Vessel Segmentation

Wenzhen Zhu
WUSTL

Notation

Color Coding

🟨	Equation
🟩	My thoughts and explanations
🟥	book segment
🟧	answer for the problem
🟪	question
🟪	assumption
🟦	intermediate answer

Intro

ArchHacks 2016 `https://archhacks.io/` 's theme is HealthTech.
I participate this hackathons and after building an app and talking to several medical students, I realized that I can actually apply what I learned from computer vision class and make some really world application to help patients who are suffering and to help doctors to be more efficient. That's why I choose to do this project.

Edge detection is an essential task in computer vision, and vessel segmentation is also an important task in clinical care and medical research.

First, doctor can diagnose many disease from small change of vessels, like diabetes. Second, some disease needs to remove vessels first in order to make further diagnosis.

Style and Interface Code

In[30]:=
```
pages676 = Import[NotebookDirectory[]<>"/1.svg.pdf"];
pages677 = Import[NotebookDirectory[]<>"/2.svg.pdf"];
```

Related Work

Morphology Operators

We define a 2D image whose range is $[N_{\min}, N_{\max}]$ as a functional $S : \mathbb{R}^2 \to [N_{\min}, N_{\max}]$ and a 2-D structuring element as a functional B: $\mathbb{R}^2 \to B$ where B is the set of neighborhood of origin.

We define basic operators, with respect to the structuring element B with scaling factor e, image S and point $M_0 \in \mathbb{R}^2$

Because Mathematica Built-in function are optimized and calls C++, therefore, the built-in Erosion, Dilation, Opening, Closing, etc are extremely fast and reliable. I always compare the built-in function result with my own implementation result. My approach to prototyping the paper is to adapt Mathematica built-in functions, which takes kernel instead our habit - "offset structure element", so I have a function that convert the offset structure element into kernel matrix.

When I implement these functions in C++, I just copy the computed offset in the form of list of lists into "Morphology.h" and defined them as 2D array.

Erosion (My C++ implementation has same result as Mathematica built-in Erosion)

$$\epsilon_B^e\ (S)\,(M_0) = \mathrm{Min}_{M \in M_0 + \epsilon\, B(M_0)}(S(M))$$

Dilation (My C++ implementation has same result as Mathematica built-in Dilation)

$$\delta_B^e\ (S)\,(M_0) = \mathrm{Max}_{M \in M_0 + \epsilon\, B(M_0)}(S(M))$$

Opening (My C++ implementation has same result as Mathematica built-in Opening)

$$\gamma_B^e\ (S)\,(M_0) = \delta_B^e(\epsilon_B^e(S))$$

Closing (My C++ implementation has same result as Mathematica built-in Closing)

$$\phi_B^e\ (S)\,(M_0) = \epsilon_B^e(\delta_B^e(S))$$

Top-hat (My C++ implementation has same result as Mathematica built-in Top-hat)

$$\mathrm{TH}_B^e\ (S)\,(M_0) = S - \gamma_B^e(S)$$

Geodesic dilation (My C++ implementation has same result as Mathematica built-in GeodesicDilation)

$$S^1 = \inf(\{\mathrm{Max}_{M \in M_0 + c}(S_m(M))\},\ S(M_0))$$
$$S^{d+1} = \Delta_{S_m}^{d+1}\ (S)\,(M_0) = \Delta_{S_m}^1\left(S^d\right)(M_0)$$

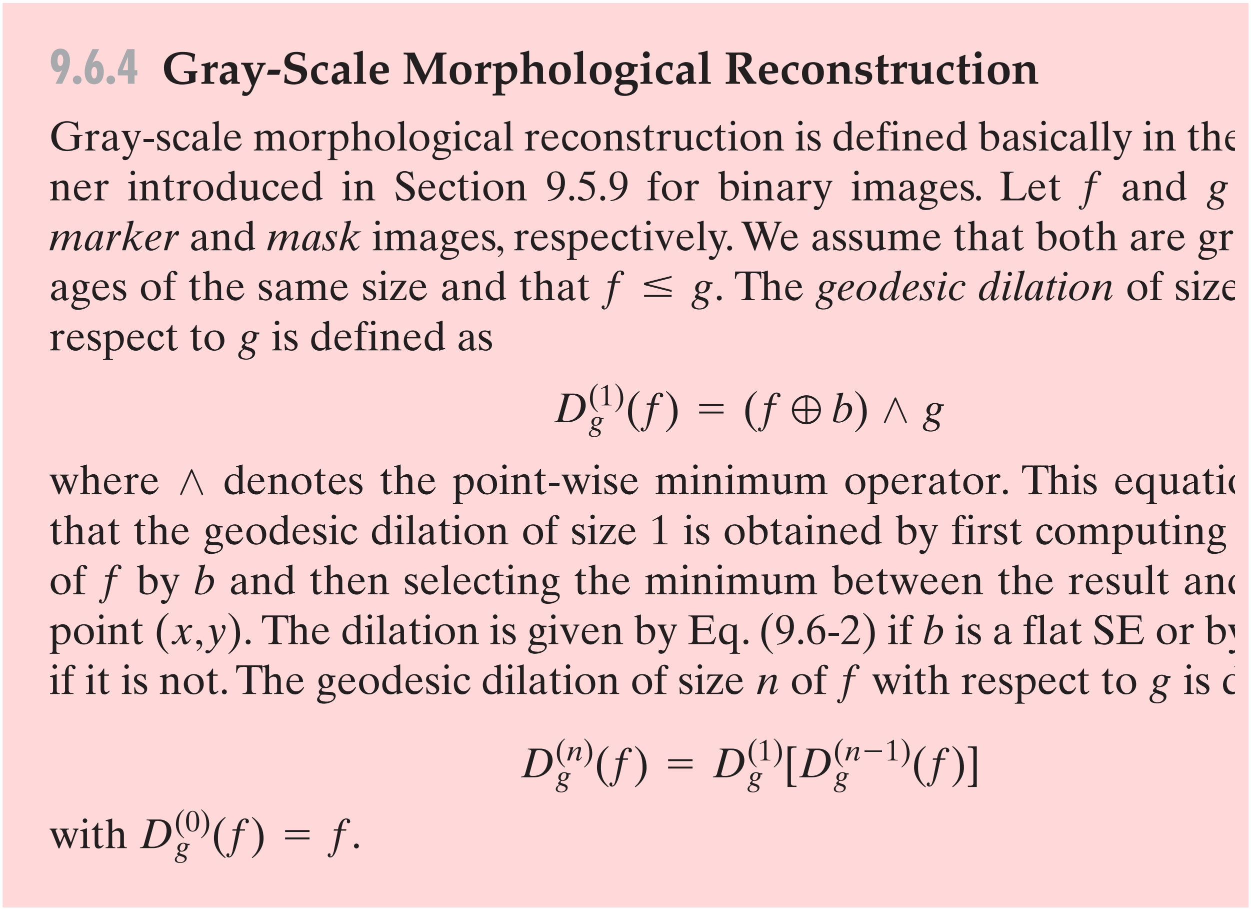
Geodesic opening (not sure, because Mathematica built-in GeodesicOpening doesn't have maskImage argument)

$$\gamma_{S_m}^{\text{rec}}(S) = \sup_{d\in\mathbb{N}}\left(\Delta_{S_m}^{d}(S)\right)$$

Geodesic closing (not sure, because Mathematica built-in GeodesicClosing doesn’t have maskImage argument)

$$\phi_{S_m}^{\text{rec}}(S) = N_{\max} - \gamma_{(N_{\max}-S_m)}^{\text{rec}}(N_{\max} - S)$$

It’s quite hard for me (and TA) to understand several definitions as Geodesic Opening and Geodesic Closing. Here is page 676 - 677, Chapter 9: Morphological Image Processing



9.6.4 Gray-Scale Morphological Reconstruction

Gray-scale morphological reconstruction is defined basically in the
ner introduced in Section 9.5.9 for binary images. Let f and g
marker and *mask* images, respectively. We assume that both are gr
ages of the same size and that $f \leq g$. The *geodesic dilation* of size
respect to g is defined as

$$D_g^{(1)}(f) = (f \oplus b) \wedge g$$

where $\wedge$ denotes the point-wise minimum operator. This equati
that the geodesic dilation of size 1 is obtained by first computing
of f by b and then selecting the minimum between the result an
point (x,y). The dilation is given by Eq. (9.6-2) if b is a flat SE or b
if it is not. The geodesic dilation of size n of f with respect to g is

