rVividImg

Hao Li

2019/2/9

Description

- The package provides functions to amplify the patterns of certain image by adjusting the pixels in the form of matrices
- The processes may include sharpening, psuedocoloring and normalizing, etc.
- Mission handling for multiple files is enabled with the tiff package.

Installation

Quickstart

With Matrix-like formed data

Let's face it. You don't want to learn anything, nor do I. So we will start by playing with our lovely volcano data(Topographic information for Maunga Whau on a 10m by 10m grid).

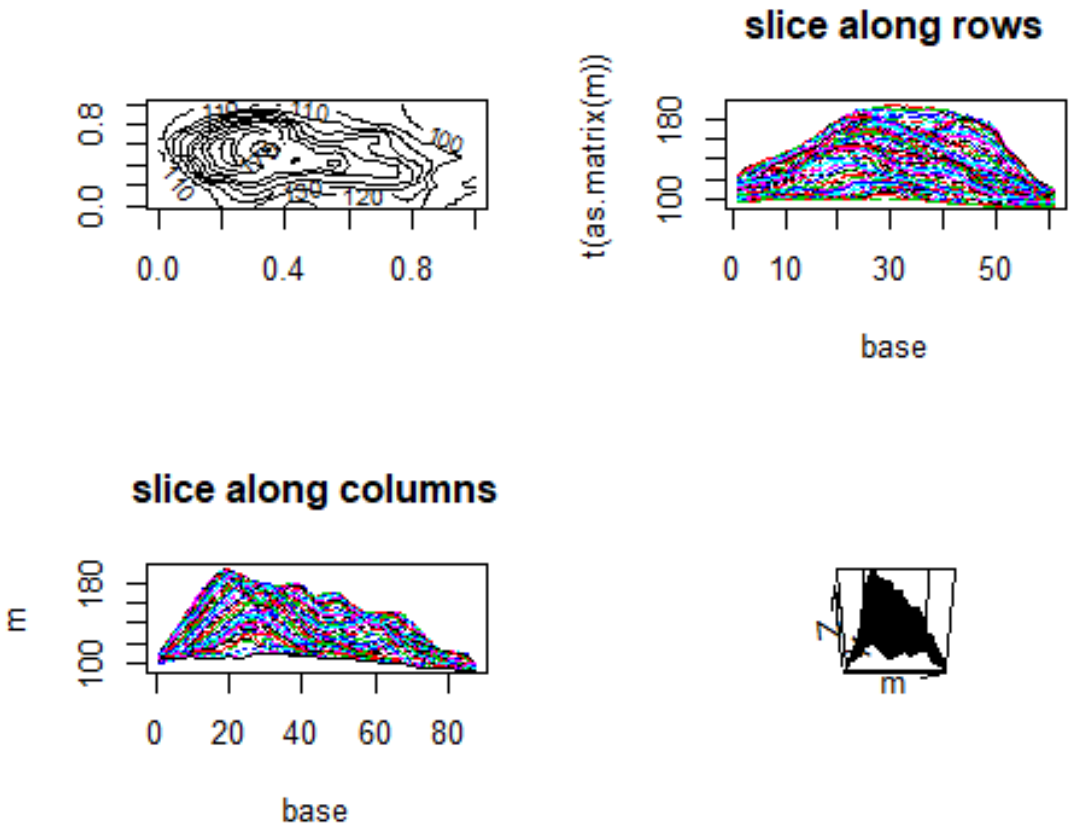
```
#head(volcano)
nrow(volcano);ncol(volcano)
```

```
## [1] 87
```

```
## [1] 61
```

We can preview the distribution of the values on grids(pixels) using image_slice() function. the plots show copies of mountains being cut in different directions. To use it, start library(rVividImg), otherwise the computer will complain.

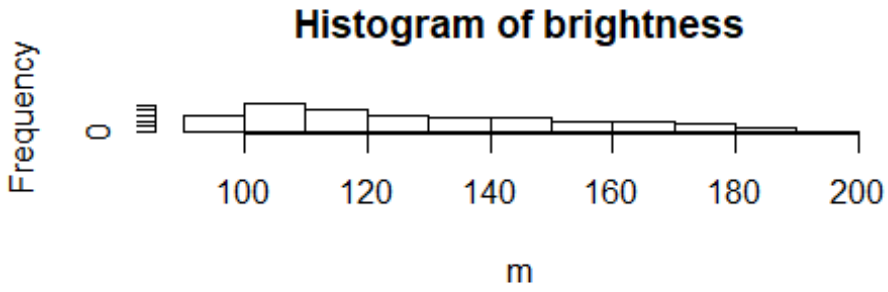
```
library(rVividImg)
image_slice(volcano)
```

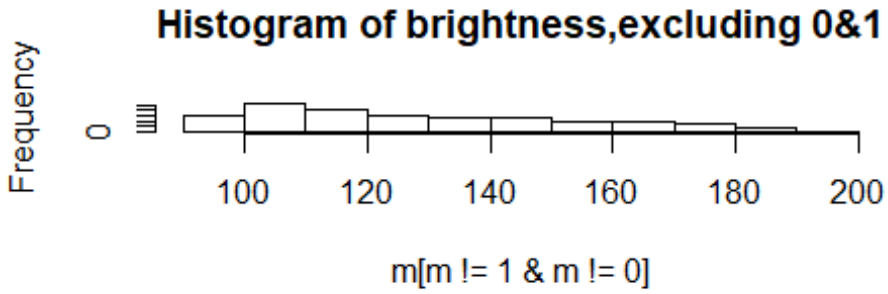


This pixel-to-shape conversion, although seemingly unrelated to image processing, will be applied to images to help you identify patterns later.

We can get the 1D summary by using image_summary()

```
image_summary(volcano)
```





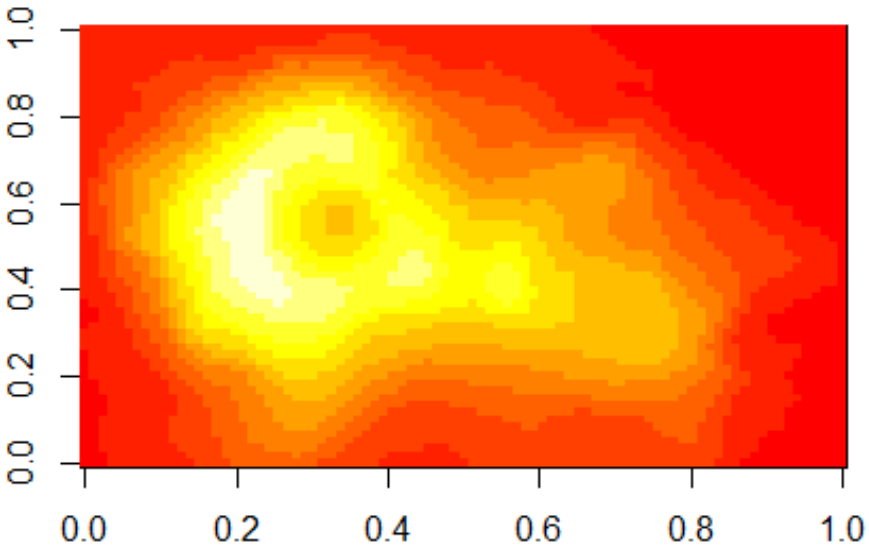

```
## $type
## [1] "greyscale"
##
## $summary
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    94.0   108.0   124.0   130.2   150.0   195.0
##
## $summaryExc
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    94.0   108.0   124.0   130.2   150.0   195.0
```

If not normalized, the graph below and the summary excluding 0 and 1 will have no meaning. Also we are not plotting brightness. So the title for this plot would be 'Histogram of height/depth velue' instead. The peak would be on the highest level of brightness, around which the frequency is small. In an image there would be very bright/ very dark points which we may not want to include or alter.

