Synthetic centroid and CCD frame generation

https://doi.org/10.5281/zenodo.7016545

Fabrizio Pinto (https://osf.io/tehp2/)
Created on: 07 September 2014
Last revised on: 10 June 2022

This is ver. 1.0 of this document. Any needed updates will appear as separate, follow-up versions.

Table of Contents

Abstract

Back to Table of Contents

In this document, we describe the creation of an idealized synthetic centroid or arbitrary coordinates described by a simple Gaussian point-spread function (PSF) including signal noise. Finally we immerse such signal within an existing digital frame and we output the result to either a TIFF or a FITS file. This synthetic image is useful in testing spacecraft auto-

1. Introduction

Back to Table of Contents

In this Jupyter Notebook, we introduce the simulation of digital stellar images by randomly extracting electron counts in individual pixels according to a pre-assigned point-spread function (PSF). The subject of PSFs in telescopic imaging has a long history preceding the wide adoption of CCDs in astronomical observations (Moffat1969, King1971, Racine1996) and continuing to this day with characterization of images produced by contemporary advanced telescopes (Melchior2018).

This is the first draft of a document conceived for the simulation of CCD imaging in various applications, such as in adaptive optics, in imaging of asteroids from auto-navigating spacecraft, and in laser beam deflection simulations (Pinto2016). As such, it is intended to be a short proof-of-concept to establish a direction of study with further elaborations to follow. We shall introduce a simple Gaussian PSF (Trujillo2001b, Stallinga2010) and populate pixels with synthetic counts by drawing random events from a Poisson distribution with average equal to the value expected from the PSF. We conclude by immersing the synthetically generated image of an object, referred to as a *synthetic asteroid*, within the image of a star field obtained from SIMBAD. No effort is made in what follows to simulate a full frame (Merline1995) including bias, dark, flat field noise, so that only source noise is included.

The synthetic images produced herein can then be processed for determination of the celestial coordinates of the synthetic asteroid as described elsewhere (Pinto2022a) and as exemplified in a forthcoming final thesis student project (see Acknowledgments). Details about documents and support material for the installation of the **Free Wolfram Engine** used in the following Sections are also found in Pinto2022a (see Introduction therein) and in the Wikis associated to that OSF Project Component (https://osf.io/pwmtr/).

2. Natural constants and useful parameters

Back to Table of Contents

These quantities are not used in the present version of this Jupyter Notebook but are stored here for future use.

In [1]:
```
cLight = 2.99792458 10^8 ;
hBar = 1.054571817 10^(-34) ;
```

3. Flux calibration and centroid coordinates

Back to Table of Contents

Assume an assigned number of photons striking perpendicularly a sensor of infinite area during an assigned exposure time — the requirement of an infinite area is just a mathematically useful idealization exploited to obtain a proportionality constant. In this introductory document, we shall not consider spectral response and efficiency issues so that this integer photon count can be considered the total electron count produced in the sensor. We can start with the following:

In [1]:
```
NTotalPhotons = 10^(7);
```

As we shall see below, we shall place the centroid image on a region of the sensor of coordinates ($x_{\min}$, $x_{\max}$) and ($y_{\min}$, $y_{\max}$), that is, we shall assume the theoretical center-of-mass centroid coordinates to be ($x_{\mathrm{cm,th}} \in [x_{\min}, x_{\max}]$, $y_{\mathrm{cm,th}} \in [y_{\min}, y_{\max}]$).

In [2]:
```
xcmth = 50.0
ycmth = 50.0
```

Out[2]:
```
50.
50.
```

Notice that these coordinates do *not* have to necessarily be integers.

4. The Point-Spread Function

Back to Table of Contents

We shall assume the Point-Spread Function (PSF) to be of the Gaussian type:

In [4]:
```
PSFCentral[xcmth01_, ycmth01_, x_, y_, Gammax_, Gammay_] = Exp[-(((x - xcmth01)^2)
```

Out[4]:
$$e^{-\frac{(x-xcmth01)^2}{4\,Gammax^2}-\frac{(y-ycmth01)^2}{4\,Gammay^2}}$$

where the subscript *Central* indicates that this is description of the PDF near the center of the centroid. As we move away from the center-of-mass, the flux becomes so small as to correspond to an absolutely minuscule number, which would be as difficult to account for as it would be useless, given the Gaussian decay. Therefore we introduce a threshold past which the intensity of the source is assumed to be negligible. In practical cases, this value can be set by considerations of S/N ratio:

In [5]:
```
PSFThreshold = 10^(-6);
```

In [6]:
```
PSF[xcmth01_, ycmth01_, x_, y_, Gammax_, Gammay_] =If[PSFCentral[xcmth01, ycmth01,
```

Out[6]:
$$\mathrm{If}\left[e^{-\frac{(x-xcmth01)^2}{4\,Gammax^2}-\frac{(y-ycmth01)^2}{4\,Gammay^2}} > \frac{1}{1\,000\,000}, \mathrm{PSFCentral}[xcmth01, ycmth01, x, y, Gammax, Gammay], 0\right]$$

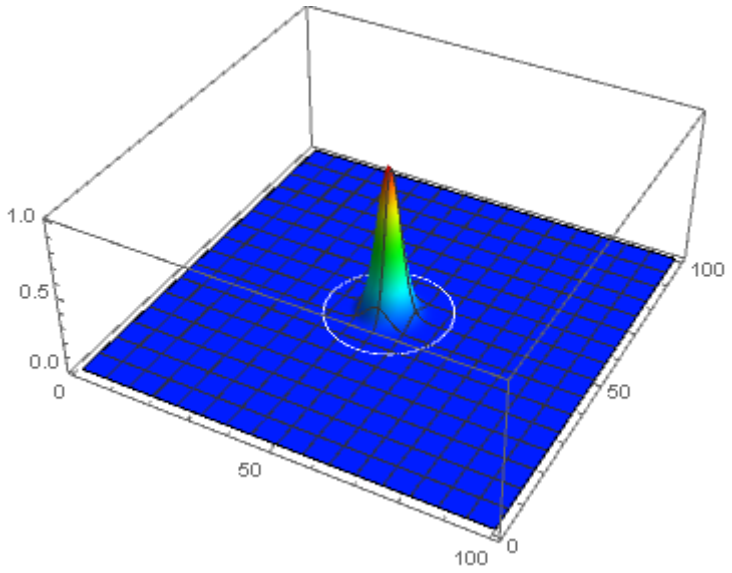
The following is a 3D plot of this PSF:

Let us choose:

In [9]:
```
Gammax0 = 2.0;
Gammay0 = 2.0;
```

In [11]:
```
PLOT1 = Plot3D [PSF[xcmth, ycmth, x, y, Gammax0, Gammay0],{x,1,100},{y,1,100},Plot
 ColorFunction -> Function[{x, y, z}, Hue[.65 (1 - z)]],PlotPoints->{200,200}]
```

Out[11]:



We now integrate over the entire plane to obtain the normalization constant connecting the PSF to the total charge count:

In [96]:
```
PSFInt[xcmth01_, ycmth01_, Gammax_, Gammay_] := NIntegrate[
 PSF[xcmth01, ycmth01, x, y, Gammax, Gammay], {x, -\[Infinity], \[Infinity]}, {y,
 Method -> {"GlobalAdaptive", "MaxErrorIncreases" -> 10000,
   Method -> "GaussKronrodRule"}, MaxRecursion -> 20,
 WorkingPrecision -> 18]
```

In [13]:
```
PSFInt[xcmth, ycmth, Gammax0, Gammay0]
```

Out[13]:
```
50.2654322007114981
```

The expression for the proportionality constant is:

In [14]:
```
Intensity0 [xcmth01_, ycmth01_, Gammax_, Gammay_]:= NTotalPhotons/PSFInt[xcmth01,
```

which, in our case, becomes:

In [15]:
```
Int0 = Intensity0 [xcmth, ycmth,Gammax0, Gammay0]
```

Out[15]:
```
198943.877774086897
```

Therefore, the theoretical photon flux on the sensor (photon count/unit area) becomes:

In [16]:
```
CCDIntensity [x_, y_]:= Int0 PSF[xcmth, ycmth, x, y, Gammax0, Gammay0]
```

For instance:

In [17]:
```
CCDIntensity [1, 1]
CCDIntensity [50, 50]
```

Out[17]:
```
0
198944.
```

Now we can verify that the integral of this flux over an infinite sensor area corresponds to the total number of photons. The cutoff we introduced for convenience produces no inaccuracy because of the rapid decay of the integrand:

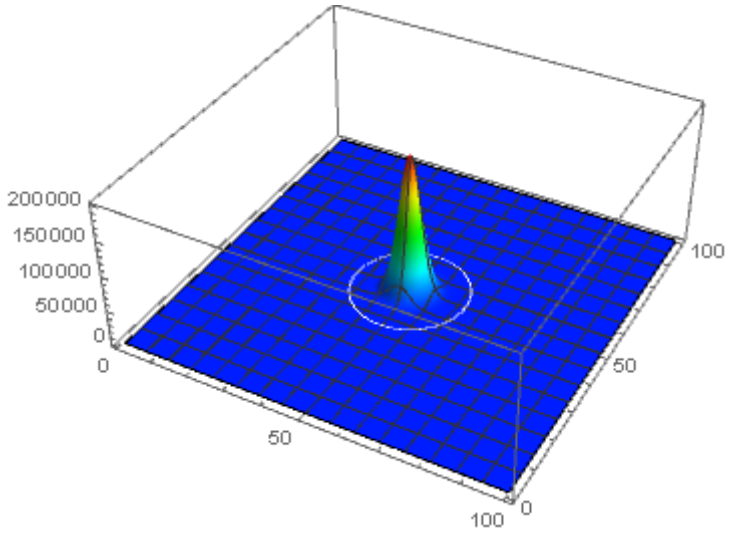
In [19]:
```
NIntegrate[
 CCDIntensity[x, y], {x, -\[Infinity], \[Infinity]}, {y, -\[Infinity], \[Infinity]
 Method -> {"GlobalAdaptive", "MaxErrorIncreases" -> 10000,
   Method -> "GaussKronrodRule"}, MaxRecursion -> 20,
 WorkingPrecision -> 18]
```

Out[19]:
$$1.0000000000000000 \times 10^{7}$$

as expected from our assumption above. The following is a 3D plot of this flux, where, on the z-axis, we have the average count per unit area corresponding to the total incident intensity:

In [20]:
```
PLOT2 =  Plot3D [CCDIntensity[x, y],{x,1,100},{y,1,100},PlotRange->All,
 ColorFunction -> Function[{x, y, z}, Hue[.65 (1 - z)]],PlotPoints->{200,200}]
```

Out[20]:



Notice that it is possible to have non-integer counts/pixel:

In [24]:
```
CCDIntensity[45, 45]
```

Out[24]:
```
8740.98
```

5. Generation of exportable synthetic centroid: Ideal case

Back to Table of Contents

Now we shall populate every pixel with a number of charges randomly extracted from a Poisson distribution consistent with the chosen PSF. In the following cell, as mentioned above, we first have a real number proportional to the average energy flux within the assigned pixel; secondly, we round that value to an integer; and, lastly, we obtain an integer number randomly drawn from a Poisson distribution of average equal to the second value below:

In [25]:
```
NIntegrate[
 CCDIntensity[x, y], {x, 40, 40+1}, {y, 50, 50+1}]

Round[NIntegrate[
 CCDIntensity[x, y], {x, 40, 40+1}, {y, 50, 50+1}]]

RandomVariate[PoissonDistribution[Abs[Round[NIntegrate[
 CCDIntensity[xpr, ypr], {xpr, 40, 40+1}, {ypr, 50, 50+1}]]]],1][[1]]
```

Out[25]:
```
729.34
729
744
```

Clicking the above cell repeatedly produces the same values for the same two quantities but it gives random values for the last one, as in a physical detector. However, we cannot simply proceed by assuming that the above result be the charge count per pixel because, at large distances from the center-of-mass, the intensity, rounded to an integer is equal to 0 and thus inappropriate to draw a random value from a Poisson distribution, as the warning issued by *Mathematica* indicates:

In [29]:
```
NIntegrate[
 CCDIntensity[x, y], {x, 35, 35+1}, {y, 50, 50+1}]

Round[NIntegrate[
 CCDIntensity[x, y], {x, 35, 35+1}, {y, 50, 50+1}]]

RandomVariate[PoissonDistribution[Abs[Round[NIntegrate[
 CCDIntensity[xpr, ypr], {xpr, 30, 30+1}, {ypr, 50, 50+1}]]]],1][[1]]
```

```
RandomVariate::posprm: Parameter 0 at position 1 in PoissonDistribution[0] is expe
cted to be positive.
```

