Pixel to circles project

Mats Hoem Olsen, Katla Marie Mayer, Oda Johanne Matre Simensen

December 5, 2022

Contents

List of Algorithms

Abstract

We designed a code that converted a pixelated image to a vector based image using Moors contour algorithm, Harris corner detector, and Postscript™.

1 Problem

As an input, a two-dimensional image or landscape is given by an a x a square of black-and-white pixels.

Your code should create an approximation of this landscape by a set of superimposed circular disks. It should aim at the best possible accuracy for a given maximum number of circular disks that can be varied as part of the input. It should also be able to generate a lossless representation of the image/landscape using an unlimited number of circular disks.

1.1 assumptions

Assume that each pixel is b/w, that is either completely black (colour value 0) or completely white (colour value 1), not on a greyscale in between.

Have your code produce a sequence of m circular discs 0 i < m, each with two coordinates of the centre, a radius, and a colour value ci, that compresses the pixel-based representation. The conversion back from the vector representation to the pixel representation could look as follows. For all pixel coordinates (0, 0) to (a-1, a-1): Initialize the pixel colour to black. Then, iteratively for all i from 0 to m-1, whenever the pixel coordinates are inside circle i, update the pixel colour to ci.

Agreement between the original and the compressed version can be quantified by the fraction of pixels for which the algorithm above reconstructs the original colour value of the pixel correctly. In the case of a lossless compression, the two versions should agree for all pixels.

You can deviate from the default recommendations; follow them just if you do not see any good reason not to.

2 Edge detector

The edge detector algorithm is composed of the following algortithm:

Algorithm 1 Contour algorithm

$P_0 \leftarrow$ first pixel of an shape
$P_{-1} \leftarrow$ previous pixel
$P_n \leftarrow P_0$
C is the contour of the shape
while $P_n \neq P_0$ **do**
    $P_{-1} \leftarrow$ look anti-clockwise from P_{-1}
    if P_{-1} is foreground **then**
        $C \leftarrow P_{-1}$
        Swap label P_{-1} and P_n
    end if
end while
return C

This algorithm will trace the contour for one blob, but this will be used multiple times until we hit the lower right corner of the image. The foreground pixels will be stored in a coordinate structure that stores x, and y coordinates.

3 Harris Corner detector

The Harris corner detector is used to find all the corners in an image (which in our case is the contour of one blob). The algorithm uses the eigen values from the first directional derivative matrix

$$M = \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}$$

After calculating the smallest eigen value for every point along the path it uses a simple threshold to determine if an eigen value is large enough to be considered a corner, or anything else. The algorithm used is

Algorithm 2 Harris corner detector

$\lambda \leftarrow \{\}$
$C \leftarrow \{\}$
for all Points in picture **do**
    Calculate I_x and I_y
    Make M matrix
    Find the smallest Eigen value
    $\lambda \leftarrow$ eigen value
end for
for all do
end for

4 Image viewer

To store the circles we have found in the image, we choose to use the language Postscript since it has simple enough syntax to export it, and convert it to other file formats that support vector graphics. The command we will be using is ”arc fill”. This command will take inn x,y coordinates, a radius, and angle of the arc. The ”fill” command just uses watershed algorithm to fill inn the circle with black.