KASP Technical notes #06:
Pupil model generator with KASP

G. Chataignier, O. Beltramo-Martin[*], C. M. Correia

Contents

1 Purpose

How to build a pupil using pupil and segment class

2 Segment class definition

The segment class constructor has 3 required inputs :

- Number of sides (3 : triangle, 6 : hexagon, inf : disk)
- External radius of a segment in meter (1.4/2 for eelt, 1.8/2 for Keck...)
- Definition in pixels for the matrix

It also has optional inputs :

[*]olivier.martin@lam.fr

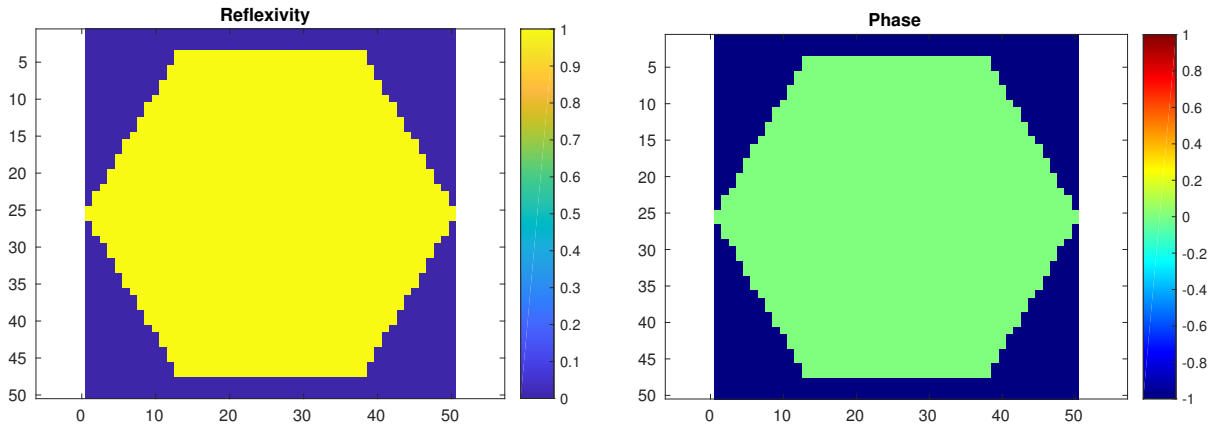


Figure 1: Illustration of a segment

- Angle in rad
- Coordinates in the pupil
- Reflexivity Error
- Coefficients for error phase modes
- Geometry errors : position, angle, size,

```
S=segment(6,0.9,50); % hexagon of 1.8 m, in a 50px matrix
```

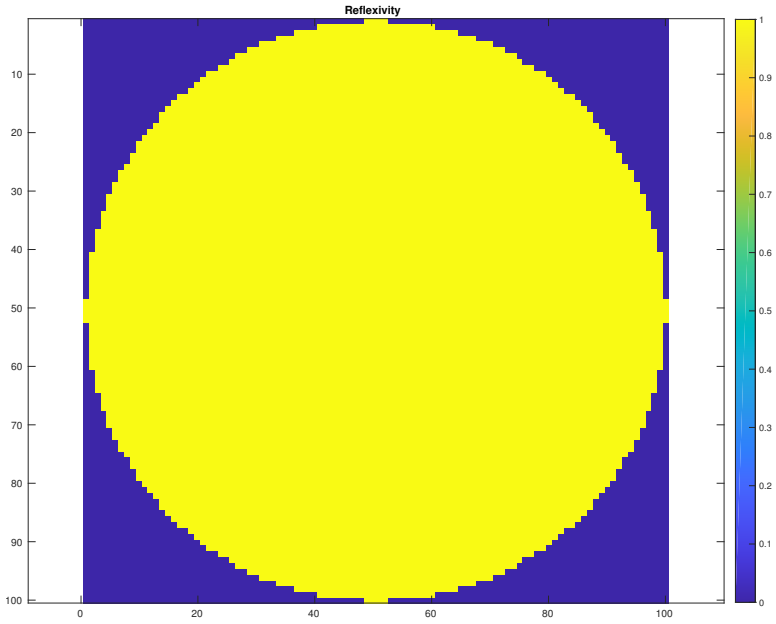
Segment can be displayed via S.disp S.disp shows reflexivity and phase S.disp('r') shows reflexivity only S.disp('p') show phase only

```
S.disp;
```

2.1 Definition of a simple Pupil

The Pupil constructor has no Required input : It builds by default a circular telescope of 1 meter with no central obstruction represented in a matrix of 100 pixels.

```
P=pupil;
P.disp('r');
```

