Oren Finard
Best-Fitting Circle Program

My program finds the best fitting circle (BFC) of an image fed through the program. The image should be a closed, solid-filled image on a square background, and it is best if the image is not touching any of the canvas edges (though not a necessity, but it may effect the accuracy of the program very slightly). The program runs an edge-detection filter that leaves just the background of the image, and the outline of the shape in the image. From there the program has two functions that can be used to find the BFC: the first is a Hough-transform function, the second a function for calculating the Least-Square Circle Fit.

The Hough-Transform function defines the BFC as the circle that intersects the most edge-points (the most points in the outline). Theoretically, it does this by drawing radii of every length at every point in the image, and then taking the circle that intersected the most edge-pixels. However, this is incredibly inefficient. To optimize this process, my function resizes the image, cutting the width and height each by half over and over again until the width and height are each less than 50 pixels. Then it runs the Hough-Transform , an easy processing task, to produce a BFC for this tiny image. We now have a point and radius for the BFC. An image twice the size (four times the area) of this tiny image will find that each pixel in the tiny image represented, at most, a 3x3 square of pixels (usually each pixel would represent a 2x2 square of pixels, but if there width and height were odd numbers, then integer rounding would cause them to downsize even further. Thus, a pixel can actually represent a 2x2 square, or a 2.5x2.5 square, which means that if we are looking for a specific pixel, we must check a 3x3 square (a 2.5x2.5 square, rounded up)). Thus, by doubling the xy-coordinates of the small BFC center, we can limit the BFC center of the larger image down to 9 pixels (and the radius down to within 6 pixels of length)(Note: the math explained here is theoretically correct, but in execution there are various trivial details that need to be accounted for; for clarity, these will be ignored in this explanation). By checking just those 9 pixels, so a total of only 54 circles, we can find the BFC of the doubled image fairly easily. By performing this sizing up loop over and over until the image reaches its original dimensions, we can fairly quickly locate the information of the BFC for any given shape. An interesting quality of this function is its ability to ignore outlying data points. If you process an image with this function, even if there are outlying pixels that are not part of the actual shape, the function will only focus on the shape because of the density of edge-points at the shape's actual location. Furthermore, even if you have a series of circles in a line, and even if they are touching, the function will almost surely focus on one circle because of the function's ability to so closely replicate a circle. This is incredibly useful, as will be discussed later. However, the program is not incredibly fast: it takes over a minute to calculate images of sizes greater than 600x600, and about 5 minutes to analyze an image of size 1200x1200.

The other function for finding the BFC is a method of finding the Least-Squared Circle (the LSC). In statistics, the BFC is defined by essentially performing a Least-Squared Line on a set of data, except then transforming that line into a circle.

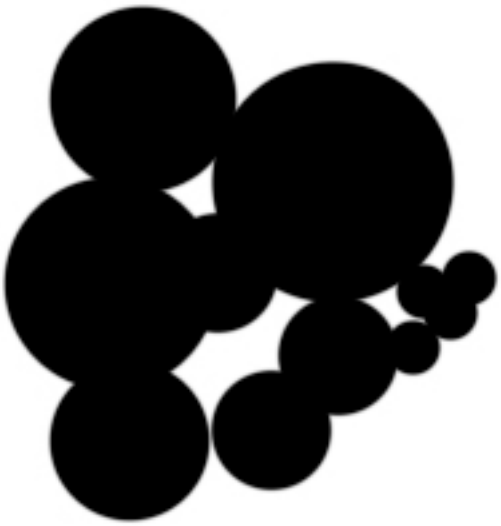
Thus, the BFC is the least far from every point in a data set, simultaneously, compare to any other possible circle. The programming in this function is minimal; it is just calculations. The real work is done in the mathematical calculation of the LFC, which was done by an R. Bullock, who kindly had uploaded a .pdf document (included) outlining the calculations needed to find the LSC for any set of points. Thus, the LSC function treats each edge-point of the shape as a data-point, and from all of them, calculates the LSC/BFC. The main functional difference between this function, and the Hough-Transform is the way they treat outliers: while the Hough-Transform ignores them, any outliers will have a significant effect on the LSC commensurate with the percentage of edge-points they represent. This is useful for comparisons of data, and analyzing a full image, rather than just a focusing on a particularly dense portion of the image. It is also an incredibly fast program, and capable of processing large images rather quickly.

