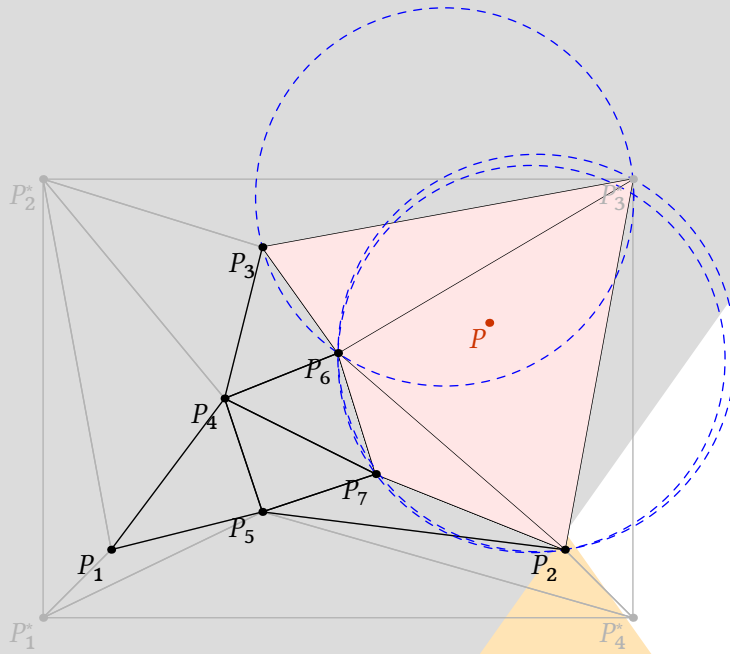


# luamesh

compute and draw meshes with Lua<sub>TEX</sub>



**Contributor**  
Maxime CHUPIN

Version 0.1, 25 novembre 2016  
<http://melusine.eu.org/syracuse/G/delaunay/>

# Luamesh: compute and draw meshes with Lua<sup>A</sup>T<sub>E</sub>X

Maxime Chupin <[mc@melusine.eu.org](mailto:mc@melusine.eu.org)>

November 28, 2016

The package `luamesh` allows to compute and draw 2D Delaunay triangulation. The algorithm is written with lua, and depending on the choice of the “engine”, the drawing is done by MetaPost (with `luampLib`) or by `tikz`.

The Delaunay triangulation algorithm is the Bowyer and Watson algorithm. Several macros are provided to draw the global mesh, the set of points, or a particular step of the algorithm.

I would like to thank Jean-Michel Sarlat, who hosts the development with a git project on the `melusine` machine:

<https://melusine.eu.org/syracuse/G/delaunay/>

I would also like to thank the first user, an intensive *test* user, and a very kind English corrector: Nicole Spillane.

## 1 Installation

Of course, you can just put the two files `luamesh.lua` and `luamesh.sty` in the working directory, but this is not recommended.

### 1.1 With T<sub>E</sub>Xlive and Linux or Mac OSX

To install `luamesh` with T<sub>E</sub>Xlive, you have to create the local `texmf` directory in your `home`.

```
user $> mkdir ~/texmf
```

Then place the files in the correct directories. First, the `luamesh.sty` file must be in the directory:

`~/texmf/tex/latex/luamesh/`

and secondly, the `luamesh.lua` must be in the directory:

`~/texmf/scripts/luamesh/`

Once you have done this, `luamesh` can be included in your document with

---

```
\usepackage{luamesh}
```

---

## 1.2 With MikTeX and Windows

We do not know these two systems, so we refer to the documentation for integrating local additions to MikTeX:

<http://docs.miktex.org/manual/localadditions.html>

## 1.3 A LuaTeX package

If you want to use this package, you must compile your document with `luatex`:

```
user $> luatex mylatexfile.tex
```

## 1.4 Dependencies

This package is built upon two main existing packages to draw the triangulations :

1. `luamplib` to use MetaPost via the LuaTeX library `mplib`;
2. and `tikz`.