$$D_g^{(n)}(f) = D_g^{(1)}[D_g^{(n-1)}(f)]$$

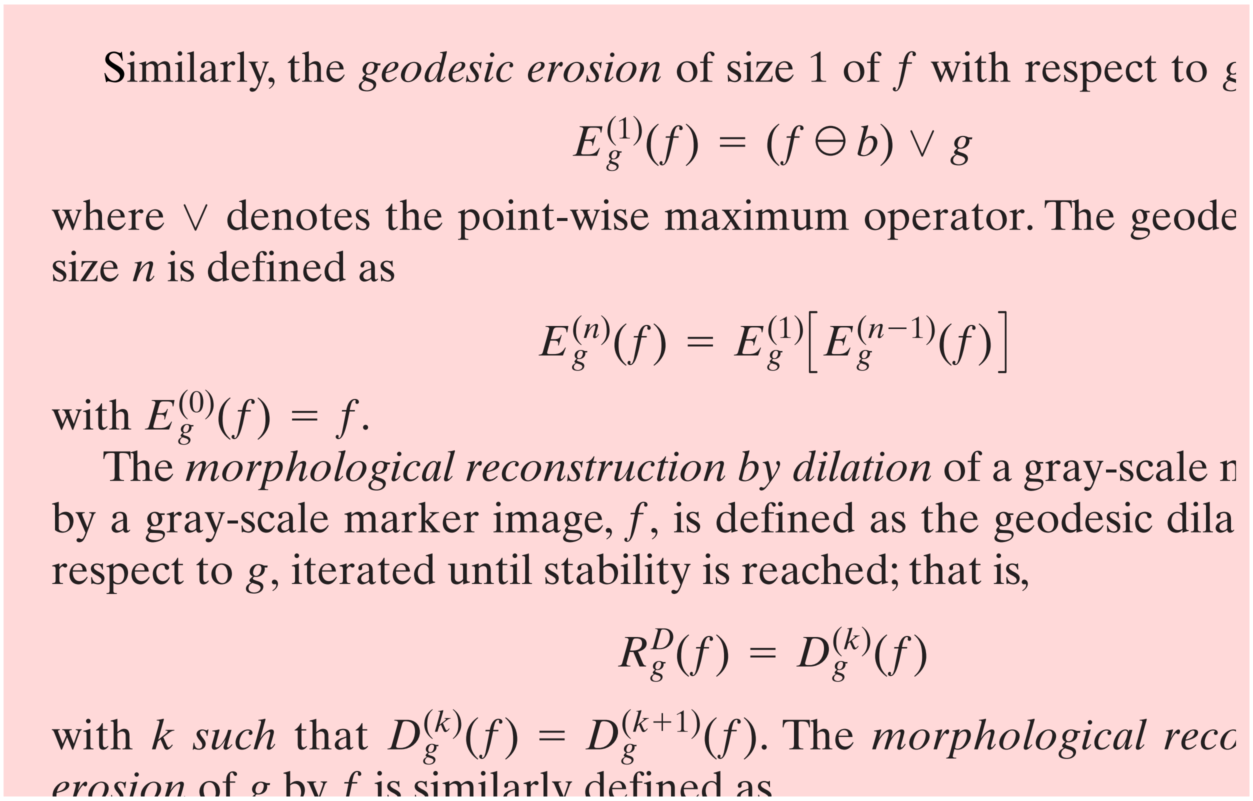
with $D_g^{(0)}(f) = f$.

Compared to the paper, $S^1 = \inf(\{\text{Max}_{M\in M_0+c}(S_m(M))\},\ S(M_0))$
$f = S_m =$ marker image,
g = mask image,
in the textbook, b means structure element.
So for the geodesic dilation of size 1, I am confident with the operation, which is first, do dilation on marker image with a neighborhood C of unit radius, then selecting the minimum between dilation result and mask image.

I am not sure about the geodesic dilation of size n, the book defined as $D_g^{(n)}(f) = D_g^{(1)}\left[D_g^{(n-1)}(f)\right]$

so when I implement this method in C++, I used the book's definition because I don't quite understand the paper's notation.
I compared the result of this implementation on some random pixel, which gives me same result as Mathematica built-in GeodesicDilation.

```
CFloatImage GeodesicDilation(CFloatImage marker, CFloatImage mask, int size) {
    int marker_w = marker.Shape().width;
    int marker_h = marker.Shape().height;
    CFloatImage resultImage(marker.Shape());

    // base case
    if (size == 0) {
        return marker;
    }
    if (size == 1) {
        CFloatImage DilationImage = DilationStructSquare(marker);
        for (int x = 0; x < marker_w; x++) {
            for (int y = 0; y < marker_h; y++) {
                //In the paper, geodesic dilation used normal dilation in the base case;
                // S1(M0) = inf({Dilation, S(M0)})
                resultImage.Pixel(x, y, 0) = min(DilationImage.Pixel(x, y, 0),
mask.Pixel(x, y, 0));
            }
        }
        return resultImage;
    }
    return GeodesicDilation(GeodesicDilation(marker, mask, size - 1), mask, 1);
}
```



Similarly, the *geodesic erosion* of size 1 of f with respect to

$$E_g^{(1)}(f) = (f \ominus b) \vee g$$

where $\vee$ denotes the point-wise maximum operator. The geode
size n is defined as

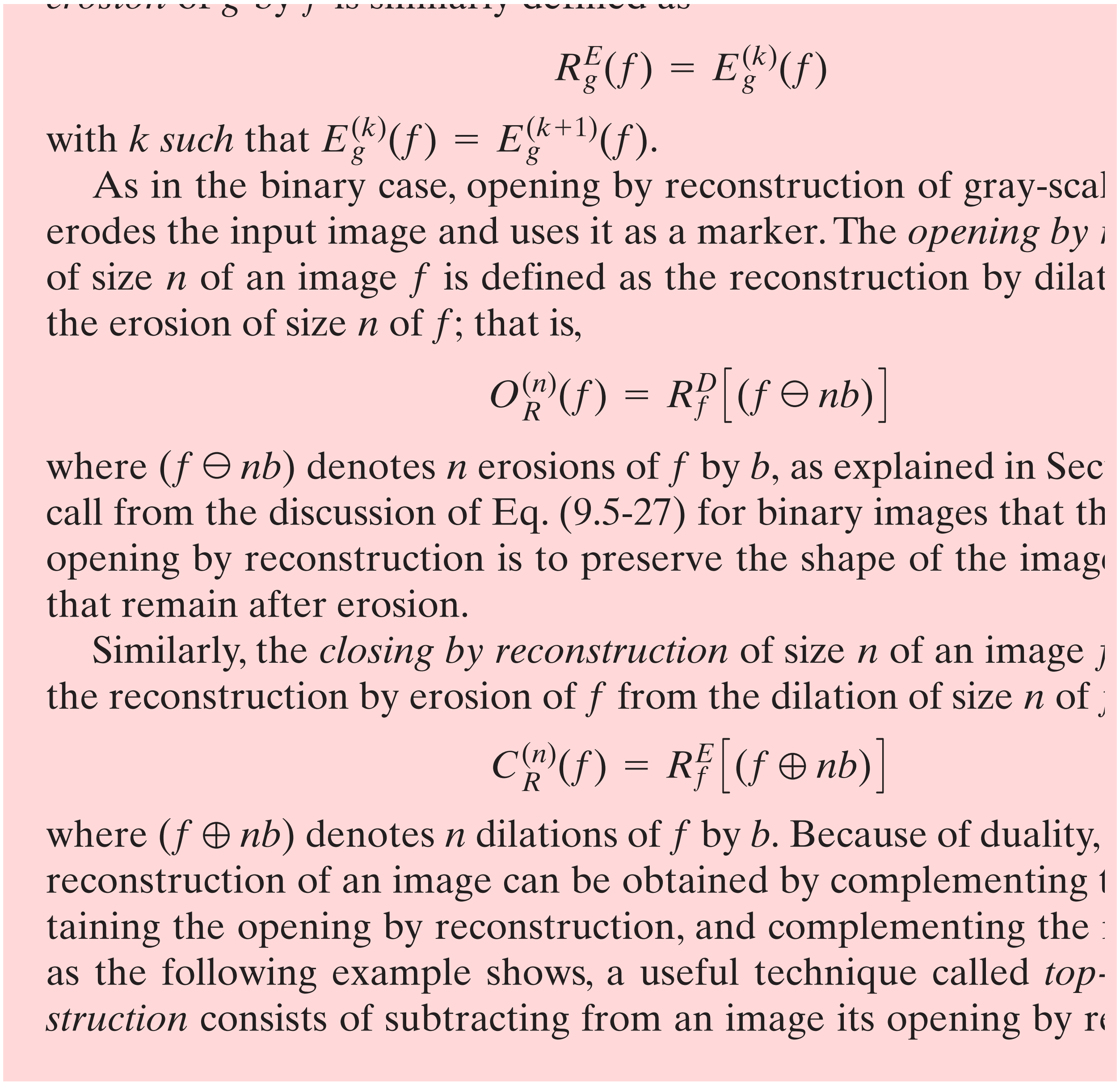
$$E_g^{(n)}(f) = E_g^{(1)}\left[E_g^{(n-1)}(f)\right]$$

with $E_g^{(0)}(f) = f$.

The *morphological reconstruction by dilation* of a gray-scale
by a gray-scale marker image, f, is defined as the geodesic dila
respect to g, iterated until stability is reached; that is,

$$R_g^D(f) = D_g^{(k)}(f)$$

with k *such* that $D_g^{(k)}(f) = D_g^{(k+1)}(f)$. The *morphological reco*
erosion of g by f is similarly defined as



erosion of g by f is similarly defined as

$$R_g^E(f) = E_g^{(k)}(f)$$

with k *such* that $E_g^{(k)}(f) = E_g^{(k+1)}(f)$.

As in the binary case, opening by reconstruction of gray-scal erodes the input image and uses it as a marker. The *opening by* of size n of an image f is defined as the reconstruction by dilat the erosion of size n of f; that is,

$$O_R^{(n)}(f) = R_f^D\left[(f \ominus nb)\right]$$

where $(f \ominus nb)$ denotes n erosions of f by b, as explained in Sec call from the discussion of Eq. (9.5-27) for binary images that th opening by reconstruction is to preserve the shape of the imag that remain after erosion.

