An Automated Technique for Fish Detection using Computer Vision Algorithms

Vu Dinh Trung

Department of Computer Science and Engineering

National Taiwan Ocean University

Keelung, Taiwan 202

R. O. C.

I. INTRODUCTION

For fish classification, shapes and measurements of component parts are needed to compare between the different species. Typically, the researchers have to measure and redraw one by one manually using magnification tools and measurement tools. This work is very time consuming, meticulous, focused, and precise requirement. Moreover, for the manual method, the fish has to be collected before, it is sometimes difficult and high cost. Therefore, a method to measure and redraw a fish automatically is greatly needed to solve these problems. For the proposed approach, we can detect fish parts in a fish image based on image processing. In addition, if we have any actual measurements of a specific part, then we can get actual measurements of all the other parts.

II. MOTIVATION

Parts of the fish are often clearly different in shape and color as shown in the figure 1. We can completely detect fish and fish features using current computer vision algorithms.

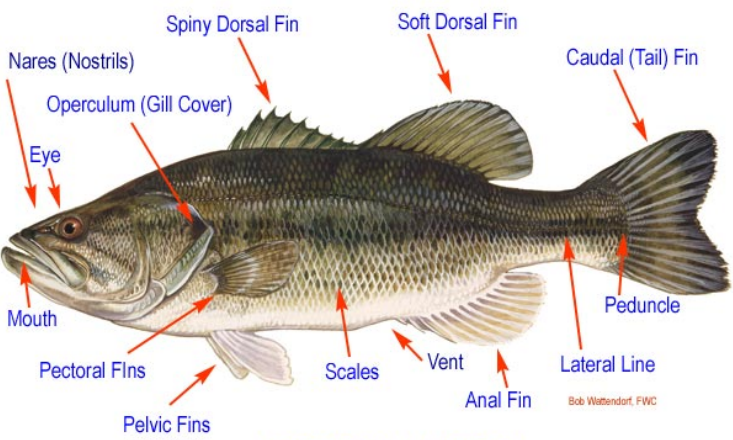


Figure 1. Fish external anatomy

III. PROPOSED APPROACH

The proposed approach is briefly described as follows:

(1) Image pre-processing enhances the visual appearance of images and improve the manipulation of datasets. Some image pre-processing methods are listed as follows:

(a) Image resampling reduces or increases the number of pixels of the dataset.

(b) Greyscale contrast enhancement improves the visualisation by brightening the dataset.

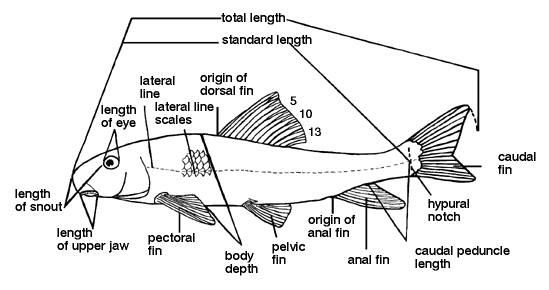
(c) Noise removal has several techniques as low-pass, high-pass, band-pass spatial filtering, mean filtering, median filtering.

(d) Mathematical operations enhances particular features. It is possible to apply to images arithmetic operations (addition, subtraction…), and morphological operations (dilation, erosion…).

(2) Fish detection, in which the fish is detected and separated from background. This process consists of identifying fish locations in an image frame (i.e., its x, y pixel coordinates), fish extent (width, height), followed by a clear segmentation of fish from background. The outcome is an image that only contains fish targets, with the background masked out, and individual non-overlapping fish targets separately labeled.

(3) Fish segmentation involves the partitioning of a fish image into distinct (usually) non-overlapping regions that is more meaningful and easier to analyze.

(4) Fish feature detection detects some of the common external features that are used to describe the differences among fish species.

****

Picture 2. Some of the fish common external features after detection and segmentation

IV. CONCLUSION

This research is expected to contribute an automated method for fish detection using computer vision algorithms which has high reliability, durability and accuracy factors; as well as minimizing cost and time needed for such task. The idea is to capture the image of the fish using a camera that uploads the picture to the software to detect and measure the common external features.

However, the problem of Algorithm 3 is the limited size of the registers. In fact, users are not able to fully control the registers, and could only utilize registers when the data size is small enough. When the data points cannot be loaded into the registers as the data dimension grows, they will be stored in local memory, which will increase the reading latency and decrease the performance significantly.

