Convex Hull BG

a new background subtraction algorithm

by Koen van Dijken

Jul 17, 2016

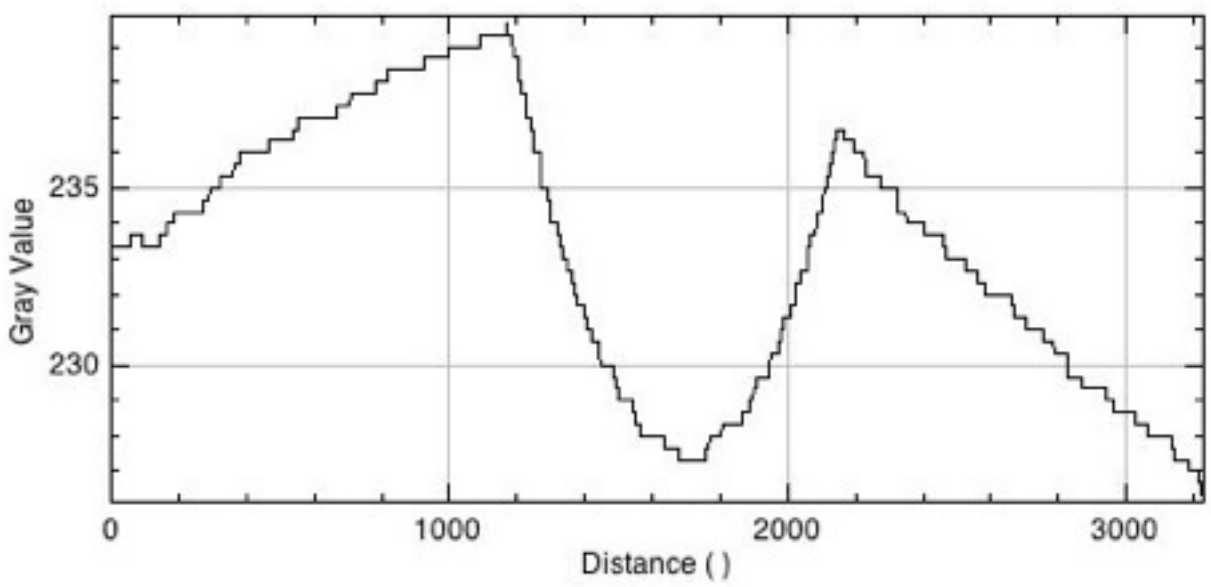
Motivation for another algorithm for background correction

In correction for brightfield microscopy images the standard Background Subtracter of ImageJ has some unwanted effects. Let me show an example.

This is a low magnification image of a part of the female genitalia of Coleophora flavipennella. What is obvious is that there is quite severe vignetting. As this is part of the genitalia this image needs to be stitched to the image of the other parts. Before stitching usually it is best to have the best even illumination of all part images of the stitch. So we need to remove the vignetting.

If we use the standard Rolling Ball algorithm (which is used in Subtract Background in ImageJ) we get the following result for the background:

When we create a profile in the horizontal direction over the middle of the background (can't seem to get the Flatten command to work) gives the following profile:

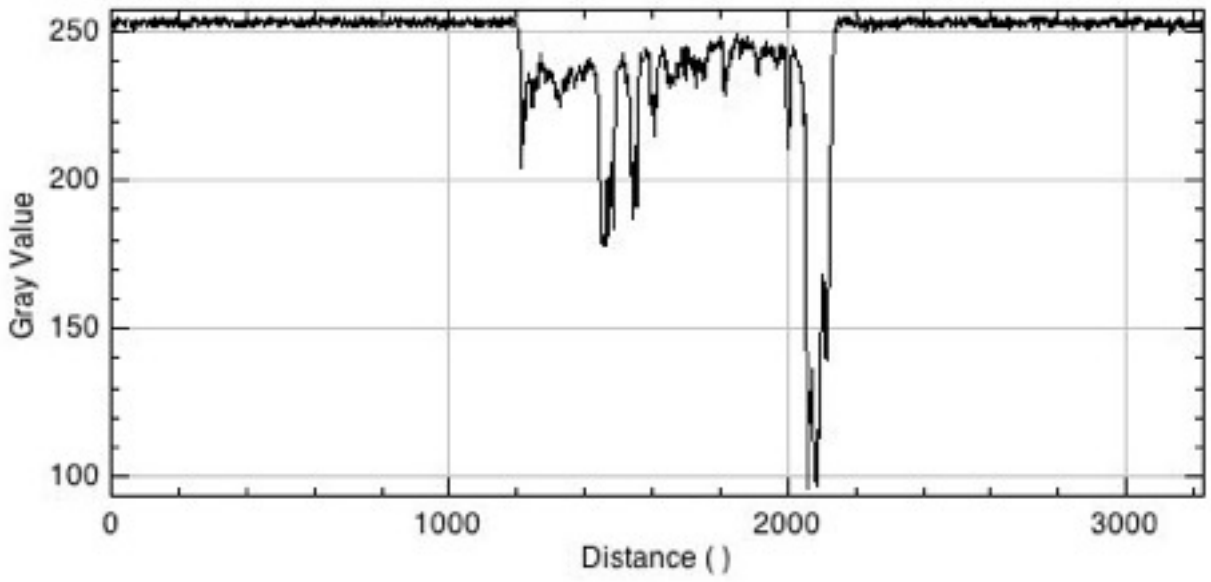


There is a dip in intensity at the location where the sack structure is in the original image. This is caused by the algorithm: a ball is rolled over the top of the image's intensity profile such that it always stays on top of the intensity profile. The intensity of the supposed background on any x-y location in the image is derived from the lowest position of the balls surface. On large structures, like this sack, the ball drops down a bit and so the supposed background intensity.

Correcting the original image with the supposed background gives the following result:

(This is with option ‘separate colors’).

The intensity profile of the final result of Rolling Ball is given in the following figure:



With the Rolling Ball algorithm the original image is corrected by adding the distance above the intensity profile of the supposed background to the intensity of the original image. This makes the background (the real bright-field background in the image) being lifted to the maximum value, but also the pixels which are part of the object being imaged are being lifted. Because of the drop in intensity of the supposed background at the x-y of the object in the image, the correction to be added to the original image (the distance above the profile of the supposed background) is even larger than on the actual background. This gives a reduction in contrast of the image. This is not what we want.

New algorithm

The drop in intensity of the supposed background can be decreased by increasing the size of the rolling ball. In extreme, this rolling ball gets an infinite diameter and reduces to a plane. With a rolling ball of infinite size the ball cannot drop down in the original image's intensity profile. By using the Rolling Ball algorithm with a ball of infinite size the rolling ball describes nothing more than the convex hull over the intensity profile. This convex hull is a 3D convex hull: 2 dimensions for x and y, and one dimension for the intensity. You can think of it in another way as wobbling a flat plane on top of the intensity plot.

Using a ball of infinite size prevents the ball from dropping into dips in the intensity profile of the original image. It can only drop towards the edges of the image. Usually this is not a problem with bright-field images if illumination is fairly even. Vignetting in the image caused by the microscope objective gives lower intensity towards the edges and corners of the image, and higher intensity in the centre of the images.

The new algorithm takes the following steps:

- create a 3D Convex Hull over the intensity profile of the original image. This hull is the intensity of the supposed background.
- extract the faces of the convex hull (only the faces on 'top' of the hull, remember the complete Convex Hull is an in itself closed plane in 3 dimensions)
- insert these faces into a spatial index
- for each pixel (x,y) in the image
  - find the corresponding face of the hull by using the spatial index
  - compute the intensity coordinate of the face at (x,y)
  - correct the pixel intensity similar as in the rolling ball algorithm by using the face intensity at (x,y) and the maximal intensity

This new algorithm is dubbed "Convex Hull BG"

Optimisation of Convex Hull BG

Do not sample every pixel for constructing the convex hull

Calculation of the Convex Hull can take a lot of time on an image of several megapixels. In its default implementation the convex hull is derived for all (x,y) in the image with their respective intensities. It turns out that this is not necessary. When scanning the image it suffices to take leaps of several pixels (in x as well as in y direction) and for every leap we sample the pixels in that leap and take the maximal value of the intensity of these pixels. The deviation from the sample-every-pixel approach is that the sampled intensity is not entirely on the exactly right position, but it is at most the leapsize pixels wrong. The derived faces of the hull ill have a slightly different slope. This slope is what counts because that is what is used in deriving the intensity of the derived background.

In practice the difference in image is not visible, whereas execution times are much shorter.