Similarly, the *closing by reconstruction* of size n of an image the reconstruction by erosion of f from the dilation of size n of

$$C_R^{(n)}(f) = R_f^E\left[(f \oplus nb)\right]$$

where $(f \oplus nb)$ denotes n dilations of f by b. Because of duality, reconstruction of an image can be obtained by complementing taining the opening by reconstruction, and complementing the as the following example shows, a useful technique called *top-struction* consists of subtracting from an image its opening by r

Algorithms

```
dir = NotebookDirectory[];
img = ColorConvert[
   Import[dir <> "DRIVE/training/images/21_training.tif"], "Grayscale"];
imgData = ImageData[img, "Byte"];
```

Step 0: Construct L_i

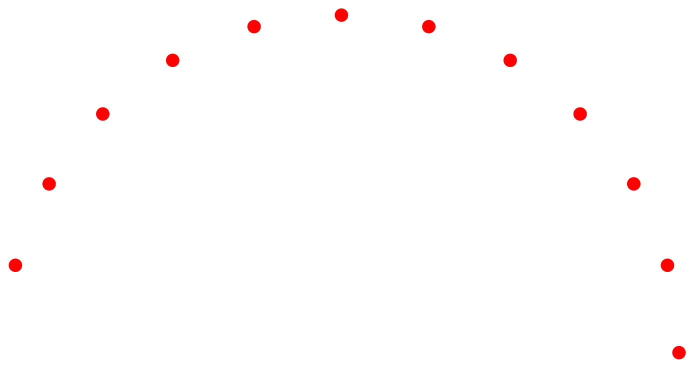
This paper is interesting because it used a linear structure element L_i which is 15-pixels long 1-pixel wide.

```
angles = Table[{Cos[t], Sin[t]}, {t, 0, 11 π / 12, π / 12}]
```

$$\left\{\{1,0\},\left\{\frac{1+\sqrt{3}}{2\sqrt{2}},\frac{-1+\sqrt{3}}{2\sqrt{2}}\right\},\left\{\frac{\sqrt{3}}{2},\frac{1}{2}\right\},\left\{\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}}\right\},\left\{\frac{1}{2},\frac{\sqrt{3}}{2}\right\},\left\{\frac{-1+\sqrt{3}}{2\sqrt{2}},\frac{1+\sqrt{3}}{2\sqrt{2}}\right\},\{0,1\},\left\{-\frac{-1+\sqrt{3}}{2\sqrt{2}},\frac{1+\sqrt{3}}{2\sqrt{2}}\right\},\left\{-\frac{1}{2},\frac{\sqrt{3}}{2}\right\},\left\{-\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}}\right\},\left\{-\frac{\sqrt{3}}{2},\frac{1}{2}\right\},\left\{-\frac{1+\sqrt{3}}{2\sqrt{2}},\frac{-1+\sqrt{3}}{2\sqrt{2}}\right\}\right\}$$

For every $\frac{\pi}{12}$, there is an linear structure element.

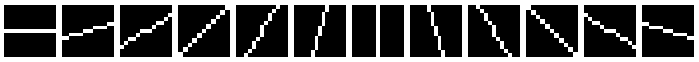
```
Graphics[{Red, PointSize → Large, Point[angles]}]
```



```
linearStrut =
  Table[Round[{Cos[t], Sin[t]} * #] & /@ (Range[15] - 8), {t, 0, 11 π / 12, π / 12}];
```

Here is a visualization of the structure element.

```
GraphicsRow[showBinaryImage[kernelFromStructure[#]] & /@ linearStrut]
```



```
kernels = kernelFromStructure[#] & /@ linearStrut
```

Step I

Calculating S_{op}

The first step is calculating S_{op}, which can be summarized in the followings:

1. Opening with 12 structure element on S_0(original images)
2. Pick the maximum from these 12 images pixel-wise and form a maximum image
3. Do geodesic opening on the maximum image, the result will be our S_{op}

$$S_{op} = \gamma_{S_0}^{rec}\ (\text{Max}_{i=1,..,12}\{\gamma_{L_i}(S_0)\})$$

```
openingImageList = Opening[img, #] & /@ kernels
```

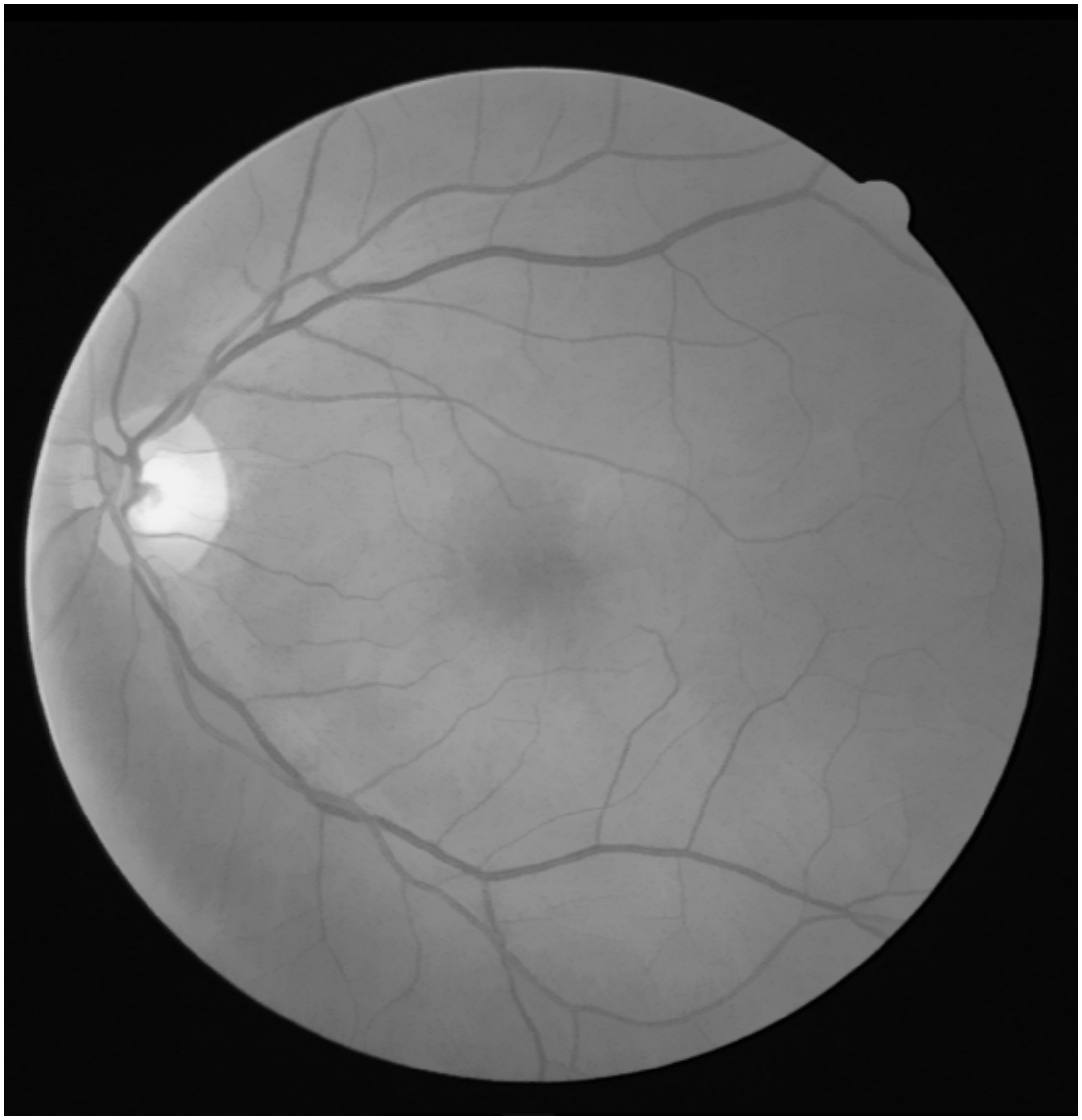
```
AbsoluteTiming[openingDataList = ImageData[#] & /@ openingImageList]
```

```
{0.0313, {{ ⋯ 1 ⋯ }, { ⋯ 1 ⋯ }, { ⋯ 1 ⋯ }, { ⋯ 1 ⋯ }, { ⋯ 1 ⋯ }, { ⋯ 1 ⋯ },
   { ⋯ 1 ⋯ }, { ⋯ 1 ⋯ }, { ⋯ 1 ⋯ }, { ⋯ 1 ⋯ }, { ⋯ 1 ⋯ }, { ⋯ 1 ⋯ }}}

large output | show less | show more | show all | set size limit...
```

```
maxOpeningData = MapThread[Max, openingDataList, 2]
```

```
{ ⋯ 1 ⋯ }

large output | show less | show more | show all | set size limit...
```

```
maxOpeningImage = Image[maxOpeningData]
```