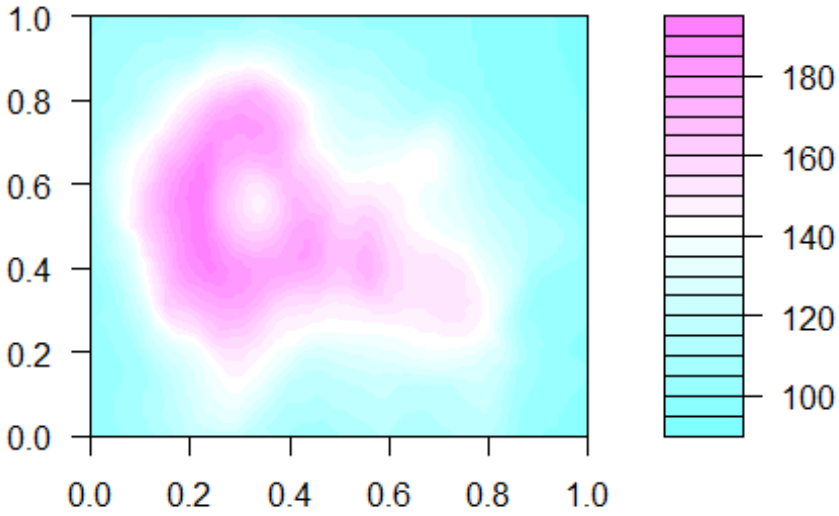
Some factors of rendering may be adjusted according to the statistical value of pixels.

In R, we can plot a image and color according to value using image() filled.contour(). This method is feasible for bitmaps which are essentially arrays, but not with the same functions. Move on to see how we handle a real image.

```
image(volcano)
```

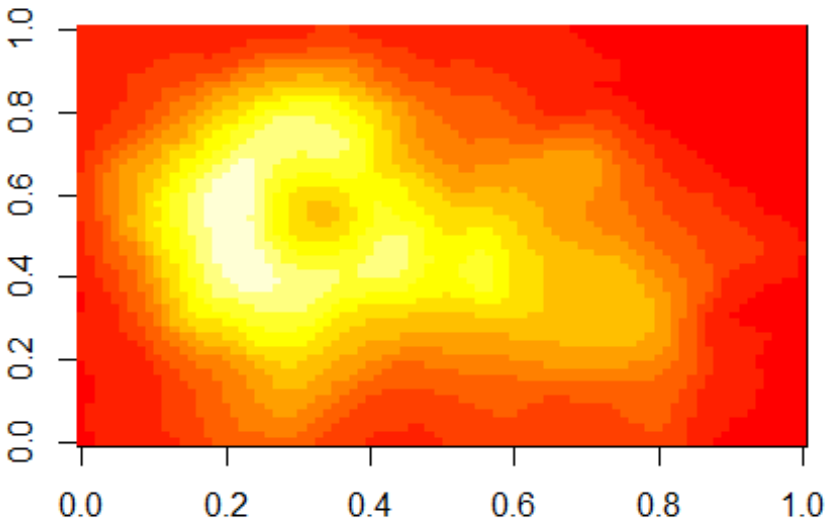



```
filled.contour(volcano)
```

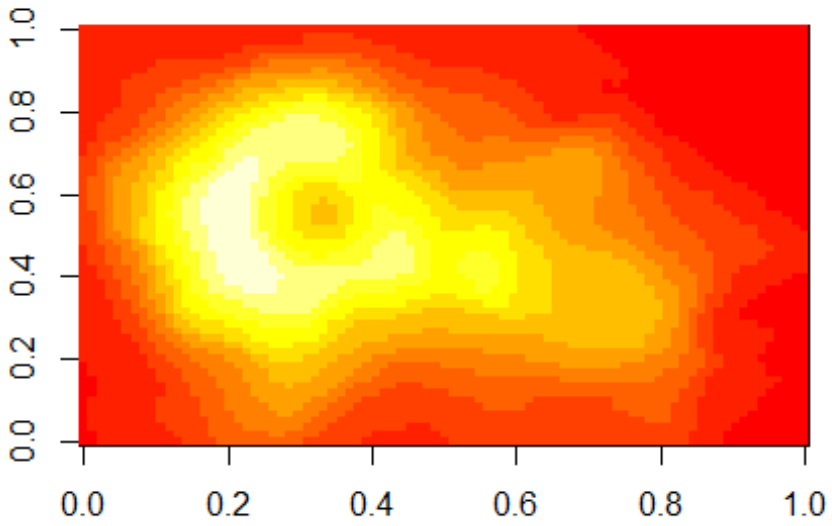


You can smooth the image or see where the gradient is large by using some of these functions.

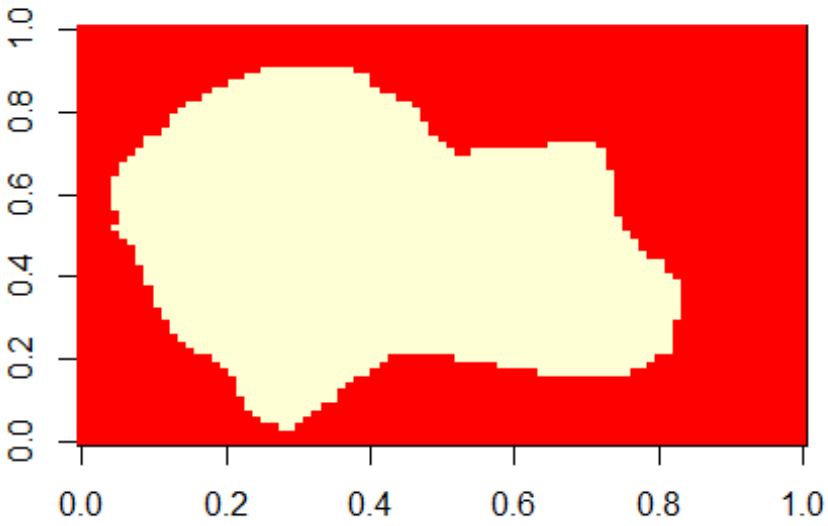
```
image(image_smooth_avg4(volcano))
```



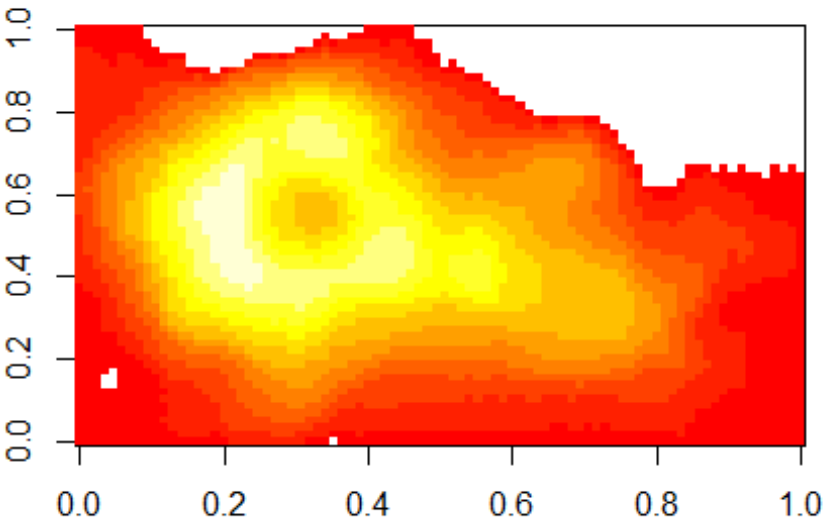

```
image(image_smooth_avg9(volcano))
```



