Document for Intergration of MATLAB and Lumerical

Yuan Yujia

May 29, 2019

Contents

1 Installing library

In order to use MATLAB to control Lumerical, we first need to "install" the library in MATLAB which contains functions allow users to control Lumerical. Here is the link where required steps are listed.

$$https://kb.lumerical.com/install\_matlab\_integration.html$$

(I am a mac OS user, so only the procedure for mac is done by me. Nevertheless, that for the two operating systems is generally same, so I only introduce the mac version here.)

Users only need to follow the tutorial shown in the link exactly, which is not difficult. However, what they should do may slightly vary from what the tutorial says. First is that in the tutorial, the first line says type:

cd /Applications/MATLAB_R2015a.app/Contents

here we may have a different version of MATLAB, so I changed it to :

cd /Applications/MATLAB_R2018b.app/Contents

Also, on the third line, it says:

sudo vim Info.plist

Personally I feel vim is not neophyte-friendly for unix users, so I use nano instead:

sudo nano Info.plist

Now MATLAB has unlocked its capability of recognizing external library for Lumerical. We just need to specify the library so it can be loaded into MATLAB:

path(path,'/Applications/Lumerical/FDTD Solutions/FDTD Solutions.app/Contents/API/Matlab');

For Windows User, I have also found a course website of ECE204(credit: Douglas Harder). This is a nice tutorial for loading libraries:

https://ece.uwaterloo.ca/∼ dwharder/Matlab/

Now the setup is done, which means we have already added the "library" of Lumerical to MATLAB. Here after, MATLAB can recognize functions needed to operate Lumerical. However, it is the procedure which user needs to do each time when they open a MATLAB session. If they hope to let the computer automatically load the library(it is needed for users who use Lumerical extensively), they need to do the following steps:

1. Find the location of installation of MATLAB. For example: Mac ▷ Applications ▷ Matlab_R2017b.
2. Find document: toolbox ▷ driving ▷ drivingdemos ▷ html ▷ startup.m
3. Write this at the most bottom of the document:

   path(path,'/Applications/Lumerical/FDTD Solutions/FDTD Solutions.app/Contents/API/Matlab');

Then each time MATLAB is launched, the library will automatically be added and users can directly use instructions provided.

2 Functions for MATLAB

appopen('fdtd'); opens the FDTD solution. If there are other solutions, then only the parameter of the function is required to change(fdtd, mode,device,interconnect). We often write it like: h=appopen('fdtd'). Here h is like a handle in C# or like a pointer points to the newly created object in C++.

appclose(h); closes the Lumerical session opened by MATLAB. Each time when the work is done, closing corresponding window is recommended, for not occupying memory and reduce conflict of multi-users.

appevalscript(h,'scriptcommand'); sends a Lumerical command described in "scriptcommand" to the opened session h. It allows MATLAB to control Lumerical through script language for Lumerical.

appgetvar(h,'T') returns the value of T in Lumerical to MATLAB. It is used for the data transfer from Lumerical to MATLAB for analysis.

appputvar(h,'T',T_value); Basically it sends value T to T_value in Lumerical. It solves the problem that users are unable to use variable as a part of parameters in Lumerical when they program in MATLAB. Further explanation will be provided in following sections. Now the demo program does not include variable, so we temporarily dismiss it.

So now the whole thing looks like what I have tested:

```
h=appopen('fdtd');
appevalscript(h,'load("A.fsp");');
appevalscript(h,'run;');
appevalscript(h,'E=getresult("monitor1","E");');
appevalscript(h,'X=E.x;');
a=appgetvar(h,'X');
appclose(h);
```

Then the value of x axis, attribute E of monitor1 will be recorded by variable a in MATLAB. Those are functions used in the demo.

3 Building structure with Lumerical Script

This paragraph introduces how to build structures with script in Lumerical. Once we are capable of doing so, we can use instruction "appevalscript" in MATLAB to establish structures in Lumerical since it is the function which sends command in MATLAB to Lumerical.

First of all, we know that we have functions to "transform" Lumerical script to MATLAB code, so we just need to learn how to build structures in Lumerical with script instead of drawing. Functions used to establish structures are defined in this website: https://kb.lumerical.com/ref_scripts_adding_objects.html. Generally the pattern of Lumerical is like:

Add an object or object group	(1)
Define parameters of the object	(2)
Redo 1 and 2 to add another object until everything is added	(3)
Run the simulation	(4)
Extract data and analyze	(5)

So according to those steps, here below is the program segment which can build a structure, add source and monitor, run the simulation and extract data for analyze.

```
switchtolayout;
selectall;
delete;

addrect;

addpower;
set("name","field_profile");
set("monitor type",7);
set("x",0);
set("x span",1.2e-6);
set("y",0);
set("y span",1.2e-6);
set("z",1.3e-6);

addfdtd;
set("dimension",2);  #  1 = 2D, 2 = 3D
set("x",0);
set("x span",1e-6);
set("y",0);
set("y span",1e-6);
set("z",1e-6);
set("z span",2e-6);

run;
```

In this code, we first switch to layout mode instead of the data mode after running simulations, so that we can change the structure. Then select all created structure and delete them, so that we can build new structures from a blank space. The for loop in the third paragraph is to create periodic structure with by the for loop. Now we can successfully create structure and simulate its reflection characteristic. If we put each line into a "appevalscrip" function, then we can execute all of them in MATLAB and FDTD solution will be controlled to perform corresponding result.

4 Automation of string operation

Considering the fact that if the code in Lumerical is long, putting each of the code into "appevalscript" will be extremely troublesome. Therefore, I have written a python script for automation.

```
inputFileName = input("Enter the file name:")
fileIn = open(inputFileName,"r")
fileOut = open("Output.m","w")
fileOut.write("clc;\n")
fileOut.write("h=appopen(\'fdtd\');\n")
lines = fileIn.readlines()
for line in lines:
line = line.strip('\n')
line = line.strip('\r')
line = line.strip('\r\n')
if lines:
line = 'appevalscript(h,\'' + line + '\');'
fileOut.write(line+'\n')
#just for demo:
fileOut.write('appevalscript(h,\'E=getresult(\"field_profile\",\"E\");\');\n')
fileOut.write('appevalscript(h,\'X=E.x;\');\n')
fileOut.write('disp(appgetvar(h,\'X\'));\n')
##############

#if not demo, then we need the next line
#fileOut.write('appclose(h);')
fileIn.close()
fileOut.close()
```

If user wants to see the demo, then just leaves the code as what it is. If they want to use it in their own projects, then they can simply disable the "just for demo" part and enable the "if not demo" part.

The way of using this script is like the following:

1. Create a lumerical script and name it. e.g: input.lsf
2. Put the python script file into same folder as input.lsf
3. Open terminal, use cd to enter directory of the script
4. Execute the script by command "python3 Interoperation.py"
5. Enter the name of file(input.lsf here)
6. Output.m will be created, which is MATLAB file already

I have tested the code for tens of times, it should be relatively stable and reliable.