In fact, the input data point and the centroid could be viewed as two matrixes data[n][d] and centroid[d][k]; the result distance could be denoted as Result[n][k]; and the distance computing process shares the same flow as matrix multiplication. Based on this observation, we design Algorithm 4 for high-dimensional data sets, by adopting the idea of matrix multiplication and utilizing registers and the shared memory together. The main idea of Algorithm 4 is decreasing the global memory access time and latency by loading the data into the shared memory tile by tile. Thus, Algorithm 4 reads each data point from global memory only once, the same as Algorithm 3. The key point of Algorithm 4 is how to access the global memory and shared memory efficiently, which is achieved by adopting 8 coalescing reading which accesses sixteen continuous address for the threads in a half warp to avoid the bank conflict. The details are described as follows.

The three matrixes data[n][d], centroid[d][k] and Result[n][k] are partitioned into TH×TW, TW×TH, and TH×TW tiles respectively. The resource of the GPU is partitioned as follows: the grid has (k/TW)×(n/TH) blocks, the ID of which is noted by blockIdx.y (by in Fig.2) and blockIdx.x (bx in Fig.2); each block has TH×TDimY threads, the ID of which is noted by threadIdx.y (ty in Fig.2) and threadIdx.x (tx in Fig.2). The computing task is dispatching as follows: each block calculates TDimY tiles in the Result, which is noted as SR[TH][TW×TDimY]; each thread computes a column of SR. For each thread, indexD points to the right position of the data, which contains the following three parts as shown in line 4: data is the beginning address of the data set; since the height of the data is divided by TH, blockIdx.y×TH×d is the address of the corresponding block; threadIdx.y×d adding threadIdx.x is the offset address inside the block.

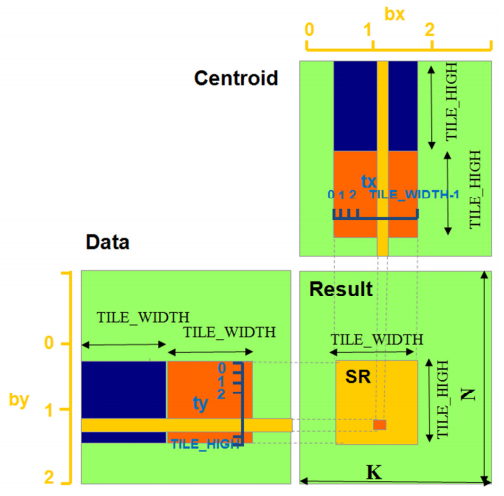


Figure 2. Tile-based distance computing process

In line 5, indexC points to the right position in centroid, which also has three parts: centroid is the beginning address of the current centroid; blockIdx.x×TW points to the corresponding block address, since the width of the centroid is divided by TW; threadIdx.y×blockDim.x adding threadIdx.x points to the address of the current thread inside the block. Obviously, the threads in one block would access centroid in continuous addresses, which is also called coalesced accessing. indexR is calculated in the same way as in line 6: the beginning address of the result, the row address, and the offset address inside the block for the current thread.

In the loop from line 11 to 16, the algorithm loads a tile of data from global memory to the shared memory, and computes the temporary distance saved in TResult which are stored in on-chip registers; the loop ends when the whole row has been calculated. Line 17-18 calculate the distance based on the results of muliplication. Line 19 waits for all the threads to finish their work. Line 20 writes the distance back from TResult to SR. The details are shown in Fig.2, and take the process of calculating a SR[TH][TW×TDimY] as an example, which is equal to data[TH][d]×centroid[d][TW×TDimY]: load the first tile (in blue color) from the data into the shared memory; multiply the blue tile in the data with the blue tile in the Centroid, which is stored in the constant memory; accumulate the temporary results into TResult, whose initial value is all zero; repeat loading the next tile (in orange color), multiplying and accumulating, until data[TH][d] and centroid[d][TW×TDimY] have been all accessed.

After calculating the distances matrix Result[n][k] between the data and the centroid, the next step is accumulating the total inverse distance, updating the memberships, and initializing. Obviously, based on the CPU, the computational complexity is O(nk). On the GPUs, each thread calculates for one data point from one row of Result, whose computational complexity is O(k).

Algorithm 4: Computing distance based on shared memory of the GPU

//

//

//

//

//

//

//

//

//

//

//

//

//

//



Algorithm 6: Computing distance based on shared memory of the GPU

//

//

//

//

//

//

//

//

//

//

//

//

//

//