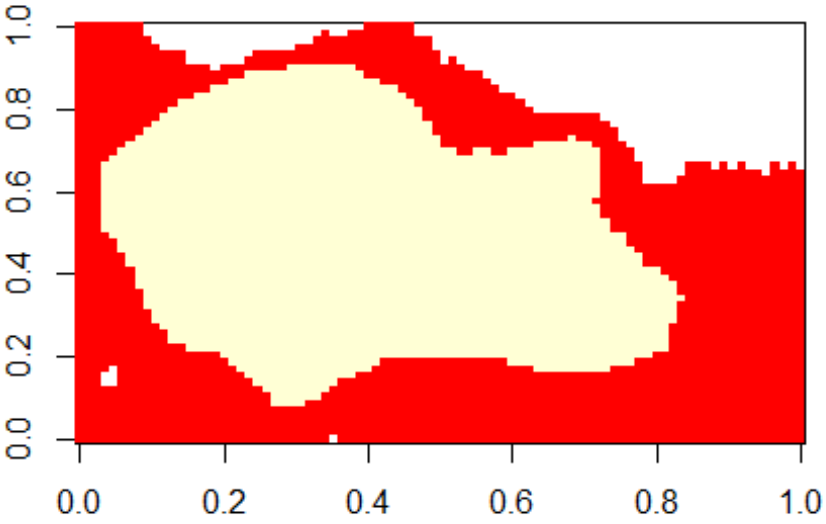

```
layout(1)
#
nvolcano = (volcano - min(volcano))/(max(volcano)-min(volcano))#normali
ze
image(image_sharp_split(nvolcano))
```



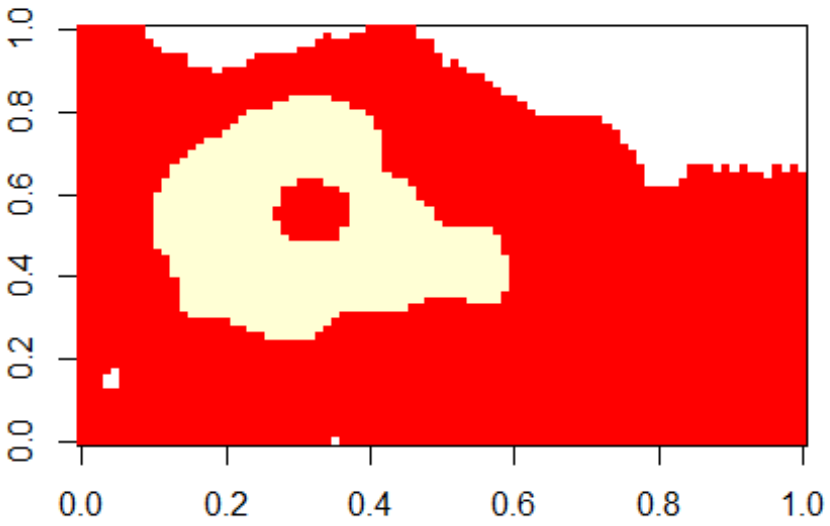

```
#

sd4 = image_sharp_sd4(nvolcano)

## Warning in sqrt(sqr - avg): 产生了 NaNs

#sd4 = (sd4 - min(sd4))/(max(sd4)-min(sd4))#normalize
image(sd4)
```




```
image(image_sharp_split(sd4,.25))
```




```
image(image_sharp_split(sd4,.5))
```



With Bitmaps

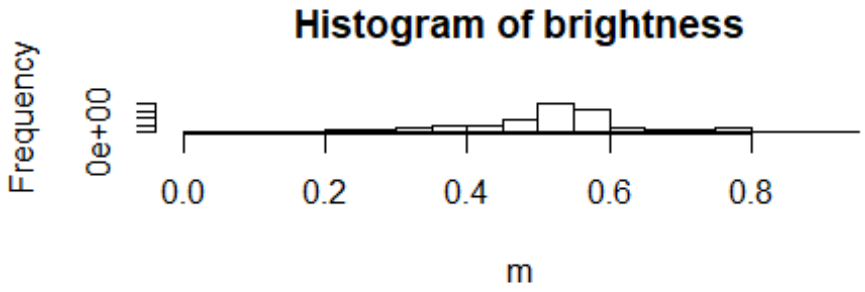
The package suggests the installation of tiff package. If not, install(from CRAN) and load it.

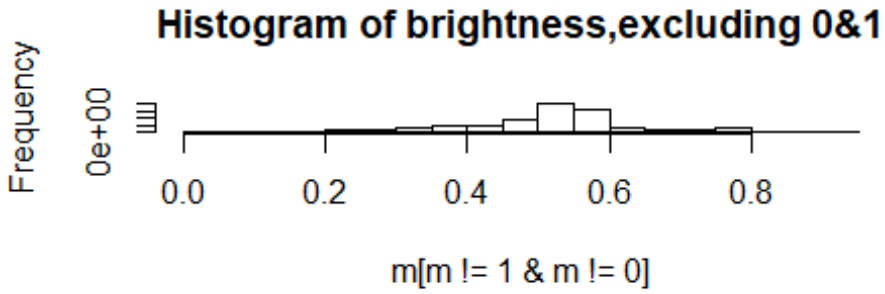
```
require(tiff)#you can go library() but I just think it is better to hav
e a verb

## Loading required package: tiff

## Warning: package 'tiff' was built under R version 3.5.2
```

Load, inspect and psuedo coloring

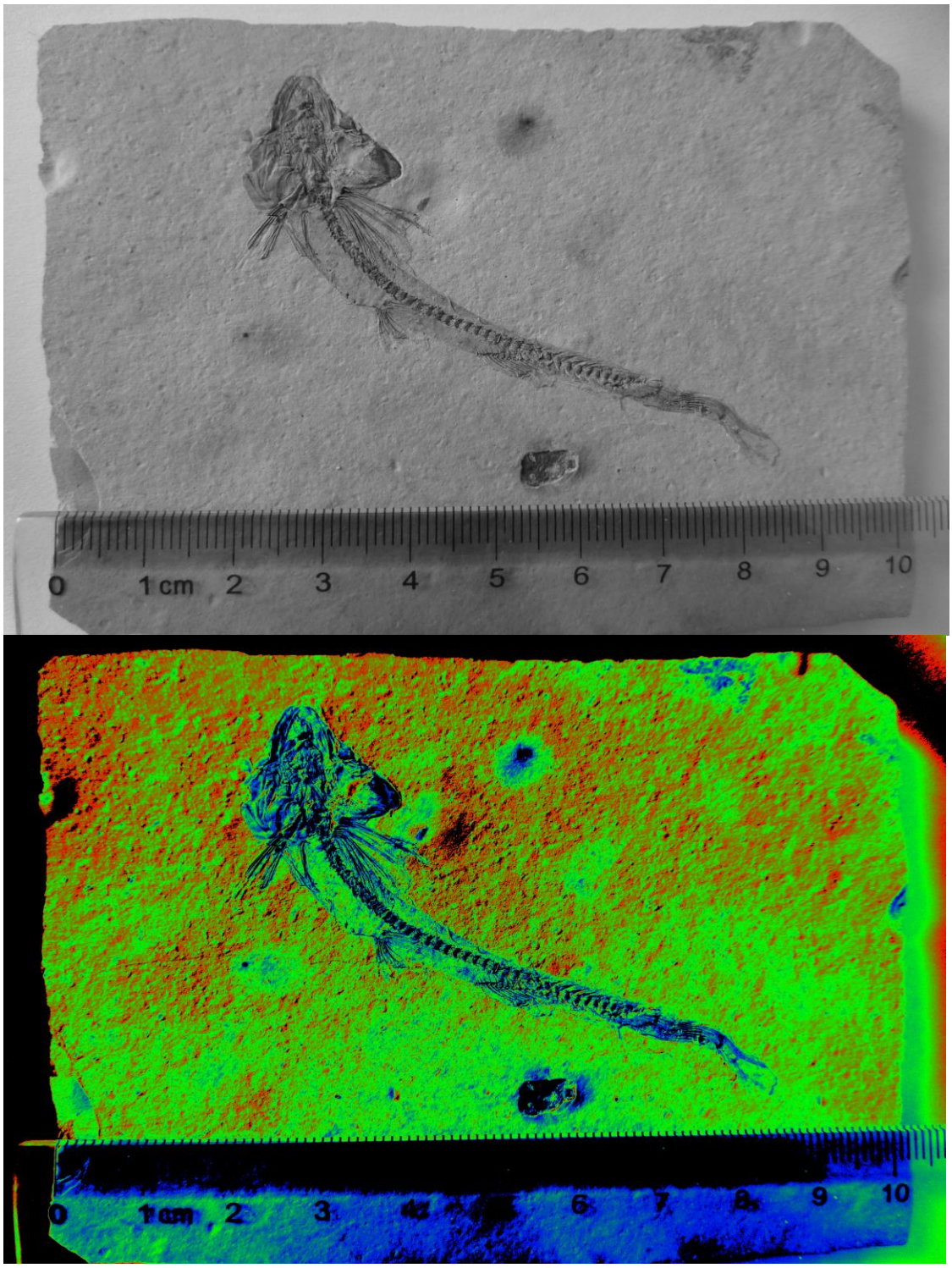
Load your images, in this case a fossil fish. Color it with image_pdc_RGB(a) and image_pdc_RGBpower(a)

```
fish = readTIFF('E:/Bio/Packages/rVividIm_manual/a.tif')
image_summary(fish)
```






```
## $type
## [1] "greyscale"
##
## $summary
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.02353 0.45882 0.52941 0.50618 0.56863 0.90980
##
## $summaryExc
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.02353 0.45882 0.52941 0.50618 0.56863 0.90980

#image_slice(fish)

writeTIFF(image_pdc_RGB(fish),'E:/Bio/Packages/rVividIm_manual/rgb.tif
')
writeTIFF(image_pdc_RGBpower(fish,Rp = .2,
                             Gp = 1,
                             Bp = 5),'E:/Bio/Packages/rVividIm_manual/r
gbpower.tif')
```