Out[29]:
```
0.409739
0
PoissonDistribution[0]
```

In order to avoid this complication, we redefine the random pixel count as equal to 0 if the intensity, rounded to an integer, is equal to 0:

In [33]:
```
RandomPixelIntensity [i_, j_] := If[Abs[Round[NIntegrate[
 CCDIntensity[xpr, ypr], {xpr, i, i+1}, {ypr, j, j+1}]]] > 0,
RandomVariate[PoissonDistribution[Abs[Round[NIntegrate[
 CCDIntensity[xpr, ypr], {xpr, i, i+1}, {ypr, j, j+1}]]]],1][[1]], 0]
```

In [34]:
```
RandomPixelIntensity [50, 50]
RandomPixelIntensity [35, 50]
```

Out[34]:
```
191555
0
```

In [43]:
```
RandomPixelIntensity[40, 40]
```

Out[43]:
```
1
```

Let us now populate a table corresponding to a small square area centered around the center-of-mass of the centroid.

In order to speed up the computation, let us first illustrate the process by choosing a very

small area $\Delta x = \Delta y = 5$ pix:

In [44]:
```
Deltax = 5;
Deltay = 5;

xmin = IntegerPart[xcmth - Deltax]
xmax = IntegerPart[xcmth + Deltax]
ymin = IntegerPart[ycmth - Deltay]
ymax = IntegerPart[ycmth + Deltay]
```

Out[44]:
```
45
55
45
55
```

Since we will be writing the results to 16 bit depth files, we shall introduce "white clipping" to simulate pixel saturation. All pixels with reading larger than, in this case, $2^{16} - 1$, shall be set to read $2^{16} - 1$. The image can be set to be unaffected by saturation by decreasing the number of photons incident upon the sensor, which corresponds to decreasing the luminosity of the source (i.e. a laser) or to decreasing the exposure time.

In [50]:
```
(*We omit the gain as we assume CCDGain = 1.0*)

GrayLevels = 2^(16) - 1
```

Out[50]:
```
65535
```

In [52]:
```
Clear[OBDspotSyntheticPixels1]
```

The [[1]] component is the CPU time required by the calculation.

In [53]:
```
OBDspotSyntheticPixels1 = Timing[Table[CCDRV = RandomPixelIntensity [i, j]; If [CC
OBDspotSyntheticPixels1[[1]]
```

Out[53]:
```
41.7813
```

The second component of OBDspotSyntheticPixels1[[2]] contains the simulated data. For instance, the first pixel reads:

In [56]:
```
N[OBDspotSyntheticPixels1[[2]][[1,1]]]
```

Out[56]:
```
15734.
```

Near the center, on the other hand:

In [55]:
```
N[OBDspotSyntheticPixels1[[2]][[3,3]]]
```

Out[55]:
```
65535.
```

5.1 Example 1

Back to Table of Contents

Let us now repeat this process by choosing a larger sensor area:

In [316…
```
Clear[OBDspotSyntheticPixels1]
```

In [373…
```
Deltax = 15;
Deltay = 15;


xmin = IntegerPart[xcmth - Deltax]
xmax = IntegerPart[xcmth + Deltax]
ymin = IntegerPart[ycmth - Deltay]
ymax = IntegerPart[ycmth + Deltay]
```

Out[373…
```
35
65
35
65
```

Attention: This calculation may require 10-20 minutes and that time obviously increases quadratically with the sides of the sensor:

In [317…
```
OBDspotSyntheticPixels1 = Timing[Table[CCDRV = RandomPixelIntensity [i, j]; If [CC
OBDspotSyntheticPixels1[[1]]
```

Out[317…
```
341.734
```

The synthetic image can now be prepared to be evaluated:

In [321…
```
OBDspotSyntheticImage1 = Image[OBDspotSyntheticPixels1[[2]]]
```

Out[321…

We are now ready to export this image to FITS (the name and folder locations can be changed as desired):

In [322…
```
Export["C:/DataFiles/Simulations/fabrizio-pinto-centroid-1.fits"
 , OBDspotSyntheticImage1]
```

Out[322…
```
C:/DataFiles/Simulations/fabrizio-pinto-centroid-1.fits
```

Notice that **ASTAP** detects the output file as a signed 64 bit format. However, the file can be immediately resaved to 16 bit format without detectable change.