We will see how to choose between these two *drawing engines*.

Moreover, the following packages are necessary:

1. `xkeyval` to manage the optional arguments;
2. `xcolor` to use colors (needed by `luamplib`);
3. `ifthen` to help the programming with TeX.

## 2 The Basic Macros

Let us recall that this package provides macros to draw two dimensional triangulations (or meshes).

### 2.1 Draw a Complete Mesh

```
\buildMeshBW[<options>]{<list of points> or <file name>}
```

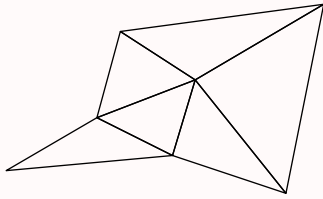
This macro produces the Delaunay triangulation (using the Bowyer and Watson algorithm) of the given *<list of points>*. The list of points must be given in the following way :

$(x_1, y_1); (x_2, y_2); (x_3, y_3); \dots; (x_n, y_n)$

---

```
\buildMeshBW{(0.3,0.3);(1.5,1);(4,0);(4.5,2.5);(1.81,2.14);(2.5,0.5);(2.8,1.5)}
```

---



### 2.1.1 The Options

There are several options to customize the drawing.

**mode = int (default) or ext:** this option allows to use either the previously described set of points in the argument, or a file, containing, line by line (2 columns), the points. Such a file looks like :

```
x1 y1  
x2 y2  
x3 y3  
...  
xn yn
```

**bbox = none (default) or show:** this option allows to draw the added points to form a *bounding box*<sup>1</sup> and the corresponding triangulation. By default, these triangles are removed at the end of the algorithm.

**color = <value> (default: black):** The color of the drawing.

**colorBbox = <value> (default: black):** The color of the drawing for the elements (points and triangles) belonging to the bounding box.

**print = none (default) or points:** To label the vertices of the triangulation. This also adds a *dot* at each vertex.

**meshpoint = <value> (default: P):** The letter(s) used to label the vertices of the triangulation. It is included in the math mode delimiters  $\$...\$$ . The bounding box points are labeled with numbers 1 to 4 and with a star exponent.

**tikz (boolean, default:false):** By default, this boolean is set to *false*, and MetaPost (with *luamplib*) is used to draw the picture. With this option, *tikz* becomes the *drawing engine*.

**scale = <value> (default: 1cm):** The scale option defines the scale at which the picture is drawn (the same for both axes). It must contain the unit of length (cm, pt, etc.).

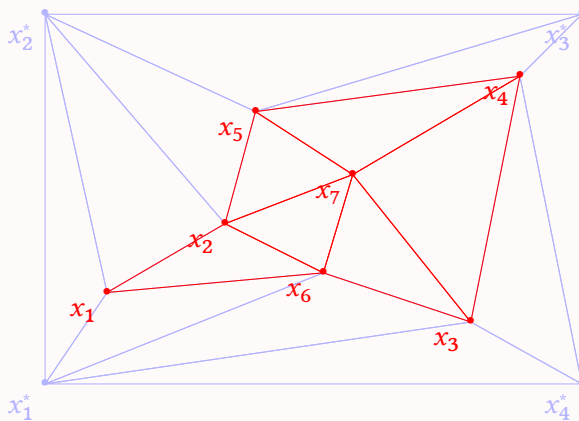
---

<sup>1</sup>The bounding box is defined by four points placed at 15% around the box defined by  $(x_{\min}, y_{\min})$ ,  $(x_{\min}, y_{\max})$ ,  $(x_{\max}, y_{\max})$ , and  $(x_{\max}, y_{\min})$ . It is used by the algorithm and will be computed in any case.

