Euclidean Distance Transform Computation

Yves Lucet

Abstract

The objective of the lab is to compute the Euclidean Distance Transform using an embarrassingly parallel algorithm.

Given a binary matrix B, the zeros are interpreted as obstacles and the ones as cells free of obstacles. The Euclidean distance transform (EDT) takes the matrix B as input and outputs a new matrix D that contains for each cell, its distance to the closest cell with a value of 0. To speed up computation, the square Euclidean distance transform (SEDT) is computed. In that case, all calculations involve integers only. (If needed a square root is applied in a final pass to obtain the distance.)

Consider the following examples.

$$B = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \text{ has square EDT } D^2 = \begin{pmatrix} 8 & 5 & 4 & 5 & 8 \\ 5 & 2 & 1 & 2 & 5 \\ 4 & 1 & 0 & 1 & 4 \\ 5 & 2 & 1 & 2 & 5 \\ 8 & 5 & 4 & 5 & 8 \end{pmatrix},$$

$$B = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \text{ has square EDT } D^2 = \begin{pmatrix} 2 & 1 & 2 & 5 & 8 \\ 1 & 0 & 1 & 2 & 5 \\ 2 & 1 & 0 & 1 & 4 \\ 5 & 2 & 1 & 2 & 5 \\ 8 & 5 & 4 & 5 & 8 \end{pmatrix},$$

and

$$B = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \text{ has square EDT } D^2 = \begin{pmatrix} 2 & 1 & 2 & 1 & 2 \\ 1 & 0 & 1 & 0 & 1 \\ 2 & 1 & 1 & 1 & 2 \\ 4 & 1 & 0 & 1 & 4 \\ 5 & 2 & 1 & 2 & 5 \end{pmatrix}.$$

1. Write a serial code to compute the SEDT using a brute force approach that is easier to parallelize: for each pixel (i, j) with value 0, assume it is the only pixel with value 0 in the binary image and write its SEDT matrix $D^2_{i,j}$. Then aggregate all the matrices together by computing for each pixel the minimum value over all the matrices $D^2_{i,j}$.

2. Write a CUDA implementation of your code.