LA-UR-17-30204
Approved for public release; distribution is unlimited.

Title: Enhancements to the Image Analysis Tool for Core Punch Experiments and Simulations (vs. 2014)

Author(s): Hogden, John Edward
Unal, Cetin

Intended for: Report

Issued: 2017-11-06

Enhancements to the Image Analysis Tool for Core Punch Experiments and Simulations (vs. 2014)

John Hogden
M.S. B297
Computer & Computational Sciences Division
Los Alamos National Laboratory
Los Alamos, NM 87545
Email: hogden@lanl.gov

Cetin Unal
M.S. F606
Decision Applications Division Office
Los Alamos National Laboratory
Los Alamos, NM 87545
Email: cu@lanl.gov

***Abstract*—A previous paper (Hogden & Unal, 2012, Image Analysis Tool for Core Punch Experiments and Simulations) described an image processing computer program developed at Los Alamos National Laboratory. This program has proven useful so developement has been continued. In this paper we describe enhacements to the program as of 2014.**

I. Purpose

In 2012, Los Alamos National Laboratory developed an image processing tool to aid in the analysis and comparison of images obtained experimentally and through simulation. The program, called ProcessImage, comprised various analyses previously performed by a collection of command line tools developed in MATLAB [1] and added a graphical user interface to simplify the use of the tools.

ProcessImage proved useful so development has continued. This paper describes enhancements made to the program.

II. Old Vs. New

A. Overview

The 2014 version of ProcessImage can also read two additional file types. The new file formats are described in Section II-B.

Some additional features are calculated in the 2014 version. The additional features are discussed in Section II-C.

The biggest changes between the two versions are the additional commands that can be used for modifying and visualizing the images. The new commands are apparent from looking at the buttons present when the program opens. These additions are summarized in Section II-D and then discussed in more depth in Section III.

B. Image File Formats

In addition to the legacy and KevinB formats that can be read by the 2012 version, the 2014 version of the code can read a slash-delimited fole format and and XYZV format.

A slash-delimited file has a one-line header like that shown in 1. The first line in the file containing eight slash-delimited values is assumed to be the header line. Although the header line must contain eight slash-delimted values, only the first four will be used. The first four entries are: the number of rows, the number of columns, the horizontal size of each pixel in centimeters, and the vertical size of each pixel in centimeters. The pixel values are then entered as slash-delimited entries in row order on subsequent rows of the file, with any number of entries per line.

The XYZV format header can specify arbitrary positions of points in three dimension space. Each line of an XYZV file after the header specifies a position in cartesian coordinates (x,y,z) followed by a value of the density at that position. While XYZV files can give arbitrary positions and values, ProcessImage can only work with values sampled on a uniform grid with the points given in row order. XYZV files that do not give samples from a uniform grid may be read, without and error, but the sampling in the image will be read incorrectly.

To read an XYZV file, ProcessImage looks for the first line containing the comma-delimited characters 'x', 'y', and 'z', as shown in 2. Subsequent lines are treated as data until a line that does not contain four comma-delimited values is encountered, at which point the program stops reading the data. The values in each data line comprise a position on the x-axis, a position in the y-axis, a position on the z-axis, and the density at that position. Positions on the z-axis are ignored.

C. New Features

In both the 2012 and 2014 versions of ProcessImage, pressing the "Get Features" or "Write Features" button will calculate features that describe the working image. However, in addition to the image features calculated in the 2012 version, the 2014 version of ProcessImage calculates 12 new features. These features are:

1) **CenterOfMassX**: The position of the center of mass of the image on the x-axis.
2) **CenterOfMassY**: The position of the center of mass of the image on the y-axis.
3) **CenterOfMassOffset**: the distance between the center of mass and the center of the image.
4) **sumM3OverR**:
5) **M3OverEquivRadius**:
6) **equivThickness**:
7) **rMOverMTotal**:
8) **eqReqTOverr1t1**:

```
10/42/0.5/1.0/5/6/7/8
```

Listing 1. Example header from a file stored in slash-delimted file format

```
        x,      y,      z,   v
```