5 Create binary object and running

Now we have all the prerequisites for the final goal. So I will explain how to create binary object and let Lumerical run all the simulations consequently by instructions from matlab. First we need to realize the way to create binary object. We have two ways to do so. We can either obtain a binary file and import it or create a binary object directly. Examples about everything related to binary object can be found in this website:

https : //kb.lumerical.com/ref_sim_obj_importing_spatial_binary.html

5.1 importbinary

It is a function to import binary file into Lumerical. The major difference between this and Importbinary2 is that this function imports object from an external file, while Importbinary2 imports object from a matrix created by users by script. In the following code segment, section 1 creates a binary matrix file and writes it to a txt file; section 2 imports the file and execute. In real life, matrix file can also be downloaded from the internet or copied from elsewhere so only section two is necessary.

```
# Section 1
# export binary data to a text file

# generate a volume of data
radius = 50;
x = linspace(-radius,radius,100);
y = linspace(-radius,radius,101);
z = linspace(-radius,radius,102);
X = meshgrid3dx(x,y,z);
Y = meshgrid3dy(x,y,z);
Z = meshgrid3dz(x,y,z);

binary = (X^2+Y^2+Z^2) <= radius^2;


# choose a filename and delete the file if it exists
filename = "usr_importbinary_3d.txt";
rm(filename);

# write the header
write(filename,num2str(length(x)) + " " + num2str(x(1)) + " " + num2str(x(length(x))));
write(filename,num2str(length(y)) + " " + num2str(y(1)) + " " + num2str(y(length(y))));
write(filename,num2str(length(z)) + " " + num2str(z(1)) + " " + num2str(z(length(z))));


# write the vector to file, reshape to a column
write(filename,num2str(reshape(binary,[length(binary),1])));

# End of section 1
################################################


################################################
# Section 2
# load binary data from a text file into import object

# add an import objectz

addimport;

# choose the desired options
filename = "usr_importbinary_3d.txt";
file_units = "nm";
x0 = 0;
y0 = 0;
z0 = 0;
reverse_index_order = 0; # choose 1 to reverse order of indices as read from file

# import the data from file
importbinary(filename,file_units,x0,y0,z0,reverse_index_order);
set("material","Au (Gold) - CRC");

# End of section 2
################################################
```

5.2 importbinary2

From the italic fonted website address above, there is code piece explaining how to create a binary object and use it. I just copied it from this website:

```
x = [-3e-6;3e-6];
y = [-2e-6;2e-6];
```

```
z = [-1e-6;1e-6];

binary = zeros(2,2,2);
binary(1,1,1) = 1;
binary(2,1,2) = 1;

addimport;
importbinary2(binary, x, y, z);
```

Here, the first 3 lines declare x, y and z as matrices with one column and two rows. Thus we can have a 2*2*2=8 grids with different sizes. Then we create a 2*2*2 matrix with only (1,1,1) and (2,1,2) filled and other elements empty. We then use "addimport" to add a object as a binary object, and assign it the arribute with "importbinary2()". If we want to create, for example, a 2*2*3 object, then we should define z as: z=[-1e-6,1e-6,5e-6](numbers are arbitrary) and binary = zeros(2,2,3). As a result, we can get a user-defined object. A sample program is provided below. It will first open the FDTD solution and build planeSource, monitor, boundary from empty file, then automatically build new objects and run simulations. It will also send data from monitors to MATLAB.

```
clc;
h=appopen('fdtd');
appevalscript(h,'switchtolayout;');
appevalscript(h,'deleteall;');
appevalscript(h,'clear;');

appevalscript(h,'addplane;');
appevalscript(h,'set("injection axis","z");');
appevalscript(h,'set("direction","backward");');
appevalscript(h,'set("x",0);');
appevalscript(h,'set("x span",2e-6);');
appevalscript(h,'set("y",0);');
appevalscript(h,'set("y span",2e-6);');
appevalscript(h,'set("z",1.7e-6);');

appevalscript(h,'addpower;');
appevalscript(h,'set("name","field_profile");');
appevalscript(h,'set("monitor type",7);');
appevalscript(h,'set("x",0);');
appevalscript(h,'set("x span",1.2e-6);');
appevalscript(h,'set("y",0);');
appevalscript(h,'set("y span",1.2e-6);');
appevalscript(h,'set("z",1.3e-6);');
appevalscript(h,'');
appevalscript(h,'');

appevalscript(h,'addfdtd;');
appevalscript(h,'set("dimension",2);  #1=2D,2=3D');
appevalscript(h,'set("x",0);');
appevalscript(h,'set("x span",1e-6);');
appevalscript(h,'set("y",0);');
appevalscript(h,'set("y span",7e-7);');
appevalscript(h,'set("z",1e-6);');
appevalscript(h,'set("z span",2e-6);');
appevalscript(h,'');
appevalscript(h,'');
appevalscript(h,'x = [-5e-7;-2e-7;-1e-7;3e-7;4e-7];');
appevalscript(h,'y = [-2e-7;2e-7];');
appevalscript(h,'z = [-3e-7;3e-7];');
appevalscript(h,'');
appevalscript(h,'f=zeros(5,2,2);');

for i=1:5
    appevalscript(h,'switchtolayout;');

    appputvar(h,'x_coordinate',i);
    appevalscript(h,'f(x_coordinate,1,1)=1;');
    appevalscript(h,'addimport;');
    appevalscript(h,'importbinary2(f,x,y,z);');
    appevalscript(h,'set("name","rect");');
    appevalscript(h,'set("x",0);');
    appevalscript(h,'set("y",0);');
    appevalscript(h,'set("z",5e-7);');
    appevalscript(h,'set("material","Au (Gold) - Palik");'

    fprintf('The %dth result.\n',i);

    appevalscript(h,'run;');
    appevalscript(h,'E=getresult("field_profile","E");');
    appevalscript(h,'X=E.x;');
    disp(appgetvar(h,'X'));

    appevalscript(h,'switchtolayout;');
    appevalscript(h,'select("rect");');
    appevalscript(h,'delete;');

    appevalscript(h,'f(x_coordinate,1,1)=0;');
end
```

6 Read Binary Object and process boundary

6.1 Read object from Lumerical by index monitor

In order to revise binary object in MATLAB, we first need to obtain the matrix of the object file in MATLAB. Basically we can do it in the same way as obtaining data from Lumerical. Here, we use the index monitor in Lumerical to show what the surface looks like by identifying different reflexivity. It is important to realize that how fine the structure looks like depends on mesh fineness of FDTD area. So if a very precise result is anticipated, then before obtaining data from index monitor, dx, dy and dz of FDTD needs to be set to small values. Here I just provide a code piece which users can use it to read real part of reflexing index and, basically, see a "0" if reflex index is different from the base which has index of 1.

```
clc;
h=appopen('fdtd');
appevalscript(h,'load("FinalTest_copy.fsp");');

appevalscript(h,'switchtolayout;');
appevalscript(h,'clear;');

% appevalscript(h,'run;');
appevalscript(h,'E=getresult("monitor_1","index preview");');
appevalscript(h,'X=E.index_x;');
E=appgetvar(h,'X');

for i=1:11
    for j=1:11
        E(i,j)=real(E(i,j));
        if(E(i,j)) < 1
            E(i,j)=0;
        end
    end
end
disp(E);
```

6.2 Read object from image

6.2.1 Functions used in MATLAB

From the above code, we know that we definitely can read the shape from Lumerical by index monitor. We can also read the 2-D object from an image. Below, MATLAB functions which may be used are listed for easy reference.