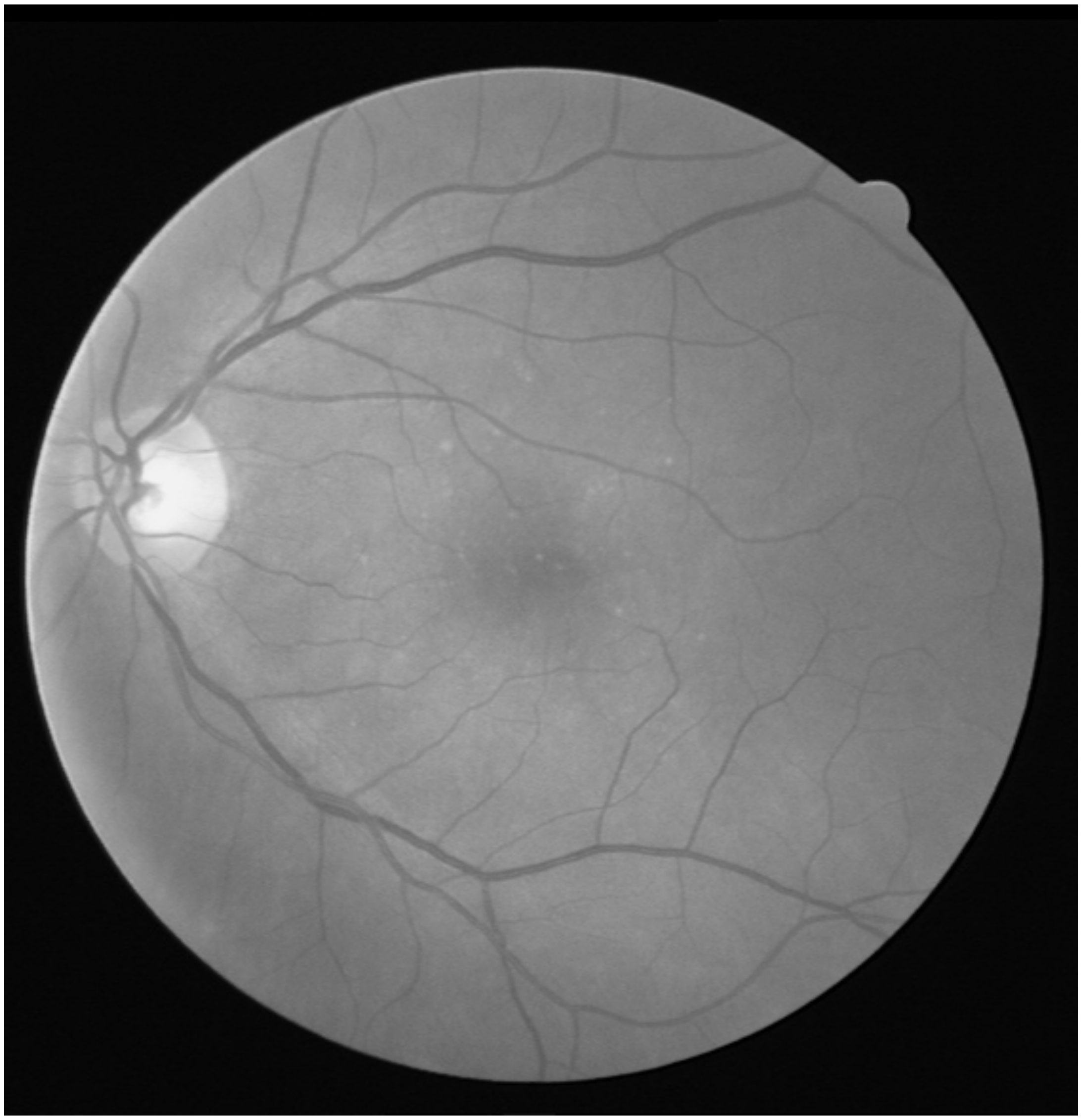
```
markerImage = img;
mask = maxOpeningImage;
```

```
SopData = geodesicOpening[markerImage, mask]
```

```
{ … 1 … }
large output    show less    show more    show all    set size limit...
```

Here is a visualized result S_{op} image.

```
SopImage = Image[SopData]
```



Step 2

Calculating S_{sum}

This transformation (a sum of top hats) reduces small bright noise and improves the contrast of all linear parts. Vessels could be manually segmented with a simple threshold on S_{sum}

$$S_{\text{sum}} := \sum_{i=1}^{12} (S_{\text{op}} - \gamma_{L_i}(S_0))$$

The second step is calculating S_{sum}, which can be summarized in the followings:
1. Opening with 12 structure element on S_0(original images).
2. Use S_{op} to minus the opening images pixel-wise.
3. Sum the 12 difference image up to get S_{sum}image

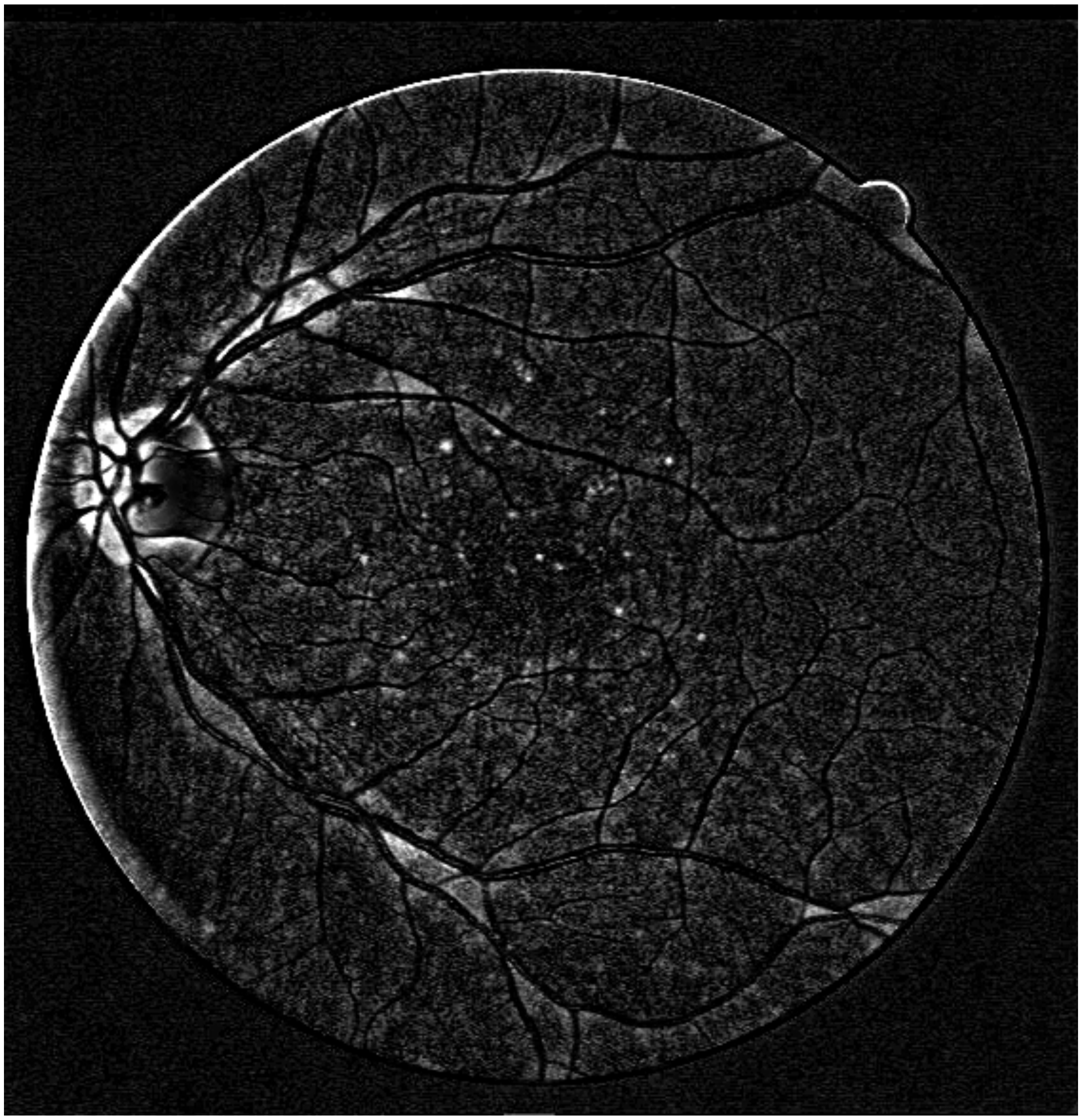
```
AbsoluteTiming[topHatData = (SopData - #) & /@ openingDataList]
```

```
{0.170847, {{ … 1 … }, { … 1 … }, { … 1 … }, { … 1 … }, { … 1 … }, { … 1 … },
  { … 1 … }, { … 1 … }, { … 1 … }, { … 1 … }, { … 1 … }, { … 1 … }}}

large output | show less | show more | show all | set size limit...
```

```
AbsoluteTiming[SsumData = Total[topHatData]]
```

```
{0.013569, { … 1 … }}

large output | show less | show more | show all | set size limit...
```

```
SsumImage = Image[SsumData]
```