To illustrate the options, let us show you an example. We consider a file `mesh.txt`:

```
0.3  0.3
1.5  1
4    0
4.5  2.5
1.81 2.14
2.5  0.5
2.8  1.5
```

```
\buildMeshBW[%
tikz,
mode = ext,
bbox = show,
color = red,
colorBbox = blue!30,
print = points,
meshpoint = x,
scale = 1.3cm,
]{mesh.txt}
```



The drawing engine is not very relevant here, but it is useful to understand how the drawing is made. However, the engine will be relevant to the so called *inc* macros (section 3), for adding code before and after the one generated by `luamesh`.

## 2.2 Draw the Set of Points

```
\drawPointsMesh[<options>]{<list of points> or <file name>}
```

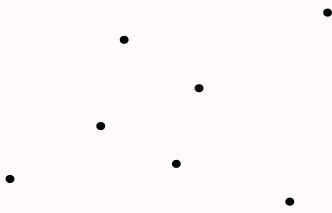
With the `\drawPointsMesh`, we plot the set of (user chosen) points from which the Bowyer and Watson algorithm computes the triangulation.

The use of this macro is quite similar to `\buildMeshBW`. Here is an example of the basic uses.

---

```
\drawPointsMesh{(0.3,0.3);(1.5,1);(4,0);(4.5,2.5);(1.81,2.14);(2.5,0.5);(2.8,1.5)}
```

---



### 2.2.1 The Options

There are several options (exactly the same as for the `\buildMeshBW`) to customize the drawing.

**mode = int (default) or ext:** this option allows to use either the previously described set of points as the argument, or a file, containing, line by line (2 columns), the points. Such a file looks like :

```
x1 y1
x2 y2
x3 y3
...
xn yn
```

**bbox = none (default) or show:** this option allows to draw the added points to form a *bounding box* and the corresponding triangulation. By default, these triangles are removed at the end of the algorithm. *Here, because we plot only the vertices of the mesh, there are no triangles, only dots.*

**color = <value> (default: black):** The color of the drawing.

**colorBbox = <value> (default: black):** The color of the drawing for the elements (points and triangles) belonging to the bounding box.

**print = none (default) or points:** To label the vertices of the triangulation. This also adds a *dot* at each vertex. Without label, there is still the dot.

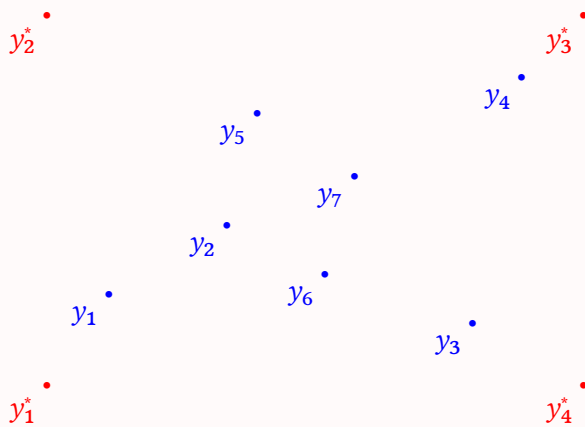
**meshpoint = <value> (default: P):** The letter(s) used to label the vertices of the triangulation. It is included in the math mode delimiters  $\$...\$$ . The bounding box points are labeled with numbers 1 to 4 and with a star exponent.

**tikz (boolean, default:false):** By default, this boolean is set to **false**, and MetaPost (with `luamplib`) is used to draw the picture. With this option, **tikz** becomes the *drawing engine*.

**scale = <value> (default: 1cm):** The scale option defines the scale at which the picture is drawn (the same for both axes). It must contain the unit of length (cm, pt, etc.).

With the same external mesh point file presented in section 2.1, we illustrate the different options.

```
\drawPointsMesh[%
tikz,
mode = ext,
bbox = show,
color = blue,
colorBbox = red,
print = points,
meshpoint = y,
scale = 1.3cm,
]{mesh.txt}
```



### 2.3 Draw a Step of the Bowyer and Watson Algorithm

`\meshAddPointBW`[*<options>*]{*<list of points>* or *<file name>*}{*<point>* or *<number of line>*}

This command allows to plot the steps within the addition of a point in a Delaunay triangulation, by the Bowyer and Watson algorithm.