I=imread(file) Takes a string as a file name as input argument. Returns a 3-D matrix. The third dimension will always has three elements: 1, 2 and 3. They indicate colors Red(R), Green(G) and Blue(B). So for example, imread(file)(:, :, 1) will return the color matrix indicating distribution of red. Each of R, G and B values from 0 to 255(no color to full color). I is the result matrix.

I=rgb2gray(I) Input I is (n x n x 3) matrix. The function turns this matrix to be a (n x n) matrix with only gray scale. Return value I is this (n x n) matrix.

J=imresize(I,[size,size]) Input I is a (n x n) matrix, we resize it with a (size x size) matrix.

J = imbinarize(J) Turns J(gray scale image) into a binary image with only 0s and 1s as its entries.

B = bwboundaries(BW) B is a cell array containing matricies. Each matrix contains X and Y values of a boundary in the binary image. e.g: If there are 3 boundaries, then there will be 3 matricies in B. B{n}(:,2) and B{n}(:,1) represents X and Y values of the nth boundary respectively. For more info: https://www.mathworks.com/help/images/ref/bwboundaries.html#bu1yffp-1

B = bwperim(I) bwperim returns boundaries of binary image I to the variable B by letting all the boundary entries to be 1 and all other entries to be 0. In another word, B is a binary 2-D object without separating each boundary from each other.

imshow(I);truesize[n,m] Shows the image I. It can take binary or RGB or gray-scaled image as input. Output will be a visualized image displayed on the new alerted window. Truesize is the function adjusts the size of the window. I usually choose (400,400) as the value of (n,m).

disp(X) Displays anything as input X in the command window. The displayed thing will be a matrix instead of graph.

plot(X, Y, 'specified', val) Displays the function of Y in respect of X. Users can also specify some features of the lines. For example, if they type "LineWidth" as "specified" and val=2, then the plot will be 2 times more thick. More specific explainations are provided here: https://www.mathworks.com/help/matlab/ref/plot.html#btzitot-LineSpec

6.2.2 Example for section 6.2

I wrote a function which can read the image and returns the value of binary information. Users can also use imbinarize directly. But for gray-scaled images with very small number of grids, its result is not ideal. So I strongly recommend all users use this function written by me.

```
function J = readImage(name,size,isBlackIndex)
    %read image and fit it into (15x15) matrix
    I=imread(name);
    %now I is 45x45x3 matrix, with third dimension to be RGB colors
    %we use rgb2gray to convert I to be a 2-Dimensional matrix
    I = rgb2gray(I);
    %imresize converts (n x n) matrix to (size x size) matrix
    J=imresize(I,[size,size]);

    %make colors eigher black or white instead of transition colors
    for index=1:numel(J)
        if J(index) >= isBlackIndex
            J(index)=255;
        end
        if J(index) < isBlackIndex
            J(index)=0;
        end
    end

    %transform gray scale matrix to binary matrix
    J = imbinarize(J);
end
```

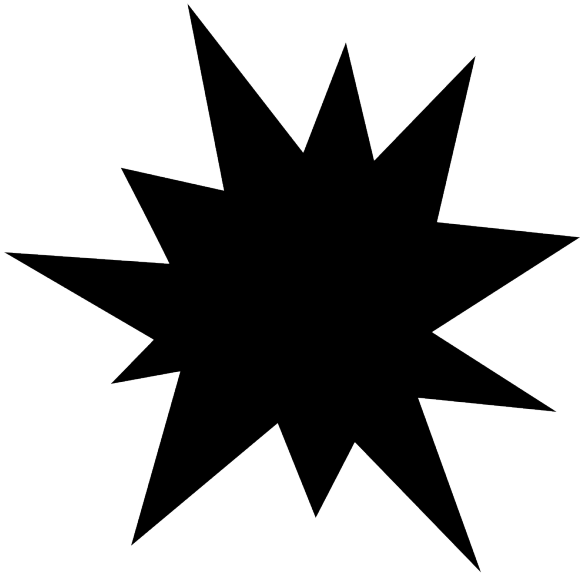
Of all the parameters, "name" is the name of the file users may wish to process. e.g: "photo.jpg". Pay attention, if users want to use input string instead of a variable, they need to add quote symbols. "size" is the resolution of the imported matrix. If size is, say, 25, then the image will be identified as a 25x25 matrix. Larger the number is, the more smooth curves will be but the matrix will be larger thus occupying more space. isBlackIndex is used for identifying whether a gray grid is black or not. Notice: sometimes they need to try several times for the most suitable value according to different matrix size of the image. From what I know, when size=15 and when size=200, isBlackIndex can vary a lot.

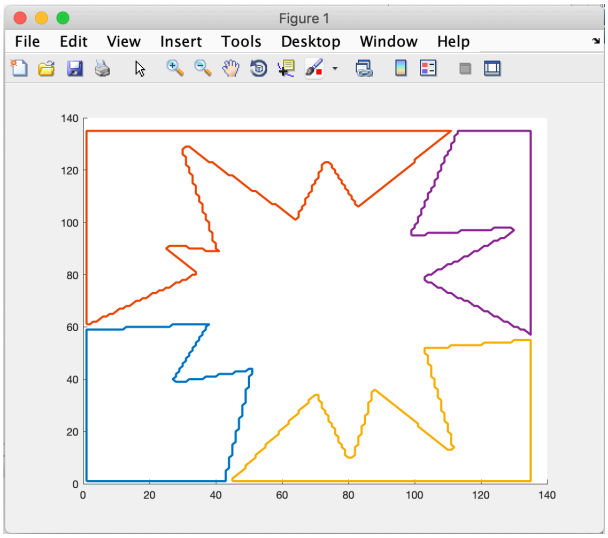
6.3 Find peripheral from a binary image

```
clc;
clear;

J=readImage('a.png',700,210);
B = bwboundaries(J);

hold on
for k = 1:length(B)
    Boundary = B{k};
    plot(Boundary(:,2), -Boundary(:,1), 'LineWidth', 2)
end
```

First we clear console and variables and read the image by the function provided above. Hold on means when plotting the second image, the first one is not cleared. Now for each boundary, we plot it. The result can be seen below. First image is the

gray-scaled image provided online. The second is the result output by the above code.