Cache last found face

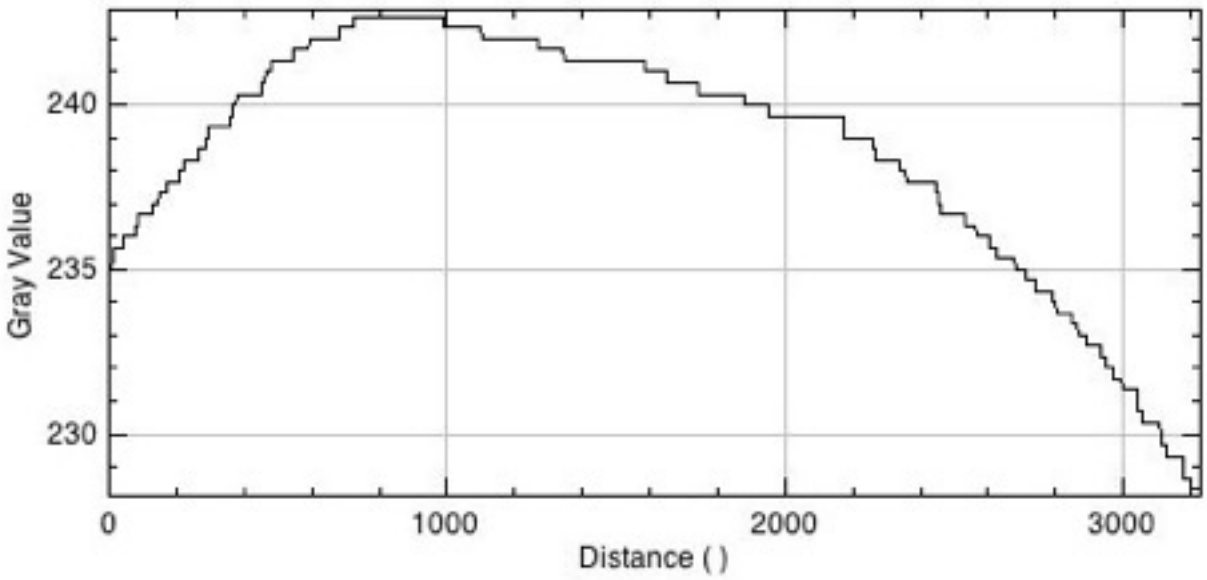
When visiting every pixel in the image to find the corresponding face in the spatial index we can first try the last found face instead of delving into the index immediately. The previously visited pixel is very close to the current pixel, and chances are high that it belongs to the same face as the current pixel. This optimisation is probably not very effective because looking up the face in the index is also quite fast.

Testing Convex Hull BG

Using the same image as was used for testing the Rolling Ball algorithm we get the following result for the derived background:

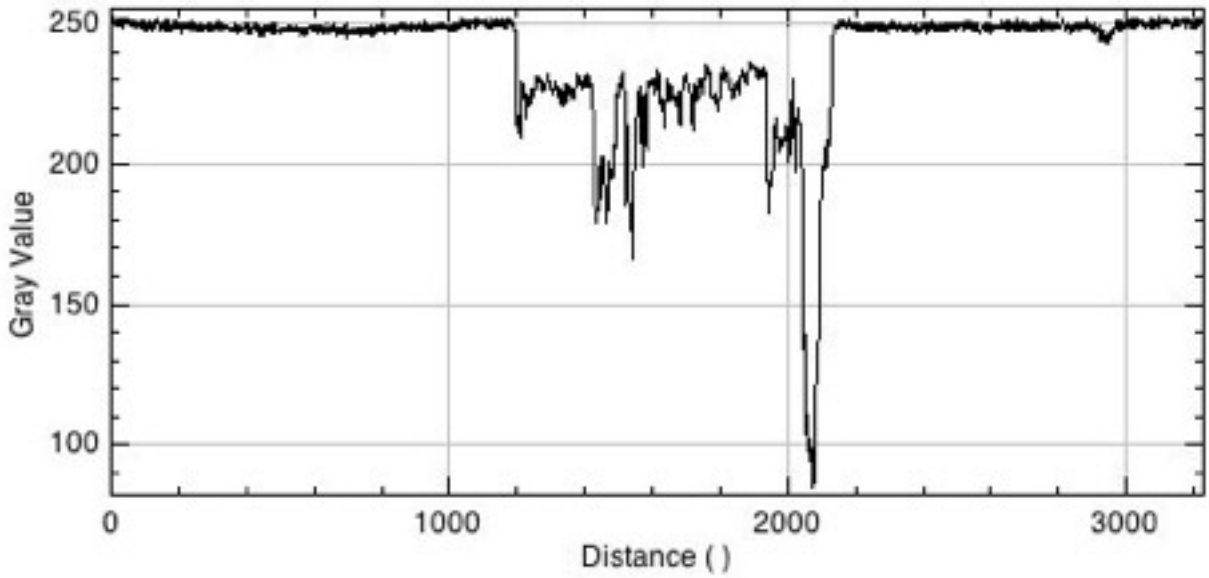
with the following intensity profile over the same selection as before:

It shows that the drop in intensity is not present to the same extent as in the Rolling Ball algorithm.