0 1 cm 2 3 4 5 6 7 8 9 10

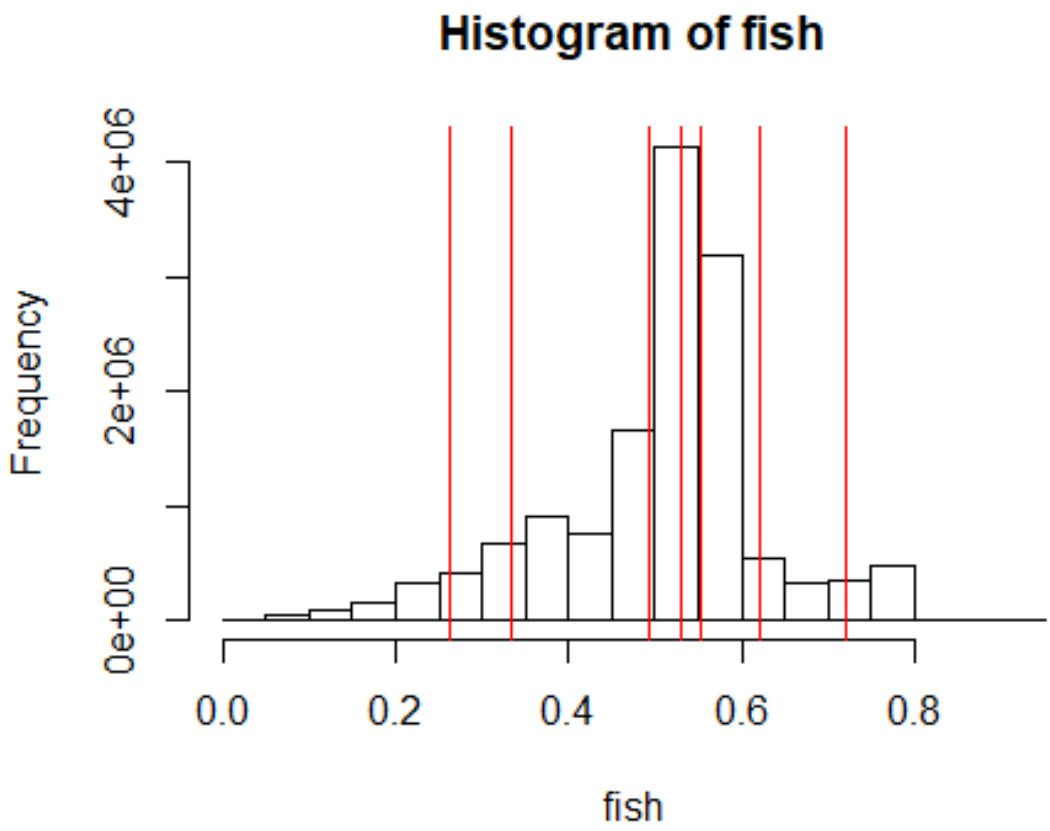
The rgb coloring function gives a high contrast, bright image which amplifies the contrast of pattern. The RGBpower function gives a colored, soft image which still keeps some consistency of the data. To know more, use help().

To turn a colored image to a greyscale one, you can extract by image_extract_avg()

Between pixels

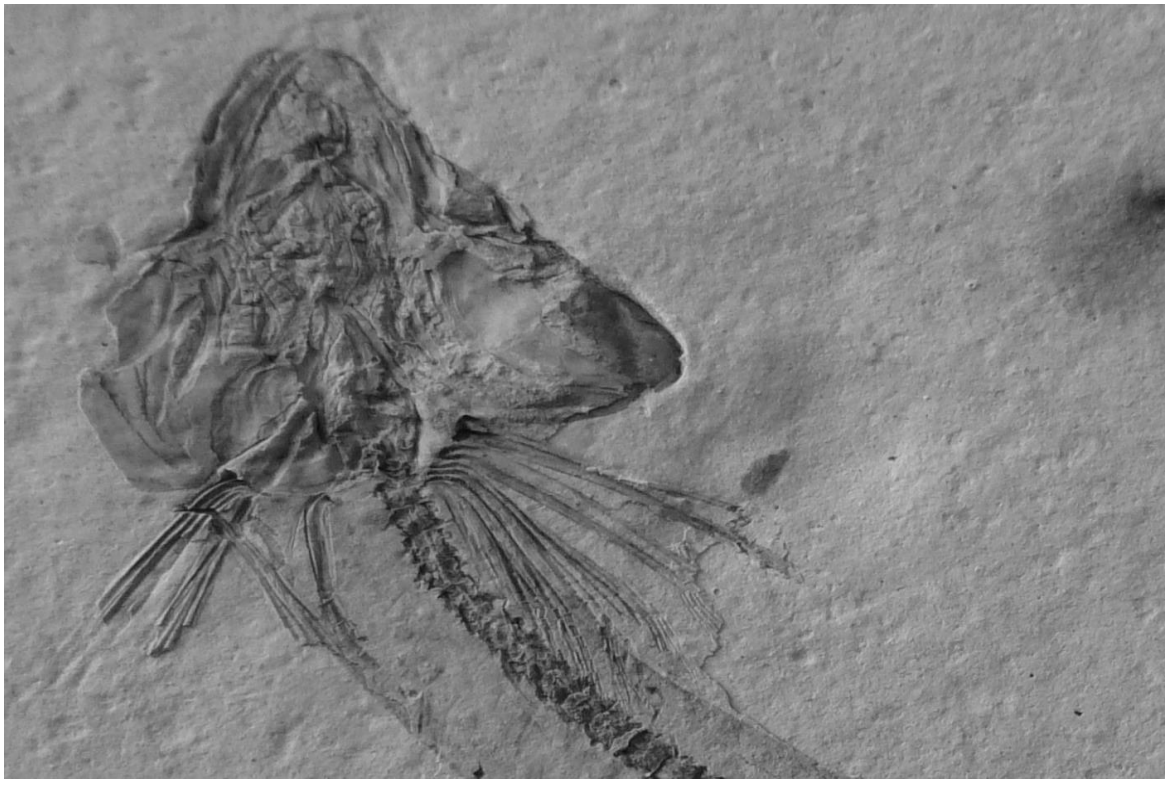
To view the general trend and distribution of certain area, we can smooth the image by taking average. In contrast, to view the gradient of certain area and sharpen the edges, we can take the standard derivation of adjacent pixels. You are encouraged to do some 1 variable statitic before working on the image. Here is an example.

```
#first check if there is any missing values. If true, fill it with some
 value
#(this example uses the mean of all other pixels)
length(is.na(fish))

## [1] 13996800

fish[is.na(fish)]=mean(fish,na.rm = T)
hist(fish)#;points(density(fish),type = 'l')
for (i in c(.05,.1,.33,.5,.67,.9,.95)) abline(v = quantile(fish,i),col
= 'red')
```



The brightness of the entire image may be complicated, which will make it difficult for normalization of the gradient(e.g. the pattern of the ruler and the edge of the rock is not required but will create some problems for scaling the gradient as sharp change in pattern appears). A easier way to work on the pattern is to first look at patterns in a smaller range. We can subset the matrix.