7 Revising binary object

7.1 Morphological operations

https://www.mathworks.com/help/images/morphological-filtering.html

The way to revise our binary object is to use **morphological operators**. All the detail about the operators is in the above link. Here we discuss functions used in this section.

J = bwmorph(I, operation) Takes a binary image and outputs the processed one. The process is specified by users by choosing different value for "operation". For specific arguments, refer to the link above.

J = bwmorph(I, 'clean') Gets rid of isolating part which is usually white noise spreading around. If a pixel is 1 and everything surround it is 0, then make this pixel 0.

J = bwmorph(I, 'majority') Obtains a (3 x 3) matrix for each pixel centered at that pixel being processed. If there are 5 or more entries of the matrix is '1', then the pixel is made to be 1. Otherwise this pixel is '0'. This can be used to clean rubbish part after erosion,

J = bwmorph(I, 'bridge') Connects disconncted pixels to their nearest neighbors.

J = imerode(I, SE) Erodes an image. "Erode" means shrink the edge of peripherals of the image. It is usually used for disconnecting two objects connecting weakly or smoothing indentations. This function will erode the image I according to rule SE. For se, users can define any of its size, shape and orientation. Refer to this website for more information of creating and defining SE: https://www.mathworks.com/help/images/ref/strel.html

J = imdilate(I, SE) Similar to Erode but is its opposite operation. It basically makes the edge of an image "fatter". So that it can reduce sharpness of an extruding part.

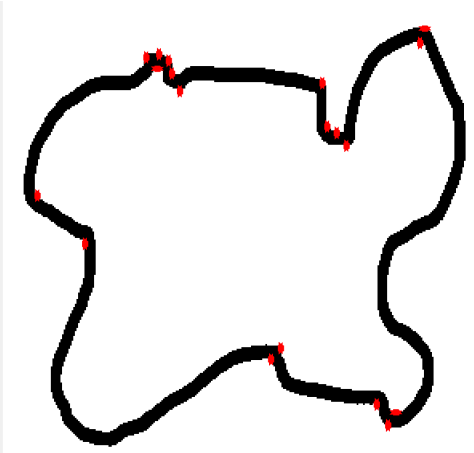
7.2 Curvature of boundaries

We will use a function "LineCurve2D". Its source code is too long so I place it in the appendix(section 8).

```
clc;
clear;

I = readImage('ddd.png',400,220);
boundaries = bwboundaries(I);

figure;
imshow(I);
truesize([600,600]);
hold on
for k=1:length(boundaries)
    Boundary = boundaries{k};
%     plot(Boundary(:,2), Boundary(:,1),'LineWidth',2)
    x = Boundary(:,2);
    y = Boundary(:,1);
    A = [x,y];

    K = LineCurvature2D(A,10);

    for index = 1 : numel(K)
        if abs(K(index)) >= 0.4
            plot(x(index),y(index),'-.r*');
        end
    end
end
```

First we clear console and variable as usual. Then read image and find boundaries as what is discussed in section 6. Now for each boundary, we list its x and y components and put them as columns of matrix A. We do so because LineCurvature2D() takes an argument in form of A. Then we set step size as 20(should be changed in different cases, just like parameter of neuroNetwork). For each index greater than a value ($k \geq 0.1$), we plot the area with a high turning speed. 0.1 here is the "CURVATURE" specified by users.

Pay attention: since all what we are doing is on the binary object, if the number of grids is too small, the function may get confused about what is the curvature of the image and what is the curvature caused by inaccuracy of binary pixels. That is, edges of binary object is not smoothened. So a relatively large number of pixels read is recommended, as well as the step size greater than 7. Below is the result of running this code:



7.3 Smooth image

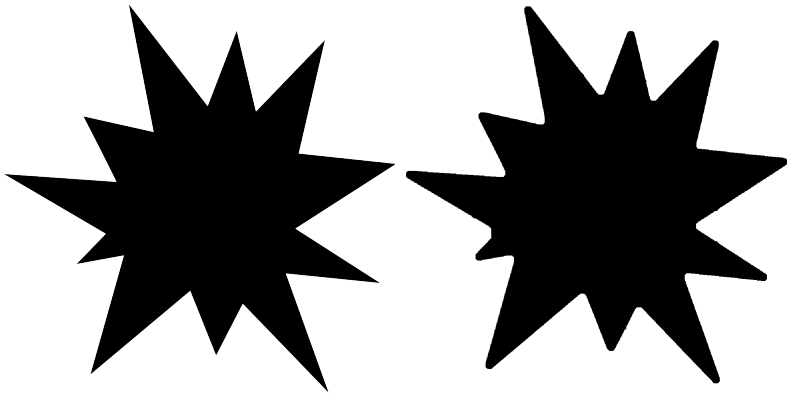
First as usual, we clear console and variables and read image. Then we use bwmorph function to get rid of white noise spreading around to prevent them being dilated when doing dilate operation. Then as sectino 7.2 did, boundaries and curvatures

are obtained. Everything between the two "%%%%%" is the core algorithm. For each point with abnormal curvature, we do image process to it. However, image process functions can only be operated on the whole image instead of a part of it. So we create a new matrix and copy everything around the point requiring for revision and call it a "tmp" matrix. If index $\geq$ 0.2, then it means the shape around the point is an indentation which requires open operation, that is, erosion followed by dilation. If index $\leq$ -0.2, then it means the shape around the point is a pointed sharp which requires close operation, that is, dilation followed by erosion. We do corresponding operation to them and cover the revised area of the original image with the new matrix. After doing all the local operations, we clean the image and open and close it for further clearance. Open and close does not affect the shape of structure since they contain two inverse operations.

```
clc;
clear;

J = readImage('ddd.png',1000,220);

J = bwmorph(J,'clean');
J = bwmorph(J,'bridge');
J = bwmorph(J,'majority');

boundaries = bwboundaries(J);

for k=1:length(boundaries)
    Boundary = boundaries{k};
    x = Boundary(:,2);
    y = Boundary(:,1);
    A = [x,y];

    K = LineCurvature2D(A,10);

    for index = 1 : numel(K)
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        if abs(K(index)) >= 0.2
            % first we create a new matrix for image process
            tmp = zeros(120);
            for i = -60:59
                for j = -60:59
                    if y(index)+j>0 && x(index)+i>0 && y(index)+j<1000 && x(index)+i<1000
                        tmp(j+61,i+61) = J(y(index)+j,x(index)+i);
                    end
                end
            end

            % process it according to its sign
            se = strel('disk',10);
            if(K(index)>= 0.2)
                  tmp = imopen(tmp,se);
            end
            if(K(index)<= -0.2)
                  tmp = imclose(tmp,se);
            end

            % put processed matrix back
            for i = -60:59
                for j = -60:59
                    if y(index)+j>0 && x(index)+i>0 && y(index)+j<1000 && x(index)+i<1000
                     J(y(index)+j,x(index)+i) = tmp(j+61,i+61);
                    end
                end
            end
%
            J = bwmorph(J,'clean');
            J = bwmorph(J,'bridge');
            J = bwmorph(J,'majority',100);
```

```
            se = strel('disk',10);
            J = imopen(J,se);
            J = imclose(J,se);

        end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    end
end

figure;
imshow(J);
truesize([600,600]);
```

Left is the original image, right is the one being revised.