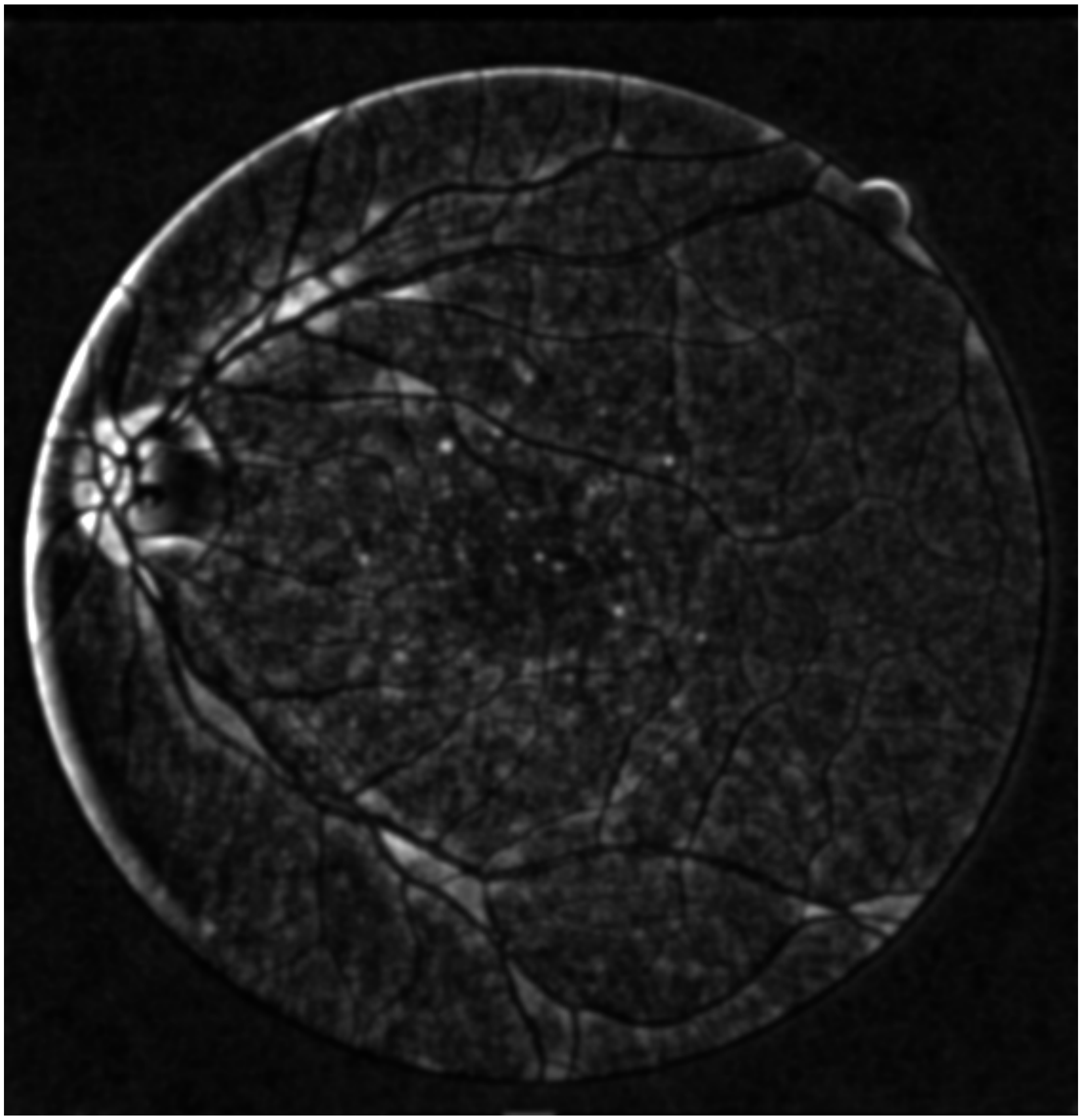
Step 3

This step is very straight forward:
1. Apply Gaussian filter with $r = 7$ and $\sigma = 7/4$
2. Apply Laplacian filter with $r = 1$.

$$S_{\text{lap}} = \text{Laplacian}\left(\text{Gaussian}^{\text{width} = 7\,\text{px}}_{\sigma = 7/4}(S_{\text{sum}})\right)$$

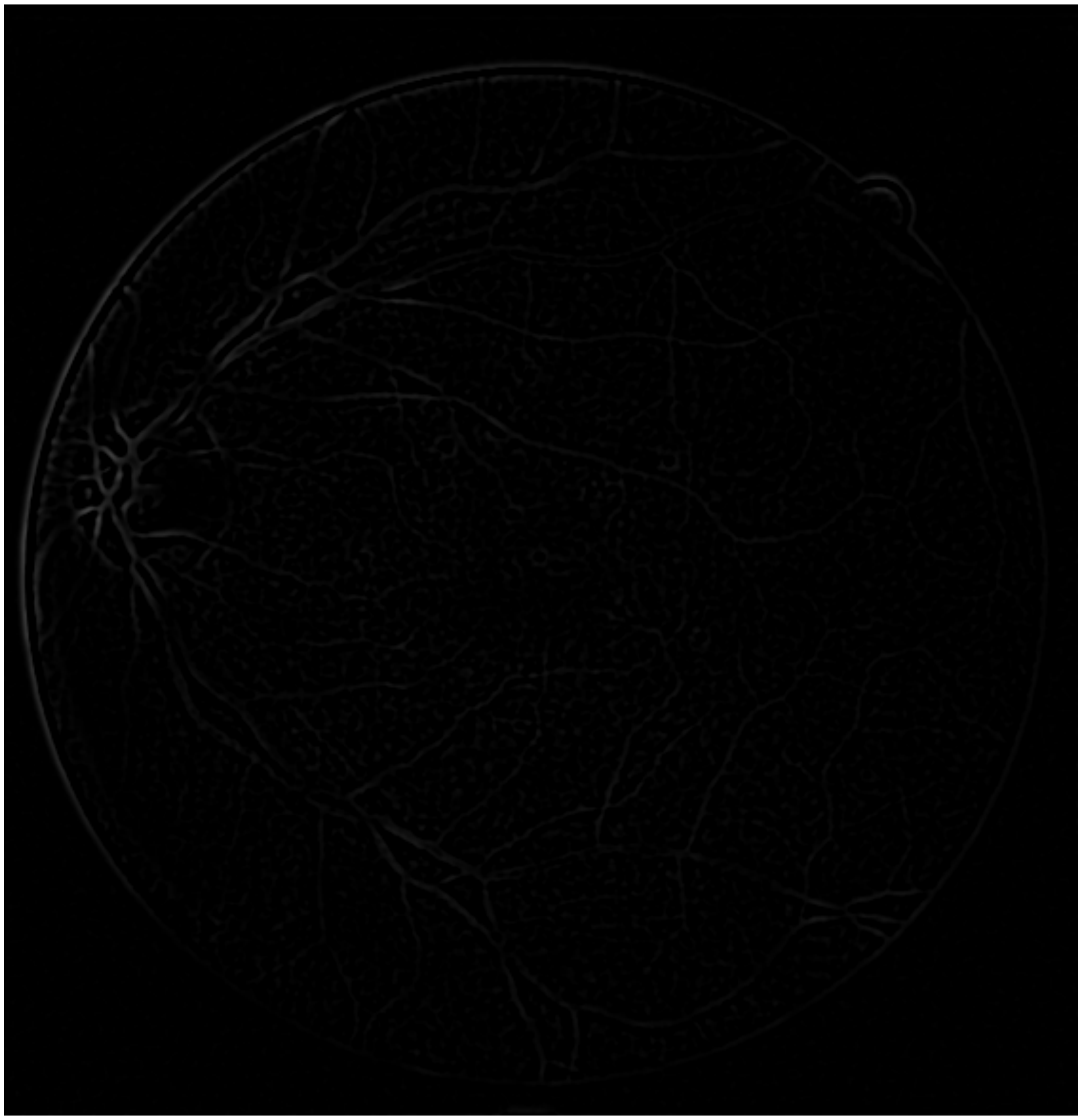
Applying Gaussian filter

```
gaussianImage = GaussianFilter[SsumImage, {7, 7 / 4}]
```



Applying Laplacian filter

```
SlapImage = LaplacianFilter[gaussianImage, 1]
```



Step 4

Calculating S_1

This step can be summarized in the followings:
1. Opening with 12 structure element on S_{lap}(Laplacian applied image).
2. Pick the maximum from these 12 images pixel-wise and form a maximum image
3. Do geodesic opening on the maximum image, the result will be our S_1

$$S_1 = \gamma^{\text{rec}}_{S_{\text{lap}}}\left(\text{Max}_{i=1,\ldots 12}\{\gamma_{L_i}(S_{\text{lap}})\}\right)$$