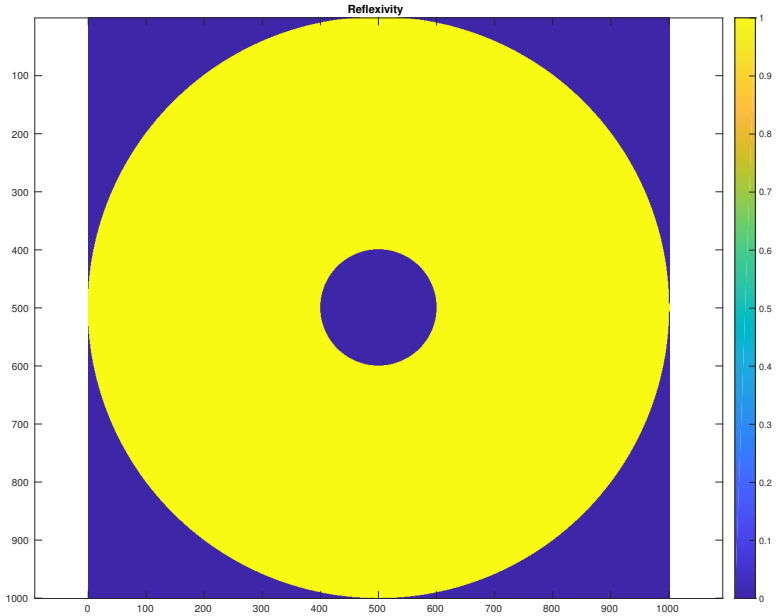


2.2 Central Obstruction and Spider

If you want to change the resolution or make the central obstruction, you have to consider one segment of a given definition, and pass the obstruction ratio as follows.

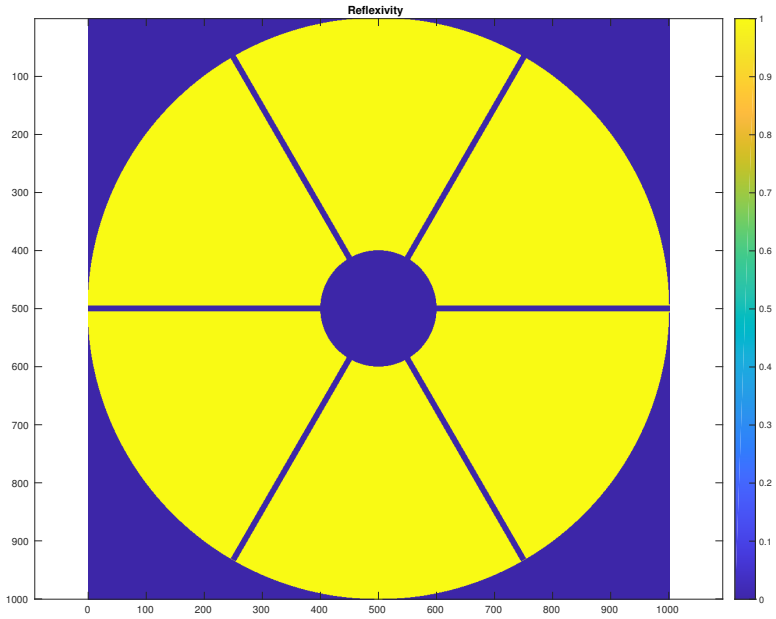
We can work with meters or with pixels. I will work with meters here and show an exemple using pixels later

```
S=segment(inf,5,1000); % disk of 10 meters, in a 1000px matrix
P=pupil('segRef',S,'obstructionRatio',0.2);
P.disp('r');
```



Let's add spider : we need to define a spider structure. if it's a simple geometry (line going through the center) it's automatic but the user can define his own mask

```
SPIDER.n=3; % number of lines
SPIDER.angle=[0 pi/3 2*pi/3]; % angle of each line
SPIDER.width=0.1; % in meter
SPIDER.usrDefined=0;% boolean to decide if spider are build by pupil or defined by user. If
SPIDER.usrMask=[];

P.mkSpiders(SPIDER);
P.disp('r');
```



Segmented Telescopes

Let's create a segmented telescope, the Keck, for example. We need a segment of reference, a list of coordinates (in meters or pixels) and a wavelength (important to create the phase). I will now work with pixels.