Listing 2. Example header from a file stored in a new format format

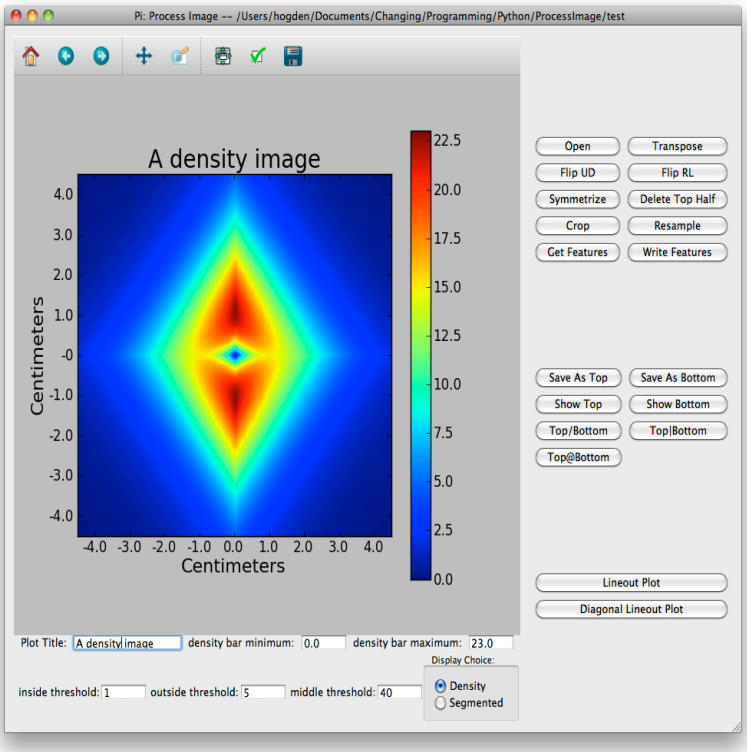
D. New Commands

Figures 1a and 1b show the top-level windows of the 2012 and 2014 version of ProcessImage. Some of the buttons from the original version have been removed in the 2014 version, but in these cases the buttons have either been relabeled or replaced by a button with identical or very similar functionality. Briefly, the deleted buttons and their replacements are:

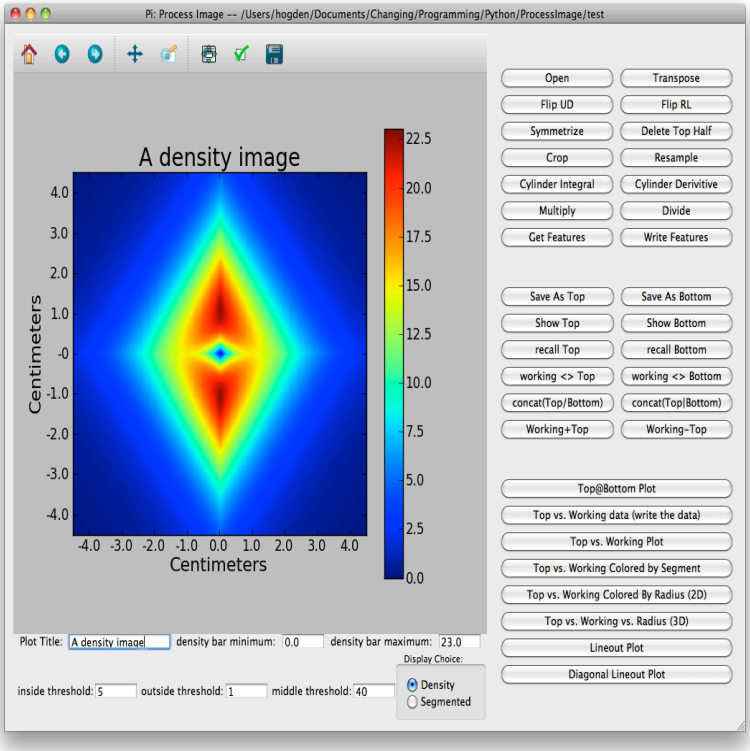
1) **Show Top**: In the 2012 version, this button opened a new window displaying the the image in the "Top" memory. This button is replaced by the new "recall Top" and "working <> Top" buttons. Both of the new buttons are described below.
2) **Show Bottom**: As with "Show Top", this button opened a new window displaying the an image in the memory – in this case, the image stored in the bottom location. This button is replaced by the new "recall Bottom" and "working <> Bottom" buttons, described below.
3) **Top/Bottom**: This button has been renamed "concat(Top/Bottom)", but the functionality is identical
4) **Top|Bottom**: This button has been renamed "concat(Top|Bottom)", but the functionality is identical
5) **Top@Bottom**: This button has been renamed "Top@Bottom Plot", and has been moved next to other buttons that create new plots, but the functionality is retained in the new program.

The new buttons, ignoring those that are merely renamed version of old buttons, are summarize below. The more complex of these are further described in separate sections.