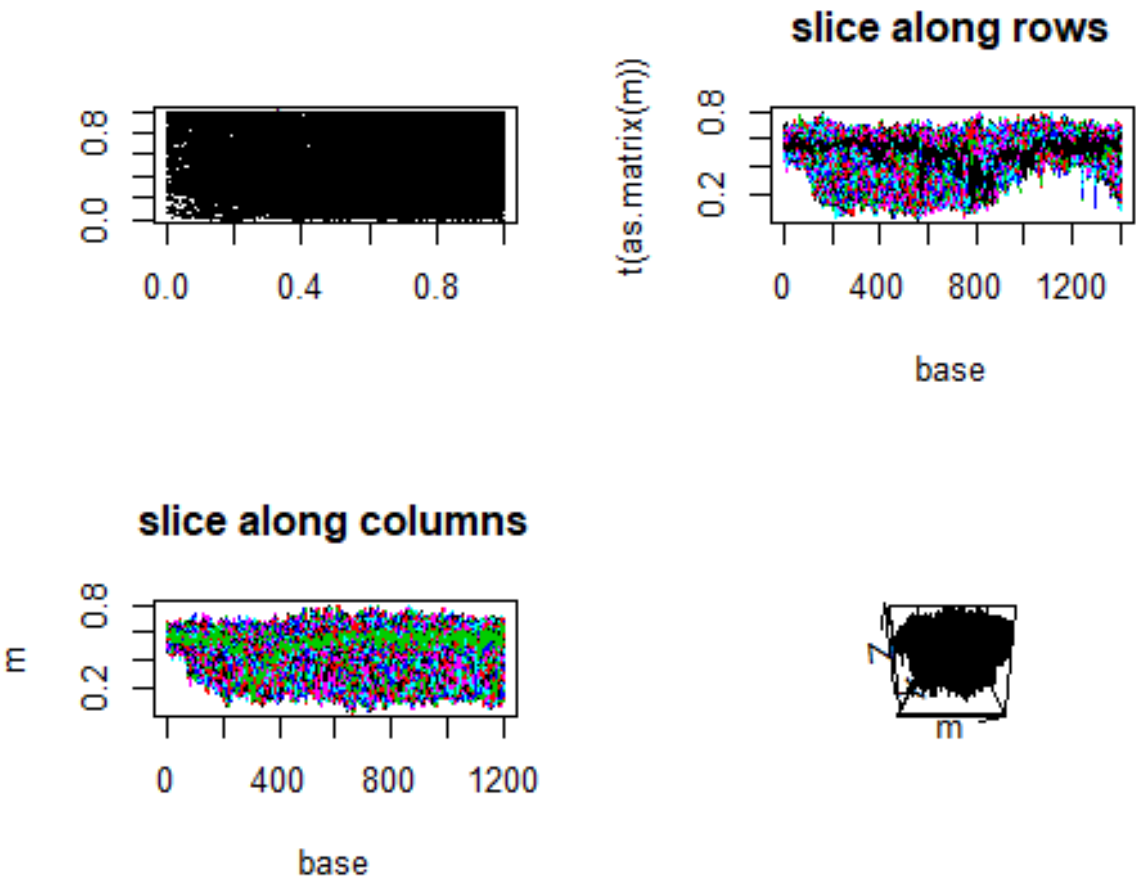
```
fish_h = fish[300:1500,1000:2400]
writeTIFF(fish_h,'fish_h.tif')

## [1] 1
```



fish_h

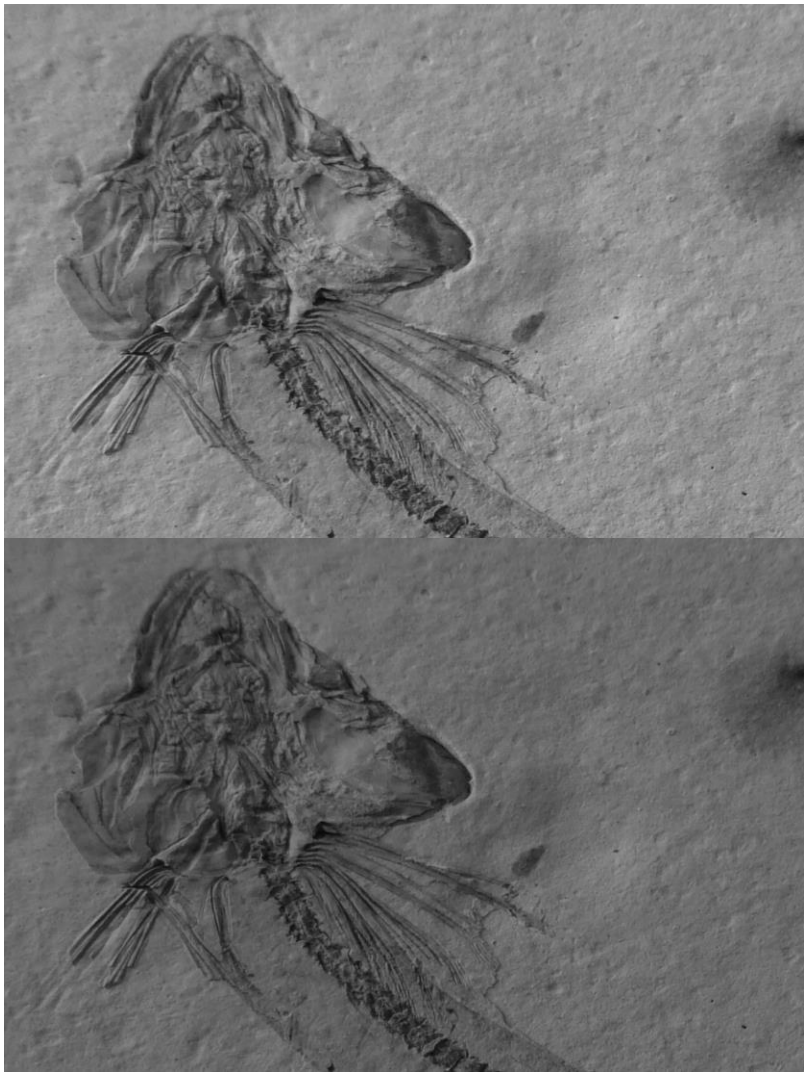
For the head of the fish, do image_slice() to visualize the pattern in the gradient.

```
image_slice(fish_h)
```



This appears to be very messy, let us smooth it a little bit.