This macro produces the Delaunay triangulation (using the Bowyer and Watson algorithm) of the given *<list of points>* and shows a step of the algorithm when the *<point>* is added. The list of points must be given in the following way:

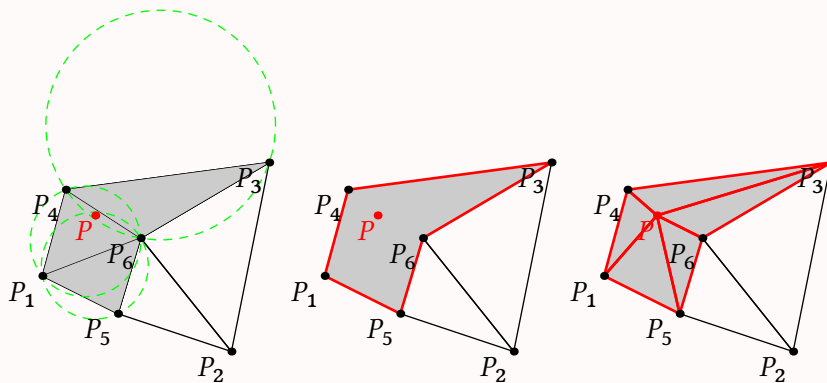
$$(x_1, y_1); (x_2, y_2); (x_3, y_3); \dots; (x_n, y_n)$$

and the point is of the form  $(x, y)$ . The *<file name>* and *<number of line>* will be explained in the option description.

One can use the macro as follows:

```
\meshAddPointBW[step=badtriangles]{(1.5,1);(4,0);(4.5,2.5);(1.81,2.14);(2.5,0.5);(2.8,1.5)
}{(2.2,1.8)}
\meshAddPointBW[step=cavity]{(1.5,1);(4,0);(4.5,2.5);(1.81,2.14);(2.5,0.5);(2.8,1.5)
}{(2.2,1.8)}
```

```
\meshAddPointBW[step=newtriangles]{(1.5,1);(4,0);(4.5,2.5);(1.81,2.14);(2.5,0.5);(2.8,1.5)}
{(2.2,1.8)}
```



The default value for `step` is `badtriangles`. Consequently, the first line is equivalent to

```
\meshAddPointBW{(1.5,1);(4,0);(4.5,2.5);(1.81,2.14);(2.5,0.5);(2.8,1.5)}{(2.2,1.8)}
```

### 2.3.1 The Options

There are several options (some of them are the same as for `\buildMeshBW`) to customize the drawing.

`mode = int (default) or ext`: this option allows to use either the previously described set of point in the first argument, or a file containing, line by line (2 columns), the points. Such a file looks like :

```
x1 y1
x2 y2
x3 y3
...
xn yn
```

For the second argument of the macro, if we are in the `mode = ext`, the argument must be the *line number* of the file corresponding to the point we want to add. The algorithm will stop the line before to build the initial triangulation for which it will add the point corresponding to the line. The subsequent lines in the file are ignored.

`bbox = none (default) or show`: this option allows to draw the added points to form a *bounding box* and the corresponding triangulation. By default, these triangles are removed at the end of the algorithm.

`color = <value> (default: black)`: The color of the drawing.

`colorBbox = <value> (default: black)`: The color of the drawing for the elements (points and triangles) belonging to the bounding box.



`colorNew = <value> (default: red)`: The color of the drawing of the “new” elements which are the point to add, the polygon of the cavity, and the new triangles.

`colorBack = <value> (default: black!20)`: The color for the filling of the region concerned by the addition of the new point.

`colorCircle = <value> (default: green)`: The color for the circumcircle of the triangles containing the point to add.

`meshpoint = <value> (default: P)`: The letter(s) used to label the vertices of the triangulation. It is included in the math mode delimiters `$....$`. The bounding box points are labeled with numbers 1 to 4 and with a star exponent.

`step = badtriangles (default) or cavity or newtriangles`: To choose the step we want to draw, corresponding to the steps of the Bowyer and Watson algorithm.

`newpoint = <value> (default: P)`: The letter(s) used to label the new point of the triangulation. It is include in the math mode delimiters `$....$`.

`tikz (boolean, default:false)`: By default, this boolean is set to `false`, and MetaPost (with `luamplib`) is used to draw the picture. With this option, `tikz` is the *drawing engine*.

`scale = <value> (default: 1cm)`: The scale option defines the scale at which the picture is draw (the same for the two axis). It must contain the unit of length (cm, pt, etc.).

Here is an example of customizing the drawing. First, recall that the external file `mesh.txt` is:

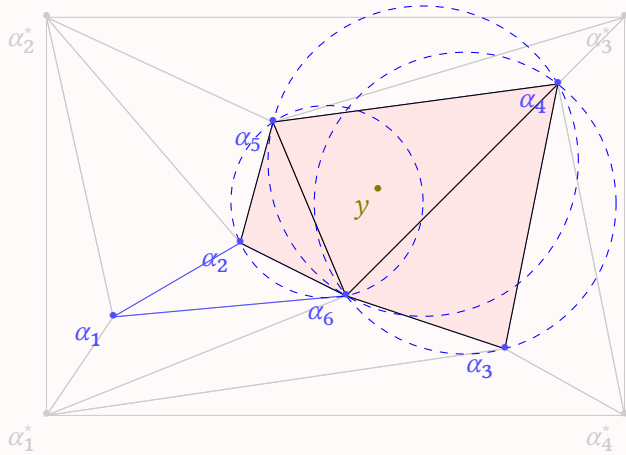
```
0.3  0.3
1.5  1
4    0
4.5  2.5
1.81 2.14
2.5  0.5
2.8  1.5
```

We draw the addition of the 6th point. The 7th line will be ignored.

---

```
\meshAddPointBW[
tikz,
mode = ext,
color = blue!70,
meshpoint = \alpha,
newpoint = y,
colorBack=red!10,
colorNew = green!50!red,
colorCircle = blue,
colorBbox = black!20,
bbox = show,
scale=1.4cm,
```

```
step=badtriangles]
{mesh.txt}{6}
```



### 3 The *inc* Macros

The three macros presented in the above sections have complementary macros, with the suffix `inc` that allow the user to add code (MetaPost or `tikz`, depending of the drawing engine) before and after the code generated by `luamesh`.

The three macros are:

```
\buildMeshBWinc[<options>]{<list of points> or <file name>}{<code before>}{<code after>}
\drawPointsMeshinc[<options>]{<list of points> or <file name>}{<code before>}{<code after>}
\meshAddPointBWinc[<options>]{<list of points> or <file name>}%
    {<point> or <number of line>}{<code before>}{<code after>}
```

#### 3.1 With MetaPost

We consider the case where the drawing engine is MetaPost (through the `luamplib` package).

We describe the feature taking one macro in example but the mechanism and the possibilities are exactly the same for all the macros.

When we use the MetaPost drawing engine, the macros previously described produced a code of the form

```
\begin{luamplib}
  u:=<scale>;
  beginfig(0);
  <code for the drawing>
  endfig;
\end{luamplib}
```

Then, the arguments *<code before>* and *<code after>* are inserted as follows:

```
\begin{luamplib}
  u:=<scale>;
  <<code before>>
  <code for the drawing>
  <<code after>>
\end{luamplib}
```



With the *inc* macros, the user has to add the `beginfig()`; and `endfig`; commands to produce a picture. Indeed, this allows to use the `\everymplib` command from the `\luamplib` package.

### 3.1.1 The $\LaTeX$ Colors Inside the MetaPost Code

The configurable colors of the  $\LaTeX$  macro are accessible inside the MetaPost code. For `\buildMeshBWinc` and `\drawPointsMeshinc`, we have `\luameshmpcolor`, and `\luameshmpcolorBbox`. For the macro `\meshAddPointBWinc` we have three additional colors: `\luameshmpcolorBack`, `\luameshmpcolorNew`, and `\luameshmpcolorCircle`. Of course, we can define MetaPost colors as well. Finally, the `luamplib` mechanism of `\mpcolor` is also available.

### 3.1.2 The Mesh Points with MetaPost

At the beginning of the automatically generated code, a list of MetaPost *pairs* are defined corresponding to all the vertices of the mesh (when the option `bbox=show`, the last 4 points are the *bounding box points*). The points are available with the `MeshPoints[]` table of variables. The `MeshPoints[i]` are defined using the unit length `u`.

### 3.1.3 Examples

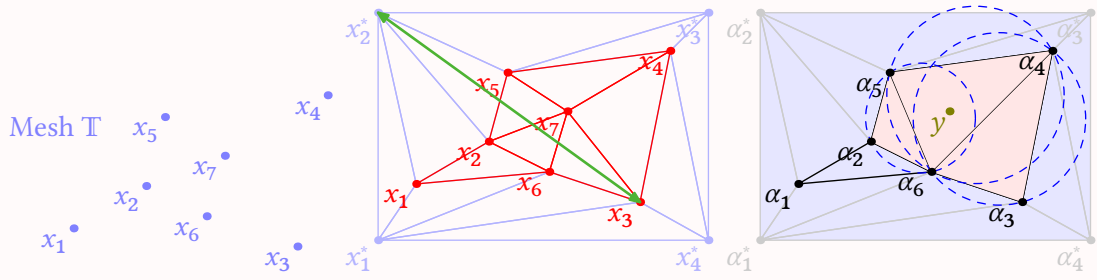
Here is three examples for the different macros.


```
\drawPointsMeshinc[
  color = blue!50,
  print = points,
  meshpoint = x,
  scale=0.8cm,
  ]{(0.3,0.3);(1.5,1);(4,0);(4.5,2.5);(1.81,2.14);(2.5,0.5);(2.8,1.5)}%
  {% code before
    beginfig(0);
  }%
  {% code after
    label(btex Mesh $\mathbb{T}$ etex, (0,2u)) withcolor \luameshmpcolor;
    endfig;
  }
\buildMeshBWinc[%
  bbox = show,
```

```

color = red,
colorBbox = blue!30,
print = points,
meshpoint = x,
scale=0.8cm
]{{(0.3,0.3);(1.5,1);(4,0);(4.5,2.5);(1.81,2.14);(2.5,0.5);(2.8,1.5)}}%
{% code before
  beginfig(0);
}
{% code after
  drawdblarrow MeshPoints[3] -- MeshPoints[9] withpen pencircle scaled 1pt
  withcolor (0.3,0.7,0.2);
  endfig;
}
\meshAddPointBWinc[
meshpoint = \alpha,
newpoint = y,
colorBack=red!10,
colorNew = green!50!red,
colorCircle = blue,
colorBbox = black!20,
bbox = show,
scale=0.8cm,
step=badtriangles]
{{(0.3,0.3);(1.5,1);(4,0);(4.5,2.5);(1.81,2.14);(2.5,0.5)}}{{(2.8,1.5)}}%
{%code before
  picture drawing;
  drawing := image(
}{%code after
  );
  beginfig(0);
  fill MeshPoints[7]--MeshPoints[8]--MeshPoints[9]--MeshPoints[10]--cycle
  withcolor \mpcolor{blue!10};
  draw drawing;
  endfig;
}

```



 The variables `MeshPoints[]` are not defined for the argument corresponding to the code to place before the code generated by `luamesh`. Hence, to use such variables, we have to define a `picture` as shown in the third example above.

3.2 With TikZ

4 Gallery of Examples