1) **Cylinder Integral**: Treat the value in each pixel as a density, and then replace the value in each pixel with the mass of the rectangular torroid it represents.
2) **Cylinder Derivitive**: Treat the value in each pixel as the mass of the corresponding torroid, and then replace the value in each pixel with the density of the torroid.
3) **Scale Density**: When this button is pressed, the user is prompted to input a floating point factor. Then the value in each pixel is multiplied by the factor.
4) **Scale Size**: When this button is pressed, the user is prompted to input a floating point factor. Then the horizontal and vertical extents of the pixels are multiplied by the factor.
5) **Draw Ellipse**: When this button is pressed, the user is prompted for the half-width and half-height in centimeters of an ellipse. The ellipse will then be drawn on the image centered at the center of the image. Drawing an ellipse does not change the actual pixel values in the image, it merely changes the display of the image. This functionality is somewhat useful for measuring the image, but is largely used in conjunction with the "Spherize top and bottom" function described below.
6) **Add Region**: This button's operation assumes that thresholds have been set to segment the working image into regions, with each region designated by an integer. Pressing the button causes a region designated by the user to be surrounded by a new region with a user-specified mass. The operation of adding a region is similar to the morphological dilate operator except that the pixels added do not generally have the same value as the region they are surrounding. The user gives a desired density for the pixels in the new region. The new region will have the mass specified by the user and a density as close to that specified by the user as practical. This operation is described more fully in Section XXX.
7) **recall Top**: Replaces the working image with the Top image.
8) **recall Bottom**: Replaces the working image with the Bottom image.
9) **working <> Top**: exchange the working image with the top image. Stores the working image in the Top memory location and stores the image from the Top memory location as the working image.
10) **working <> Bottom**: exchange the working image with the bottom image. Stores the working image in the Bottom memory location and stores the image from the Bottom memory location as the working image.
11) **Working + Top**: Replace the working image with a new image created by adding the values of the pixels in the working image to the values of the pixels stored in the top image location, pixel-by-pixel. If the size or sampling of the top image is different from that of the working image, the top image is resampled to match the sampling of the working image.
12) **Working − Top**: Identical to "Working + Top" except that pixel values from the top image are subtracted from the working image.
13) **Stitch bot(top)bottom**: This button opens a new window that allows the user to create a new image made by copying the pixels values inside a circle centered at the origin of the top image into the corresponding positions in the bottom image. If the sample positions of the top and working image are not the same, the top image will be resampled to match the sample positions in the working image. An example is shown in Section



(a) A window from the original version of ProcessImage.



(b) A window from the 2014 version of ProcessImage.

Fig. 1. A comparison of functions in the original and 2014 versions of ProcessImage

XXX.

14) **Top vs. Working Colored by Radius (2D)**: This creates a new window with a two-dimensional plot of the values of pixels in the top image versus the values of pixels in the working image. The points in the plot are given a color corresponding to their distance from the center of the image. An example is shown in Section XXX.
15) **Top vs. Working vs. Radius (3D)**: This creates a new window with a three-axis plot of the values of pixels in the top image versus the values of pixels in the working image with the third axis being the distance of the pixel center from the center of the image. The points in the plot are given a color corresponding to their distance from the center of the image. An example is shown in Section XXX.
16) $\Sigma\rho * r$: Pressing this button opens a new window showing a plot of the summed density times the line length along lines passing through the image at different angles. An example is shown in Section XXX.
17) **Spherized top and bottom**: Pressing this button displays a plot of the spherized density of the top and bottom images. The notion of a spherized density is discussed in Section XXX.
18) **Region Contours top and bottom**: Opens a window allowing the user to choose different threshold for the top and bottom images and plot the region boundaries for both images in a single plot. An example is shown in Section XXX.
19) **Ray length distribution**: This open a new window showing the distribution of lengths of rays originating in pixels of region one. The directions of the rays are random and travel through three-dimensional space, i.e., the image is rotated around the axis of symmetry to create a three-dimensional object, and rays are traced through the object. The lengths of the rays are only calculated for the parts that are in region one. This is discussed more fully in Section XXX.

III. New Commands In Depth

IV. Features

When the "Get Features" button is pressed, a window like that shown in Fig. 2 is displayed. The window shows information about the image (pixel size, rows, columns, etc.) and features calculated for the seed, shell, and husk regions. In this example there are features calculated for three regions but if no shell region is defined the features for two regions will be given.

Most of the features are derived from a few primitive features, e.g., bounding box, volume, mass, surface area, variance of the region, the minimum enclosing circle, and the Legendre moments. For those who want more details of the computations, we describe the primitive features in Section **??**.

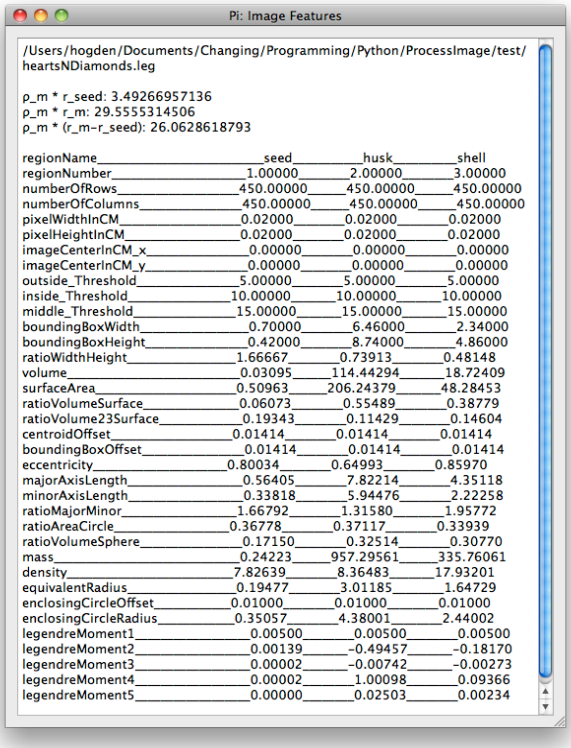


Fig. 2. A window displaying features calculated for the segments of the window shown in Fig. **??**, which is the heartsNDiamonds.leg file analyzed with thresholds of 10, 5, and 15 for the inside, outside, and middle, respectively