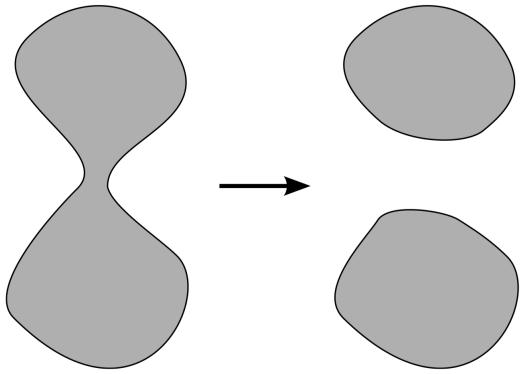


7.4 Bridging and Disconnection

7.4.1 Remove Bridges

If boundaries of two objects are too close to each other but not connected, the narrow empty space can cause difficulty in optical fabrication. Or if the bridge between two objects is too narrow, it also causes problems. So our main target in this section is to get rid of bridges or disconnections which are too small for fabrication. In order to save the running time, it is not suitable for us to detect all the boundaries and to see the smallest route length from this boundary to that: it will takes $O(n^4)$ of iterations for each pixel to find its behavior. So instead, we do something to the whole image which will not affect the main structure but will change what is inside.

Luckily, we have those functions defined in MATLAB. They are imdilate() and imerode(). In order to get rid of bridge, we first erode the image. The result of erosion is like what is shown in the picture[AUTOMATED DESIGN OF PHOTONIC DEVICES, Alexander Y. Piggott].

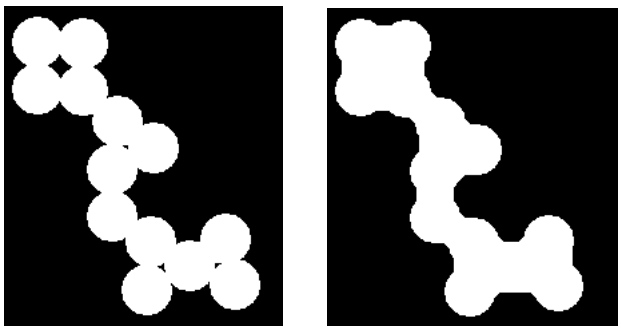


Now we do dilation again, which is the inverse operation of erosion. we can get two "balls" with same shape as before, but the connection is not there. Do not worry if the bridge is not thick enough and will be extremely thin after processing. The dilation will return the bridge to its original thickness if it is not thin enough to be get rid of.

There is a function called imopen. It basically does erosion first and then dilation. So by this function, we can remove bridges being too thin.

7.4.2 Remove disconnections

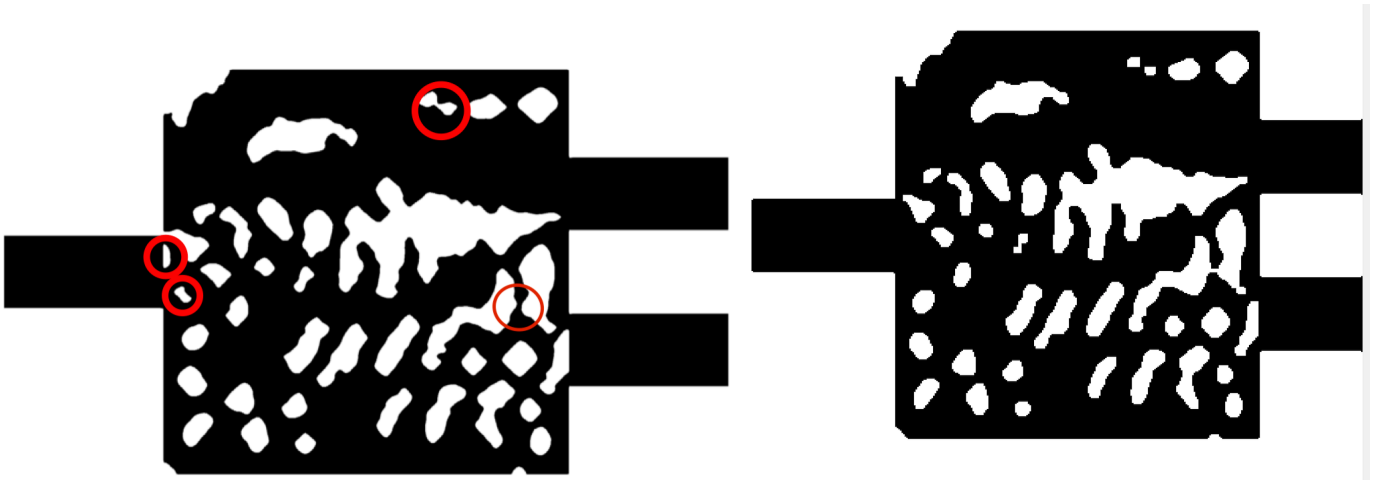
Also for two objects close to each other, we may wish to connect those two instead of leaving them disconnected. Here we use the "imclose", which does dilation first and then erosion. It can also get rid of holes which are small inside an object. The sample result of close operation is given below[MathWork forum].



7.4.3 Implementation

```
clc;
clear;

J = readImage('a.png',400,220);
J = bwmorph(J, 'clean');
J = bwmorph(J, 'majority');
J = bwmorph(J, 'bridge');

se = strel('disk',3);
J = imopen(J,se);
J = imclose(J,se);

figure;
imshow(J);
truesize([600,600]);
```

Left is the input image and right is output from the above code



The document finishes here. Users can reconstruct all the result by copying codes included. THE CODE EXCEPT LIBRARIES, WHICH DOES THE JOB OF ALL ABOVE SECTIONS IS INTEGRATED INTO ONE FUNCTION AND CAN BE SEEN IN THE APPENDIX.