Note that those coordinates are "optimised" for segments of 200px exactly The fact is that segments are not perfect : for exemple, the up-left side of an hexagon does not perfectly fit the down-right side, so we cannot perfectly mesh the pupil. We need to avoid overlapping segments, but we must also minimize the gaps between each segment. We also need to have a regular mesh to be able to fill the remaining gaps. Thats why it is "easier" to work in pixels. We can adjust manually the position of each segment, with one pixel precision.

Segments are ordered by their apparition in the coordinates list

```
SVpx=[   772         535;
         622         448;
         472         535;
         472         709;
         622         796;
         772         709;
         922         448;
         772         361;
```

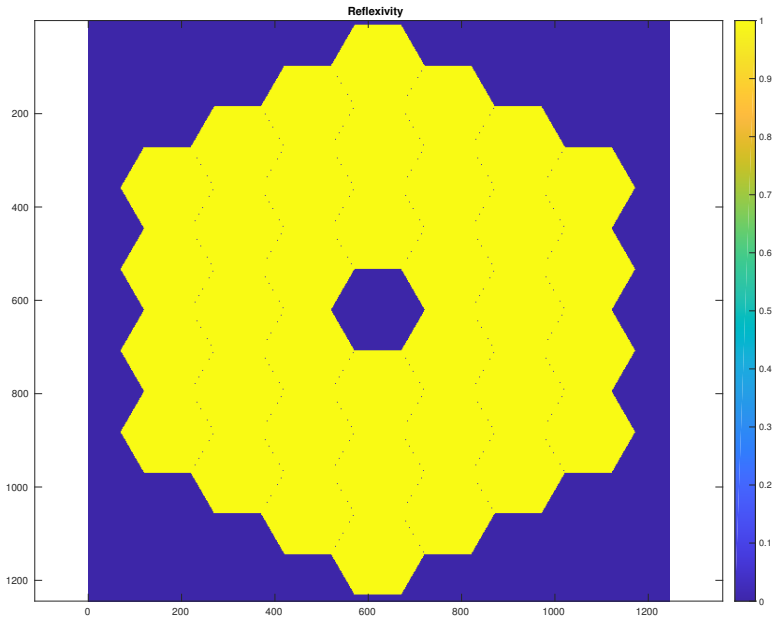
```
 622         274;
 472         361;
 322         448;
 322         622;
 322         796;
 472         883;
 622         970;
 772         883;
 922         796;
 922         622;
1072         361;
 922         274;
 772         187;
 622         100;
 472         187;
 322         274;
 172         361;
 172         535;
 172         709;
 172         883;
 322         970;
 472        1057;
 622        1144;
 772        1057;
 922         970;
1072         883;
1072         709;
1072         535];
```

Of course it is easier to load a .txt ...

```
S=segment(6,0.9,200);
lambda=2.12e-6;
P=pupil('segRef',S,'segCoord',SVpx,'wavelength',lambda,'unit','px');
```

DO NOT forget to specify the unit ! (or expect to have memory issue... Here, the max coordinate is 1134. If the class thinks it's in meter, it will create a HUGE matrix)

```
P.disp('r');
```



3 Pupil including phase and amplitude errors

3.1 Introducing Phase and Reflexivity errors

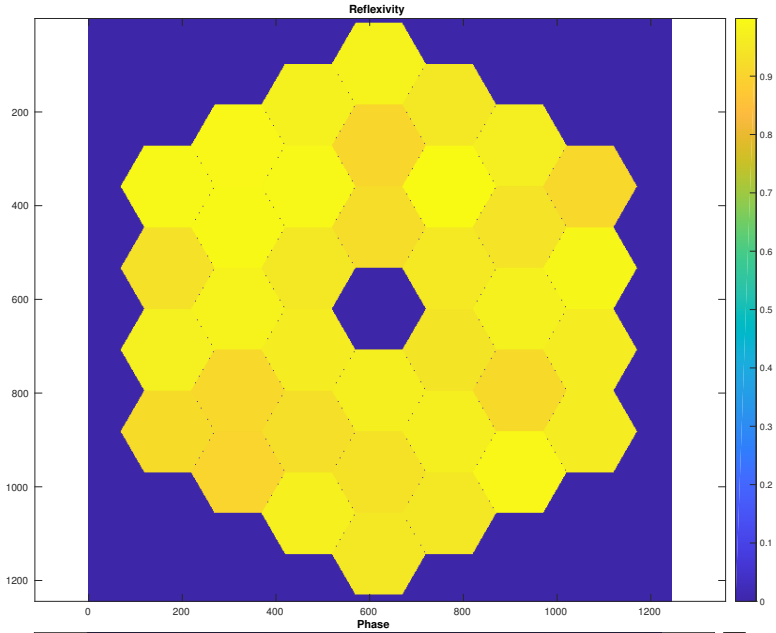
Phase : only specify the coefficients and the type of modes (Zernike by default). It's a [number of segments x number of Modes] matrix.