A. Volume

The most straightforward way to calculate the volume of an arbitrary pixelized region is to sum the volumes associated with each pixel in the region. The volume added by a pixel is the volume of a rectangular toroid (or cylinder for pixels touching the horizontal mid-line) created by rotating the pixel 360 degrees around the horizontal mid-line. Letting r be the distance from the center of the pixel to the horizontal mid-line, and keeping in mind that r must either be ih, $i \in \mathbb{N}$ (for an odd number of rows) or $ih + \frac{h}{2}$ (for an even number of rows), the volume added by a pixel is

$$dV_i = \begin{cases} \pi \left(\frac{1}{2}h\right)^2 w & \text{if } r = 0, \\ 2\pi r h w & \text{if } r \neq 0 \end{cases} \quad (1)$$

.

We average the volumes obtained from the top and bottom halves of the image to get a final volume.

A complication is that we are actually interested in the volume of the object that has been pixelized instead of the volume of the rotated region, and these are not the same. Attempts have been made to increase the accuracy of area estimates from pixelized images, and these techniques can be extended to volume estimates as well, e.g., MATLAB's "bwarea" function [1] [2]. Nonetheless, for our work to date,

the volume of the pixels in the image region appears to be a sufficiently accurate approximation to the volume of the imaged object.

B. Mass

This feature assumes that the pixel values give densities. The mass added by a pixel is the volume added by that pixel times the density of the pixel.

C. Perimeter and Surface Area

In this work we use "boundary" to mean the edge of an object and "perimeter" to be the length of the edge. Both the perimeter and the surface area are commonly used to calculate features for selected sets of pixels. For example, we can get a feature characterizing the circularity of a region from the area and the perimeter because $area/perimeter^2$ is largest for a circle and has a maximum value of $1/4\pi$. Our interest is in three-dimensional objects, so we focus on surface area rather than perimeter. Nonetheless, since the surface area is calculated for the surface of rotation of the perimeter of a region, we first discuss the perimeter of a pixelized region.

As with calculating the volume, pixelation of an object introduces errors in our measurements of the perimeter length; however, the errors caused by pixelation cannot be ignored because they can become very large when estimating the length of perimeter. Consider a region containing a single pixel. The obvious approach to calculating the perimeter is to simply measure the length of the border of the pixelized object, so for a single pixel the perimeter is $2(w + h)$. For regions comprising many pixels, using the distance around the edges of the pixels will only give an accurate measure of the perimeter if the edges of the object are vertical or horizontal. For edges at any other angle the edge length will be overestimated.

The three images in Fig. IV-C illustrate the problem. If the perimeter is estimated by tracing the pixel edges around the region each of these objects will have the same perimeter. In fact, a pixelized square has a larger value of $area/perimeter^2$ than a pixelized circle — a result that contradicts standard geometry and which would invalidate a measure of circularity.

A common (although not necessarily accurate) approximation of the boundary of an object from a pixelized image is the polygon (call it the edge polygon) connecting the centers of adjacent pixels that are included in the object but which share a side with a pixel that is not in the object. This is the approach used at Los Alamos National Laboratory in the past. As discussed in [3], using the perimeter of the edge polygon as an approximation of the perimeter of the object is accurate for edges that are oriented at an integer multiple of a 45 degree angle, but overestimates the length for edges that are at other angles. Since this measure of the perimeter always results in lengths that are equal or larger than the actual perimeter, better estimates can be obtained using a variety of other techniques [3], [4], e.g., subtracting a bias term. However, even in cases where all of the edges are at integer multiples of 45 degrees, the volume of the region is not completely encompassed by the perimeter, so the ratio of area to perimeter will not match our geometric expectations.

There is no single accurate way to measure the perimeter of every object from its pixelized image. Following what has been done at LANL in the past, we use the edge polygon as our estimate of the boundary. Rotating the portion of the edge polygon that lies above the horizontal mid-line by 360 degrees around the horizontal mid-line produces a surface. Rotating the portion of the polygon that lies below the mid-line produces a second surface. We use the average area of these two surfaces as the surface area.

We calculate the perimeter and surface area in one pass through the data. The approach we use is to move a 2x2 pixel window over the image looking for horizontal, vertical, or diagonal edges. Since a 2x2 window of binary pixel values can have only 16 different pixel combinations, we pre-calculate the amount of surface area added by each of the 16 windows.

With this approach, single pixels do not contribute to the surface area, nor do isolated sets of pixels centered on the horizontal mid-line. In software previously used at LANL, small values were added to the surface area for isolated pixels and pixels on the mid-line. Differences between the surface area calculations of previous LANL software and this package appear to be minimal.

D. Region Statistics

The centroid and variance of selected sets of pixels are used in some of our image features. For the purpose of this calculation, the image is binary with pixels in the region having value 1 and pixels outside the region having value 0. The image variance calculation is not obvious, so we describe it here.

