AToM2Ti*k*Z Documentation

version 1.0.

Vojtěch Neuman
neumavo1@fel.cvut.cz

March 27, 2020

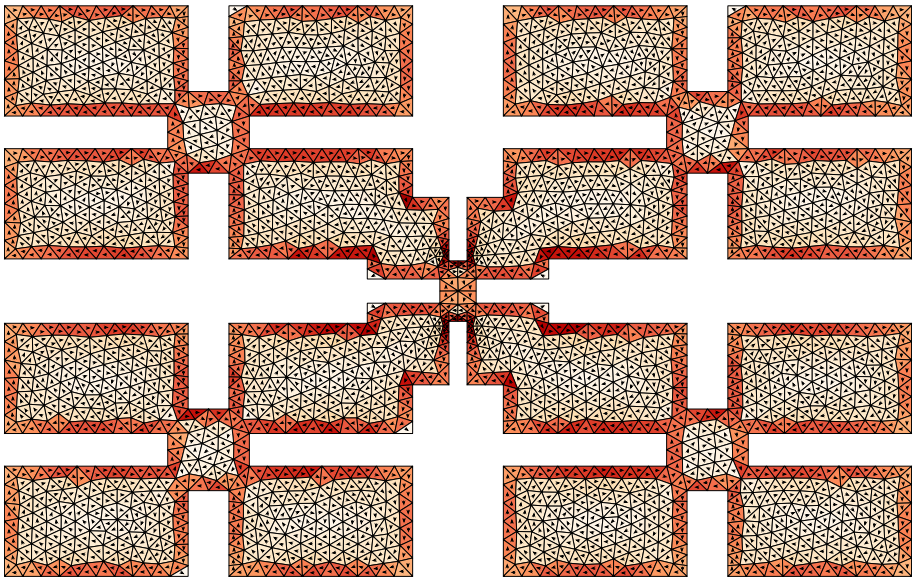


Figure 1: Example of structure converted from MATLAB /AToM plot into LaTeX/Ti*k*Z output.

1 General

This is a package for conversion of MATLAB plots generated by AToM [1] into LaTeX/Ti*k*Z macros. The strategy used to generate the plots is similar as for Matlab2TikZ package and is as follows: AToM/MATLAB figures are searched for all required data and properties and they are converted into corresponding PGF/Ti*k*Z commands written into created `.tex` files. All the outputs/PDFs can be animated via `animate` package (*i.e.*, to change camera position). Package

offers several settings for personification of `.tex` file output of functions. The following AToM data can be converted:

(a) triangulation,

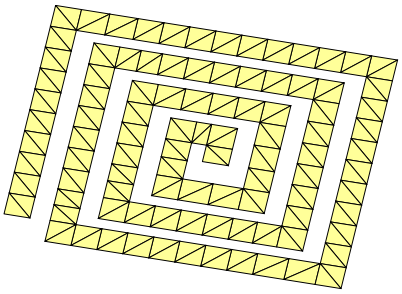
(b) surface charge density,

(c) current density,

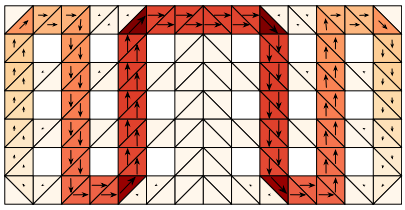
(d) radiation pattern,

(e) radiation pattern cut,

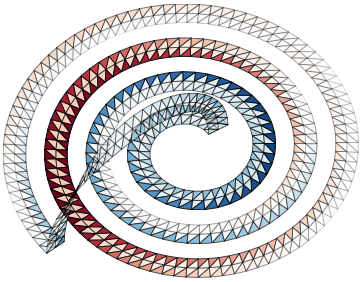
(f) topological sensitivity sample.



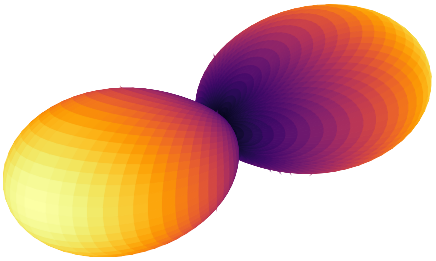
(a) An example of the mesh plot.



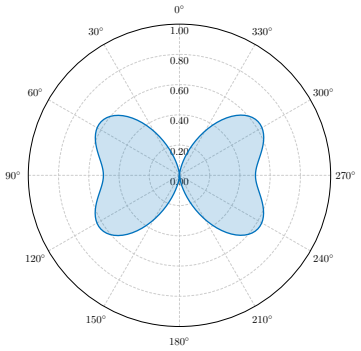
(b) An example of the current plot.



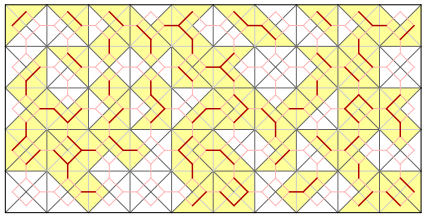
(c) An example of the charge plot.



(d) An example of the far field plot.



(e) An example of the far field cut plot.



(f) An example of the topological sensitivity plot.

For further information, see AToM documentation. In case of `plotMesh()`, functions described latter in this document operates only with option `showTriangle` enabled. The package can be used with properly installed Antenna Toolbox for Matlab (AToM). For advanced functionality, LaTeX with Ti*k*Z packages have to be installed as well. Whole converted picture have layout depicted in Figure 3.

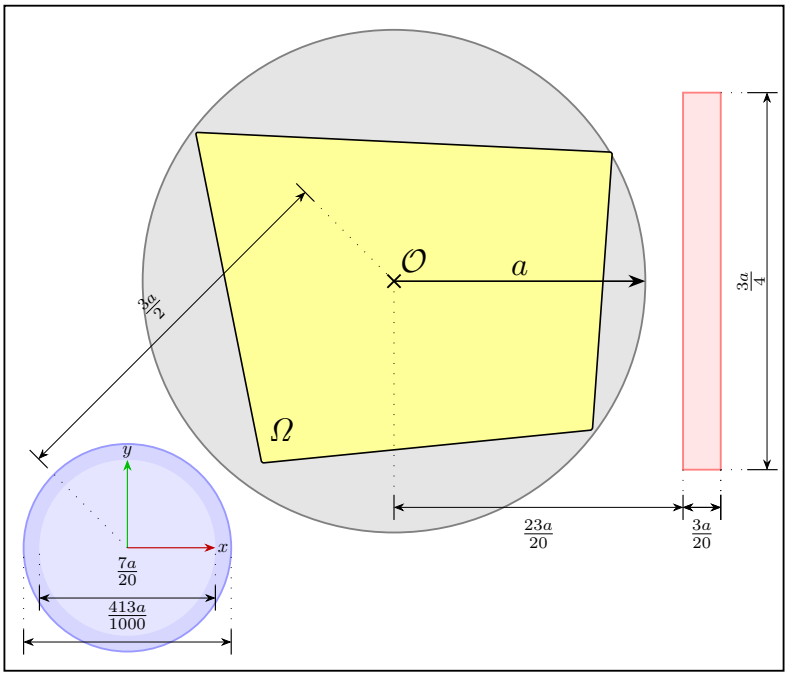


Figure 3: Radius of the smallest circumscribed sphere a is a key parameter which sets sizes of every additional object in picture. Yellow color stands for structure Ω itself. Gray color is used for bounding sphere. Shades of blue represent coordinate cross and coordinate cross labels. Red color stands for colorbar. Symbol $\mathcal{O}$ indicates point (0, 0) in Ti*k*Z figure.

2 Introduction

The following code snippet serves as an example of package usage with AToM. The example begins with mesh generation and evaluation of the impedance matrix **Z**. Project is closed at the end, everything in following is only the post-processing of results.

Listing 1: Generation of trial data to be exported with `AToM2TikZ`.

```
% preparation of workspace
```

```
clc; clear; close all

% start of AToM session
fileName = models.utilities.prepareExampleSession(...
   'ConversionExample2','atom');
atom = Atom.start(false);
atom.createProject(fileName);

% creation of rectangle
atom.selectedProject.geom.addParallelogram(...
   '[0,0,0]','[0,0.5,0]','[1,0,0]','rect');

% start of solvers
atom.selectedProject.physics.setFrequencyList(100e6);
atom.selectedProject.mesh.setGlobalMeshDensity(32);
atom.selectedProject.mesh.getMesh();
atom.selectedProject.solver.MoM2D.solve();

% operators acquisition
mesh = atom.selectedProject.solver.MoM2D.results.mesh;
BF = atom.selectedProject.solver.MoM2D.results.basisFcns;
Z = atom.selectedProject.solver.MoM2D.results.zMat;

% exit
atom.quit;
```

At first, structure discretised into triangles is plotted and then converted into Ti*k*Z macros. See the results of a code below (Listing 2) in Figure 4.

Listing 2: Visualisation of meshed structure and its conversion with `AToM2TikZ`.

```
% visualisation of mesh
options.showTriangles = true;
hndl = results.plotMesh(mesh,'options',options);
hndl.type = 'mesh';

% colors settings
colorMap = results.colors.colorLib.paired12.getColorMap(1);
hndl.triangles.FaceColor = 'flat';
hndl.triangles.CDataMapping = 'scaled';
hndl.triangles.FaceVertexCData = ones(mesh.nTriangles,1).*...
   colorMap(11,:);

% export
export.plot2TikZ(hndl);
```

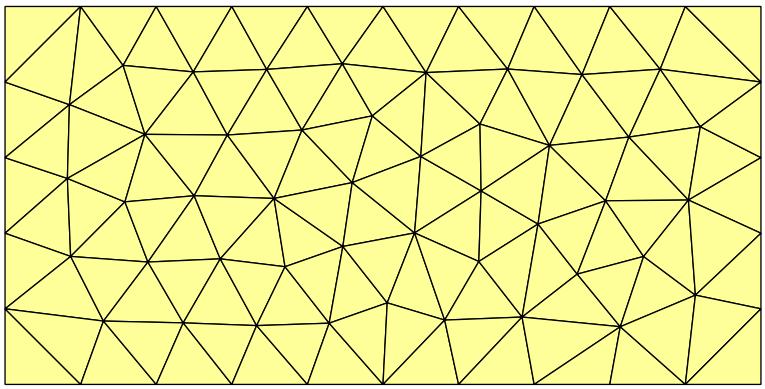


Figure 4: Plate with dimensions $L \times L/2$ made from perfect electric conductor (PEC) discretised into 118 triangles (*i.e.*, 162 basis functions).

Listing 3: Visualisation of meshed structure and its conversion with `AToM2TikZ`.

```
% calculation of characteristic mode
[I,~] = eigs(real(Z.data),imag(Z.data),1,'lm');

% plotting function
hndl = results.plotCurrent(mesh,'basisFcns',BF,...
   'iVec', I,'arrowScale', 'proportional');
hndl.type = 'current';

% export
export.plot2TikZ(hndl);
```

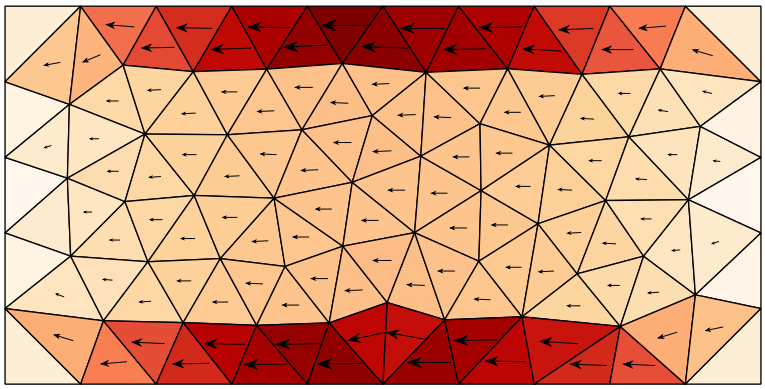


Figure 5: Visualisation of surface current density, real part of time harmonic current depicted (first characteristic mode at $ka = 3.75$, k being wavenumber in vacuum and a being radius of the smallest circumscribing sphere).

Listing 4: Visualisation of meshed structure. and its conversion with `AToM2TikZ`.

```
% plotting function
hndl = results.plotCharge(mesh,'basisFcns',BF,...
   'iVec', I);
hndl.type = 'charge';

% export
export.plot2TikZ(hndl);
```

Surface charge density plot could be obtained in similar way.

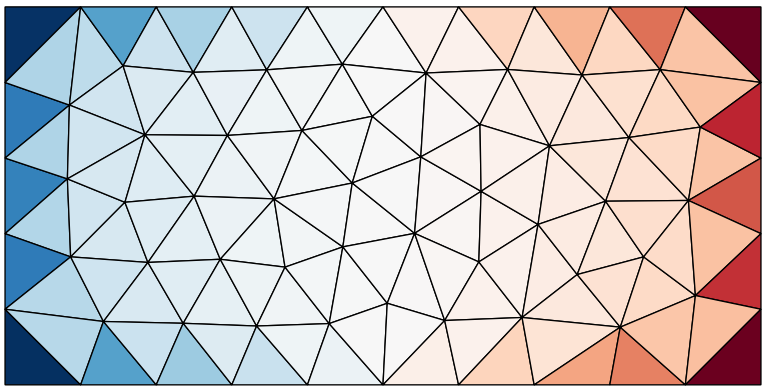


Figure 6: Visualisation of surface charge density.

Although in case of electrical charge, red standing for positive charge and blue for the negative one is a standard colormap, someone might want to change this to his familiar colormap. One possible way how to do this is depicted in the following snippet.

Listing 5: Visualisation of meshed structure and its conversion with `AToM2TikZ`.

```
% change of colormap
hndl.axes.Colormap = results.colors.colorMap.viridis;

% export
export.plot2TikZ(hndl);
```

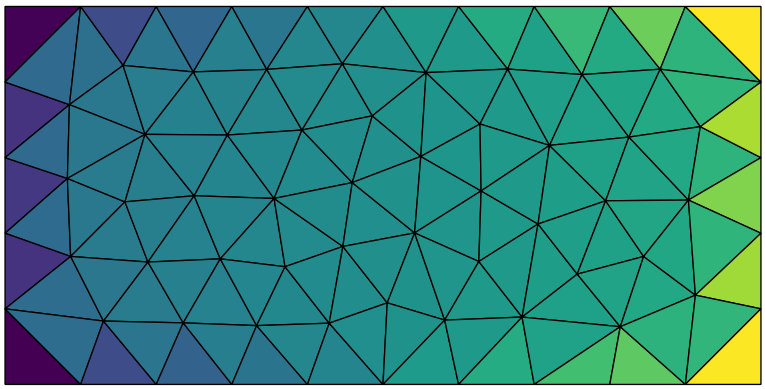


Figure 7: Visualisation of surface charge density with different, though badly, used colormap.

In some cases, user may need to visualise different data over mesh. This could also be done relatively easily.

Listing 6: Visualisation of meshed structure and its conversion with `AToM2TikZ`.

```
% data modification
hndl.triangles.CData = randn(mesh.nTriangles,1);
hndl.axes.CLim = [min(hndl.triangles.CData) max(hndl.triangles.CData)];

% export
export.plot2TikZ(hndl);
```

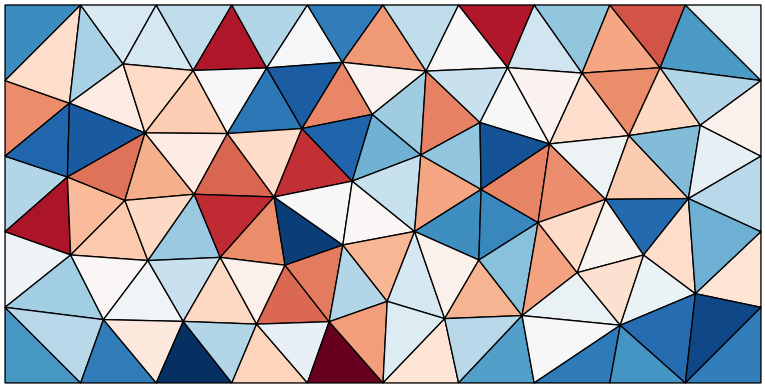


Figure 8: Random data visualised on triangular mesh – a useful test of used colormap.

3 Functions

3.1 `export.plot2TikZ()`

3.1.1 Overview

The advanced users may utilize the low-level functions directly. It is, however, advised to use `plot2TikZ()` in case of any troubles (it is well-optimized).

Perspective projection into x-y plane is used to achieve planar representation of 3D objects. This ensures easy postprocessing of the data.

3.1.2 Arguments

Plot handles are the only mandatory argument. Once specified, the function do everything by itself. If desired, the `plot2TikZ()` returns structures `data` and `settings`. The first one contains data needed for plot recreation and for conversion. It is rather specific to each type of visualising method. The `settings` consists of items in Table 1.

Optional input arguments are divided into two categories by their use. The first category affects `.tex` file output, these arguments are elements of structure `settings`. The other one performs pre-processing of converted plot.

In the first one, the arguments affecting only inputs of functions and the other ones, which modify the `.tex` output file. `settings`. Optional parameters offers possibility of adjusting the `.tex` output. These parameters can be following.

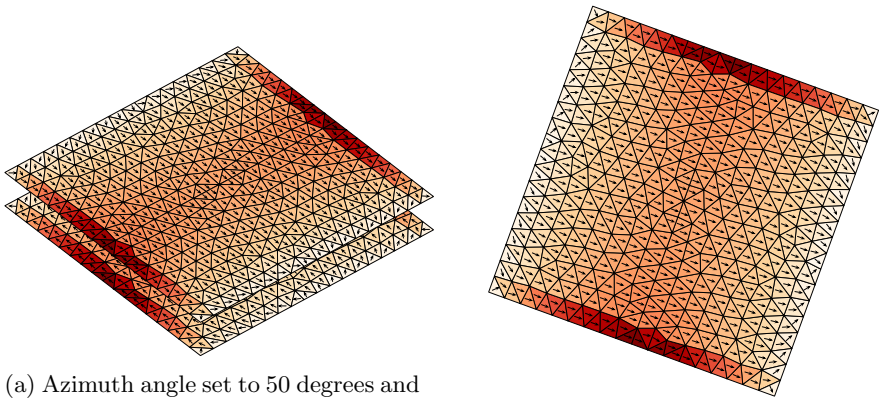
Parameter	Format	Adjustable	Default Value
`ArrowThresh`	double	*yes*	`0.1`
`BoundingSphere`	boolean	*yes*	`false`
`CircumSphere`	boolean	*yes*	`false`
`Colorbar`	boolean	*yes*	`false`
`CompileAutomatically`	boolean	*yes*	`false`
`Compression`	boolean	*yes*	`true`
`CoordinatesCross`	boolean	*yes*	`false`
`FileName`[1]	character string	*yes*	`AToM2TikZ_`
`OpenAfter`	boolean	*no*	`true`
`OriginCoordSystem`	2×1 double	*yes*	`[0;0]`
`Standalone`	boolean	*yes*	`true`
`TransMat3D`	3×3 double	*no*	`eye(3)`
`VectorsCoordSystem`	3×3 double	*no*	`eye(3)`
`TikZ_Corners`	double	*yes*	`0.1`
`TikZ_Opacity`	N×1 double	*yes*	`1`
`TikZ_Scale`	double	*yes*	`5`
`TikZ_Shading`	boolean	*yes*	`true`

Table 1: Table of optional arguments. It is recommended to not modify arguments with *no* in adjustable column. N reads as number of triangle/rectangle faces.

3.1.3 Arguments Description

`Azimuth` and `Elevation`

These arguments have to be set both at once. They are mainly utilised by higher layer function for animations, `animatePlot2TikZ()`. In case of being set, the view angle of former plot are also set for easier comparison of results. Angles are set in degrees. Be aware of restrictions of spherical coordinates system usage.



(a) Azimuth angle set to 50 degrees and elevation angle set to 40 degrees.

(b) Azimuth angle set to 20 degrees and elevation angle set to 90 degrees.

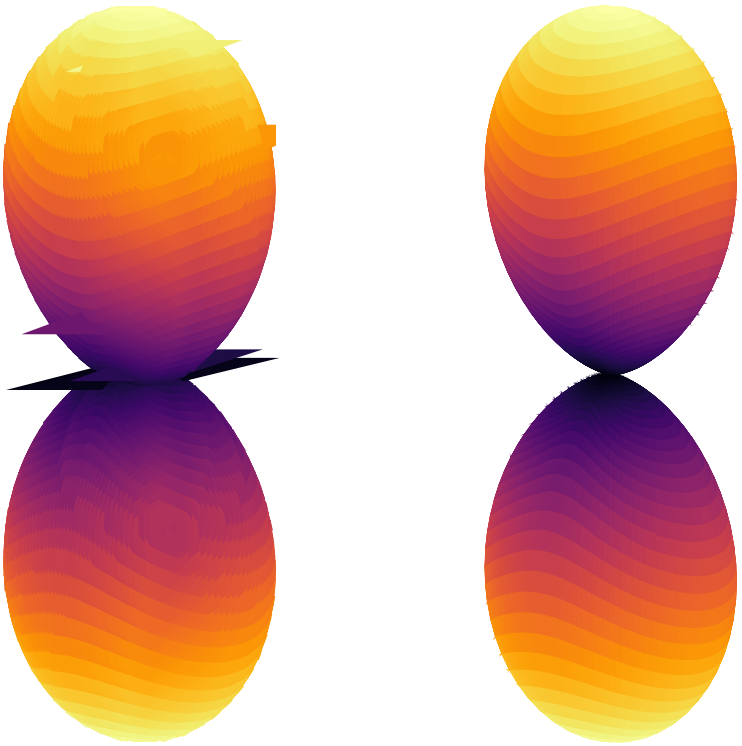
Figure 9: Angles settings example.

Listing 7: Snippet showing usage of `Azimuth` and `Elevation` setting.

```
hndl = results.plotCurrent(mesh,'basisFcns',basisFcns,...
   'iVec',iVec(:,1),'arrowScale','linear');
hndl.type = 'current';

% equivalent to view(50,40)
export.plot2TikZ(hndl,'Azimuth',50,'Elevation',40);

% equivalent to view(20,90)
export.plot2TikZ(hndl,'Azimuth',20,'Elevation',90)
```

`Correction`

`Correction` argument serves as scaling to vertices in plot handle. Ti*k*Z has problem with low resolution. This can result in bad conversion, see (a) in figure below. `Correction` could fix this problem by scaling of vertices.



(a) Faces vertices have low resolution which results in bad conversion.

(b) Fixed conversion by `Correction` factor.

Figure 10: Low resolution example and its correction.

Listing 8: Artificially created situation which represents problem described above.

```
theta = linspace(0, pi, 80);
phi = linspace(0, 2*pi, 128);
farfield = results.calculateFarField(mesh,1e8,'basisFcns',basisFcns,...
   'theta',theta,'phi',phi,'iVec',iVec(:,1));
hndl = results.plotFarField('theta',theta,'phi',phi,...
   'farField',farfield.D);
hndl.type = 'farField';
view(130,15)

% purposely scaled vertices
hndl.farFieldSurface.Vertices = hndl.farFieldSurface.Vertices / 50;

% this picture also has to be changed manually in resulting .tex file
export.plot2TikZ(hndl,'TikZ_Scale',50);

% correction of vertices
export.plot2TikZ(hndl,'Correction',50);
```

`Data` and `Settings`

These two parameters utilise prepared **`data`** and **`settings`** structures. Be aware of following possible combinations of these input arguments together with plot handles

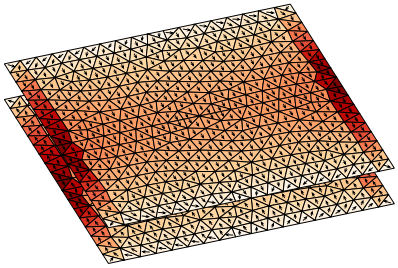
(a) plot handles,

(b) plot handles + **`settings`** structure,

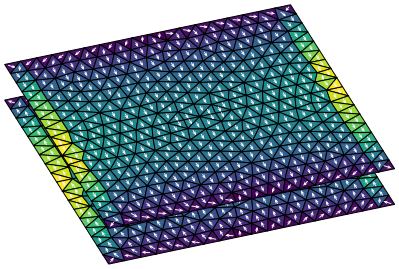
(c) **`data`** structure + **`settings`** structure,

(d) **`data`** structure.

Any other combination will result in error. Observe following figures with example. Converted plots differ only in used colormap and arrow color which is set manually in resulting `.tex` file.



(a) Although **`ArrowThresh`** is set to value 0.5, it does not have effect on resulting picture because of linear scaling.



(b) Recreated plot with proportional scaling. **`ArrowThresh`** is set to value 0.5.

Listing 9: Different possibilities of plot conversion.

```
hndl = results.plotCurrent(mesh,'basisFcns',basisFcns,...
   'iVec',iVec(:,1),'arrowScale','linear');
hndl.type = 'current';
view(70,35)

% export with generation of data and settings structures
[data,settings] = export.plot2TikZ(hndl);

% colormap changed
data.cMap = results.colors.colorMap.viridis;

% TikZ
export.plot2TikZ([],'data',data,'settings',settings);
```

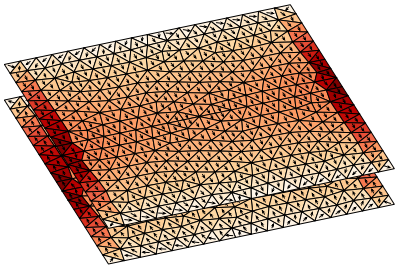
`ArrowThresh`

This parameter is used to set arrows threshold, under which will be arrows laying on triangles with small amplitude omitted. **`ArrowThresh`** takes as its

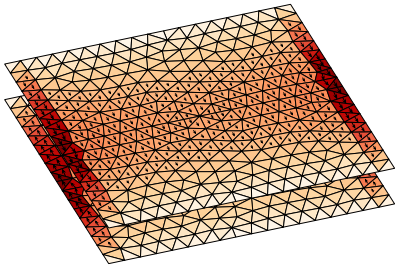
value scalar $\xi \in [0, 1)$ with $\xi = 0.5$ being a default value. The simplified logic behind this process plots n-th arrow only if the following relation holds:

$$\frac{\|\boldsymbol{J}_n\|}{\max_n \|\boldsymbol{J}_n\|} > \xi. \tag{1}$$

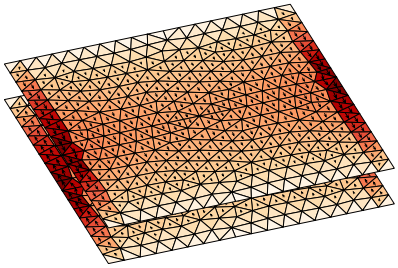
The parameter respects settings `ArrowScale` possibilities of function `plotCurrent()` in AToM [1]. In case of scaling being linear, all arrows will be drawn.



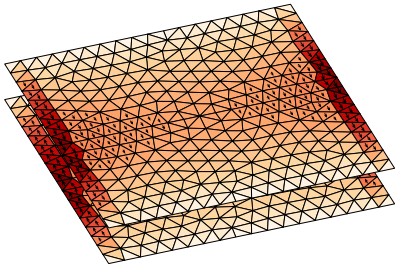
(a) Although `ArrowThresh` is set to value 0.5, it does not have effect on resulting picture because of linear scaling.



(b) Recreated plot with proportional scaling. `ArrowThresh` is set to value 0.5.



(c) `ArrowThresh` is set to value 0.25.



(d) `ArrowThresh` is set to value 0.75.

Figure 12: Effect of `ArrowScale` in different scenarios.

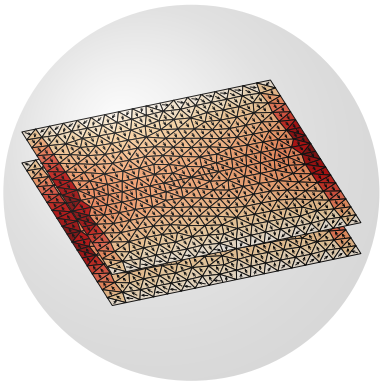
Listing 10: See previous figure for results of this snippet.

```
% case (a)
hndl = results.plotCurrent(mesh,'basisFcns',BF,...
   'iVec', I,'arrowScale','linear');
hndl.type = 'current';
view(70,35);

export.plot2TikZ(hndl,'ArrowThresh',0.5)

% case b), c) and d)
hndl = results.plotCurrent(mesh,'basisFcns',BF,...
```

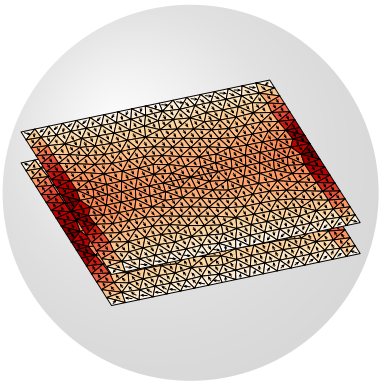
```
   'iVec', I,'arrowScale','proportional');
hndl.type = 'current';
view(70,35);

export.plot2TikZ(hndl,'ArrowThresh',0.5)
export.plot2TikZ(hndl,'ArrowThresh',0.25)
export.plot2TikZ(hndl,'ArrowThresh',0.75)
```

`BoundingSphere`

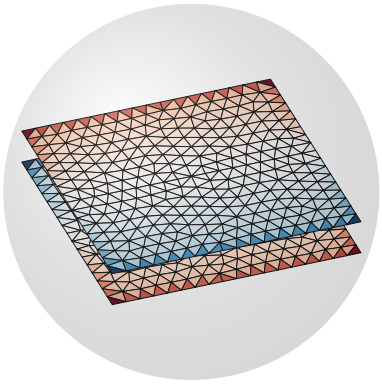
`BoundingSphere` draws grey opaque circumscribing sphere over geometrical structure. Boolean value with `false` being default settings. Grey sphere is implicitly drawn after the geometrical structure although this can be manually reverted in resulting `.tex` file, see Figure 13 for comparison.



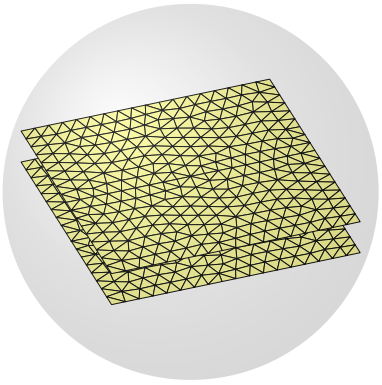
(a) Grey sphere is drawn after geometrical structure.



(b) Grey sphere is drawn before geometrical structure.



(c) `BoundingSphere` applied for charge plot.



(d) `BoundingSphere` applied for mesh plot.

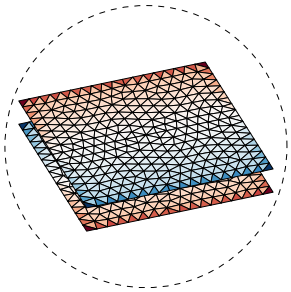
Figure 13: `BoundingSphere` effect examples.

Listing 11: Code used for generation of pictures above.

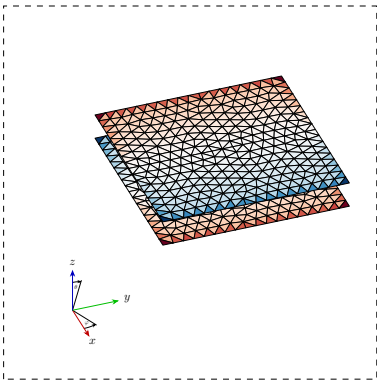
```
hndl = results.plotCurrent(mesh,'basisFcns',basisFcns,...
   'iVec', iVec(:,1),'arrowScale','proportional');
hndl.type = 'current';
view(70,35)

% same picture for a), b)
export.plot2TikZ(hndl,'BoundingSphere',true)

hndl = results.plotCharge(mesh,'basisFcns',basisFcns,...
   'iVec', iVec(:,1));
hndl.type = 'charge';
view(70,35);

% picture c)
export.plot2TikZ(hndl,'BoundingSphere',true)

options.showTriangles = true;
hndl = results.plotMesh(mesh,'options',options);
hndl.type = 'mesh';
colorMap = results.colors.colorLib.paired12.getColorMap(1);

hndl.triangles.FaceColor = 'flat';
hndl.triangles.CDataMapping = 'scaled';
hndl.triangles.FaceVertexCData = ones(mesh.nTriangles,1).*colorMap(11,:);
view(70,35);

% picture d)
export.plot2TikZ(hndl,'BoundingSphere',true)
```

`CircumSphere`

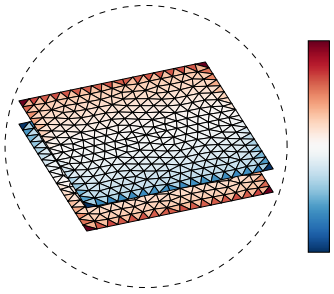
`CircumSphere` adds white invisible space over converted plot. This argument is primarily used for ensuring smooth animation. Boolean value with `false` being default settings.



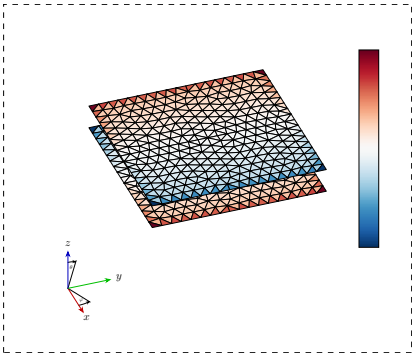
(a) White space has shape of circumscribing sphere.



(b) White space is rectangle optimised for size of mesh and coordinates cross.



(c) White space for this situation is not prepared at this moment.



(d) White space for this situation is still in development phase.

Figure 14: White space borders illustration. This argument is still in development, see picture (c), and will be done in next release.

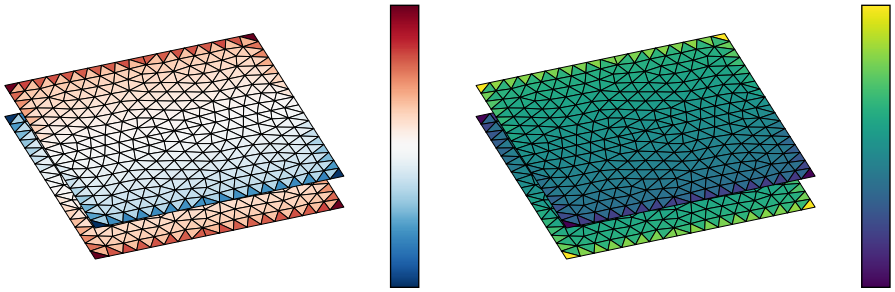
Listing 12: See previous figure for results of this snippet.

```
hndl = results.plotCharge(mesh,'basisFcns',basisFcns,...
   iVec',iVec(:,1));
hndl.type = 'charge';
view(70,35);

% basic conversion
export.plot2TikZ(hndl,'CircumSphere',true);

% conversion with added coordinates
export.plot2TikZ(hndl,'CircumSphere',true,...
   'CoordinatesCross',true);

% conversion with added colorbar
```

```
export.plot2TikZ(hndl,'CircumSphere',true,...
   'Colorbar',true);

% conversion with added colorbar and coordinates
export.plot2TikZ(hndl,'CircumSphere',true,...
   'Colorbar',true,'CoordinatesCross',true);
```

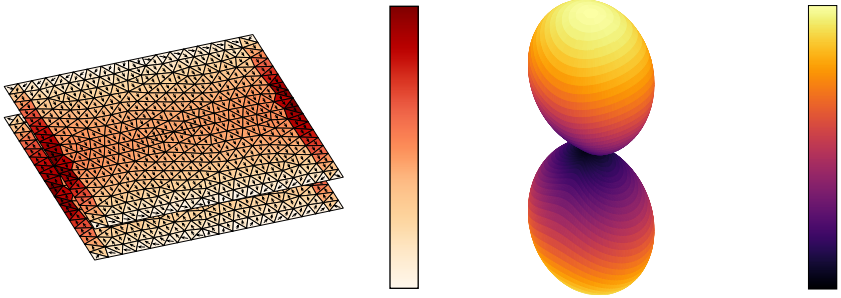
`Colorbar`

`Colobar` draws in resulting picture colorbar with respect to used colormap. This argument is in development phase and its current form will be changed in future release. Boolean type with default value set as `false`.



(a) Charge plot.

(b) Charge plot with changed colormap



(c) Current plot.

(d) Far fied plot.

Figure 15: Different colorbars for different types of colormap.

Listing 13: Different colorbar with respect to used plots and colormaps.

```
hndl = results.plotCharge(mesh,'basisFcns',basisFcns,...
   'iVec',iVec(:,1));
hndl.type = 'charge';
view(70,35);

% charge plot
```

```
export.plot2TikZ(hndl,'Colorbar',true);

% charge plot with changed colormap
hndl.axes.Colormap = results.colors.colorMap.viridis;
export.plot2TikZ(hndl,'Colorbar',true);

hndl = results.plotCurrent(mesh,'basisFcns',basisFcns,...
   'iVec',iVec(:,1));
hndl.type = 'current';
view(70,35);

% current plot
export.plot2TikZ(hndl,'Colorbar',true);

theta = linspace(0, pi, 80);
phi = linspace(0, 2*pi, 128);
farfield = results.calculateFarField(mesh,1e8,
   'basisFcns',basisFcns,'theta',theta,
   'phi',phi,'iVec',iVec(:,1));
hndl = results.plotFarField('theta',theta,'phi',phi,...
   'farField',farfield.D);
hndl.type = 'farField';
view(130,45)

% far field plot
export.plot2TikZ(hndl,'Colorbar',true);
```

`CompileAutomatically`

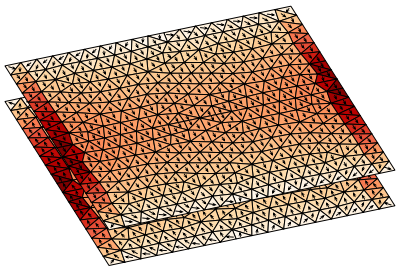
`CompileAutomatically` compiles resulting `.tex` file. Prepare for unforeseen consequences due to use of controversial MATLAB function `evalf()`. Use of this argument requires pdfLaTeX being installed. Actual version supports only Microsoft Windows® system.

Listing 14: Demonstration of `CompileAutomatically` usage.

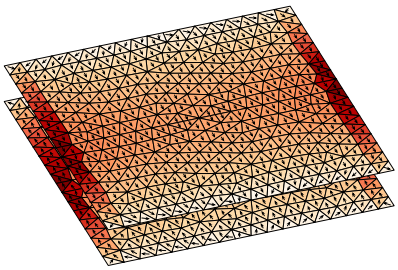
```
hndl = results.plotCurrent(mesh,'basisFcns',basisFcns,...
   'iVec',iVec(:,1),'arrowScale','linear');
hndl.type = 'current';

export.plot2TikZ(hndl,'CompileAutomatically',true);
```

`Compression`

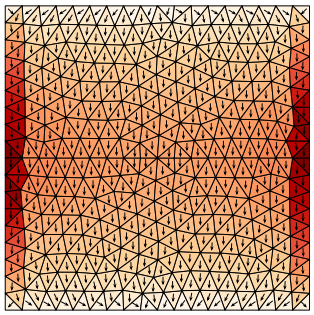
`Compression` turns on/off visible patches compression This can significantly shorten compilation time and memory usage of afterwards `.pdf` file. Compression is usable only with `TikZ_Opacity` equal to one (scalar). In some cases, it is convenient to turn off compression. For example, if object is planar with single layer. Takes values of true or false. See [2] to observe details of compression algorithms.



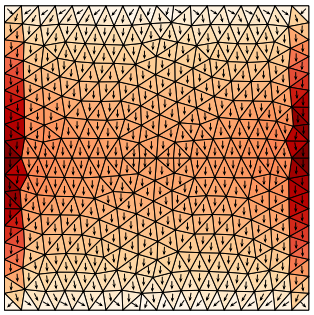
(a) 693 triangles, size of file is 86 KB.



(b) 1048 triangles, size of file is 127 KB.



(c) 524 triangles, size of file is 67 KB.



(d) 1048 triangles, size of file is 127 KB.

Figure 16: Figure presents two different angles settings and compression effect. Pictures on left side are same as the one at right side but without indirectly visible triangles.

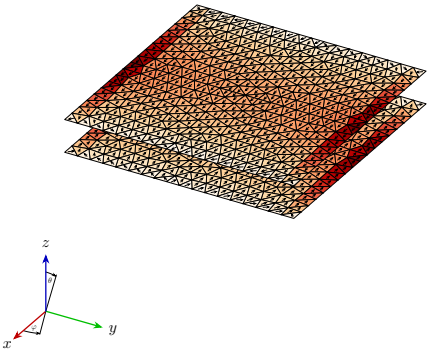
Listing 15: Code for generation of pictures above.

```
hndl = results.plotCurrent(mesh,'basisFcns',basisFcns,...
   'iVec', iVec(:,1),'arrowScale','linear');
hndl.type = 'current';

% first row in figure
view(70,35);
export.plot2TikZ(hndl,'Compression',true);
export.plot2TikZ(hndl,'Compression',false);

% second row in figure
view(90,90);
export.plot2TikZ(hndl,'Compression',true);
export.plot2TikZ(hndl,'Compression',false);
```

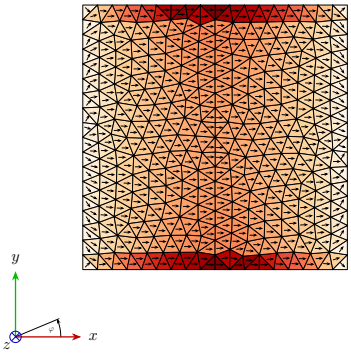
`CoordinatesCross`

`CoordinatesCross` sets coordinates cross in left bottom corner. Coordinates cross is set with respect to plot view angles. In resulting `.tex` file, strings

representing direction 'in screen' and 'out of screen' are prepared. Mentioned direction can be then changed manually replaced by strings, see following figure.



(a) Azimuth is 120 degrees and elevation is 30 degrees.



(b) Azimuth is 0 degrees and elevation is 90 degrees. Coordinates cross is modified in resulting `.tex` file.

Listing 16: Code for generation of pictures above.

```
hndl = results.plotCurrent(mesh,'basisFcns',basisFcns,...
   'iVec',iVec(:,1),'arrowScale','linear');
hndl.type = 'current';

% coordinates cross is drawn without modifications needed
view(120,30);
export.plot2TikZ(hndl,'CoordinatesCross',true);

% coordinates cross have to be modified in resulting file
view(0,90);
export.plot2TikZ(hndl,'CoordinatesCross',true);
```

`FileName`

`FileName` allows to specify resulting name of `.tex` file. In case of `animatePlot2TikZ()`, this specifies name of folder in which particular `.pdf` files are saved. These files are differentiated by concatenated number at the end of file name string. The function does not control possible overwriting of already existing file, so be aware of choose of file name. Default value is `AToM2TikZ` with concatenated string created from return value of MATLAB `now()` function.

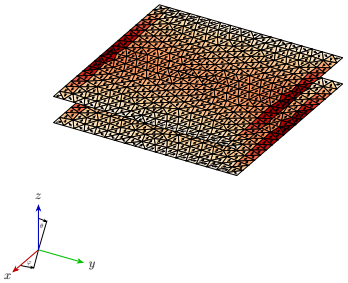
Listing 17: Change of resulting file name example.

```
hndl = results.plotCurrent(mesh,'basisFcns',basisFcns,...
   'iVec',iVec(:,1),'arrowScale','linear');
hndl.type = 'current';
```

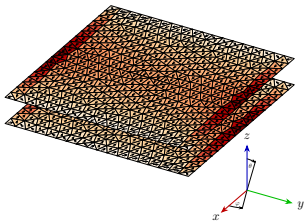
```
export.plot2TikZ(hndl,'FileName','current_plot');
```

OriginCoordSystem

`OriginCoordSystem` is complementary argument to `CoordinatesCross` and offers possibility to set offset of beginning of the coordinates cross. See following figure.



(a) Coordinates cross at its default position with respect to layout in Figure 3.



(b) Coordinates cross has offset $x = 0.85$ and $y = 0.2$ in TikZ picture.

Listing 18: Coordinates cross positioning.

```
hndl = results.plotCurrent(mesh,'basisFcns',basisFcns,...
   'iVec', iVec(:,1));
hndl.type = 'current';
view(120,30);

% default position
export.plot2TikZ(hndl,'CoordinatesCross',true);

% offset position
export.plot2TikZ(hndl,'CoordinatesCross',true,...
   'OriginCoordSystem',[0.85;0.2]);
```

Standalone

`Standalone` creates independent `.tex` file. Otherwise, the `tikzpicture` environment is used. The header of standalone file is shown in following code snippet.

```
\documentclass[tikz]{standalone}
\usepackage{amsmath}
\usepackage{tikz}
\usetikzlibrary{arrows.meta}
```

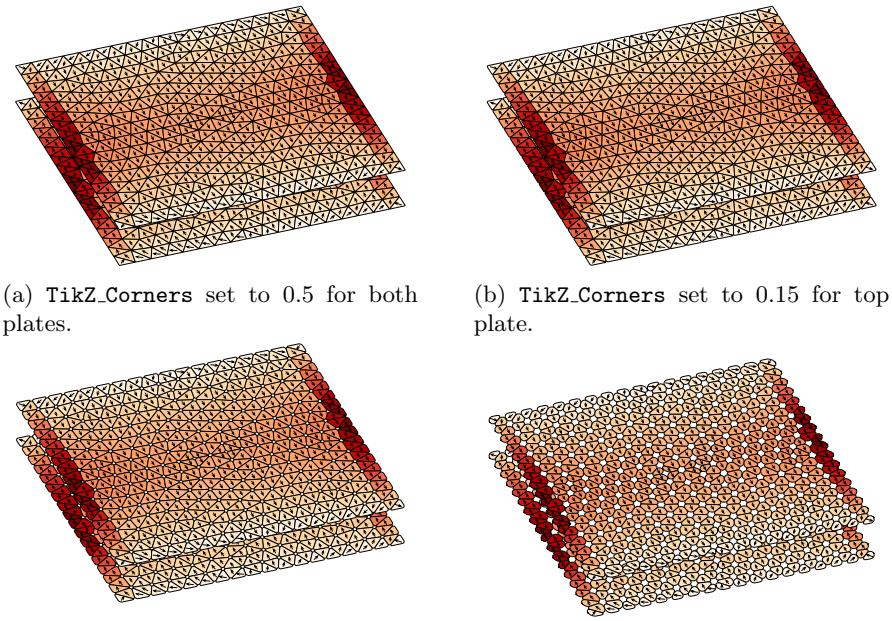
```
% Converted by AToM2TikZ
\begin{document}
\begin{tikzpicture}[scale=5.00]

% Converted plot

\end{tikzpicture}
\end{document}
```

Listing 19: Change of standalone settings.

```
hndl = results.plotCurrent(mesh,'basisFcns',basisFcns,...
   'iVec',iVec(:,1),'arrowScale','linear');
hndl.type = 'current';

export.plot2TikZ(hndl,'Standalone',true);
```

`TikZ_Corners`

`TikZ_Corners` sets rounding of triangles corners. This can produce interesting effect of highlighted vertices.



(a) `TikZ_Corners` set to 0.5 for both plates.

(b) `TikZ_Corners` set to 0.15 for top plate.

(c) `TikZ_Corners` is proportional to absolute value of charge density.

(d) `TikZ_Corners` set to 0.5.

Figure 19: Effect of `TikZ_Corners` in different scenarios.

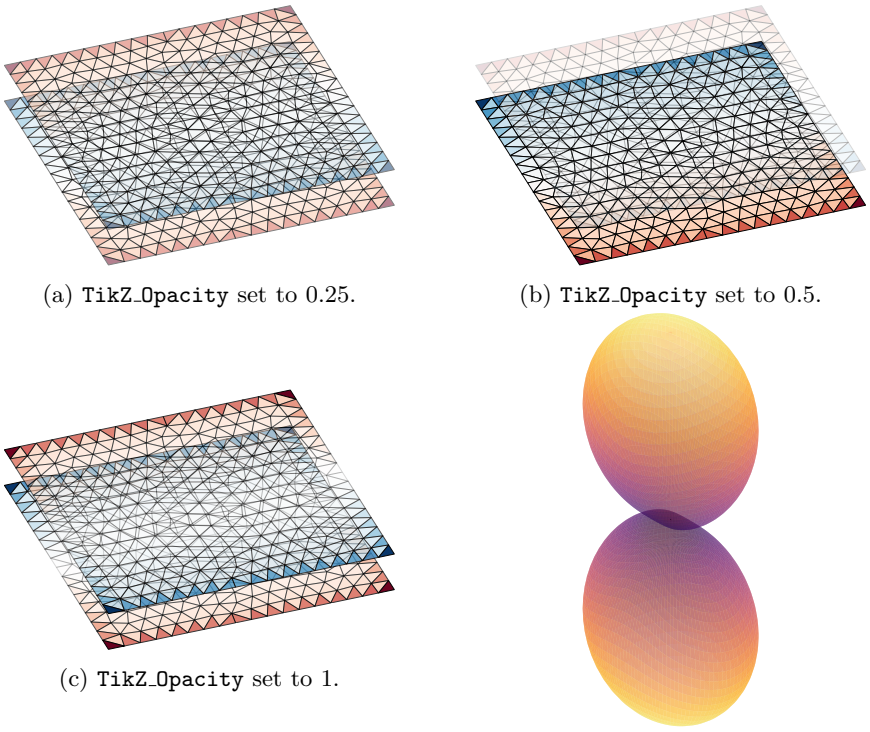
Listing 20: See previous figure for results of this snippet.

```
% charge plot
hndl = results.plotCharge(mesh,'basisFcns',basisFcns,...
   'iVec',iVec(:,1));
hndl.type = 'charge';
view(70,35);

% opacity set as scalar
export.plot2TikZ(hndl,'TikZ_Opacity',0.5);

% opacity set as vector
opacity = ones(mesh.nTriangles,1);
opacity(mesh.triangleCentroids(:,3) > 0.05) = 0.15;
export.plot2TikZ(hndl,'TikZ_Opacity',opacity);

% opacity set as vector
opacity = hndl.triangles.FaceVertexCData;
opacity = sqrt(abs(opacity)/max(abs(opacity)));
export.plot2TikZ(hndl,'TikZ_Opacity',opacity);

% far field calculation
theta = linspace(0,pi,80);
phi = linspace(0,2*pi,128);
farfield = results.calculateFarField(mesh,1e8,...
   'basisFcns',basisFcns,'theta',theta,...
```

```
   'phi',phi,'iVec',iVec(:,1));
hndl = results.plotFarField('theta',theta,'phi',phi,...
   'farField',farfield.D);
hndl.type = 'farField';
view(130,45)

% opacity set as scalar
export.plot2TikZ(hndl,'TikZ_Opacity',0.5);
```

`TikZ_Opacity`

`TikZ_Opacity` allows to set opaqueness of whole geometrical object, if set as scalar, or to specify particular opacity to particular triangle. Setting value must be specified as scalar or as vector of size equal to number of patches. Values also have to be in interval $(0, 1]$.



(a) `TikZ_Opacity` set to 0.25.

(b) `TikZ_Opacity` set to 0.5.

(c) `TikZ_Opacity` set to 1.

(d) `TikZ_Opacity` set to 2.

Figure 20: Effect of `TikZ_Opacity` in different scenarios.

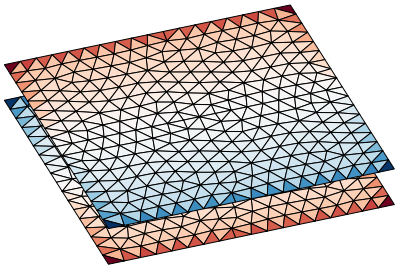
Listing 21: Examples of opacity possibilities.

```
% charge plot
```

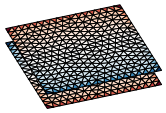
```
hndl = results.plotCharge(mesh,'basisFcns',basisFcns,...
   'iVec',iVec(:,1));
hndl.type = 'charge';
view(70,35);

% opacity set as scalar
export.plot2TikZ(hndl,'TikZ_Opacity',0.5);

% opacity set as vector
opacity = ones(mesh.nTriangles,1);
opacity(mesh.triangleCentroids(:,3) > 0.05) = 0.15;
export.plot2TikZ(hndl,'TikZ_Opacity',opacity);

% opacity set as vector
opacity = hndl.triangles.FaceVertexCData;
opacity = sqrt(abs(opacity)/max(abs(opacity)));
export.plot2TikZ(hndl,'TikZ_Opacity',opacity);

% far field calculation
theta = linspace(0,pi,80);
phi = linspace(0,2*pi,128);
farfield = results.calculateFarField(mesh,1e8,...
   'basisFcns',basisFcns,'theta',theta,...
   'phi',phi,'iVec',iVec(:,1));
hndl = results.plotFarField('theta',theta,'phi',phi,...
   'farField',farfield.D);
hndl.type = 'farField';
view(130,45)

% opacity set as scalar
export.plot2TikZ(hndl,'TikZ_Opacity',0.5);
```

`TikZ_Scale`

`TikZ_Scale` set scaling in `tikzpicture` environment.



(a) Scaling set to 5 (default).



(b) Scaling set to 2.

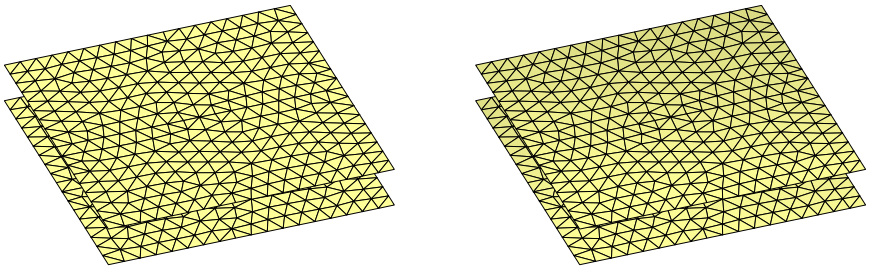
Figure 21: Comparison of `TikZ_Scale` effect.

Listing 22: Usage of `TikZ_Scale` effect.

```
hndl = results.plotCharge(mesh,'basisFcns',basisFcns,...
   'iVec',iVec(:,1));
hndl.type = 'charge';
view(70,35);

% value 5 is default
export.plot2TikZ(hndl,'TikZ_Scale',5);

export.plot2TikZ(hndl,'TikZ_Scale',2);
```

TikZ_Shading

TikZ_Shading turns on shading of meshed structure for better perspective. This argument is exclusive for mesh structures plots. Compare pictures on following figure.



(a) Mesh structure without shading. (b) Mesh structure with shading.

Figure 22: Comparison of **TikZ_Shading** effect.

Listing 23: Observe figure above for results of this code snippet.

```
options.showTriangles = true;
hndl = results.plotMesh(mesh,'options',options);
hndl.type = 'mesh';
colorMap = results.colors.colorLib.paired12.getColorMap(1);

hndl.triangles.FaceColor = 'flat';
hndl.triangles.CDataMapping = 'scaled';
hndl.triangles.FaceVertexCData = ones(mesh.nTriangles,1).*colorMap(11,:);
view(70,35);

% without shading
export.plot2TikZ(hndl,'TikZ_Shading',false);

% with shading
export.plot2TikZ(hndl,'TikZ_Shading',true);
```

3.2 `export.animatePlot2TikZ()`

3.2.1 Overview

`animatePlot2TikZ` is special functions which produces animated plots via TikZ macros with usage of
`usepackageanimate`. Be aware of quality dependence on number of created `.pdf` files.

3.2.2 Arguments

The arguments of `animatePlot2TikZ()` are the handle of plot together with vectors containing angle for azimuth and elevation. The latter ones can be either both vectors or one scalar value with second one as vector. The optional arguments are pretty much same as in `plot2TikZ`. However, parameters `'CircumSphere'` and `'OriginCoordSystem'` are automatically set to their default values to assure smooth animation.

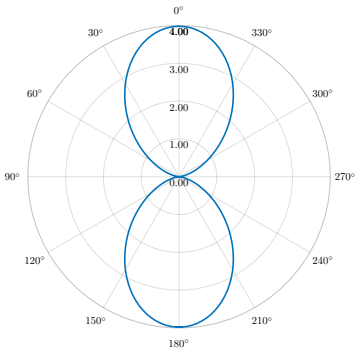
3.3 `export.plotLine2TikZ()`

3.3.1 Overview

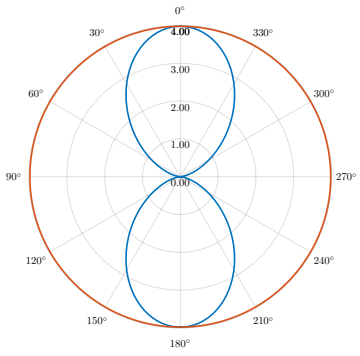
This function serves as a interface between AToM and TikZ for planar plots. `plotFarFieldCut()` is the only one which is supported at this time. This is done for potential backward compatibility. Unlike the `plotFarFieldCut()` it is implicitly rotated with zero at top. Additional functionalities are present.

3.3.2 Arguments

Handles of far field cut plots are the only mandatory arguments. Converted plot could contain more cut plots. This can be achieved by using vector of plot handles instead of one particular handle.



(a) Original coordinates system.



(b) Enhanced coordinates system.

Listing 24: Demonstration of `CompileAutomatically` usage.

```
theta = linspace(0, pi, 90);
phi = linspace(0, 2*pi, 180);
farfield = results.calculateFarField(mesh,1e8,'basisFcns',basisFcns,...
   'theta',theta,'phi',phi,'iVec',iVec(:,2));

% Preparation of plotting.
phiCut = pi/2;
hndl1 = results.plotFarFieldCut('theta',theta,'phi',phi, ...
   'farField',farfield.D,'phiCut',phiCut);
thetaCut = 0;
hndl2 = results.plotFarFieldCut('theta',theta,'phi',phi, ...
   'farField',farfield.D,'thetaCut',thetaCut);

% Handles vector.
hndl = [hndl1; hndl2];

% One picture
export.plotLine2TikZ(hndl1);

% Two pictures.
export.plotLine2TikZ(hndl);
```

Optional arguments serves for slight modification of `.pdf` output, see Table 2.

Parameter	Format	Adjustable	Default
`CompileAutomatically`	boolean	*yes*	`false`
`Coordinates`	boolean	*yes*	`true`
`FileName`[2]	character string	*yes*	`AToM2TikZ_`
`Filling`	boolean	*yes*	`false`
`OpenAfter`	boolean	*no*	`true`
`Standalone`	boolean	*yes*	`true`

Table 2: Table of optional arguments. It is recommended to not modify arguments with *no* in adjustable column.

3.3.3 Arguments Description

`CompileAutomatically`

`CompileAutomatically` compiles resulting `.tex` file. Prepare for unforeseen consequences due to use of controversial MATLAB function `evalf()`. Use of this argument requires pdfLaTeX being installed. Actual version supports only Microsoft Windows® system.

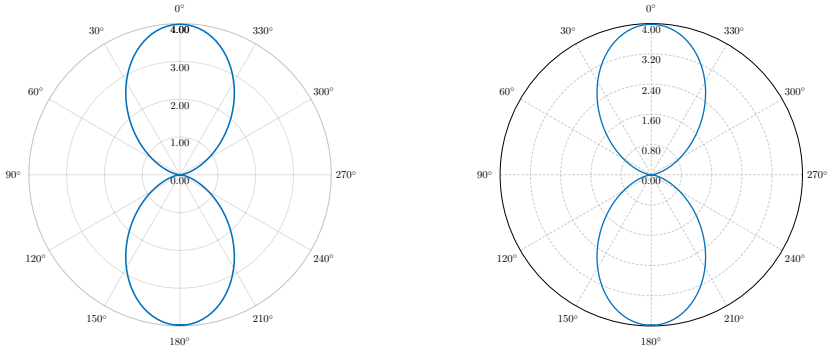
Listing 25: Demonstration of `CompileAutomatically` usage.

```
hndl = results.plotCurrent(mesh,'basisFcns',basisFcns,...
   'iVec',iVec(:,1),'arrowScale','linear');
hndl.type = 'current';
```

```
export.plot2TikZ(hndl,'CompileAutomatically',true);
```

Coordinates

`Coordinates` turns on enhanced coordinates system instead of coordinates system taken from MATLAB figure.



(a) Original coordinates system.

(b) Enhanced coordinates system.

Listing 26: Observe figure above for results of this snippet.

```
% calculation of far field
theta = linspace(0, pi, 90);
phi = linspace(0, 2*pi, 180);
farfield = results.calculateFarField(mesh,1e8,'basisFcns',basisFcns,...
   'theta',theta,'phi',phi,'iVec',iVec(:,1));

% preparation of plot.
phiCut = pi/2;
hndl = results.plotFarFieldCut('theta', theta, 'phi', phi, ...
   'farField', farfield.D, 'phiCut', phiCut);

% with default coordinates
export.plotLine2TikZ(hndl,'Coordinates',false);

% with enhanced coordinates
export.plotLine2TikZ(hndl,'Coordinates',true);
```

FileName

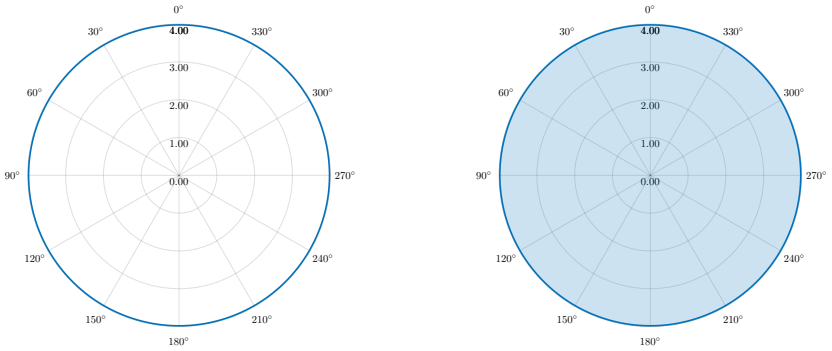
`FileName` allows to specify resulting name of `.tex` file. In case of `animatePlot2TikZ()`, this specifies name of folder in which particular `.pdf` files are saved. These files are differentiated by concatenated number at the end of file name string. The function does not control possible overwriting of already existing file, so be aware of choose of file name. Default value is `AToM2TikZ` with concatenated string created from return value of MATLAB `now()` function.

Listing 27: Change of resulting file name example

```
hndl = results.plotCurrent(mesh,'basisFcns',basisFcns,...
   'iVec',iVec(:,1),'arrowScale','linear');
hndl.type = 'current';

export.plot2TikZ(hndl,'FileName','current_plot');
```

`Filling`

`Filling` turns on filling of inner space bounded by radiation pattern. Color used for line is used also although with opaque effect.



(a) Without filling. (b) With filling.

Figure 25: Comparison of effect of `filling`.

Listing 28: Observe figure above for results of this snippet.

```
% calculation of far field
theta = linspace(0, pi, 90);
phi = linspace(0, 2*pi, 180);
farfield = results.calculateFarField(mesh,1e8,'basisFcns',basisFcns,...
   'theta',theta,'phi',phi,'iVec',iVec(:,2));

% preparation of plot.
thetaCut = 0;
hndl = results.plotFarFieldCut('theta',theta,'phi',phi,...
   'farField',farfield.D,'thetaCut',thetaCut);

% without filling
export.plotLine2TikZ(hndl,'Filling',false);

% with filling
export.plotLine2TikZ(hndl,'Filling',true);
```

`Standalone`

`Standalone` creates independent `.tex` file. Otherwise, the `tikzpicture` environment is used. The header of standalone file is shown in following code snippet.

```
\documentclass[tikz]{standalone}
\usepackage{amsmath}
\usepackage{tikz}
\usetikzlibrary{arrows.meta}

% Converted by AToM2TikZ
\begin{document}
\begin{tikzpicture}[scale=5.00]

% Converted plot

\end{tikzpicture}
\end{document}
```

Listing 29: Change of standalone settings.

```
hndl = results.plotCurrent(mesh,'basisFcns',basisFcns,...
   'iVec',iVec(:,1),'arrowScale','linear');
hndl.type = 'current';

export.plot2TikZ(hndl,'Standalone',true);
```

3.4 `export.advanced.current2TikZ()`

This method is implemented for conversion of `plotCurrent()` resulting plot. Input arguments are structures `settings`, see Table 1, and `data` described below. Both these structures have to contain all data mentioned in those tables. It is recommended to not directly utilise this function and using `plot2TikZ()` instead.

Parameter	Format	Description
`faces_b`	$N_\mathrm{F} \times 3$	connectivity list of triangles vertices
`vert`	$N_\mathrm{V} \times 3$	vertices vector
`cMap`	$N_\mathrm{C} \times 3$	colormap vector
`cVec`	$N_\mathrm{F} \times 3$	vector of faces colors
`arrowsVert`	$N_\mathrm{V} \times 3$	arrow head vertices
`arrows`	$N_\mathrm{F} \times 6$	arrow shaft
`type`	`string`	control parameter for higher-level functions

Table 3: N_V denotes length of vertices vector, N_F stands for faces matrix and N_C is length of colormap vector.

All these data are obtained from plot handles. However, color vector `cVec` is in MATLAB represented as a `double` and it has to be first transformed into integers $c_n \in [1, N_\mathrm{C}]$. This transformation is described below.

$$T = \begin{pmatrix} C_\mathrm{max} & 1 \\ C_\mathrm{min} & 1 \end{pmatrix}, \tag{2}$$

where C_max stands for maximal colorbar value and C_min represents minimal colorbar value. Scaling a and translation b coefficients are obtained by matrix inversion of T

$$\begin{pmatrix} b \\ a \end{pmatrix} = T^{-1} \begin{pmatrix} N_\mathrm{C} \\ 1 \end{pmatrix} \tag{3}$$

Transformed colormap indices c_n are then obtained as

$$c_n = b\,\mathrm{cVec}_n + a \tag{4}$$

c_n is then used as `data.cVec`. This process is same for every other plot which is based on coloring of faces.

3.5 `export.advanced.farField2TikZ()`

This method is implemented for conversion of `plotFarField()` resulting plot. Input arguments are structures `settings`, see Table 1, and `data` described below. Both these structures have to contain all data mentioned in those tables. It is recommended to not directly utilise this function and using `plot2TikZ()` instead.

Parameter	Format	Description
`faces_b`	$N_\mathrm{F} \times 4$	connectivity list of triangles vertices
`vert`	$N_\mathrm{V} \times 3$	vertices vector
`cMap`	$N_\mathrm{C} \times 3$	colormap vector
`cVec`	$N_\mathrm{F} \times 3$	vector of faces colors
`type`	`string`	control parameter for higher-level functions

Table 4: N_V denotes length of vertices vector, N_F stands for faces matrix and N_C is length of colormap vector.

3.6 `export.advanced.farFieldCut2TikZ()`

This method is implemented for conversion of `plotFarFieldCut()` resulting plot. Input arguments are structures `settings`, see Table 2, and `hndl` (output argument of `plotFarFieldCut()`) described in [1].It is recommended to not directly utilise this function and using `plot2TikZ()` instead.

3.7 `export.advanced.charge2TikZ()`

This method is implemented for conversion of `plotCharge()` resulting plot. Input arguments are structures `settings`, see Table 1, and `data` described below. Both these structures have to contain all data mentioned in those tables. It is recommended to not directly utilise this function and using `plot2TikZ()` instead.

Parameter	Format	Description
`faces_b`	$N_\mathrm{F} \times 3$	connectivity list of triangles vertices
`vert`	$N_\mathrm{V} \times 3$	vertices vector
`cMap`	$N_\mathrm{C} \times 3$	colormap vector
`cVec`	$N_\mathrm{F} \times 3$	vector of faces colors
`type`	`string`	control parameter for higher-level functions

Table 5: N_V denotes length of vertices vector, N_F stands for faces matrix and N_C is length of colormap vector.

3.8 `export.advanced.mesh2TikZ()`

This method is implemented for conversion of `plotMesh()` resulting plot. Input arguments are structures `settings`, see Table 1, and `data` described below. Both these structures have to contain all data mentioned in those tables. It is recommended to not directly utilise this function and using `plot2TikZ()` instead.

Parameter	Format	Description
`faces_b`	$N_\mathrm{F} \times 3$	connectivity list of triangles vertices
`vert`	$N_\mathrm{V} \times 3$	vertices vector
`cVec`	$N_\mathrm{F} \times 3$	vector of faces colors
`type`	`string`	control parameter for higher-level functions

Table 6: N_V denotes length of vertices vector and N_F stands for faces matrix.

3.9 `export.advanced.topoSens2TikZ()`

This method is implemented for conversion of topological sensitivity results. It is still in early phase of development and is primarily meant for internal usage only. Input arguments are structures `settings`, see 1 and `data` described below.

Parameter	Format	Description
`Mesh`	`struct`	connectivity list of triangles vertices
`BF`	`struct`	vertices vector
`Temp`	$N_\mathrm{B} \times 3$	boolean gene vector
`axes.View`	2×1	view angle
`type`	`string`	control parameter for higher-level functions

Table 7: N_B denotes length of gene vector. `Mesh` and `BF` structures are described in [1].

Resulting plot differs active and inactive mesh element, see figure below, where the top row is active.

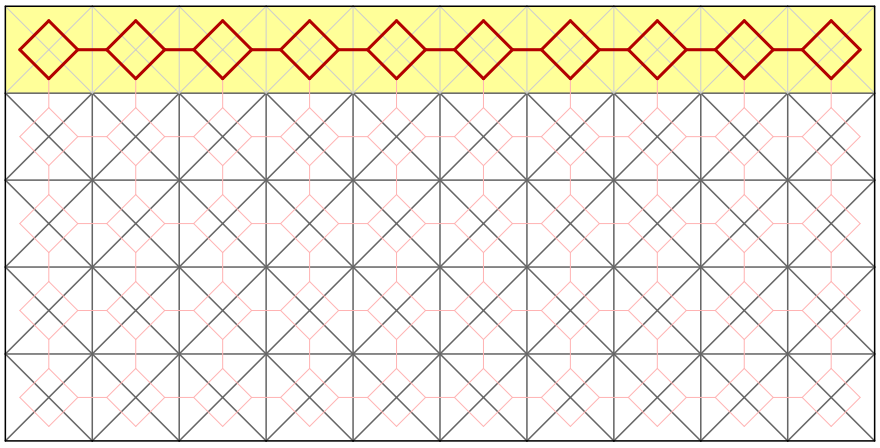


Figure 26: Active and inactive basis functions.

Listing 30: Example of topological sensitivity utilisation.

```
% Generation of structure
P = 2*ones(5,10);

% Computation of mesh and basis functions structures
[~,~,Mesh] = models.utilities.meshPublic.pixelGridToMesh(P, 1);
BF = models.solvers.MoM2D.basisFcns.getBasisFcns(Mesh);

% Setting active edges
actEdges = (237:5:277)' + (0:4);
temp = false(BF.nUnknowns,1);
temp([actEdges(:); (282:285)']) = true;

% Data assignment
hndl.Mesh = Mesh;
hndl.BF = BF;
hndl.Temp = temp;
hndl.axes.View = [0 90];
hndl.type = 'topoSens';

% Use this instead of direct function topoSens2TikZ()
export.plot2TikZ(hndl,'TikZ_Scale',1);
```

4 Known Problems

4.1 Sorting

Graphic primitives representing faces have to be correctly sorted for correct visualisation. However, the algorithm for this sort fails sometimes. This negative effect is represented in following figure.

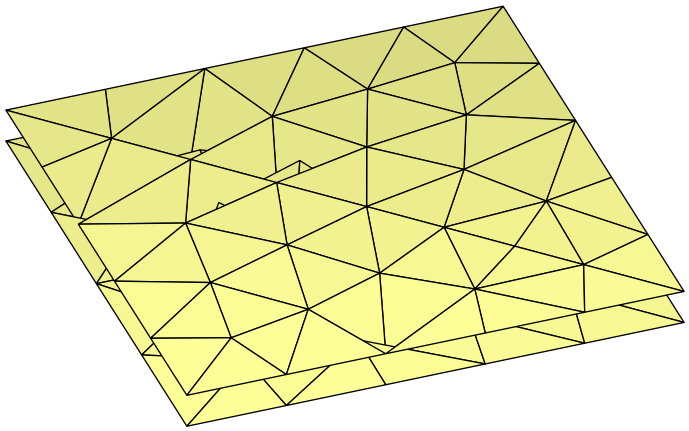


Figure 27: Triangles are not visualised correctly due to low mesh density.

This problem can be solved with more detail mesh, or with different view angle, although this is not always working.

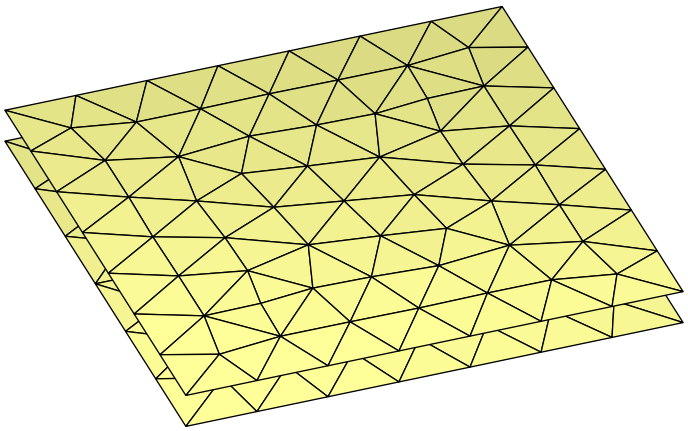


Figure 28: Triangles are visualised correctly in this case.

4.2 Memory

Another problem can occur when TeX memory exceeds its limit. This can be solved through few following steps.

1. Open command window and enter following command.

```
initexmf --edit-config-file=latex
```

2. Configuration file is opened. Add line below to this file and save it.

```
main_memory=12000000 (at default 3000000)
```

3. Last step is again in command window, use the following line

```
initexmf --dump=latex
```

In case of need, repeat with *pdflatex* and *xelatex* [3].

References

[1] (2019) Antenna Toolbox for MATLAB (AToM). Czech Technical University in Prague. [Online]. Available: www.antennatoolbox.com

[2] V. Neuman, *Volumetric Method of Moments and Post-Processing of Results*. Czech Technical University, 2019.

[3] F. Rappl. (2019, Oct.) Increase LaTeX memory. [Online]. Available: https://florian-rappl.de/Articles/Page/239/latex-memory