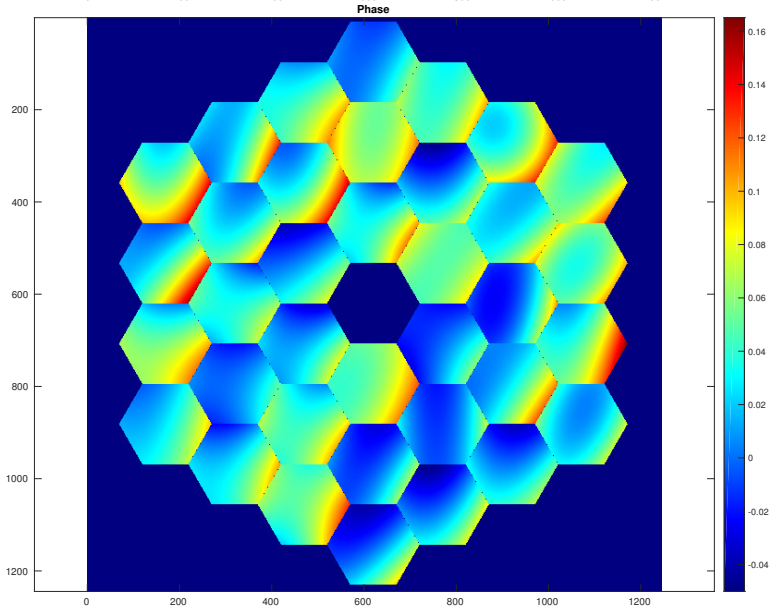
Reflexivity : an arrow of nSegments giving reflexivity value. It could be a "cube" [segDefinition x segDefinition x number of segment] but it has not been tested yet.

```
R=0.9+rand(1,length(SVpx))*(1-0.9); % random reflexivity btw .85 and 1

piston=rand(1,length(SVpx))*25e-9; % random piston btw 0 and 25 nm
tip=rand(1,length(SVpx))*10e-9; % random tip btw 0 and 10 nm
tilt=rand(1,length(SVpx))*10e-9; % random tilt btw 0 and 10 nm
focus=rand(1,length(SVpx))*5e-9; % random focus
astig1=rand(1,length(SVpx))*5e-9; % random astig
astig2=rand(1,length(SVpx))*5e-9;

PE=[piston' tip' tilt' focus' astig1' astig2'];

P=pupil('segRef',S,'segCoord',SVpx,'wavelength',lambda,'unit','px',...
    'coeffReflexion',R,'coeffPhaseModes',PE);

P.disp
```

```
 @(zernike polynomials)> Created!
___ ZERNIKE POLYNOMIALS ___
 . 6 modes: 1,2,3,4,5,6
```

```
-------------------------------------------------------
 @(zernike polynomials)> Terminated!
```





Also available by calling P.applyReflexivity and P.applyPhaseError :

