title: “Steganographie”
author:
Do Duy Huy Hoang
Nguyen Thi Mai Phuong

University de Limoges
date: October 22, 2020

List of Tables

List of Figures

Contents

Introduction

We denote Steganographie as **Stena**, for short.

In this project, we have done the following:

- Stena with CPU
- Stena with GPU and CUDA (5-15x faster) using shared memory
- Hiding multiple characters in an image
- Can use filter any size (with width, height <= 31, odd numbers)
- Automated testing for correctness and speed
  - Benchmark at the end

Technical details

In this part, we highlight the important ideas and implementation details in the program. Note that this section is about CPU version, the GPU version will be explained later.

Overall, Stena is divided into 4 main phases:

- Initialization: We allocate memory for the arrays we use during computation
- Convolution: the filter is convoluted with the image to create image V. Pixel of V at the border (that cannot fit the filter) will have value 0
- Sorting: 8*n pixels with the largest values in V is selected, where n is the length of the input string. Each pixel will contain 1 bit of that string
- Output: given 8*n positions, the characters are hidden in the original image using the described method

The convolution section is the most important part in this project. The final two is self-explanatory in the source code.

Utilities functions

- We use ppm_lib.h for image reading/write. Since the memory does not include any memory management, we have to allocate/free memory manually
- For measuring time, we use C++ chrono libraries. The utility class is implemented in my_timer.h
- Definitions of important variables:
  - H, W: height and width of the image
  - M, N: height and width of the filter
  - n: length of the hidden string
  - K = 8*n: the number of pixels needed to hide the string, 1 bit per pixel

Convolution

The CPU implementation is straightforward. We loop over each pixel (i,j) and place the center of the filter on top of it. Then, we loop over each cell of the filter to compute the convolution sum. That sum will be the new value of V(i,j)

Note that for pixels at the border, the filter cannot fit inside the image. In that situation, V(i,j) = 0.

Time complexity: O(H∗W∗M∗N)

Sorting

In this part, we need to find 8*n pixels (i,j) with the largest V(i,j).

The simplest way to this is to create an array size H∗W, where each element contains a value V(i,j) and its index in array V (which is implemented as a 1D array, like PPMImage). After that, we sort the array decreasingly and choose the first 8∗n pixel position to hide the string.

Time complexity for sorting is: O(H∗W∗log(H∗W))

This is wasteful because n << H∗W. For example, for a 1080p images, there are 1920*1080 = 2073600 pixels to sort. If we want to hide a 10-character string, we only need the largest 80 pixels. Sorting the entire array is a waste.

Therefore, we use QuickSelect algorithm to partially fort and find only the K largest elements. This is a popular and common algorithm, and more details can be seen in the code's comments.

/TODO: add more code desb hereeee

Time complexity: ~~O(H∗W∗log(K))