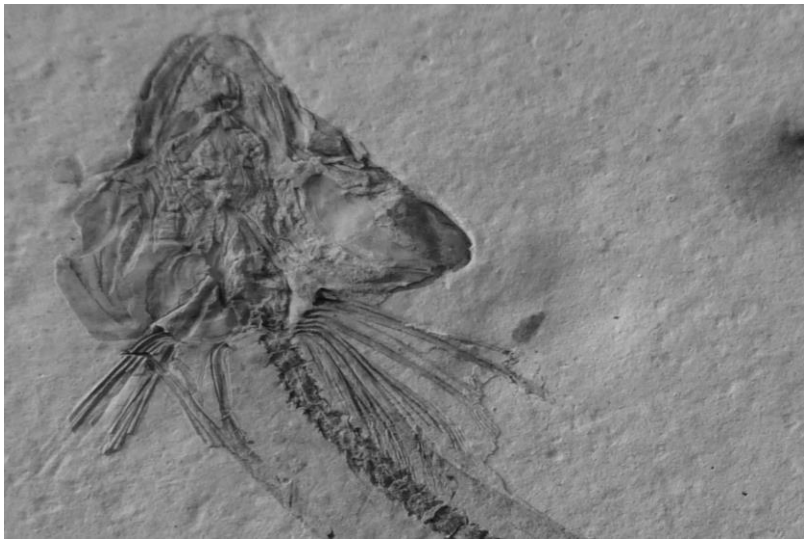
```
smooth4 = image_smooth_avg4(fish_h)

writeTIFF(smooth4,'smooth4.tif')

## [1] 1

smooth9 = image_smooth_avg9(fish_h)
#image_slice(smooth9)
writeTIFF(smooth9,'smooth9.tif')

## [1] 1
```



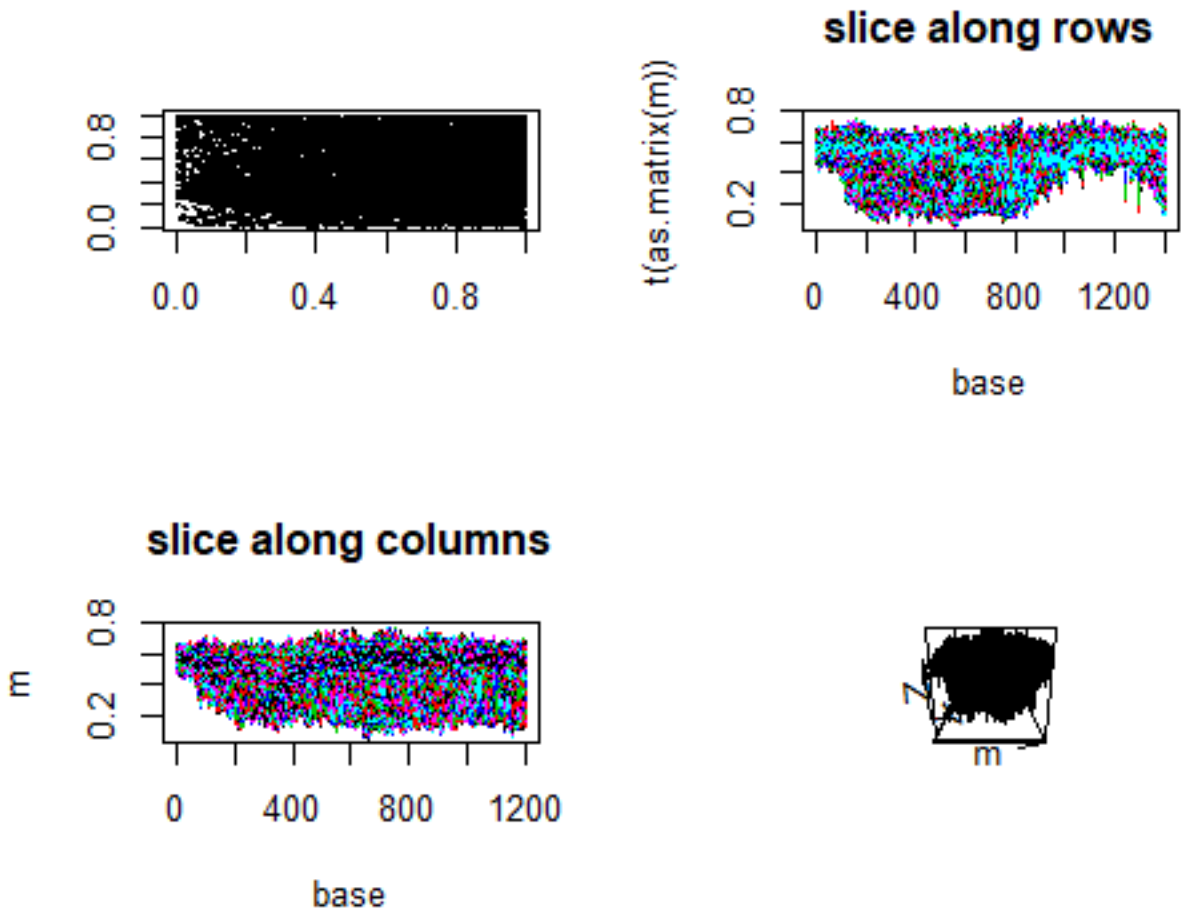
Smoothing makes the brightness closer to the mean and will help ignore lots of outliers. But it also omits lots of features and let the values to be less different.

You can smooth a smoothed plot again, taking more rounds of average.

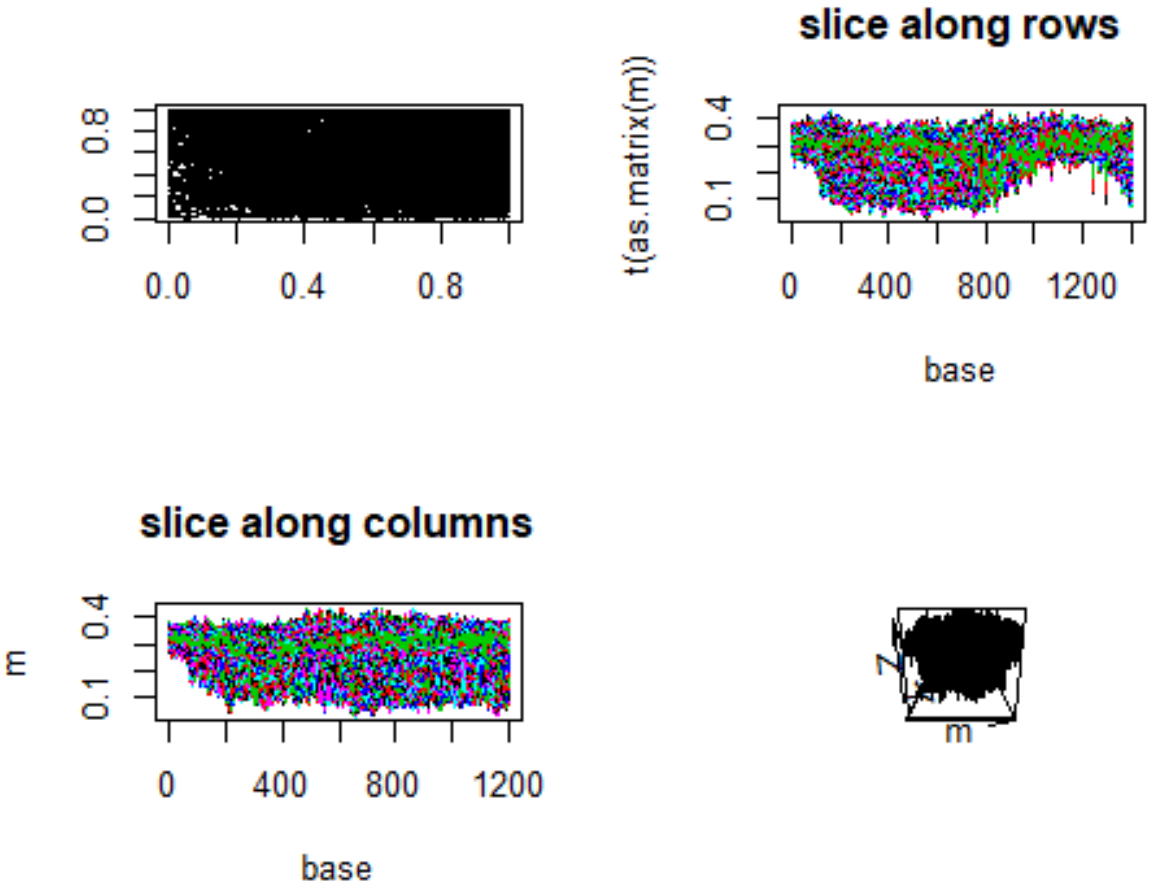
```
layout(1)
smooth44 = image_smooth_avg4(smooth4)#each time the image has 1 less ro
w and 1 less col
writeTIFF(smooth44,'smooth44.tif')

## [1] 1

smooth99 = image_smooth_avg9(smooth9)#2lessrow, 2less col
writeTIFF(smooth9,'smooth9.tif')

## [1] 1
```


smooth99 Directly plotting the gradient is feasible, but the performance may not be ideal.

```
image_slice(smooth44)
```