segment n1 and 5 will have a reflexivity of 0.25 and 0.443 respectively

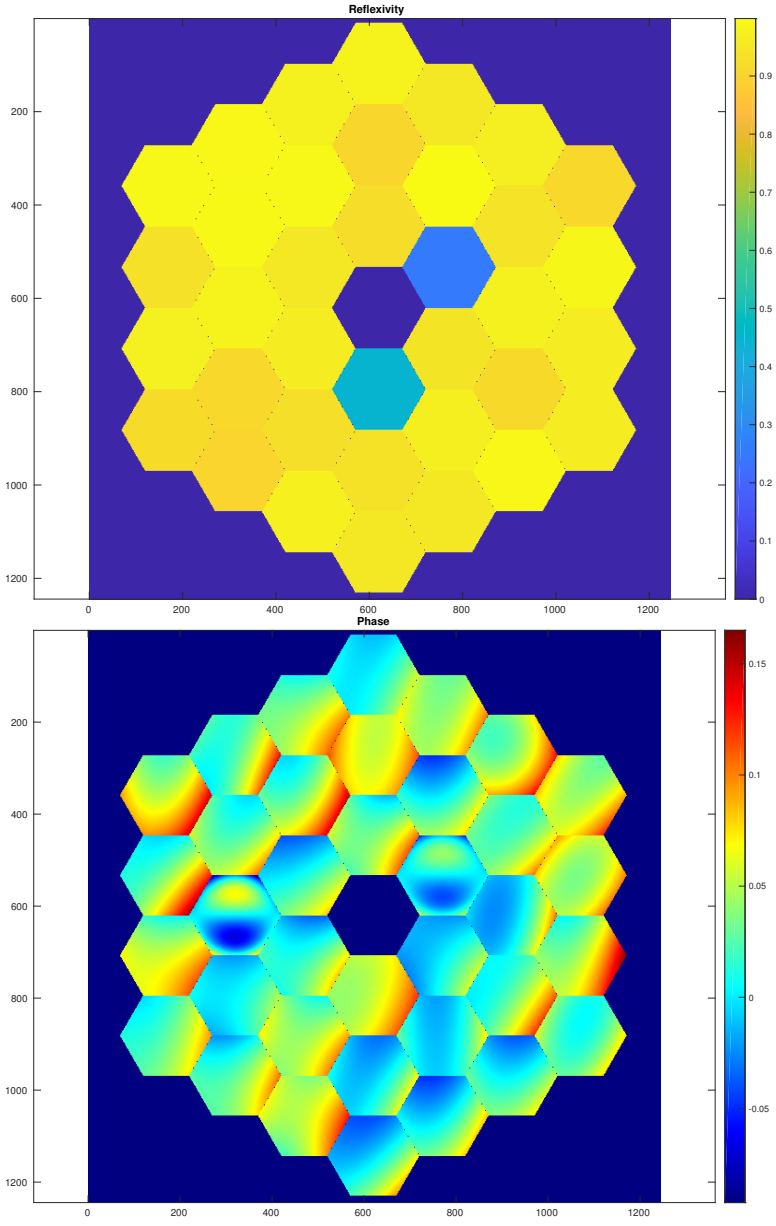
```
P.applyReflexivity([1 5],[0.25 0.443]);
```

segments n 1 and 12 will get 8 and 13 nm of coma

```
P.applyPhaseError([1 12], [0 0 0 0 0 0 8; 0 0 0 0 0 0 13]*1e-9);
```

```
P.disp;
```

```
 @(zernike polynomials)> Created!
___ ZERNIKE POLYNOMIALS ___
 . 7 modes: 1,2,3,4,5,6,7
-----------------------------------------------------
 @(zernike polynomials)> Terminated!
```



Geometry Errors

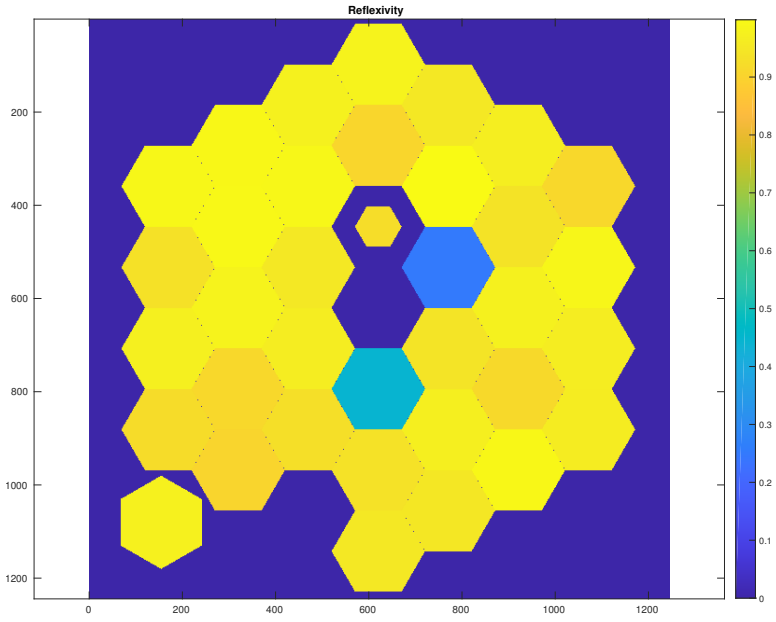
Operations available :

- Shift the segment giving X and Y displacement. Those values will be stored in segment.posXError and segment.posYError /!\ Warning : x-axis = left to right, y-axis = down to top
- Rotate the segment giving an angle. The rotation value is stored in segment.angleError

- Shrink segment giving a scale factor between 0 and 1. This factor is stored in segment.sizeError