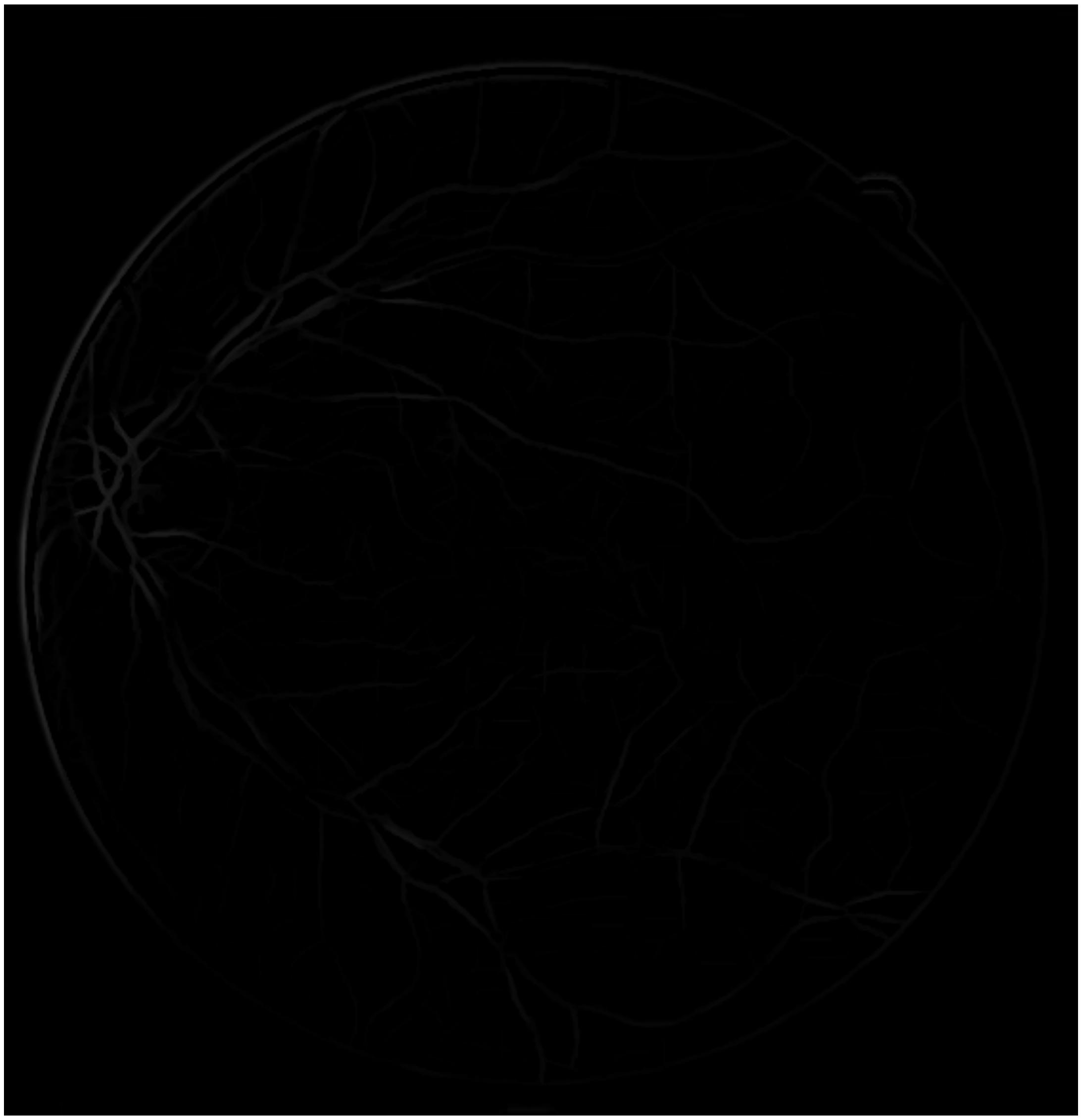
```
AbsoluteTiming[openingOnSlapData = ImageData /@ (Opening[SlapImage, #] & /@ kernels)]
```

```
{0.213361, {{ … 1 … }, … 10 … ,
  {{0.000290349, 0.000318456, 0.000355327, 0.000355327, 0.000355327, 0.000267228,
    0.00015834,  … 552 … , 0.0000839143, 0.0000715021, 0.0000585977,
    0.0000535731, 0.0000487266, 0.0000456603},  … 582 … , { … 1 … }}}}
large output | show less | show more | show all | set size limit...
```

Pick the maximum

```
maxOpeningOnSlapData = MapThread[Max, openingOnSlapData, 2];
```

```
maxOpeningOnSlapImage = Image[maxOpeningOnSlapData]
```



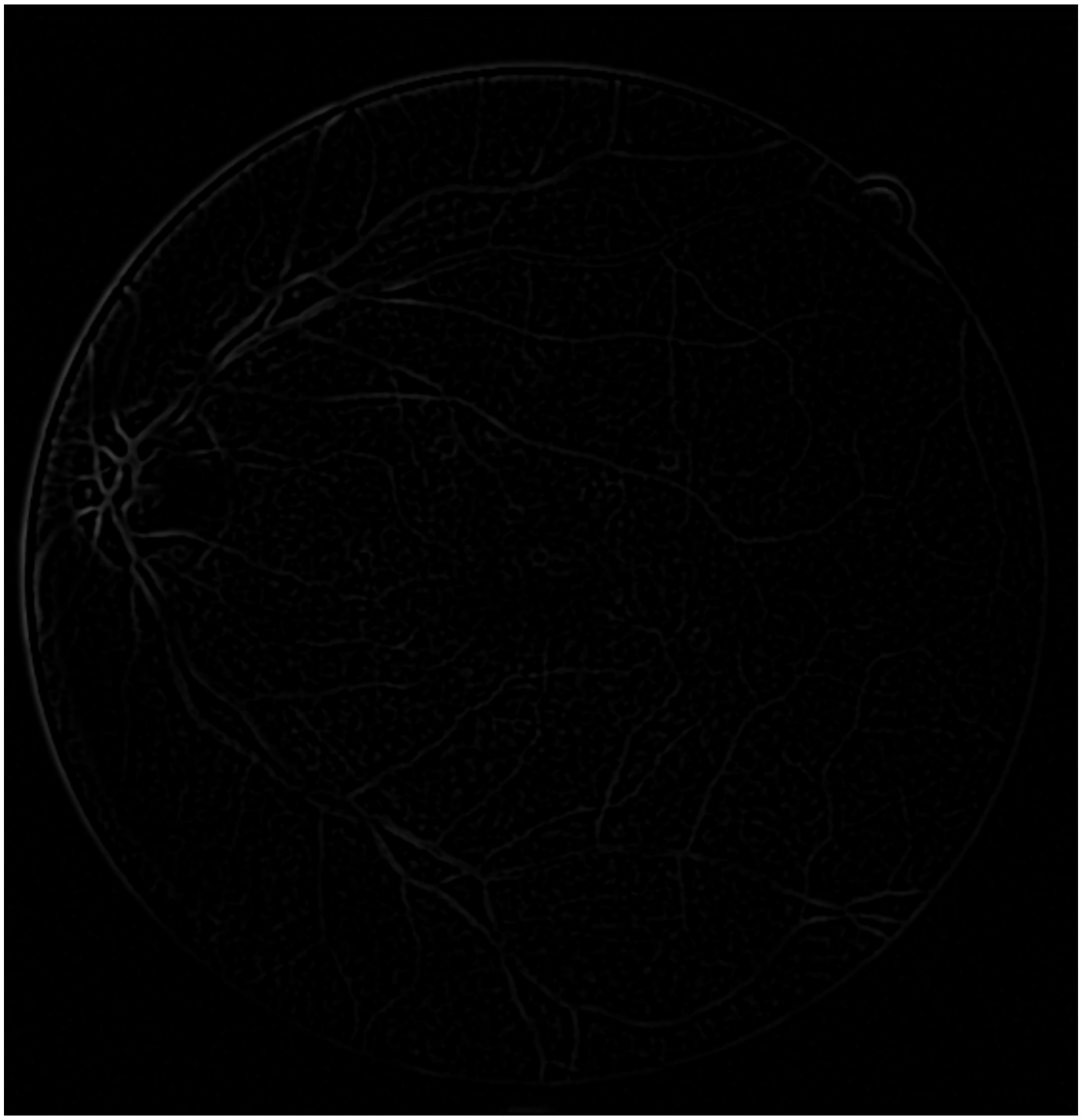
Apply Geodesic opening on the maximum

```
S1Data = geodesicOpening[SlapImage, maxOpeningOnSlapImage]
```

```
{{0.000290349, 0.000318456, 0.000358895, 0.000383397, 0.000355327, 0.000267228,
  0.00015834, 0.0000754616, ··· 550 ···, 0.000158945, 0.0000839143, 0.0000715021,
  0.0000585977, 0.0000535731, 0.0000487266, 0.0000456603}, ··· 582 ···, {··· 1 ···}}
```

large output | show less | show more | show all | set size limit...

```
S1 = Image[S1Data]
```



Calculating S_2

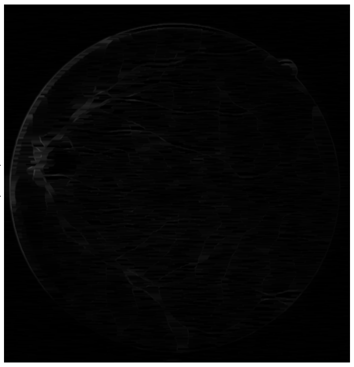
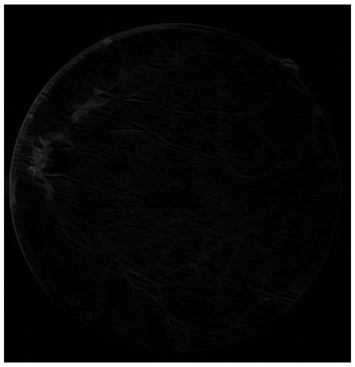
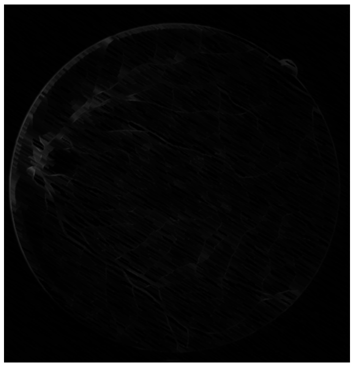
This step can be summarized in the followings:
1. Closing with 12 structure element on S_1.