8 Replace Lumerical object with revised one

9 APPENDIX: sample code

9.1 readImage()

It is included in the text. However, for convenience, I write it again here as "library" which can be used easily by users by just copying and pasting. Name is name of image users may wish the program to read. Size is how large the matrix representing binary image will be. isBlackIndex is to see when a gray pixel is identified as black, usually its value is 220. IF THE INPUT IMAGE IS BINARY: REMOVE "I=rgb2gray" SENTENCE FOR CORRECT RESULT

```
function J = readImage(name,size,isBlackIndex)
    %read image and fit it into (15x15) matrix
    I=imread(name);
    %now I is 45x45x3 matrix, with third dimension to be RGB colors
    %we use rgb2gray to convert I to be a 2-Dimensional matrix
    I = rgb2gray(I);
    %imresize converts (n x n) matrix to (size x size) matrix
    J=imresize(I,[size,size]);

    %make colors eigher black or white instead of transition colors
    for index=1:numel(J)
        if J(index) >= isBlackIndex
            J(index)=255;
        end
        if J(index) < isBlackIndex
            J(index)=0;
        end
    end

    %transform gray scale matrix to binary matrix
    J = imbinarize(J);
end
```

9.2 LineCurve2D()

It is a function updated from the Internet. Basically it uses module length of a higher dimensioned object and project it onto a 2-D surface. It succeeds all tests. Specific documentation is inside the code as annotation.

```
function k=LineCurvature2D(Vertices,divisionNumber)
% Part of this function is written by D.Kroon University of Twente (August 2011)

% This function calculates the curvature of a 2D line. It first fits
% polygons to the points. Then calculates the analytical curvature from
% the polygons;
%
%  k = LineCurvature2D(Vertices,Lines)
%
% inputs,
%   Vertices : A M x 2 list of line points.
%   (optional)
%   divisionNumber : A number indicating step size
%
% outputs,
%   k : M x 1 Curvature values
%
%%%%%%%%%%%%% BEING CHANGED %%%%%%%%%%%%%%%%%%
% If no line-indices, assume a x(1) connected with x(2), x(3) with x(4) ...
% if(nargin<2)
%     Lines=[(1:(size(Vertices,1)-1))' (2:size(Vertices,1))'];
% end
% USERS CHANGE DIVISIONNUMBER SO THEY CAN SET THE STEP SIZE
numVertices = floor( size(Vertices,1) / divisionNumber );
Lines=[(1:(numVertices-1))' (2:numVertices)'];
Lines=Lines .* divisionNumber;
```

```
% Get left and right neighbor of each points
Na=zeros(size(Vertices,1),1); Nb=zeros(size(Vertices,1),1);
Na(Lines(:,1))=Lines(:,2); Nb(Lines(:,2))=Lines(:,1);

% Check for end of line points, without a left or right neighbor
checkNa=Na==0; checkNb=Nb==0;
Naa=Na; Nbb=Nb;
Naa(checkNa)=find(checkNa); Nbb(checkNb)=find(checkNb);

% If no left neighbor use two right neighbors, and the same for right...
Na(checkNa)=Nbb(Nbb(checkNa)); Nb(checkNb)=Naa(Naa(checkNb));

% Correct for sampeling differences
Ta=-sqrt(sum((Vertices-Vertices(Na,:)).^2,2));
Tb=sqrt(sum((Vertices-Vertices(Nb,:)).^2,2));

% If no left neighbor use two right neighbors, and the same for right...
Ta(checkNa)=-Ta(checkNa); Tb(checkNb)=-Tb(checkNb);

% Fit a polygons to the vertices
% x=a(3)*t^2 + a(2)*t + a(1)
% y=b(3)*t^2 + b(2)*t + b(1)
% we know the x,y of every vertice and set t=0 for the vertices, and
% t=Ta for left vertices, and t=Tb for right vertices,
x = [Vertices(Na,1) Vertices(:,1) Vertices(Nb,1)];
y = [Vertices(Na,2) Vertices(:,2) Vertices(Nb,2)];
M = [ones(size(Tb)) -Ta Ta.^2 ones(size(Tb)) zeros(size(Tb)) zeros(size(Tb)) ones(size(Tb)) -Tb Tb.^2];
invM=inverse3(M);
a(:,1)=invM(:,1,1).*x(:,1)+invM(:,2,1).*x(:,2)+invM(:,3,1).*x(:,3);
a(:,2)=invM(:,1,2).*x(:,1)+invM(:,2,2).*x(:,2)+invM(:,3,2).*x(:,3);
a(:,3)=invM(:,1,3).*x(:,1)+invM(:,2,3).*x(:,2)+invM(:,3,3).*x(:,3);
b(:,1)=invM(:,1,1).*y(:,1)+invM(:,2,1).*y(:,2)+invM(:,3,1).*y(:,3);
b(:,2)=invM(:,1,2).*y(:,1)+invM(:,2,2).*y(:,2)+invM(:,3,2).*y(:,3);
b(:,3)=invM(:,1,3).*y(:,1)+invM(:,2,3).*y(:,2)+invM(:,3,3).*y(:,3);

% Calculate the curvature from the fitted polygon
k = 2*(a(:,2).*b(:,3)-a(:,3).*b(:,2)) ./ ((a(:,2).^2+b(:,2).^2).^(3/2));

function  Minv  = inverse3(M)
% This function does inv(M) , but then for an array of 3x3 matrices
adjM(:,1,1)=  M(:,5).*M(:,9)-M(:,8).*M(:,6);
adjM(:,1,2)=  -(M(:,4).*M(:,9)-M(:,7).*M(:,6));
adjM(:,1,3)=  M(:,4).*M(:,8)-M(:,7).*M(:,5);
adjM(:,2,1)=  -(M(:,2).*M(:,9)-M(:,8).*M(:,3));
adjM(:,2,2)=  M(:,1).*M(:,9)-M(:,7).*M(:,3);
adjM(:,2,3)=  -(M(:,1).*M(:,8)-M(:,7).*M(:,2));
adjM(:,3,1)=  M(:,2).*M(:,6)-M(:,5).*M(:,3);
adjM(:,3,2)=  -(M(:,1).*M(:,6)-M(:,4).*M(:,3));
adjM(:,3,3)=  M(:,1).*M(:,5)-M(:,4).*M(:,2);
detM=M(:,1).*M(:,5).*M(:,9)-M(:,1).*M(:,8).*M(:,6)-M(:,4).*M(:,2).*M(:,9)+M(:,4).*M(:,8).*M(:,3)+M(:,7).*M(:,2).
Minv=bsxfun(@rdivide,adjM,detM);
```