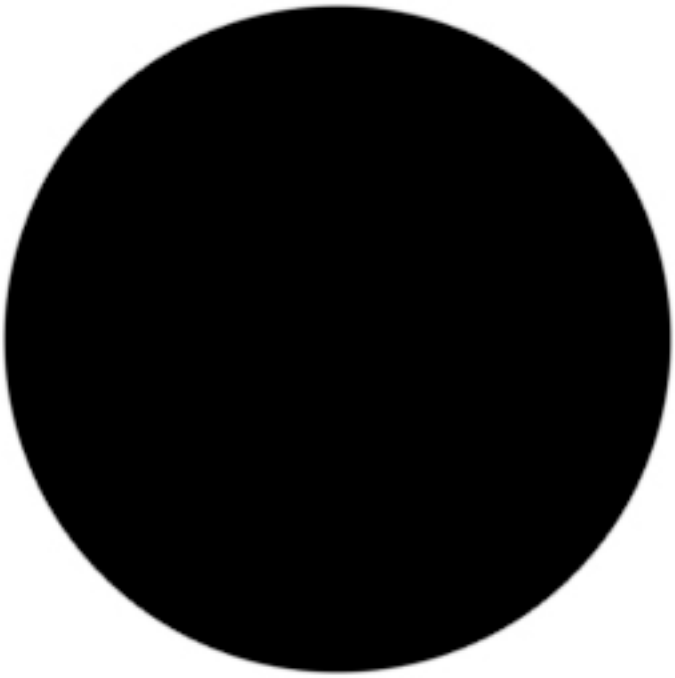
The ultimate purpose of this program is to be used in the research and analysis of plasma properties by Wesleyan Physics Professor Lutz Huwel. The program is intended for studying pictures of plasma bubbles in a water-sealed container, which often look like a series of circles in a line. This program will later be extended so that, after finding the BFC of an individual bubble, it outlines the whole image using a series of circles of relatively the BFC's size. Because of the nature of the program's ultimate purpose, I have also included functions to translate the text matrices that the plasma pictures come in into jpegs, and functions to crop out background space of the photos.

User Guide:

Start program in Idle. Place any image you want analyzed in the same folder as the program. For the Hough-Transform function, type ```kindle(-name_of_file-)```, and for the LSC function, type ```lscf(-nam_of_function-)```. If you would like to test how the programs handle different sizes of circles, call the producer function: ```producer(int)``` to create photo to analyze of radius int. Feel free to make your own solid, closed shapes in photoshop or mspaint, and try them out! I will include two cropped photos of the plasma bubbles, so you can see how they are analyzed, and a text file of the matrix of a plasma image so you can try the txt2pic function and the cropper function. Please enjoy!

Note: also included below is the pdf that includes the math used to create the LSC program





Least-Squares Circle Fit

R. Bullock

Given a finite set of points in $\mathbb{R}^2$, say $\{\,(x_i, y_i)\,|\,0 \le i < N\,\}$, we want to find the circle that "best" (in a least-squares sense) fits the points. Define

$$\overline{x} = \frac{1}{N}\sum_i x_i \qquad \text{and} \qquad \overline{y} = \frac{1}{N}\sum_i y_i$$

and let $u_i = x_i - \overline{x}, \quad v_i = y_i - \overline{y}$ for $0 \le i < N$. We solve the problem first in (u, v) coordinates, and then transform back to (x, y).

Let the circle have center (u_c, v_c) and radius R. We want to minimize $S = \sum_i \left(\, g(u_i, v_i)\,\right)^2$, where $g(u, v) = (u - u_c)^2 + (v - v_c)^2 - \alpha$, and where $\alpha = R^2$. To do that, we differentiate $S(\alpha, u_c, v_c)$.

$$\begin{aligned}\frac{\partial S}{\partial \alpha} &= 2\sum_i g(u_i, v_i)\,\frac{\partial g}{\partial \alpha}(u_i, v_i)\\ &= -2\sum_i g(u_i, v_i)\end{aligned}$$