With the following resulting final image:

with this intensity profile:



Performance

On the test image (which is a 16 megapixel RGB-image) I get the following times for running the algorithm

Algorithm	settings	time (seconds)
Rolling Ball	radius = 1200	44.3
Convex Hull BG	speedup = 1	25.8
Convex Hull BG	speedup = 4	1.8

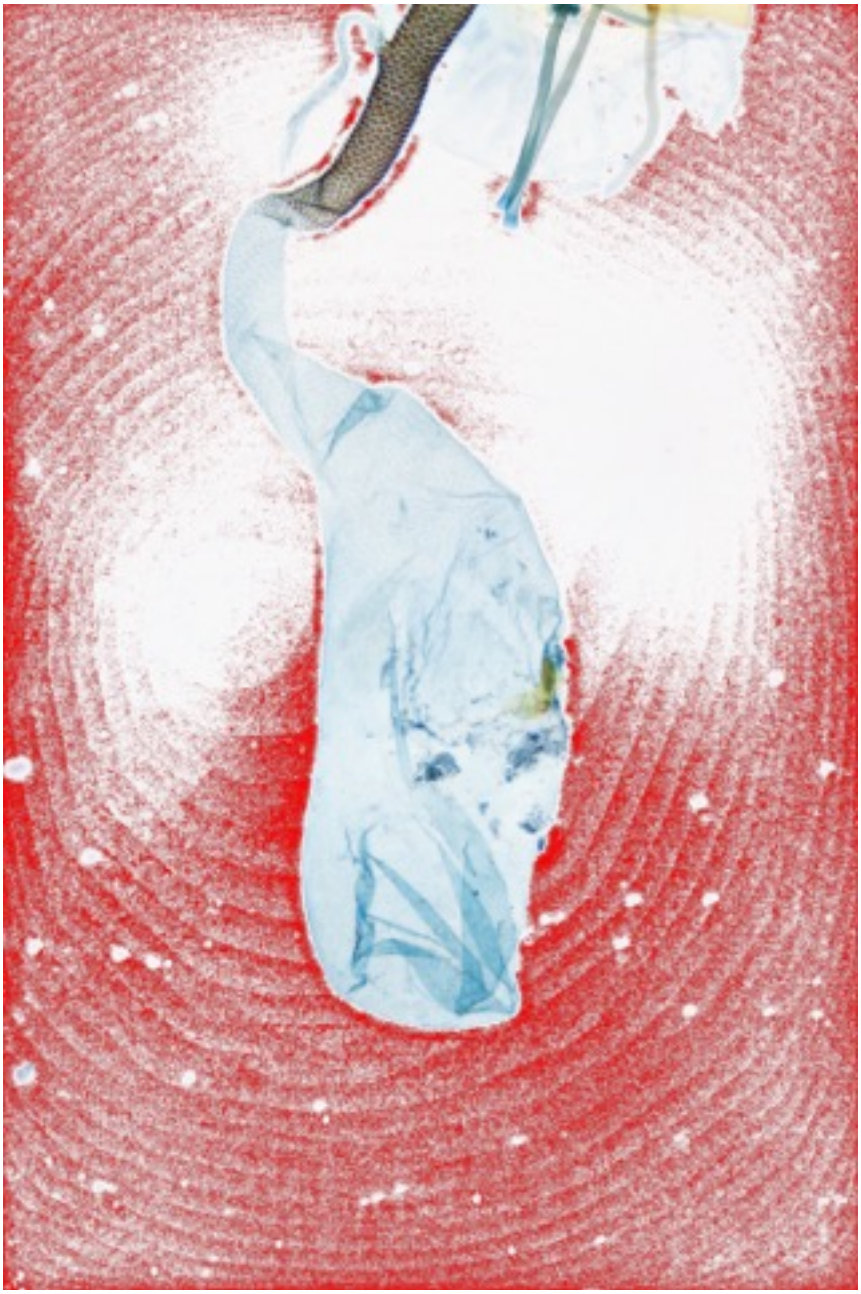
Implementation issues

When testing a face for a pixel (x,y) we have to take robustness issues into account. These robustness issues always play a role in computation geometry algorithms. The part to look out for is the ‘inside’ test. This tests whether a certain (x,y) is inside a polygon. Special cases as (x,y) on an edge or equal to one of the vertices of the polygon have to be tested and taken care off.

