# Gmsh & Toast++ Instruction guide

*Written by Ron Milutin and Oren Schor*

Contents

[Gmsh & Toast++ Instruction guide 1](#_Toc156255952)

[Create a simple 2D shape 3](#_Toc156255953)

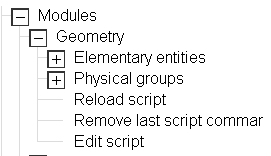
[Integrating Gmsh with Toast++ (example of DOT scan simulation) 6](#_Toc156255954)

# Create a simple 2D shape

Utilizing Gmsh's scripting language offers an efficient and precise approach for geometric construction.

* Open a new file script editor:

navigate to Modules -> Geometry -> Edit script.



* Initiating Points:

Start by defining a variable to represent the desired mesh size, for example:

‘sz = 0.01’. Subsequently, define the points with four parameters: X, Y, Z coordinates, and the mesh size. For instance:

Point(1) = {0, 0, 0, sz };

Point(2) = {0.1, 0, 0, sz};

Point(3) = {0, 0.1, 0, sz};

Ensure that each point is uniquely indexed.

* Surface Formation:

Initially, create a curve loop that interlinks all the curves. Then, define a plane surface utilizing these curve loops. Example:

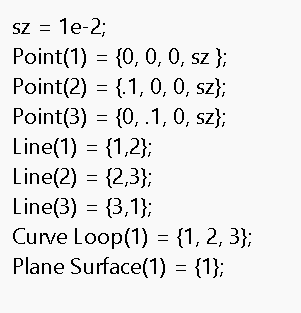
Line(1) = {1,2};

Line(2) = {2,3};

Line(3) = {3,1};

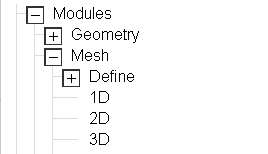
Plane Surface(1) = {1};

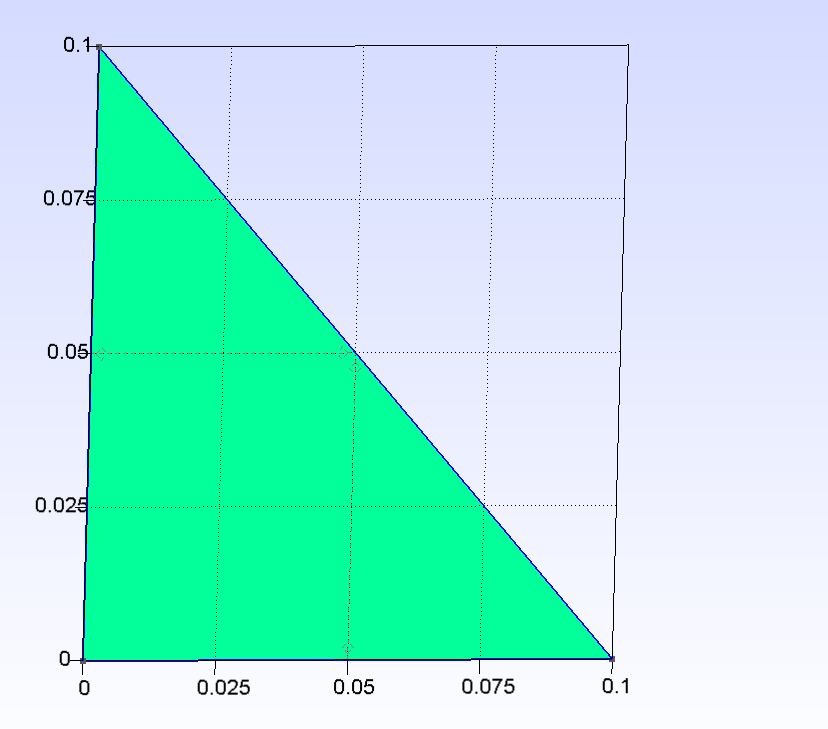
* All in all, you should get:



* Visualization:

To observe the created shape, navigate to Modules -> Mesh -> 2D.

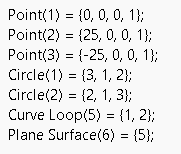




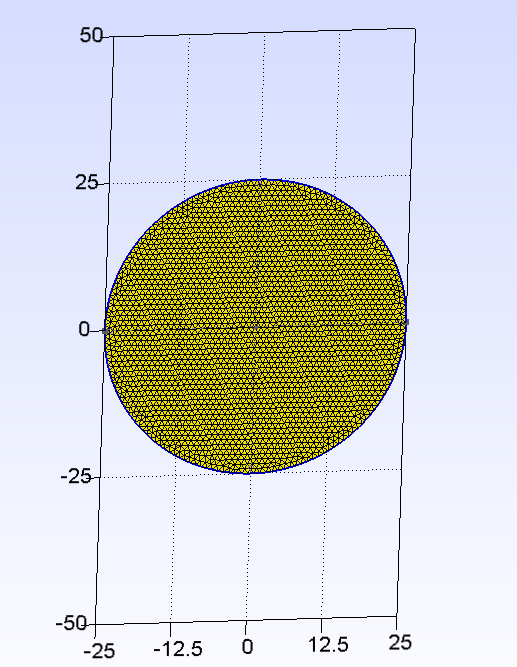
This process can be adapted to create various polygons.

* In a similar way:

For a circle, position one point at the center and two additional points either vertically or horizontally aligned. Use "circle" instead of "line" in the script. Note that the center point should be placed second in the order.



Executing this process and selecting the 2D view will display the desired shape –



* Saving:

The shape can be saved in both ‘geo’ and ‘msh’ file formats.