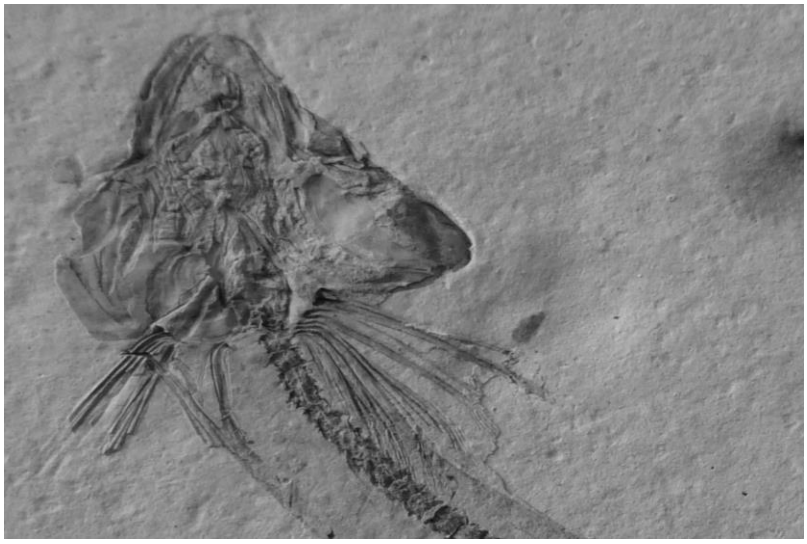
```
image_slice(smooth99)
```



Note that brightness is NOT proportional to height, but could be useful to extract boundaries.

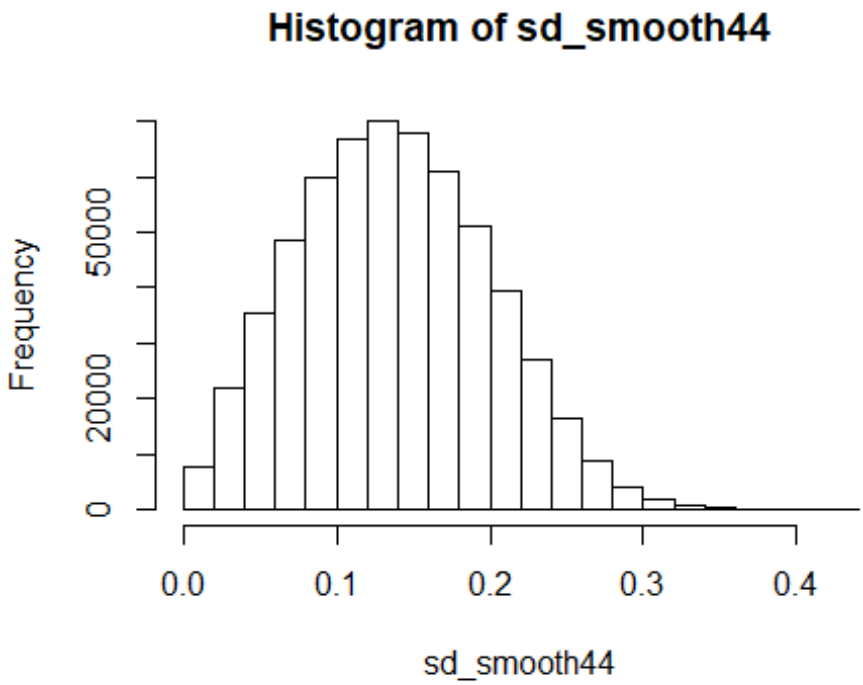
Split the value by the means of them, let the larger value go 1 and smaller go 0.

```
layout(1)
writeTIFF(image_sharp_split(smooth44),'split44.tif')

## [1] 1

writeTIFF(image_sharp_split(smooth99),'split99.tif')

## [1] 1
```


smooth99

Extract sharp edge, normalize, and repeat the same process.

```
sd_smooth44 = image_sharp_sd4(smooth44)

## Warning in sqrt(sqr - avg): 产生了 NaNs

hist(sd_smooth44)
```




```
sd_smooth44 = (sd_smooth44 - min(sd_smooth44,na.rm = T))/(max(sd_smooth
44,na.rm = T) - min(sd_smooth44,na.rm = T))
```

```
writeTIFF(sd_smooth44,'sd44.tif')
```

```
## [1] 1
```

```
writeTIFF(image_sharp_split(sd_smooth44),'splitsd44.tif')
```