A set of pixels can be treated as the convolution of a single pixel, centered at the origin, with a set of Dirac delta functions positioned at the pixel centers [5]. It is well known that the variance of a convolution is the sum of the variances of the terms in the convolution [6] — a fact we use when calculating the variance:

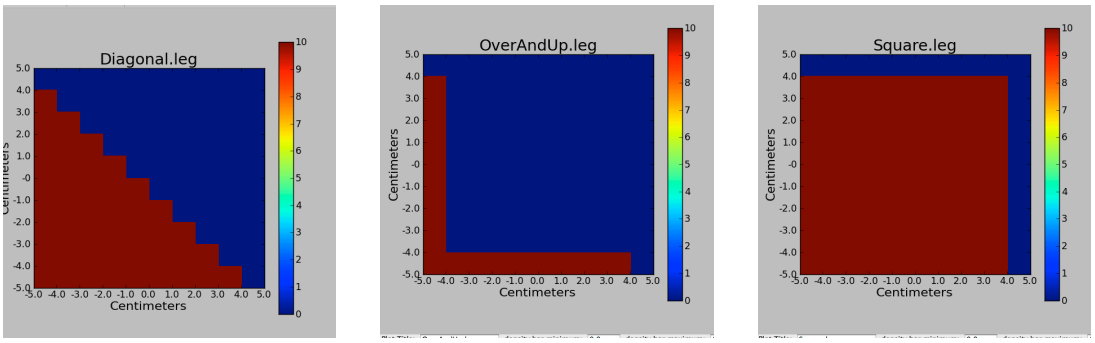
$$\mathtt{Var}_x = \mathtt{Var}_x\left(\mathtt{centers}\right) + \mathtt{Var}_x\left(\mathtt{pixel}\right), \tag{2}$$

where the subscript x shows that this is the variance along the x-axis. The equations for the variance on the x-axis and the y-axis are nearly identical so we only give equations for the x-axis variance.

Calculating the variance of the pixel centers is straightforward. Let x_i be the x-axis position of the center of the i^{th} pixel and N be the number of pixels in the image. The standard formulas for the variances of the pixel centers on the x-axis are given in EQs 3 and 4.

$$\mathtt{Centroid}_x = \frac{\sum_i x_i}{N} \tag{3}$$

$$\mathtt{Var}_x\left(\mathtt{centers}\right) = \frac{\sum_i x_i^2}{N} - \left(\mathtt{Centroid}_x\right)^2 \tag{4}$$



(a) (b) (c)

Fig. 3. A straight line is the shortest distance between two points. However, when a straight line is pixelized, as in the case of the diagonal edge in Fig. 3a, the straight line is not necessarily shorter than other paths. Using the pixel edges to measure the distance, the pixelized straight line is the same length as paths that we know to be longer, such as those in Figs. 3b and 3c.

The variance of a single pixel centered on the origin (and therefore having mean 0) can be calculated from the 0^{th} and 2^{nd} order moments, m_0 and m_2, respectively:

$$m_0 = \int_{\frac{-w}{2}}^{\frac{w}{2}} \int_{\frac{-w}{2}}^{\frac{w}{2}} 1 dx dy = w^2 \tag{5}$$

$$m_2 = \int_{\frac{-w}{2}}^{\frac{w}{2}} \int_{\frac{-w}{2}}^{\frac{w}{2}} x^2 dx dy = \frac{w^4}{12} \tag{6}$$

$$\texttt{Var}_x\left(\texttt{pixel}\right) = \frac{m_2}{m_0} = \frac{w^2}{12} \tag{7}$$

The covariance value for uniform distribution with the support being a single pixel centered on the origin is always 0, so the covariance value for a region is simply the covariance value of the pixel centers. Using the standard equation:

$$\texttt{Cov} = \frac{\sum_i x_i y_y}{N} - (\texttt{Centroid}_x)(\texttt{Centroid}_y) \tag{8}$$

V. Example Calculations

For those who wish to test their understanding of the image features and/or verify the accuracy of the calculations, we calculate the features for the example image shown in Fig. 4.

The image is a 20x20 grid of pixels with each pixel having height and width set to 0.5 cm. The image is a set of concentric squares. The center of the set of squares is offset 0.25 cm to the right and 0.25 cm below the center of the image. The density values in each square, from the center out, are 0, 1, 2, 3, 4, 5, 4, 3, 2, 1, 0.

A segmented version of the image is shown in Fig. 5. When the image is segmented we have a seed that is a 2.5×2.5 cm offset square extending from -1 cm to 1.5 cm on the x-axis and from -1.5cm to 1 cm on the y-axis. The seed is surrounded by a hollow square (the shell) that extends from -2.5cm to 3 cm on the x-axis and from -3 cm to 2.5 cm on the y-axis. The density values for each pixel in the seed and shell are shown in Fig. 7.