9.3 Integrated code(image from picture)

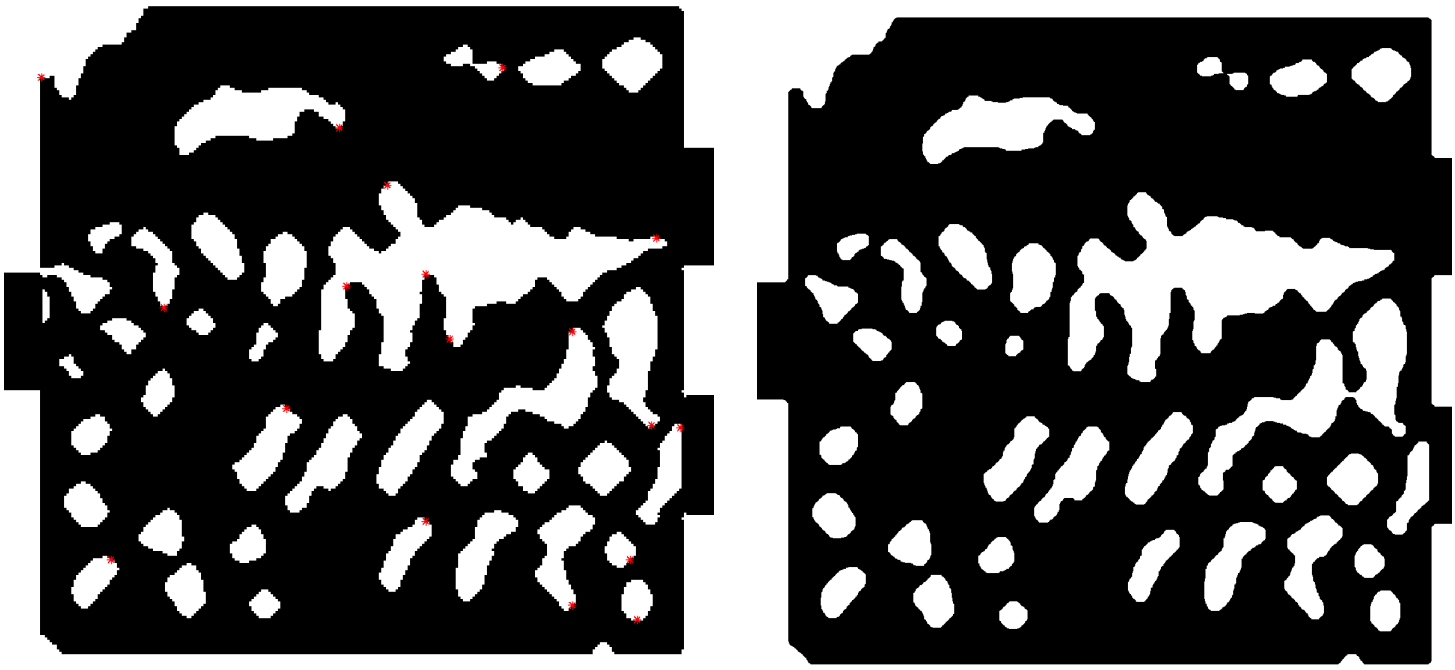
Function readImage() and LineCurve2D are provided in this document. Users are required to include them in order to reconstruct what this document have done. **If an image is gray-scaled, uses need to delete "RGB2gray" function in readImage.** Otherwise errors occur. Then, the main code is below:

```
% Created By Yuan Yujia on 23rd, May, 2019
% For revising binary objects
% This program implements the algorithm described by Alexandar Piggott in his
% PhD dissertation "AUTOMATED DESIGN OF PHOTONIC DEVICES". It
% can get rid of parts in binary object by smoothening its surface, disconnecting
% bridges which are too narrow and connecting two components which are very
% close to each other. Everything is automated.
%
% INPUT:
% Each user who wish to use this program should specify:
% Resolution, which is how

% many x and y pixels will an image be imported as a binary matrix.
% IntendedCurvature, which is the curvature allowed by users. The program will
% fix all the areas with curvature greater than this.

% localMorphArea: decides how much users may wish to revise the image. Too
% small of this value will cause the clearance not complete, too large its value
% will make the shape of large structure be changed once the program wants to fix
% a small local part. Also it is the size of copy matrix

% localMorphRadius: radius of the strel for morphological operation, can
% also be interpreted as radius of the disk which do morphological
% operation
%
% StepSizeOfLC2D: how long a distance for the interpolator to walk each time. The
% recommened value is between 8 to 15. Too large, the program may walk pass the
% point; too small, the program may think each diagonalized pixel as the point being
% fixed.

% globalMorphRadius: after fixing the shape, the program will do clearance, open and close operation
% again. This tells the extent of being globally changed. Usually I set it to be 2.

% isBlackorWhite: decides whether a pixel is black or white if the picture is RGB
% or gray-scale. Usually I set it to be 220, so each value from 220 to 255 is identified
% as "white", otherwise it is "black".
%
% OUTPUT:
% The program will output a matrix which represents the revised binary object.

clc;
clear;

%%%%%%%%%%%%%%%%%%%% random variable %%%%%%%%%%%%%%%%%%%%%%%%
resolution = 1000;
intendedCurvature = 0.25;
localMorphRadius = 8;
localMorphArea = 40;
% Users need to specify them according to different projects

%%%%%%%%% these are variables with recommended value %%%%%%
stepSizeOfLC2D=15;
globalMorphRadius = 7;
isBlackorWhite = 220;
%%%%%%%%%%%%% just leave it if not necessary %%%%%%%%%%%%%%%
```

```
J = readImage('ddd.png',resolution,isBlackorWhite);

for iteration = 1 : 3 % multi-time run for higher accuracy

J = bwmorph(J,'clean');
J = bwmorph(J,'bridge');
J = bwmorph(J,'majority');

boundaries = bwboundaries(J);

for k=1:length(boundaries)
    Boundary = boundaries{k};
    x = Boundary(:,2);
    y = Boundary(:,1);
    A = [x,y];

    K = LineCurvature2D(A,stepSizeOfLC2D);

    for index = 1 : numel(K)
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        if abs(K(index)) >= intendedCurvature
            % first we create a new matrix for image process
            tmp = zeros(2*localMorphArea);
            for i = -(localMorphArea):(localMorphArea-1)
                for j = -localMorphArea:(localMorphArea-1)
                    if y(index)+j>0 && x(index)+i>0 && y(index)+j<resolution && x(index)+i<resolution
                        tmp(j+localMorphArea+1,i+localMorphArea+1) = J(y(index)+j,x(index)+i);
                    end
                end
            end

            % process it according to its sign
            se = strel('disk',localMorphRadius);
            if(K(index)>= intendedCurvature)
                  tmp = imopen(tmp,se);
            end
            if(K(index)<= -intendedCurvature)
                  tmp = imclose(tmp,se);
            end

            % put processed matrix back
            for i = (-localMorphArea):(localMorphArea-1)
                for j = (-localMorphArea):(localMorphArea-1)
                    if y(index)+j>0 && x(index)+i>0 && y(index)+j<resolution && x(index)+i<resolution
                     J(y(index)+j,x(index)+i) = tmp(j+localMorphArea+1,i+localMorphArea+1);
                    end
                end
            end
%
            J = bwmorph(J,'clean');
            J = bwmorph(J,'bridge');
            J = bwmorph(J,'majority',100);
            se = strel('disk',globalMorphRadius);
            J = imopen(J,se);
            J = imclose(J,se);

        end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    end
end
```

```
end % large loop

figure;
imshow(J);
truesize([600,600]);
```

left is original image and right one is revised image. **Experimental show after 5 or 50 iterations of the largest loop, result is exactly same. So we only need 4 to 5 times loop. As a result, I do not set it as a variable for users to define. Instead, I have put it into the code as a fixed part.**