Let us extract some useful information about the sensor area under study:

In [347…
```
xminImage = 1
xmaxImage = ImageDimensions[OBDspotSyntheticImage1][[1]]
yminImage = 1
ymaxImage = ImageDimensions[OBDspotSyntheticImage1][[2]]

Max[ImageData[OBDspotSyntheticImage1][[yminImage ;; ymaxImage,
     xminImage ;; xmaxImage ]]]

Min[ImageData[OBDspotSyntheticImage1][[yminImage ;; ymaxImage,
     xminImage ;; xmaxImage ]]]
```

Out[347…
```
1
31
1
31
65535.
0.
```

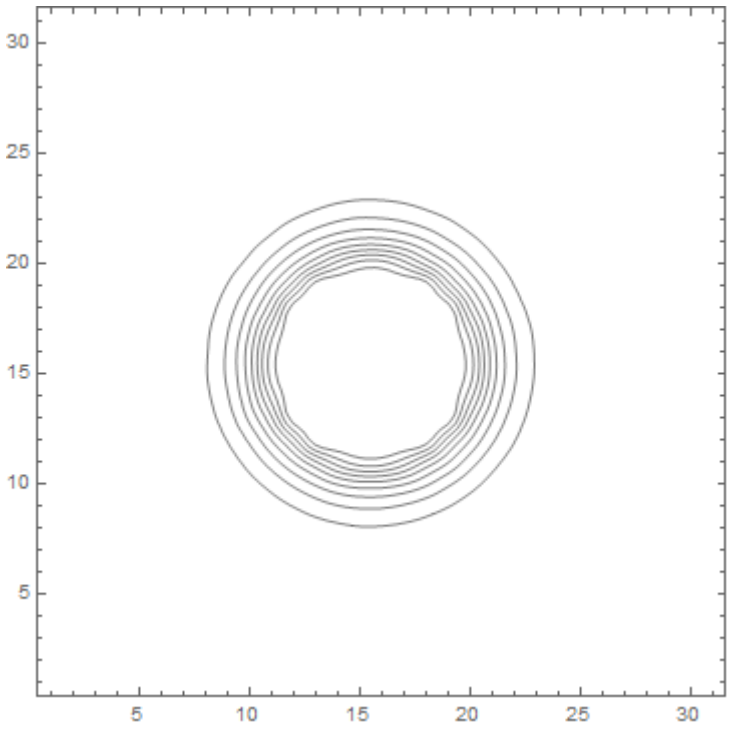
We do not discuss the issue of establishing a threshold here, which is dealt with elsewhere (Pinto2022a):

In [328…
```
PixelThreshold = 0.0 ;
```

The contour plot is:

In [353…
```
OBDspotSyntheticImageRenorm1 =
 ListContourPlot[
  ImageData[OBDspotSyntheticImage1][[yminImage ;; ymaxImage, xminImage ;; xmaxImag
   Max[ImageData[OBDspotSyntheticImage1][[yminImage ;; ymaxImage,
     xminImage ;; xmaxImage ]]], Contours -> Automatic, ContourStyle -> Black,
  ContourShading -> None, InterpolationOrder -> 2,
  PlotRange -> {Full, Full, {PixelThreshold, 1.0}}]
```

