Programming Parallel Computers

Aalto 2023

IS: image segmentation

Please read the general instructions on this page first, and then check the individual tasks for more details. For each task you can download a zip file that contains the code templates you can use for development.

Task	Attempts	Expected	Points	Max	Rating	Rec.	Deadline for full points
IS4: fast solution Using all resources that you have in the CPU, solve the task **as fast as possible**. You are encouraged to exploit instruction-level parallelism, multithreading, and vector instructions whenever possible, and also to optimize the memory access pattern. Please do all arithmetic with **double-precision** floating point numbers.	0	–	–	5 + 2	★★	R	2023-05-21 at 23:59:59
IS6a: fast CPU solution for 1-bit images In this task, the input is always a monochromatic image: each input pixel is either entirely **white** with the RGB values (1,1,1) or entirely **black** with the RGB values (0,0,0). Make your solution to IS4 faster by exploiting this property. It is now enough to find a solution for only one color channel, and you will also have much less trouble with rounding errors. In this task, you are permitted to use **single-precision** floating point numbers.	0	–	–	5 + 2	★★★	R	2023-06-04 at 23:59:59
IS6b: fast GPU solution for 1-bit images Port your solution to IS6a to the **GPU**, again, make it run **as fast as possible**.	0	–	–	5 + 2	★★	R	2023-06-04 at 23:59:59
IS9a: better algorithm Design a more efficient algorithm that (at least in typical cases) does not need to try out all possible locations of the rectangle. Implement the algorithm efficiently on the CPU.	0	–	–	5	★★★		2023-06-04 at 23:59:59

General instructions for this exercise

Find the best way to partition the given figure in two parts: a monochromatic rectangle and a monochromatic background. The objective is to minimize the sum of squared errors.

Interface

We have already defined the following type for storing the result:

```
struct Result {
    int y0;
    int x0;
    int y1;
    int x1;
    float outer[3];
    float inner[3];
};
```

You need to implement the following function:

```
Result segment(int ny, int nx, const float* data)
```

Here `data` is a color image with `ny*nx` pixels, and each pixel consists of three color components, red, green, and blue. In total, there are `ny*nx*3` floating point numbers in the array `data`.

The color components are numbered `0 <= c < 3`, x coordinates are numbered `0 <= x < nx`, y coordinates are numbered `0 <= y < ny`, and the value of this color component is stored in `data[c + 3 * x + 3 * nx * y]`.

Correct output

In the `Result` structure, the first four fields indicate the location of the rectangle. The upper left corner of the rectangle is at coordinates (`x0`, `y0`), and the lower right corner is at coordinates (`x1-1`, `y1-1`). That is, the width of the rectangle is `x1-x0` pixels and the height is `y1-y0` pixels. The coordinates have to satisfy `0 <= y0 < y1 <= ny` and `0 <= x0 < x1 <= nx`.

The last two fields indicate the color of the background and the rectangle. Field `outer` contains the three color components of the background and field `inner` contains the three color components of the rectangle.

Objective function

For each pixel (`x`,`y`) and color component `c`, we define the error `error(y,x,c)` as follows:

- Let `color(y,x,c) = data[c + 3 * x + 3 * nx * y]`.
- If (`x`,`y`) is located outside the rectangle: `error(y,x,c) = outer[c] - color(y,x,c)`.
- If (`x`,`y`) is located inside the rectangle: `error(y,x,c) = inner[c] - color(y,x,c)`.

The total cost of the segmentation is the sum of squared errors, that is, the sum of `error(y,x,c) * error(y,x,c)` over all `0 <= c < 3` and `0 <= x < nx` and `0 <= y < ny`.

Your task is to find a segmentation that minimizes the total cost.

Algorithm

In IS4, IS6a, and IS6b tasks you are expected to use an algorithm that tries out all possible locations 0 ≤ y0 < y1 ≤ ny and 0 ≤ x0 < x1 ≤ nx for the rectangle and finds the best one. However, for each candidate location you should only perform O(1) operations to evaluate how good this position is. To achieve this, some preprocessing will be needed.

In IS9a you are expected to design a more efficient algorithm that (at least in typical cases) does not need to try out all possible locations of the rectangle. In IS9a your submission will be graded using a structured input that might resemble e.g. a real-world image in which some candidate positions are much better than others.

Examples

These examples show the segmentation produced by a correct implementation (right) for some sample images (left). Hover the mouse on the output to better see the segmentation.






General hints

Hints for IS9a