9.4 Integrated code(image from Lumerical)

This function is not fully developed, mainly because I do not know how to use put variable back to Lumerical. Maybe we can write another file for interaction? But how to revise the file to prevent duplication? Anyway, other code passed tests and can be used. They are here:

```
% Created By Yuan Yujia on 23rd, May, 2019
% For revising binary objects
% This program implements the algorithm described by Alexandar Piggott in his
% PhD dissertation "AUTOMATED DESIGN OF PHOTONIC DEVICES". It
% can get rid of parts in binary object by smoothening its surface, disconnecting
% bridges which are too narrow and connecting two components which are very
% close to each other. Everything is automated.
%
% INPUT:
% Each user who wish to use this program should specify:
% Resolution, which is how
% many x and y pixels will an image be imported as a binary matrix.
% IntendedCurvature, which is the curvature allowed by users. The program will
% fix all the areas with curvature greater than this.
% localMorphRadius: decides how much users may wish to revise the image. Too
% small of this value will cause the clearance not complete, too large its value
% will make the shape of large structure be changed once the program wants to fix
% a small local part.
%
% StepSizeOfLC2D: how long a distance for the interpolator to walk each time. The
% recommened value is between 8 to 15. Too large, the program may walk pass the
% point; too small, the program may think each diagonalized pixel as the point being
% fixed.
% globalMorphRadius: after fixing the shape, the program will do clearance, open and close operation
% again. This tells the extent of being globally changed. Usually I set it to be 2.
% isBlackorWhite: decides whether a pixel is black or white if the picture is RGB
% or gray-scale. Usually I set it to be 220, so each value from 220 to 255 is identified
% as "white", otherwise it is "black".
% YOU NEED TO WRITE FILENAME YOURSELF SINCE LUMERICAL DOESN'T SUPPORT
% STRING VARIABLE

% OUTPUT:
% The program will output a matrix which represents the revised binary object.


clc;
clear;
%%%%%%%%%%%%%%%%%%% random variable %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
resolution = 1000;
intendedCurvature = 0.35;
localMorphRadius = 3;
localMorphArea = 3;
% YOU NEED TO WRITE fileName YOURSELF
% Users need to specify them according to different projects
%%%%%%%%% these are variables with recommended value %%%%%%
stepSizeOfLC2D=12;
globalMorphRadius = 10;
isBlackorWhite = 220;
%%%%%%%%%%%%% just leave it if not necessary %%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%% From Lumerical to MATLAB %%%%%%%%%%%%%%%%%
h=appopen('fdtd');
appevalscript(h,'load("FinalTest_copy1.fsp");');
```

```
appevalscript(h,'switchtolayout;');
appevalscript(h,'clear;');

% appevalscript(h,'run;');
appevalscript(h,'E=getresult("monitor_1","index preview");');
appevalscript(h,'X=E.index_x;');
J = appgetvar(h,'X');

sizeJ = size(J);
J = real(J);

for i=1:sizeJ(1)
    for j=1:sizeJ(2)
        if(J(i,j)) < 1
            J(i,j)=0;
        end
    end
end

J = imbinarize(J);
%%%%%%%%%%%%%%%%%%%%%%%%%%% END %%%%%%%%%%%%%%%%%%%%%%%%%%%

for iteration = 1 : 3 % multi-time run for higher accuracy

J = bwmorph(J,'clean');
J = bwmorph(J,'bridge');
J = bwmorph(J,'majority');

boundaries = bwboundaries(J);

for k=1:length(boundaries)
    Boundary = boundaries{k};
    x = Boundary(:,2);
    y = Boundary(:,1);
    A = [x,y];

    K = LineCurvature2D(A,stepSizeOfLC2D);

    for index = 1 : numel(K)
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        if abs(K(index)) >= intendedCurvature
            % first we create a new matrix for image process
            tmp = zeros(2*localMorphArea);
            for i = -(localMorphArea):(localMorphArea-1)
                for j = -localMorphArea:(localMorphArea-1)
                    if y(index)+j>0 && x(index)+i>0 && y(index)+j<resolution && x(index)+i<resolution
                        tmp(j+localMorphArea+1,i+localMorphArea+1) = J(y(index)+j,x(index)+i);
                    end
                end
            end

            % process it according to its sign
            se = strel('disk',localMorphRadius);
            if(K(index)>= intendedCurvature)
                  tmp = imopen(tmp,se);
            end
            if(K(index)<= -intendedCurvature)
                  tmp = imclose(tmp,se);
            end
```

```
            % put processed matrix back
            for i = (-localMorphArea):(localMorphArea-1)
                for j = (-localMorphArea):(localMorphArea-1)
                    if y(index)+j>0 && x(index)+i>0 && y(index)+j<resolution && x(index)+i<resolution
                     J(y(index)+j,x(index)+i) = tmp(j+localMorphArea+1,i+localMorphArea+1);
                    end
                end
            end
%
            J = bwmorph(J,'clean');
            J = bwmorph(J,'bridge');
            J = bwmorph(J,'majority',100);
            se = strel('disk',globalMorphRadius);
            J = imopen(J,se);
            J = imclose(J,se);

        end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    end
end

end % large loop


%%%%%%%%% return the matrix back to Lumerical%%%%%%%%%
% create a file for exchange

file = fopen('tempFile.txt','wt');
for ii = 1:size(J,1)
    fprintf(file,'%g\t',J(ii,:));
    fprintf(file,'\n');
end
fclose(file);

% let Lumerical read the matrix
appevalscript(h,'M=readdata("tempFile.txt");');
sizeImage = size(J);
appputvar(h,'sizeX',sizeImage(1));
appputvar(h,'sizeY',sizeImage(2));

appevalscript(h,'x = linspace(1,sizeX,sizeX);');
appevalscript(h,'y = linspace(1,sizeY,sizeY);');
appevalscript(h,'z = [-1;1];');
appevalscript(h,'');
appevalscript(h,'binary = zeros(sizeX,sizeY,2);');
appevalscript(h,'');
appevalscript(h,'for(i = 1:sizeX) {');
appevalscript(h,'    for(j = 1:sizeY) {');
appevalscript(h,'        if(M(i,j)==1){binary(i,j,:) = 0;}');
appevalscript(h,'        else {binary(i,j,:) = 1;}');
appevalscript(h,'    }');
appevalscript(h,'}');

appevalscript(h,'addimport;');
appevalscript(h,'importbinary2(binary, x, y, z);');
appevalscript(h,'set("x",0);');
appevalscript(h,'set("y",0);');
appevalscript(h,'set("z",4.01e-7);');
appevalscript(h,'set("z span",1e-7);');
appevalscript(h,'set("x span",5e-7);');
appevalscript(h,'set("y span",5e-7);');
```

```
appevalscript(h,'set("material","etch");');

% delete file created

%%%%%%%%%%%%%%%%%%%% end %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure;
imshow(J);
truesize([600,600]);
```

NOTE: BUGS IN PROGRAM

1.When binary object is large, e.g. 1000x1000, the program will be extremely slow even deadlocked

2.Folder of output file of MATLAB should correspond to folder of input file of Lumerical, which is not done

3.Hidden bugs may exist when integrating this code into larger projects and require additional debugging time

Result: