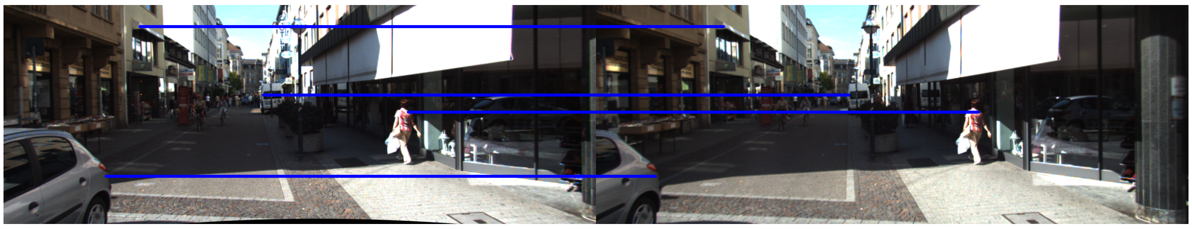
Left-right consistency check for stereo matching

In this challenge problem, you will implement left/right consistency check for a stereo matcher and optimize its performance.

Stereo Block Matching Mini-Primer

Stereo vision cameras estimate the distance to an object by measuring the disparity (i.e., pixel shift, or parallax) between two image features. The larger the disparity between the two features, the closer the object (mathematically, disparity[pixels] = camera focal length[pixels] * baseline width between cameras[m]/range[m]). For example, hold your arm in front of you and put your thumb up (your thumb is the feature in this example). Alternate between closing your left eye and right eye, and notice that the position of the thumb moves relative to the background (the background should be much farther away than your thumb). As you bring your thumb closer to your nose, you will see that this shift (aka, disparity) increases. This is the principle for estimating depth using stereo vision.

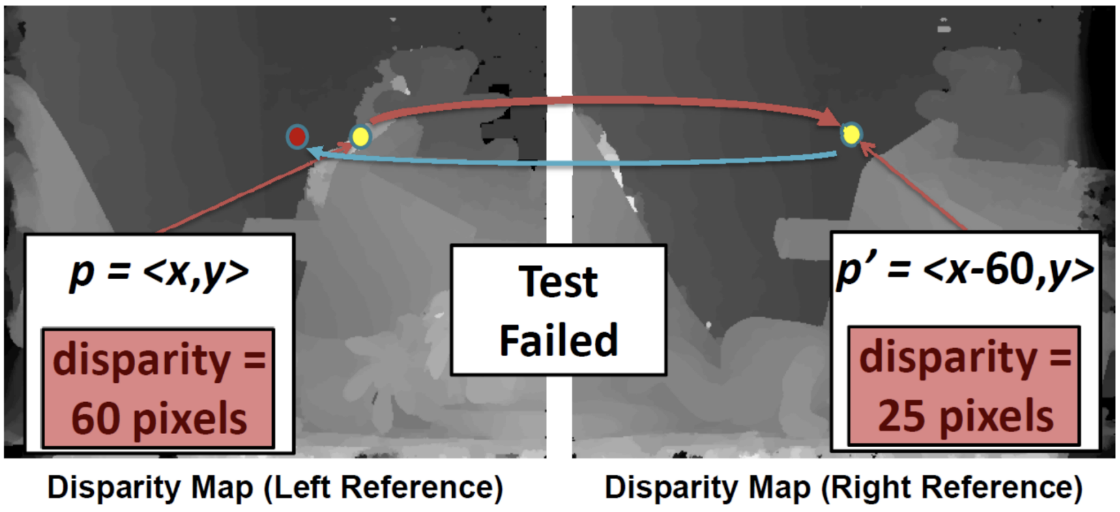
A computer estimates disparity for by iteratively comparing left to right sub-windows of an image. Typically, the subwindow, or *block*, that is used for matching features from left to right images is 5x5, 9x9, 11x11, 15x15, or 21x21 pixels, depending on the scale and format of the image. After the left and right images are *rectified*, then the block matching only needs to occur on the same row -- a 1D search rather than a 2D search. Rectification means that the images are row aligned so that, for example, the corner of the same object is in the same row in the left and right rectified images. The fancy way of saying this is that the epipolar lines are aligned. Below is a visualization of several matches along the epipolar lines.



If you want to read more about stereo block matching in plain english, (here)[https://python.plainenglish.io/the-depth-ii-block-matching-d599e9372712] is a webpage that explains the process with pretty pictures.

Your task

In its simplest form, the stereo matching algorithm calculates the disparity algorithm by looping over all pixels in the left image and searching for their match in the right image along the horizontal line. Disparity maps can be calculated relative to the left or right image. A popular method of verifying the correctness of the measurement is the so-called left-right consistency check. It verifies that the corresponding values in the left and right disparity maps do not differ more than some predefined tolerance.



Source: https://mordohai.github.io/classes/cs532_f15/cs532f15_Week4.pdf

Main steps:

- Install the OpenCV library. You will find the instruction following (this link) [https://docs.opencv.org/4.x/d7/d9f/tutorial_linux_install.html].
- Implement left-right consistency check kernel in CUDA.
- How does the application of this filter change the results? How does it depend on the defined tolerance?
- Profile the code and optimize it. What is the optimal number of blocks and threads for executing the CUDA kernel? Can memory passing be improved? What else can we do to speed-up the application?
- How does the performance depend on the type of the GPU? Does it scale with the number of CUDA cores and the type of CUDA compute capability?

Deliverables:

- Clean, production level code. Please remember about tests and exception handling.
- Presentation or a text document explaining your solution.

Attached files

- main.cpp - Short program loading disparity maps and displaying them. Please fill in the gaps in this program. To compile it, run `` g++ -Wall main.cpp -o challenge.out `pkg-config --libs opencv4` ``
- disp_left.png, disp_right.png - Disparity maps computer with respect to the left and right camera frames respectively. Every pixel of disparity is represented by 4bits. The disparity maps is saved in a 16-bit format, but only 12 bits are used.
- img_left_rect.png, img_right_rect.png - left and right rectified images respectively. The images come from the (KITTI dataset)[http://www.cvlibs.net/datasets/kitti/].

Note

The challenge problem is designed with CUDA implementation in mind. However, feel free to use OpenCL if you think it's better suited to solve the problem. Please be prepared to explain your choice.