Please be careful : always give the value in the unit used at pupil creation (px or meter)

```
% move segment n36 of -300 px on x and -15px on y
% and then rotate it of pi/6 rad
P.shiftSegment(30,-315,-25);
P.rotateSegment(30,pi/6);
P.shrinkSegment(2,0.5); % segment 2 will be shrink at 0.5* original size

P.disp('r');
```



Modifying the pupil

You can zeroPad, remove the borders and resize

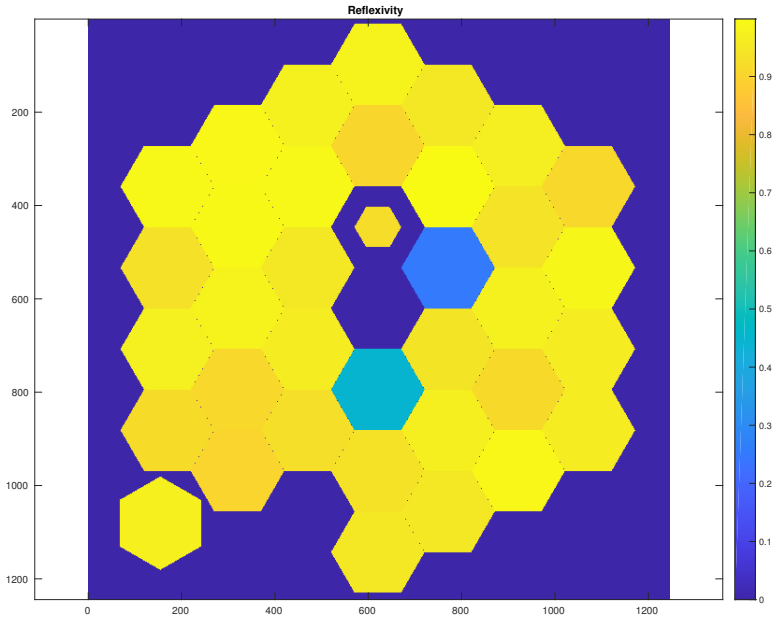
- ZeroPad : add zeros around the pupil for sampling purposes (frequency domain), giving a coefficient
- removeZeroBorder : remove all the border full of zeros, but keep the matrix square
- resize : resize the pupil giving a new number of pixels. It is using interp2 Matlab function, so you can specify the method (linear, nearest,spline,cubic). Nearest, by default, seems to be the best method (no strange pixel in reflexivity nor phase)

Those methods will produce a temporary matrix and will change the pupil matrix. If you want only the tmp matrix but do not want the original pupil to be modified, you can specify 0 or 1 after the main arguments

Please note that rmZeroBorder and Resize prevent you from modifying the pupil : all the methods (geometry, reflexivity and phase) are based on the segments geometry. As we resize the whole pupil matrix and not each segment one by one, trying the methods above will produce weird results