We focus on the calculations of the primitive features but mention the features derived from them. We do not check

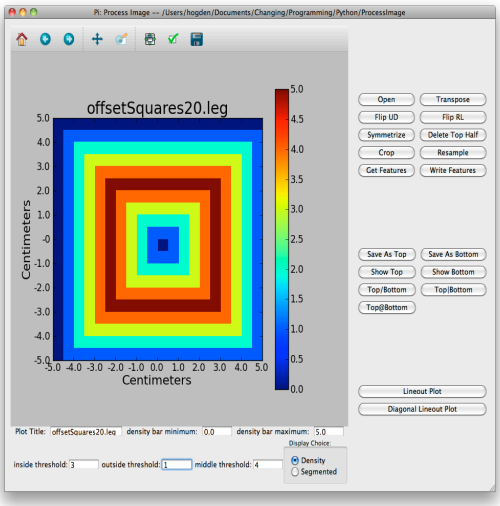


Fig. 4. A simple image used for checking the accuracy of the calculations

the calculations of the Legendre moments by hand in this paper, but the Legendre moment algorithms have be validated in previous work [7]. We compared results from our independently programmed algorithms to the previous results for confirmation.

A. Seed Features

1) Bounding box: The bounding box is the same size as the seed so has width and height of 2.5 cm. The center of the bounding box is the same as the center of the seed so is at (0.25, -0.25). Therefore the distance (or equivalently the offset) between the center of the bounding box and the center of the image is $\sqrt{0.25^2 + 0.25^2} = .353553$. The ratio of the bounding box height and width is 1.
2) Volume: When rotated 360 degrees around the horizontal mid-line, the section of the seed above the horizontal mid-line is a cylinder with radius 1 cm and width 2.5 cm. The cylinder with these dimensions has volume = $\pi r^2 w \approx 7.853982$. The rotated section below the horizontal mid-line has radius 1.5 so volume ≈ 17.671459.

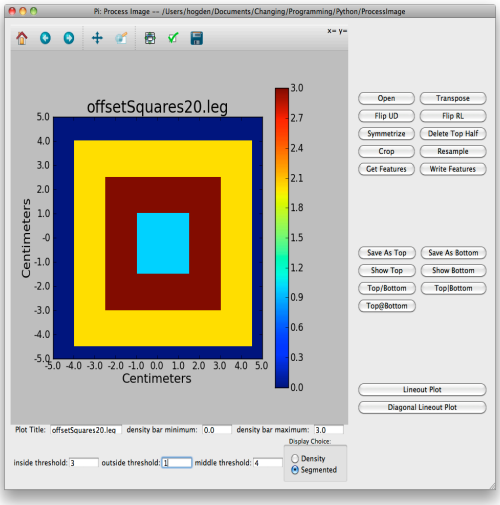


Fig. 5. The segmented version of simple image from Fig. 4

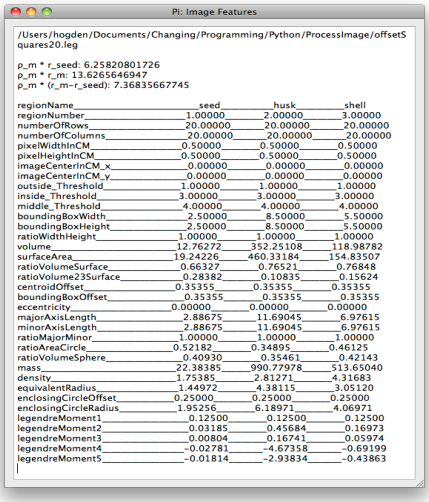


Fig. 6. Features calculated from the image in Fig. 4

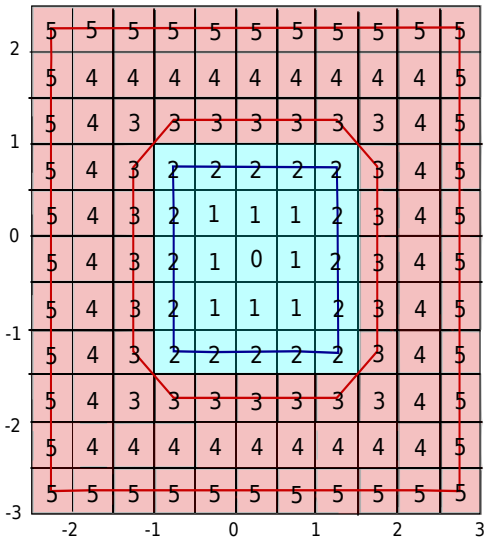


Fig. 7. The shell and seed portion of Fig. 4 with the density values shown by the numbers in each pixel. The boundary pixels of the seed are connected by the blue polygon. The inner and outer boundary pixels of the shell are connected by the red polygons. The x and y coordinates are shown by the numbers outside the pixels.

The average volume of the rotated seed is shown in Fig. 6. The volume V is used to calculate various ratios and is also used to calculate the equivalent radius using the relationship $r = \sqrt[3]{\frac{3V}{4\pi}} = 1.449722$.