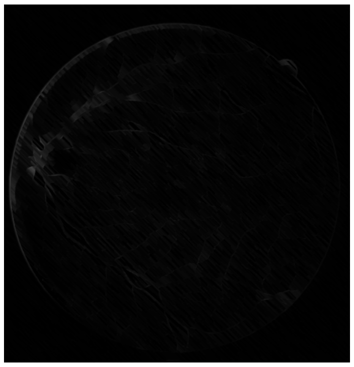
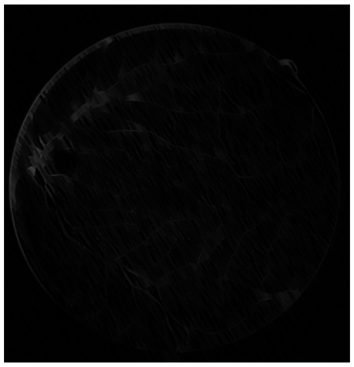
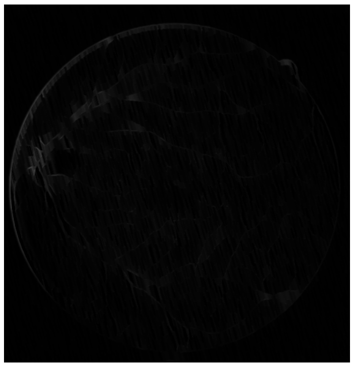
2. Pick the minimum from these 12 images pixel-wise and form an image.
3. Do geodesic closing with marker being S_1, mask being the minimum image.

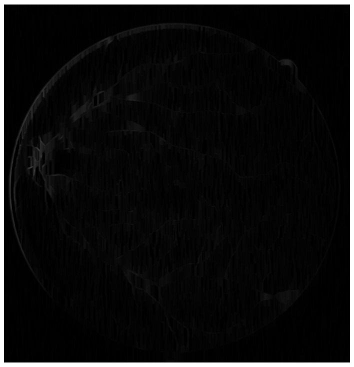
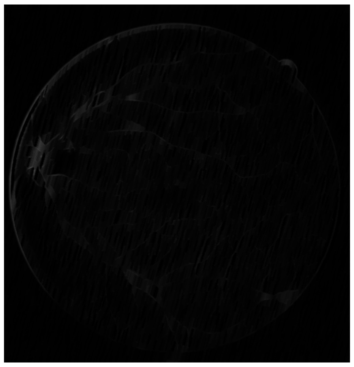
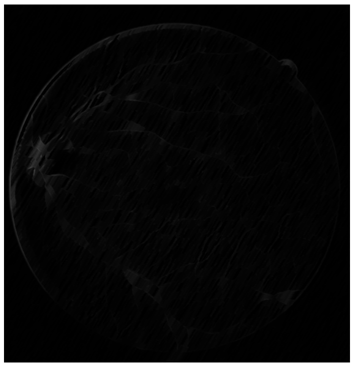
$$S_2 = \phi_{S_1}^{\text{rec}}\ (\text{Min}_{i=1,\ldots,12}\,\{\phi_{L_i}(S_1)\})$$

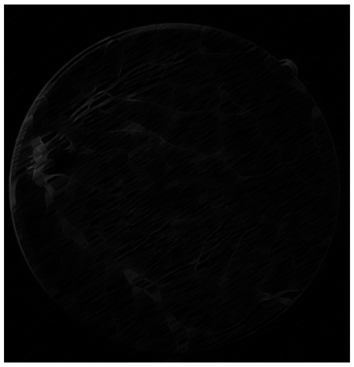
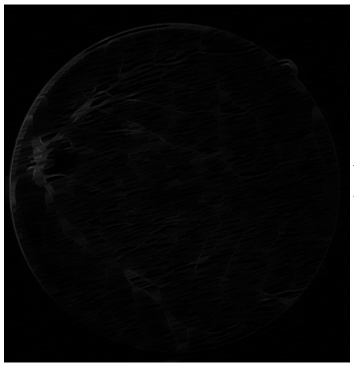
```
AbsoluteTiming[closingImageList = Closing[S1, #] & /@ kernels]
```

{0.240477,

{, , ,

, , ,

, , ,

, , }}

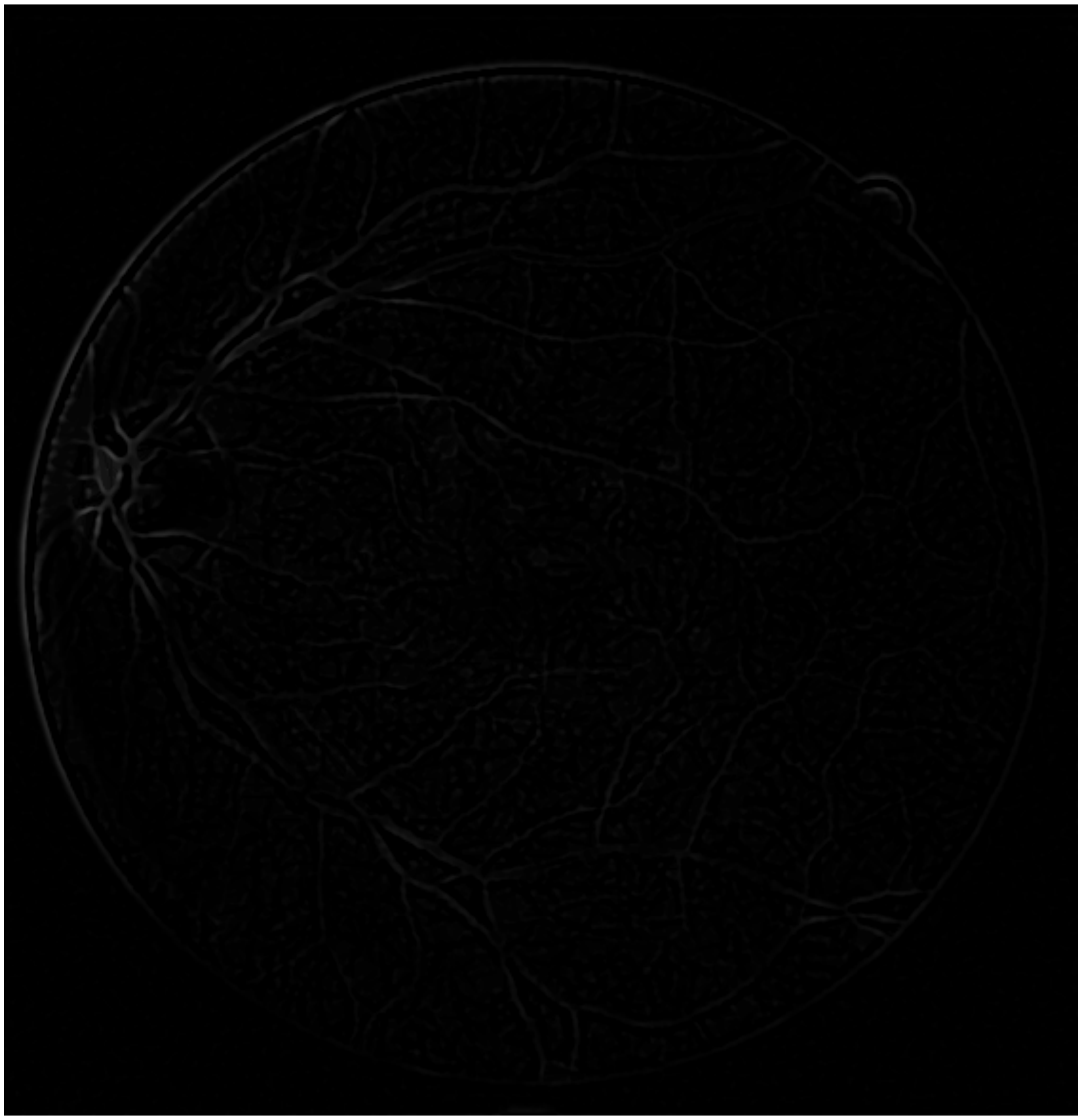
```
AbsoluteTiming[closingDataList = ImageData /@ closingImageList]
```

```
{0.000021,
 {{ … 1 … }, {{0.000703308, 0.000703308, 0.000550335, 0.000383397, 0.000355327,
    0.000355327, … 553 … , 0.000189741, 0.000189741, 0.000189741, 0.0000535731,
    0.0000487266, 0.0000487266}, … 582 … , { … 1 … }}, … 9 … , { … 1 … }}}
```

large output | show less | show more | show all | set size limit...

```
minClosingData = MapThread[Min, closingDataList, 2];
```

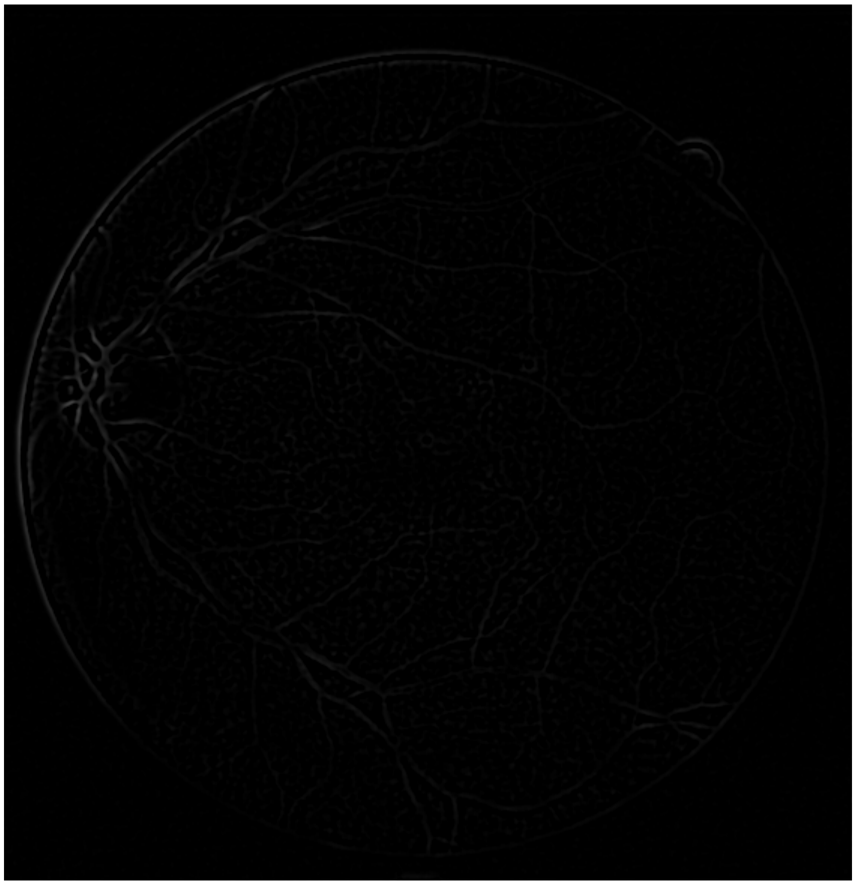
```
minClosingImage = Image[minClosingData]
```



```
S2Data = geodesicClosing[S1Data, minClosingData]
```

```
{{0.000290349, 0.000318456, 0.000358895, 0.000383397, 0.000355327, 0.000267228,
  0.00015834, 0.0000754616, ··· 550 ···, 0.000158945, 0.0000839143, 0.0000715021,
  0.0000585977, 0.0000535731, 0.0000487266, 0.0000456603}, ··· 582 ···, {··· 1 ···}}
large output    show less    show more    show all    set size limit...
```

```
S2 = Image[S2Data]
```