# Integrating Gmsh with Toast++ (example of DOT scan simulation)

The Following example is using Toast++ 2.0.2 MATLAB toolbox and Gmsh 4.12.0. Note that MSH file should be saved in an older revision (2.x) as Toast++ last update was in 2017 while Gmsh is updated monthly (But thankfully has a backward compatibility).

* Initial Setup:

Start by clearing all previous MATLAB workspace variables and closing any open figures. This ensures a fresh start for your simulation.

clear all

close all

* Open Mesh:

Load a pre-defined mesh named using Toast++ built-in function.

mesh = toastMesh('circle.msh','gmsh');

* Setting Background Parameters:   
    
  Define the background optical properties for your simulation. This includes the absorption coefficient (mua\_bkg), scattering coefficient (mus\_bkg), and refractive index (ref\_bkg). In this example we randomize ‘mua’ and ‘mus’ within defined ranges for each node in the mesh.

mua\_bkg = 0.01;

mus\_bkg = 1.0;

ref\_bkg = 1.4;

nnd = mesh.NodeCount;

mua = 0.099.\*rand(nnd,1) + 10^-3;

mus = rand(nnd,1) + 1;

ref = ones(nnd,1) \* ref\_bkg;

* Visualizing Parameters:   
    
  Display the mesh and the optical properties using MATLAB's figure functions. This will help you visualize ‘mua’, ‘mus’, and the mesh structure with source-detector arrangements.

figure

mesh.Display(mua);

title('mua display')

figure

mesh.Display(mus);

title('mus display')

figure

mesh.Display;

title('mesh and source-detector display')

A circular object with a colorful pattern

Description automatically generated with medium confidence

A circular object with a chart

Description automatically generated with medium confidence

A diagram of a mesh

Description automatically generated

* Source and Detector Positions:   
  Define 16 source and detector positions around the mesh. They should be evenly spaced in a circular arrangement.

rad = 25;

nq = 16;

for i=1:nq

phi\_q = 2\*pi\*(i-1)/nq;

Q(i,:) = rad \* [cos(phi\_q) sin(phi\_q)];

phi\_m = 2\*pi\*(i-0.5)/nq;

M(i,:) = rad \* [cos(phi\_m) sin(phi\_m)];

end

mesh.SetQM(Q,M);

hold on

plot(Q(:,1),Q(:,2),'ro','MarkerFaceColor','r');

plot(M(:,1),M(:,2),'bs','MarkerFaceColor','b');

* Source and Boundary Projection Vectors:

Create the source (‘qvec’) and boundary (‘mvec’) projection vectors using the mesh's ‘Qvec’ and ‘Mvec’ functions.

qvec = mesh.Qvec('Neumann', 'Gaussian', 2);

mvec = mesh.Mvec('Gaussian', 2, 0);

* Solving the FEM Linear System:

Use the ‘dotSysmat’ function to create the system matrix (K) and solve the linear system to obtain the photon distribution (Phi). This is a key step in simulating a Diffuse Optical Tomography (DOT) scan.

K = dotSysmat(mesh,mua,mus,ref,0);

Phi = K\qvec;

Y = mvec.' \* Phi;

* Displaying Results:
  + Sinogram:

Display a sinogram (a graphical representation of the detected light) using ‘imagesc’. This plot shows the log of the light intensity against source and

detector indices.

* + Boundary Profile:   
      
    Plot the logarithmic intensity of light against angular source-detector separation. This provides insights into how light intensity varies across different angles.

\* \*

% Display sinogram

figure;

imagesc(log(Y));

xlabel('source index q');

ylabel('detector index m');

axis equal tight;

colorbar

% Display boundary profile

figure;

hold on;

angle = [360/32:360/16:360];

for i=1:size(Y,2)

ywrap = [Y(i:end,i); Y(1:i-1,i)];

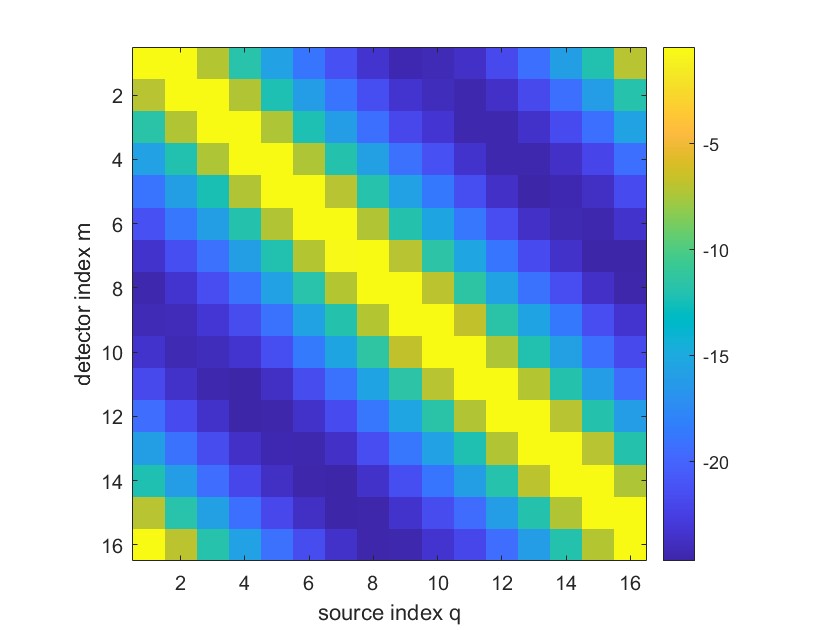
plot(angle,log(ywrap),'o-');

end

axis([0 360 -13 -2]);

xlabel('angular source-detector separation');

ylabel('log intensity');



A graph of a function

Description automatically generated