3) Surface area: recall that, for the purpose of calculating surface area, the boundary of an area is the polygon comprising the line segments that connect the centers of the boundary pixels. The polygons connecting the boundary pixels for the shell and the seed are shown in Fig. 7. The surface area of the seed is not the surface area of the two half-cylinders used for finding the volume because the polygons are inset a half pixel from the edges used for finding the volume. For finding the surface area the top cylinder has radius = 0.75 and width = 2. The lower cylinder has radius = 1.25 and width = 2. The surface area of a cylinder of radius r and width w is the area of the two ends plus the area of the side, which amounts to $2\pi r(r + w)$. Therefore the surface area of the top cylinder is 12.959070 cm^2 and the surface area of the bottom cylinder is 25.525440. The average of the surface areas is 19.242255.
4) Region Statistics: The centroid is at (0.25, -0.25), just like for the bounding box, so the centroid offset is the same as the bounding box offset. The easiest way to calculate the seed variance for both axes is to use EQ. 7, treating the seed like a 2.5×2.5 cm pixel: $2.5^2/12 = 0.520833$. The covariance for a uniform square (note that density values are not used) is always 0. EQ. **??** then gives the result 2.886751 cm for the Major Axis length, EQ. **??** gives the same result for the minor axis length: 2.886751 cm, and the eccentricity is seen to be 0.
5) Minimum Enclosing Circle: Inspection of the Fig. 5 shows the circle with its center on the line $y = 0$ that touches the lower-left and lower-right corners of the seed is the minimum enclosing circle – if it becomes any smaller at least one point in the seed will be excluded. EQ. **??** shows that the center of the circle is at (0.25,0) so the enclosing circle offset is 0.25. Furthermore, EQ. **??** gives us the radius of the circle: 1.952562 cm. Since the area of the seed is $2.5^2 = 6.25$, and the area of a circle with radius 1.952562 is 11.977322, the ratio of the seed's area to the area of the Minimum Enclosing Circle is 0.521819. The volume of a sphere with radius 1.952562 is $\frac{4}{3}\pi r^3 = 31.181958$. Therefore, dividing the volume (calculated above) by the volume of the minimum enclosing sphere, we get 0.409298, which is

r	dV
0.25	0.196349541
0.75	0.589048623
1.25	0.981747704
1.75	1.374446786
2.25	1.767145868
2.75	2.159844949

TABLE I
THE VALUES GIVEN IN THIS TABLE SIMPLIFY THE PROBLEM OF CALCULATING THE MASS OF THE SEED AND THE SHELL FOR THE EXAMPLE PROBLEM. THE DISTANCE BETWEEN THE PIXEL CENTER AND THE HORIZONTAL MID-LINE IS GIVEN BY r. THE VOLUME ADDED BY A PIXEL ROTATED 360 DEGREES AROUND THE HORIZONTAL MID-LINE IS dV.

the ratio of the volume to the sphere.

6) Mass: Holding the pixel height and width constant, EQ. 1 shows that dV (the amount of volume associated with a rotated pixel) varies only with the distance between pixel center and the horizontal mid-line. Given that, the easiest way to calculate the mass is to first find dV for each row of pixels in the image. Table 5 shows the relationship of r to dV for the rows of pixels in the example image. The density of every pixel in the rows that are 0.25 cm from the horizontal mid-line is multiplied by the same dV so we can sum those densities and multiply the sum by dV. The sum of the densities in rows that are 0.25 cm from the horizontal mid-line is 13. The summed densities for rows with $r = 0.75$ is 17. The summed densities for $r = 1.25$ is 10. Multiplying these values by the corresponding dV and summing gives a total mass of 22.383848. Dividing the mass by the volume gives the density of 1.75385.

B. Shell Features

1) Bounding box: The bounding box is the same size as the outside of the shell, so has width and height of 5.5 cm. The center of the bounding box is the same as the center of the seed so the offset between the center of the bounding box and the center of the seed is the same as that of the seed. The ratio of the bounding box height and width is 1.
2) Volume: We can calculate the volume of the shell by calculating the volume of the shell with the seed and then subtracting the volume of the seed. The section of the seed and shell above the horizontal mid-line is a cylinder with radius 2.5 cm and width 5.5 cm. The cylinder with these dimensions has volume = $\pi r^2 w \approx 107.992247$. The rotated section below the horizontal mid-line has radius 3.0 so volume $\approx$ 155.508836. The average volume of the rotated shell and seed is 131.750542. Subtracting the volume of the seed we get 118.987822. The equivalent radius using the relationship $r = \sqrt[3]{\frac{3V}{4\pi}} = 3.051204$.
3) Surface area: The surface area of the shell includes the surface area of the outside of the shell and the inside of the shell. Note that the inside of the shell is not the same as the outside of the seed for the purposes of calculating the surface area because the boundary polygons are inset by half a pixel. The surface area of the outside of the shell is the average of the surface areas of the top cylinder and the bottom cylinder. The top cylinder has radius = 2.25 and width = 5 and so a surface area of 102.494460. The lower cylinder has radius = 2.75, width = 5, and so surface area of 133.910387. The average outer surface area is 118.202424. Five lines make up the polygon for the top of the inside portion of the shell. The left and right vertical lines contribute the surface area of disks with radius 0.75 cm, which comes to 3.534292 combined. The left and right diagonal lines contribute the surface area of the side of a truncated cone, which comes to 8.885766 combined. The horizontal line at the top of the inside polygon contributes a surface area of 15.707963. The total for the top inside surface area for the shell is 28.128021. The bottom half of the polygon making up the inside of the shell also has 5 lines contributing to the surface area. The left and right edges contribute 9.817477 combined. The diagonal lines contribute 13.328649 cm^2 when combined. The bottom horizontal line contributes 21.991149 cm^2. So the bottom half of the inside surface of the shell has a surface area of 45.137275 cm^2. The average of the top and bottom inside surface areas is 36.632648. Adding the inside and the outside surface area give a combined surface area for the shell of 154.835072.
4) Region Statistics: The centroid is at (0.25, -0.25), just like for the bounding box, so the centroid offset is the same as the bounding box offset. The n^{th} order moment for the shell can be calculated by subtracting the n^{th} order moment of the seed from the n^{th} order moment of the combined shell and seed. Since the width of the seed is 2.5 cm and the width of the shell is 5.5 cm, EQ.5 shows the 0^{th} order moment of the shell is $5.5^2 - 2.5^2 = 24$. The second-order moment of the shell is $\frac{5.5^4}{12} - \frac{2.5^4}{12} = 73$. Then from EQ. 7 we get that the variance of the shell is 3.041666. The variance on the y-axis is the same, so EQs. **??**, **??**, and **??** give values of 6.976150, 6.976150, and 0 for the major axis, the minor axis, and the eccentricity, respectively.
5) Minimum Enclosing Circle: Similar to the situation for the seed, the minimum enclosing circle touches the lower-left and lower-right corners of the shell with its center at (0.25,0). The enclosing circle offset is clearly 0.25, and EQ. **??** gives us the radius of the circle: 4.069705 cm. Since the area of the shell is $5.5^2 - 2.5^2 = 24$, and the area of a circle with radius 4.069705 is 52.032628, the ratio of the seed's area to the area of the minimum enclosing circle is 0.461249. The

volume of a sphere with radius 4.069705 is 282.343274. Therefore, dividing the volume (calculated above) by the volume of the minimum enclosing sphere, we get .421430 for the ratio of the volume to the sphere.

6) Mass: As with the seed, we use Table 5 to help calculate the mass of the shell. The sum of the densities for the shell in rows that are 0.25 cm from the horizontal midline is 48. The summed density values for progressively more distant rows are: 48, 63, 85, 101, and 55. Multiplying these values by the corresponding dV and summing gives a total mass of 513.65040. Dividing the mass by the volume gives the density of 4.316831.

C. Combined Features

The three features listed at the top of the features window can be confirmed by hand calculation from the volumes and masses calculated above. To simplify comparisons, note that ρ_m is the same as the density of the shell and r_seed is the same as the equivalent radius of the seed. One number that has not been directly calculated above is r_m, which, as stated above, is the radius of the sphere having the same volume as the seed plus the shell. This value is 3.156612464.

VI. Conclusion

We have described a program that allows the user to quickly and easily perform many of the most common tasks needed to compare density images, tasks that were previously performed using MATLAB tools with a command line interface. Not only were a graphical user interface and new image features added, but details of feature analyses and example calculations have been presented here to enable other researchers to provide feedback and refine the features. The example calculations help verify that the values generated by the software are accurate. However, the example calculations are performed on typical images – images with an even number of rows, for which the pixel height and width are the same, and for which reasonable thresholds have been set. Further testing is necessary to verify the calculations for images that do not meet these criteria.

References

[1] MATLAB, *version 7.10.0 (R2010a)*. Natick, Massachusetts: The MathWorks Inc., 2010.

[2] W. K. Pratt, *Digital Image Processing (2nd Edition)*. New York, New York: John Wiley & Sons, Inc., 1991.

[3] L. Dorst and A. W. M. Smeulders, "Length estimators for digitized contours," *Computer Vision, Graphics, and Image Processing*, vol. 40, no. 3, pp. 311–333, Dec. 1987.

[4] D. Coeurjolly and R. Klette, "A comparative evaluation of length estimators of digital curves," Nov. 06 2007. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00185088/en/; http://hal.archives-ouvertes.fr/docs/00/18/50/88/PDF/116125-2.pdf

[5] R. M. Haralick and L. G. Shapiro, *Computer and Robot Vision*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1992.

[6] R. N. Bracewell, *The Fourier transform and its applications*, 2000.

[7] K. McLenithan, F. Hemez, and R. Krajcik, "Analysis of late-time implosion images (U)," Los Alamos National Laboratory, Fiscal Year 2010 Report of the Penetrating Image Project LA-CP-10-1778, 2010.