Coverage maps on domains with obstacles

Oriol Balló*[1], Narcís Coll†[2], and Marta Fort‡[2]

[1]Universitat de Girona
[2]Graphics and Imaging Laboratory, Universitat de Girona

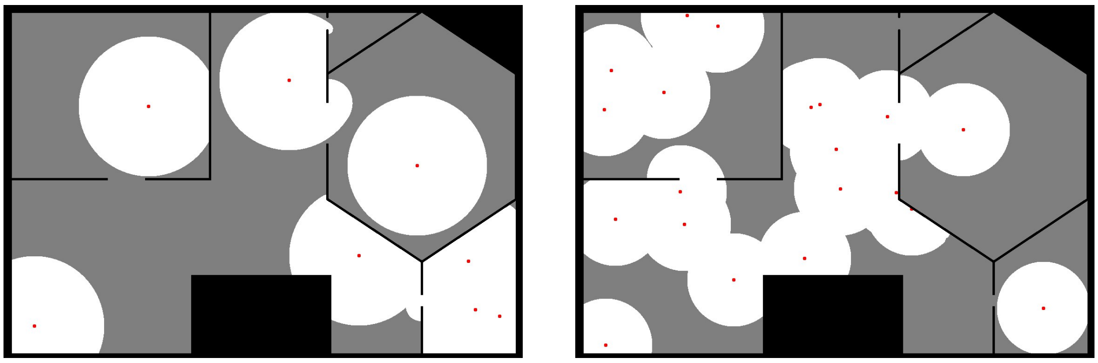
Figure 1: Coverage map of eight facilities (left) and twenty facilities (right)

Abstract

The coverage map of a set of facilities represents, for each point within a domain, whether at least one facility covers it (see Figure 1). That is, we know if at least one facility is at most at a given distance (the facility coverage radius) through the free space of each point of the domain. In this work, we propose a parallel method that runs on the GPU to compute coverage maps over domains given by binary images. The input is a binary image, where each pixel is marked depending on whether it is free space or not, the set of facilities, and their coverage radius. The output is an image where each pixel is marked as covered or uncovered by the set of facilities.

We use a two-step iterative process that combines a quasi-Euclidean distance [1] propagation along free space and an exact Euclidean distance computation (without propagation). We iteratively repeat these steps until no updates occur. During the process, we obtain the distance of each pixel to its nearest site and the pixel-*id* of the last corner (of the current shortest path) by using two CUDA-kernels executed on 2d-grids and 2d-blocks and considering a thread per pixel.

The quasi-Euclidean distance propagation along free space uses a GPU parallel Bellman-Ford algorithm. The used graph (computed on the fly from the binary image) has as vertices the free pixels, and connects a free pixel with its, at most 8, neighboring free pixels. At the beginning of the process, pixels containing sites store a 0 as distance and its pixel-*id* as last-corner-*id*. The rest store ∞ as distance and -1 as last-corner-*id* (uncovered pixel). Throughout the process, in a similar way to the Euclidean distance transformation algorithms [2], each pixel propagates towards itself the paths that reach its free neighbors. If the length of any of these paths is less than the current distance, this distance and the last corner-*id* are updated accordingly.

An inner-block propagation followed by an inter-block propagation expands the current paths through free space according to the quasi-Euclidean distance. A boolean variable and two synchronizing points (per block) stop the inner-block propagation when no updates occur within the block. The inter-block propagation uses a global boolean to keep calling the kernel while updates occur.

The second step computes the exact Euclidean length of the obtained paths. The covered-pixel threads retrieve the path reaching the pixel while determining its Euclidean length. They add the Euclidean distance from the pixel to its last-corner-*id* to the distance of this last corner to the previous one, and so on, until reaching the original site. This two-step iteration leads to exact free-space coverage maps.

Funding

Research supported by grants PID2019-106426RB-C31 and PDC2021-120997-C32 funded both by MCIN/AEI/10.13039/501100011033 and the 2nd one also by European Union NextGenerationEU/PRTR.

References

[1] Montanari, U. (1968). A Method for Obtaining Skeletons Using a Quasi-Euclidean Distance. Journal of the ACM (JACM), 15, 600–624.

[2] Maurer, C. R., Qi, R., and Raghavan, V. (2003). A linear time algorithm for computing exact Euclidean distance transforms of binary images in arbitrary dimensions. IEEE Transactions on Pattern Analysis and Machine Intelligence, 25(2), 265–270.

*Email: u1962391@campus.udg.edu
†Email: narcis.coll@udg.edu
‡Email: marta.fort@udg.edu