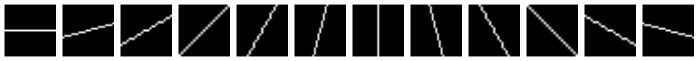
Calculating S_{res}

This step can be summarized in the followings:
1. Opening with 12 structure, r = 2 indicates the structure element 29 pixels long, 1 pixel wide on S_1.
2. Pick the maximum from these 12 images pixel-wise and form an image.
3. If the pixel value ≥ 1, assign 1 on that pixel, otherwise assign 0.

$$S_{\text{res}} = \left(\text{Max}_{i=1..12}\left\{\gamma^2_{L_i}(S_2)\right\} \geq 1\right)$$

scaling e = 2

```
linearStruct2 =
 Table[Round[{Cos[t], Sin[t]} * #] &/@ (Range[29] - 15), {t, 0, 11 π / 12, π / 12}]

GraphicsRow[showBinaryImage[kernelFromStructure[#]] &/@ linearStruct2]
```



```
kernels2 = kernelFromStructure[#] &/@ linearStruct2;

openingOnS2 = Opening[S2, #] &/@ kernels2

openingOnS2Data = ImageData /@ openingOnS2
```

```
{{{0.0000166485, 0.0000166485, 0.0000166485, 0.0000166485,
   0.0000166485, 0.0000166485, 0.0000166485, 0.0000166485, 0.0000166485,
   0.0000166485, 0.0000166485, 0.0000252046, … 541 …, 0., 0.,
   0., 0., 0., 0., 0., 0., 0., 0., 0., 0.}, … 583 …}, … 11 …}
```

large output | show less | show more | show all | set size limit...

```
SresData = MapThread[Max, openingOnS2Data, 2];

sres = Image[SresData]

sresByte = ImageData[sres, "Byte"]
```

```
{ … 1 … }
```

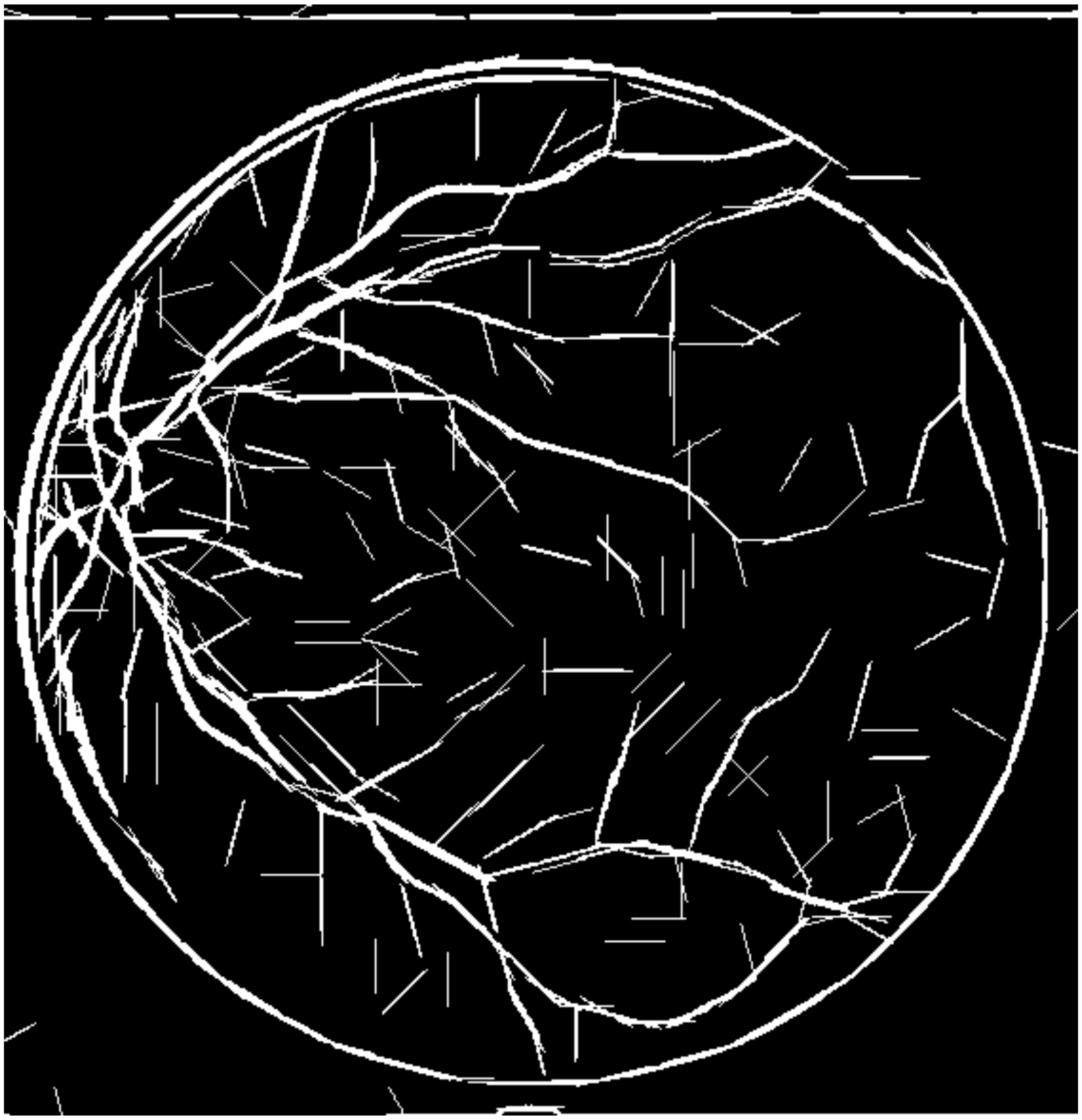
large output | show less | show more | show all | set size limit...

```
convertToBinary[x_] := If[x ≥ 1, 1, 0]

result = (convertToBinary /@ sresByte[[#]]) &/@ Range[Length[sresByte]]
```

```
{ … 1 … }
```

large output | show less | show more | show all | set size limit...

```
Image[result]
```



Summary

This paper is supposed to be very basic morphology's application. But I am very sorry that I didn't get the expected result.

I checked both my Mathematica prototyping and C++ implementation over and over again.
The possible mistakes could happen are as followings, I still need more time to check them or talk to people who are more experienced in this field.

1. From GeodesicOpening/Closing, I cannot check my implementation with Mathematica built-in functions anymore because Mathematica built-in function has different arguments. (It doesn't take mask image, only marker image)

2. Therefore, to follow the paper, I need to implement geodesicOpening/closing in Mathematica myself.

From my understanding, geodesic opening is by applying geodesic dilations on original image (gray) with a marker image (doesn't change) again and again until it gets "stable" (all pixels don't change anymore).

Then from a list of geodesic dilated images, we can pick the maximum point-wise, which will give us the geodesic opening result.

Geodesic closing is very straightforward hence easy to implement if Geodesic opening is correct.

I used **FixedPointList** function in Mathematica, which should work exactly as the theory described.

I used a threshold = 10^{-3} in C++, since I cannot compare two floating number are equal directly.

3. GaussianFilter

Mathematica's kernel for GaussianFilter is probably normalized. But the ratio between GaussianFilter's result and GaussianKernel's result is the same.

My question is, is that really so easy? The paper didn't include a parameter in the kernel. And I don't quite understand section III. C. Evaluation of the Curvature using the Laplacian.

Future Direction

1. For glaucoma detection, there is one approach from a paper which extract the optic disk first, then replace vessels, because vessels are regarded as noises. I already implement the first step that extract the optic disk automatically. I will implement this paper as well.
2. Google Brain is working on similar things
`http://research.google.com/teams/brain/healthcare/`
3. I think the most important thing is what theory I learned from this paper. I built an App this semester doing segmentation on Corneal Ulcer eyes. But the algorithm I developed is to the binarize on a threshold first and do morphology operations on binary images. Which works well for the fluorescent images but not working well on images under natural lights. I want to try using the morphology I learned from this paper to solve the natural light problem. There are three medical school students promised me to give me images for testing.
4.