Original Matrix

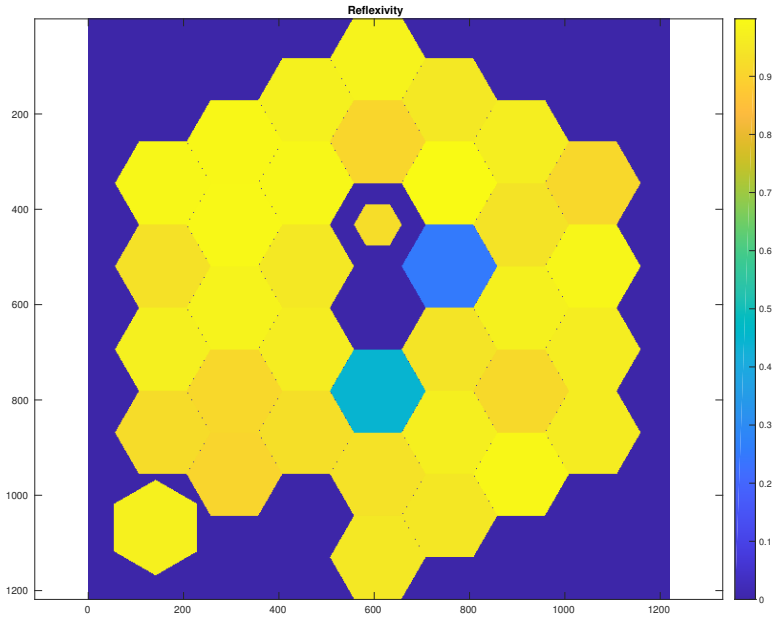
```
P.disp('r');
```



Removing empty border

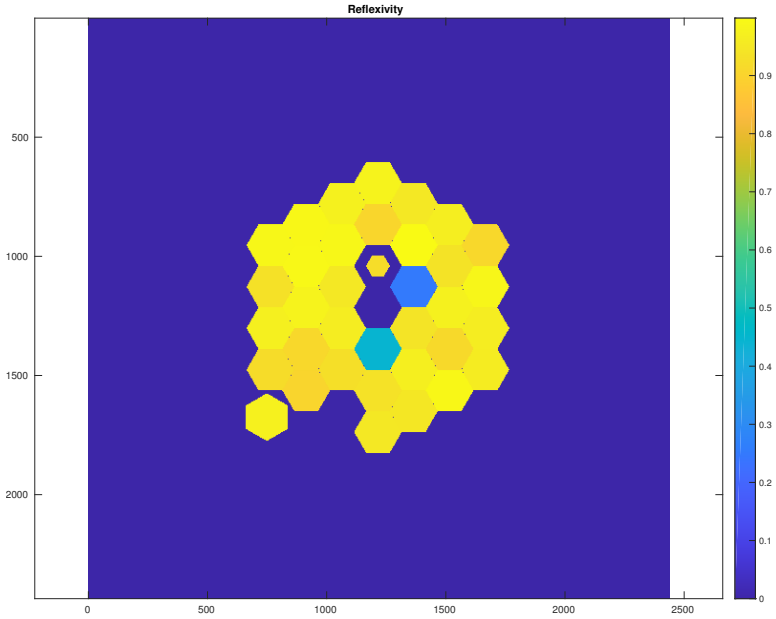
```
tmpMat=P.rmZeroBorder(0); % flagReplace at 0 -> you'll get the tmp Matrix in  tmpMatrix but

P.rmZeroBorder(1); % flagReplace at 1 -> pupil will be modified inside the object
P.disp('r');
```

```
removing border full of zeros
removing border full of zeros
```



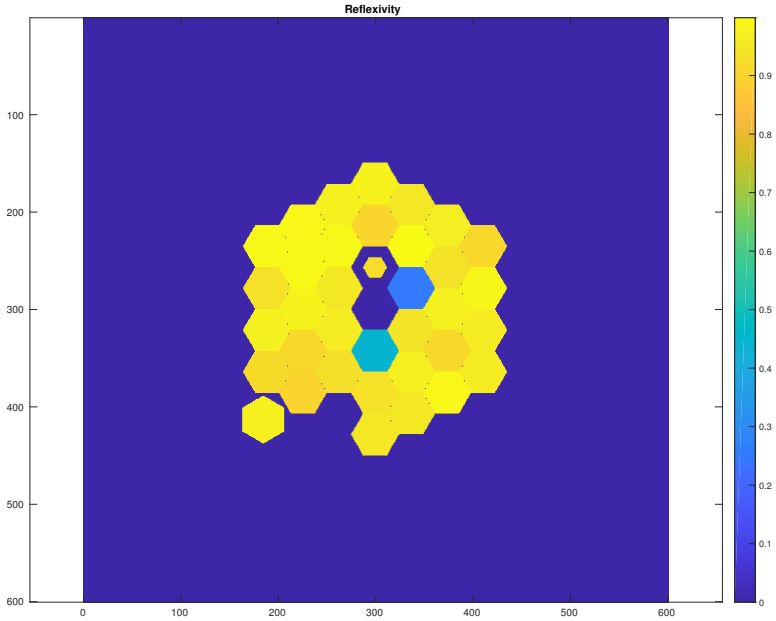
ZeroPad

```
P.zeroPad(2,1); % pupil will be 2x larger -> Nyquist sampling in frequency domainP.disp('r'
P.disp('r');
```



Resize

```
P.resize(600,1,'nearest'); % pupil will be resized at 300px, replaced in the object, using
P.disp('r');
```



3.2 Filling the gaps

As said above, the segment class does not produce good shape for meshing. Moreover, if coordinates are given in meters, the conversion meters-> pixels could produce rounding errors. Consequences : pixels may overlap, or create gaps between segments.