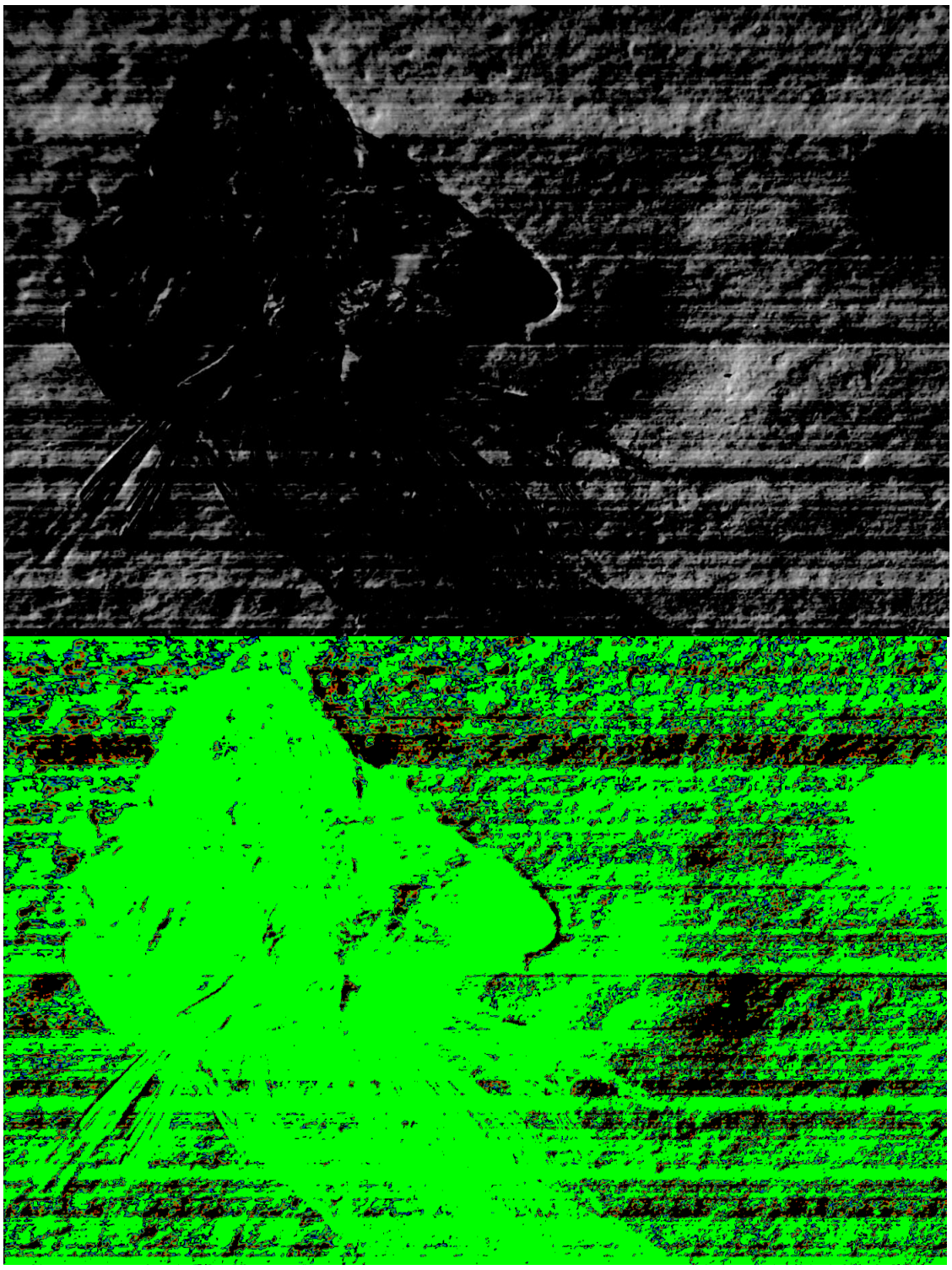
```
## [1] 1
```

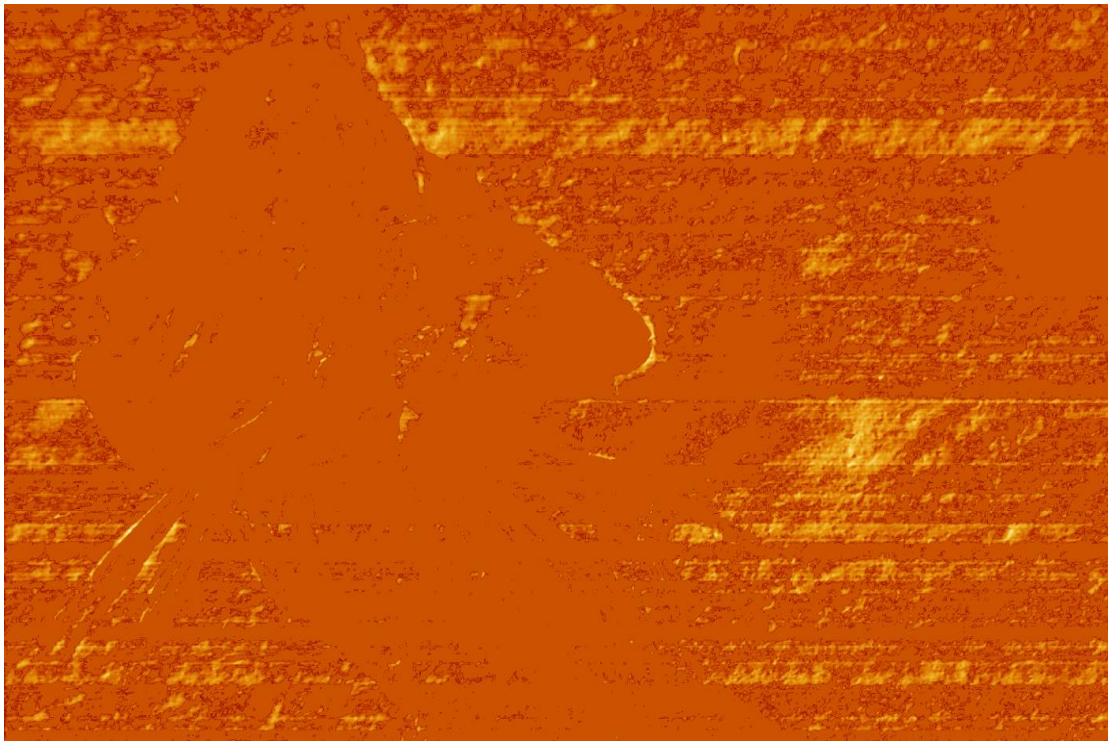
```
sd_smooth44[is.nan(sd_smooth44)] = mean(sd_smooth44,na.rm = 1)
writeTIFF(image_pdc_RGB(sd_smooth44),'rgbsd44.tif')
```

```
## [1] 1
```

```
writeTIFF(image_pdc_RGBpower(sd_smooth44),'rgbpowersd44.tif')
```

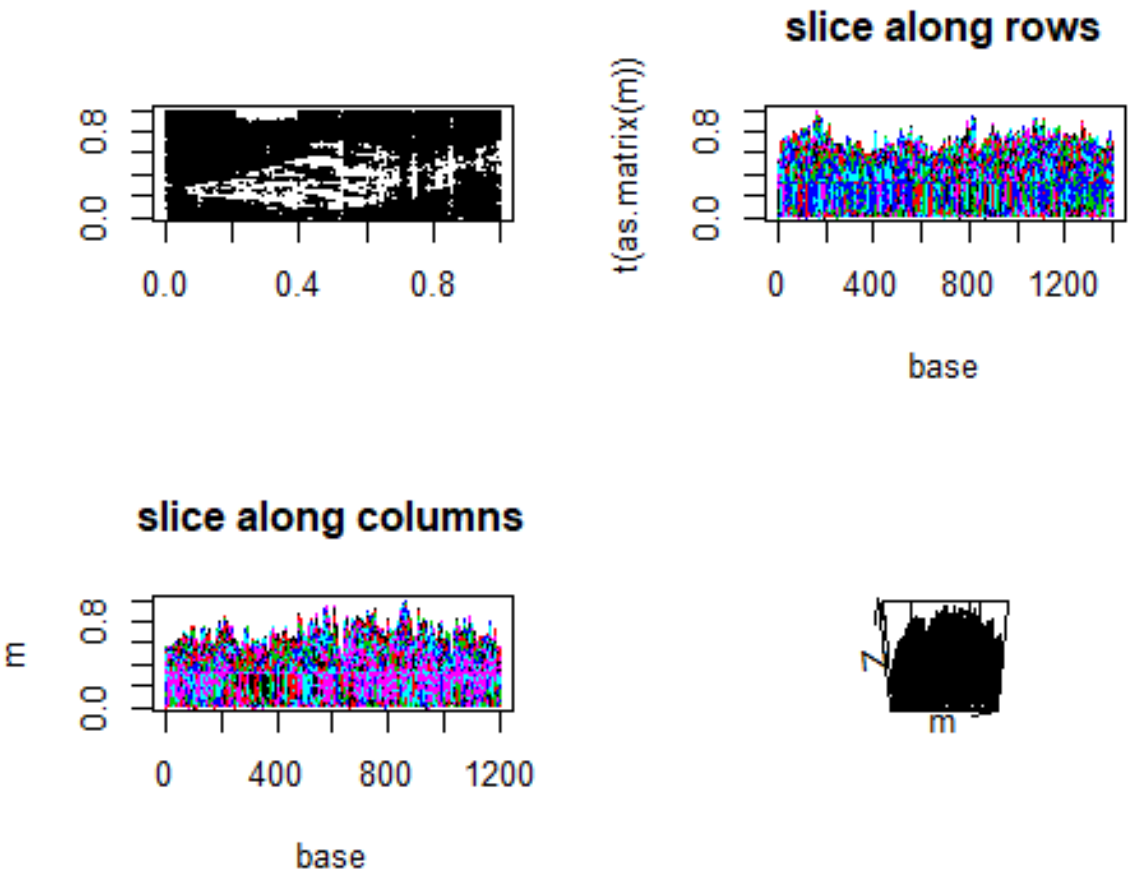
```
## [1] 1
```





More ambitiously, let us do the slice of gradient.

```
image_slice(sd_smooth44)
```



Then we can subset the original one with the gradient, setting the hig-gradient area to the brightest value. But I am not a professional archeaologist so I can get rid of it when things get too complicated. Further processes will be written in other documents and phew… You've read a lot.

More

This package is a part of my hobby projects and is free to all. I am also building a toolbox of packages for mathematical modelling and system modelling. Any suggestions would be helpful and I thank you for using and sharing this work. To report bugs or collaborate please contact be by email on GitHub.

*This document follows GNU FDL