwahara Filter with Shared Memory**
  - We define shared memory for each block to speed up calculation
  - The pixels from the original image array are copied into the shared memory before being used to calculate variances and means of the quadrants.

4 Resulting image

All 3 version of our code gives the result:



We can observe that the image of the dog was blurred and have the appearance like a painting. In this case, we use kernel with size 9.

5 Performance Evaluation and Analysis

We calculate the times it takes for each version of the code using the timer() function.

We do not include the part where the image is being converted from rgb to hsv into the evaluation and only includes the transformation part of the filter.

The running time for each version is as followed:

- CPU: 130s
- GPU without share memory: 0.43s
- GPU with share memory: 0.65s

  In our case, the version without share memory actually performed best. This suggest better optimization is needed for improvement for the version with share memory.