Thus $\partial S/\partial\alpha = 0$ iff

$$\boxed{\sum_i g(u_i, v_i) = 0} \qquad \textbf{Eq. 1}$$

Continuing, we have

$$\begin{aligned}\frac{\partial S}{\partial u_c} &= 2\sum_i g(u_i, v_i)\,\frac{\partial g}{\partial u_c}(u_i, v_i)\\ &= 2\sum_i g(u_i, v_i)\,2(u_i - u_c)(-1)\\ &= -4\sum_i (u_i - u_c)\,g(u_i, v_i)\\ &= -4\sum_i u_i\,g(u_i, v_i) + 4\,u_c\,\underbrace{\sum_i g(u_i, v_i)}_{=\,0 \text{ by } \textbf{Eq. 1}}\end{aligned}$$

Thus, in the presence of **Eq. 1**, $\partial S/\partial u_c = 0$ holds iff

$$\boxed{\sum_i u_i\,g(u_i, v_i) = 0} \qquad \textbf{Eq. 2}$$

Similarly, requiring $\partial S/\partial v_c = 0$ gives

$$\boxed{\sum_i v_i\,g(u_i, v_i) = 0} \qquad \textbf{Eq. 3}$$

Expanding **Eq. 2** gives

$$\sum_i u_i \left[u_i^2 - 2\,u_i\,u_c + u_c^2 + v_i^2 - 2\,v_i\,v_c + v_c^2 - \alpha \right] = 0$$

Defining $S_u = \sum_i u_i,\ S_{uu} = \sum_i u_i^2,$ *etc.*, we can rewrite this as

$$S_{uuu} - 2\,u_c\,S_{uu} + u_c^2\,S_u + S_{uvv} - 2\,v_c\,S_{uv} + v_c^2\,S_u - \alpha\,S_u = 0$$

Since $S_u = 0$, this simplifies to

$$\boxed{u_c\,S_{uu} + v_c\,S_{uv} = \frac{1}{2}\left(S_{uuu} + S_{uvv}\right)} \qquad \textbf{Eq. 4}$$

In a similar fashion, expanding **Eq. 3** and using $S_v = 0$ gives

$$\boxed{u_c\,S_{uv} + v_c\,S_{vv} = \frac{1}{2}\left(S_{vvv} + S_{vuu}\right)} \qquad \textbf{Eq. 5}$$

Solving **Eq. 4** and **Eq. 5** simultaneously gives (u_c, v_c). Then the center (x_c, y_c) of the circle in the original coordinate system is $(x_c, y_c) = (u_c, v_c) + (\overline{x}, \overline{y})$.

To find the radius R, expand **Eq. 1**:

$$\sum_i \left[u_i^2 - 2\,u_i\,u_c + u_c^2 + v_i^2 - 2\,v_i\,v_c + v_c^2 - \alpha \right] = 0$$

Using $S_u = S_v = 0$ again, we get

$$N\left(u_c^2 + v_c^2 - \alpha\right) + S_{uu} + S_{vv} = 0$$

Thus

$$\boxed{\alpha = u_c^2 + v_c^2 + \frac{S_{uu} + S_{vv}}{N}} \qquad \textbf{Eq. 6}$$

and, of course, $R = \sqrt{\alpha}$.

See the next page for an example!

Example : Let's take a few points from the parabola $y = x^2$ and fit a circle to them. Here's a table giving the points used:

i	x_i	y_i	u_i	v_i
0	0.000	0.000	-1.500	-3.250
1	0.500	0.250	-1.000	-3.000
2	1.000	1.000	-0.500	-2.250
3	1.500	2.250	0.000	-1.000
4	2.000	4.000	0.500	0.750
5	2.500	6.250	1.000	3.000
6	3.000	9.000	1.500	5.750

Here we have $N = 7$, $\overline{x} = 1.5$, and $\overline{y} = 3.25$. Also, $S_{uu} = 7$, $S_{uv} = 21$, $S_{vv} = 68.25$, $S_{uuu} = 0$, $S_{vvv} = 143.81$, $S_{uvv} = 31.5$, $S_{vuu} = 5.25$. Thus (using **Eq. 4** and **Eq. 5**) we have the following 2×2 linear system for (u_c, v_c):

$$\begin{bmatrix} 7 & 21 \\ 21 & 68.25 \end{bmatrix} \begin{bmatrix} u_c \\ v_c \end{bmatrix} = \begin{bmatrix} 15.75 \\ 74.531 \end{bmatrix}$$

Solving this system gives $(u_c, v_c) = (-13.339, 5.1964)$, and thus $(x_c, y_c) = (-11.839, 8.4464)$. Substituting these values into **Eq. 6** gives $\alpha = 215.69$, and hence $R = 14.686$. A plot of this example appears below.