Characteristics and how to use

float-int conversions

In RGB images the calculation of the final intensity involves a float to int conversion. This results in banding in the final image which is visible when we threshold the final image:



This banding looks worse than it actually is, because these bands represent a deviation of an intensity value of at most 1.

If we recognise this problem and take the following steps
- split channels of original image
- convert each channel from 8 or 16-bit to 32-bit float
- run Convex Hull BG on each channel separately
- merge these three channels

- convert to RGB

we get rid of the banding as seen in this figure

The first version of the plugin works on stacks in ImageJ, but as a result of something still unexplained a stack of float images results in trouble getting a displayable result. Because of that we will have to split any original tiff into its separate channels and run Convex Hull BG on it. Another possibility is running Window/Level Reset on any slice in the stack.

Irregular backgrounds

This algorithm cannot handle backgrounds which decrease and increase in intensity. In other words, the intensity can only rise and then lower again in any direction. It cannot handle dips in intensity. Local dips in background intensity will not be corrected.

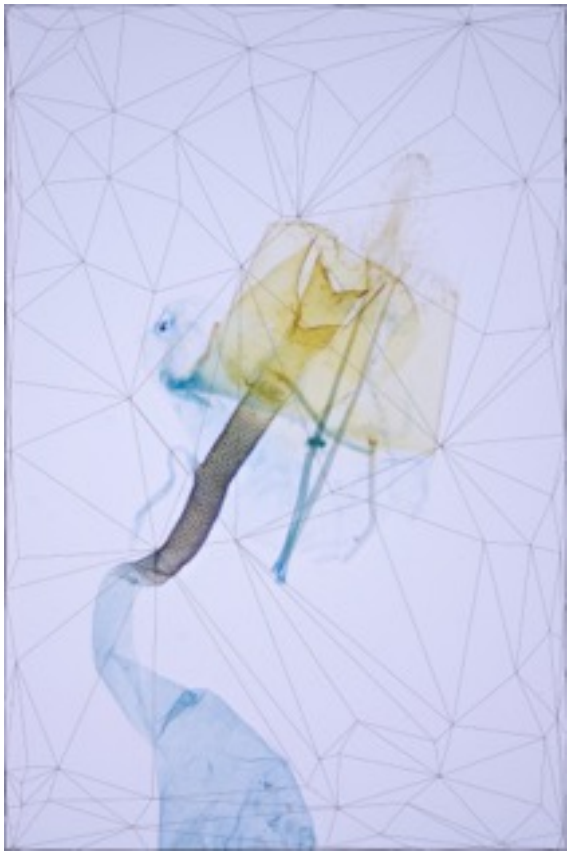
Susceptibility to outliers

Pixels surrounding pixels with extreme high intensity will not be regarded as the background, but as part of an object. This will result in lower final intensity that what is wanted for background pixels. This is not unique to this algorithm however. Also the Rolling Ball algorithm suffers from it, as the ball also has to roll over such an outlier. Clever removing of outliers is the clue, but can be hard.

In this example there is an outlier in the top right corner of the image which results in a darker than expected background. This is visible as the white patch in following thresholded image:

If we remove the outlier in the original image this results in a cleaner background, as visible in this thresholded image:

The difference is obvious when we overlay the Convex Hull over the original image, left with outlier, right without outlier



Implementation

In its current implementation Convex Hull BG for ImageJ is implemented as a plugin. It is distributed as a .jar file Convex_Hull_BG-0.9.jar.

It makes use of the following libraries:

QuickHull3d by Richard van Nieuwenhoven, see https://github.com/Quickhull3d/quickhull3d
JSI by aled, see https://github.com/aled/jsi

through dependencies of JSI it also needs the following jars

trove_3.0.3.jar
slf4j-api-1.5.11.jar
slf4j-nop-1.5.6.jar

These required jars are included in Convex_Hull_BG-0.9.jar

The jar of this plugin can be dropped in the plugins folder of ImageJ and shows itself as “Convex Hull BG” in the Plugins menu.

Issues

- although Convex Hull BG does work on stacks, it has problems with stacks of 32-bit float images. After running Convex Hull BG for every slice of the stack Image -> Adjust -> Window/ Level -> Reset -> Apply has to be executed.

If you find any issues, bugs etc, mail koen.van.dijken@gmail.com