Solution : create ”optimized” coordinates in pixel for a given segment size. They must avoid overlapping pixels, but be as ”tight” as possible (= a few gaps as possible). The segments must be regulary placed. Example :

```
  1   2  3
1     0
2  0     0
3     0
4  0     0
5     0
```

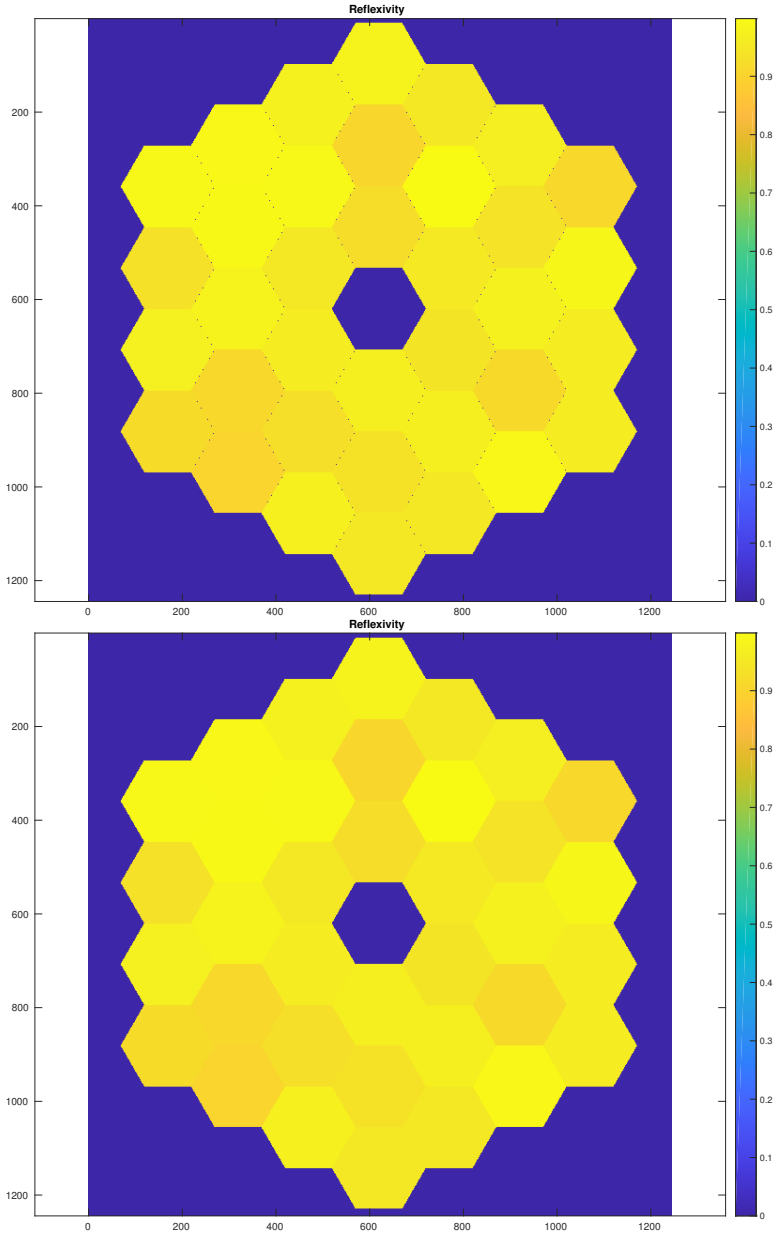
in the scheme above :

```
segment(2,3) = segment(2,1)+[0 segDiameter]
segment(4,3) = segment(4,1)+[0 segDiameter]
segment(4,1) = segment(2,1)+[segHeight 0]
segment(4,3) = segment(2,3)+[segHeight 0]
segment(1,2) = segment(2,1)+[segHeight/2  cos(30)*segDiameter]
segment(3,2) = segment(4,1)+[segHeight/2  cos(30)*segDiameter]
segment(5,2) = segment(4,1)+[segHeight/2  cos(30)*segDiameter]
```

Once it’s done, just add the parameter ”flagNoGap” at 1.

```
S=segment(6,0.9,200);
lambda=2.12e-6;
P1=pupil('segRef',S,'segCoord',SVpx,'wavelength',lambda,...
    'flagNoGap',0,'unit','px','coeffReflexion',R,'coeffPhaseModes',PE);

P2=pupil('segRef',S,'segCoord',SVpx,'wavelength',lambda,...
    'flagNoGap',1,'unit','px','coeffReflexion',R,'coeffPhaseModes',PE);

P1.disp('r');
P2.disp('r');
```

```
 @(zernike polynomials)> Created!
___ ZERNIKE POLYNOMIALS ___
 . 6 modes: 1,2,3,4,5,6
-------------------------------------------------
 @(zernike polynomials)> Terminated!
 @(zernike polynomials)> Created!
___ ZERNIKE POLYNOMIALS ___
 . 6 modes: 1,2,3,4,5,6
-------------------------------------------------
 @(zernike polynomials)> Terminated!
```



4 Application to KECK telescope

5 Conclusions