Out[353…

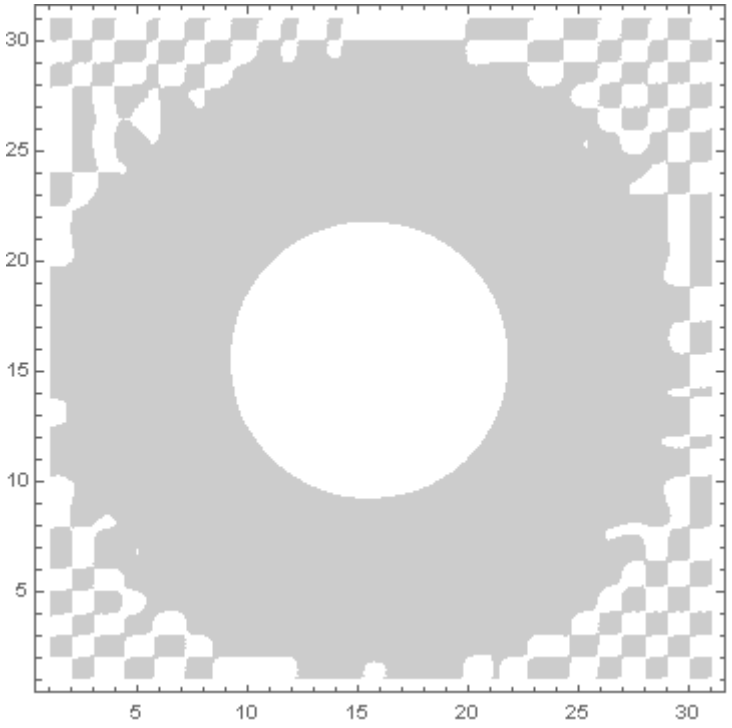


The density plot is:

In [354…

```
OBDScience1DP =
  ListDensityPlot[
   ImageData[OBDspotSyntheticImage1][[yminImage ;; ymaxImage, xminImage ;; xmaxIma
   InterpolationOrder -> 2, ColorFunction -> GrayScale,
   PlotRange -> {Full,
     Full, Automatic}]
```

Out[354…

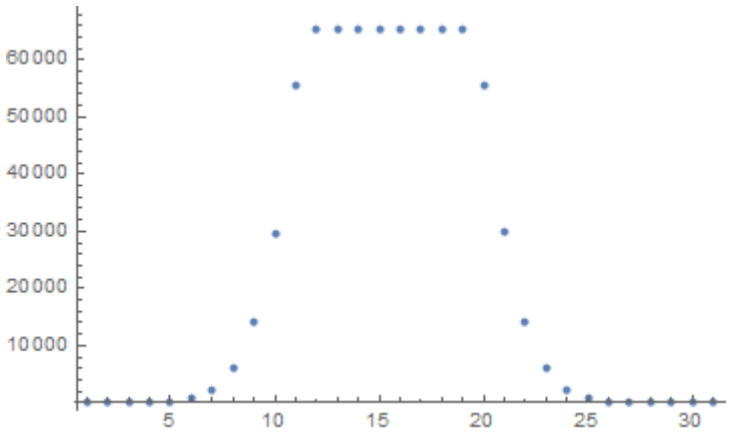


Important to the determination of the centroid coordinates is a scan of the sensor across either axis:

In [355…

```
OBDspotxCross =
 ListPlot[ImageData[OBDspotSyntheticImage1][[15, xminImage ;; xmaxImage ]],
  PlotRange -> All]
```

Out[355…



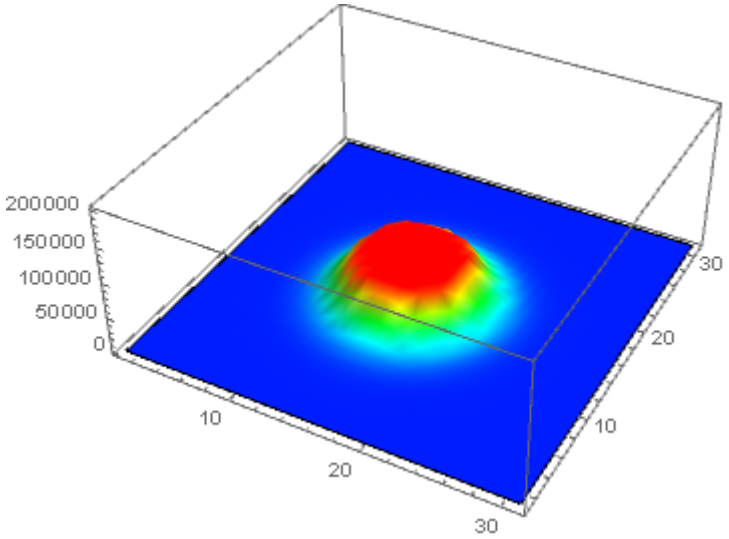
This result clearly shows the saturation at the center.

The 3D plot of the centroid area is:

In [356…

```
OBDspotxy3D =
 ListPlot3D[ImageData[OBDspotSyntheticImage1][[yminImage ;; ymaxImage, xminImage
  PlotRange -> {All,All, {0,200000}},Mesh->None,
 ColorFunction -> Function[{x, y, z}, Hue[.65 (1 - z)]]]
```

Out[356…

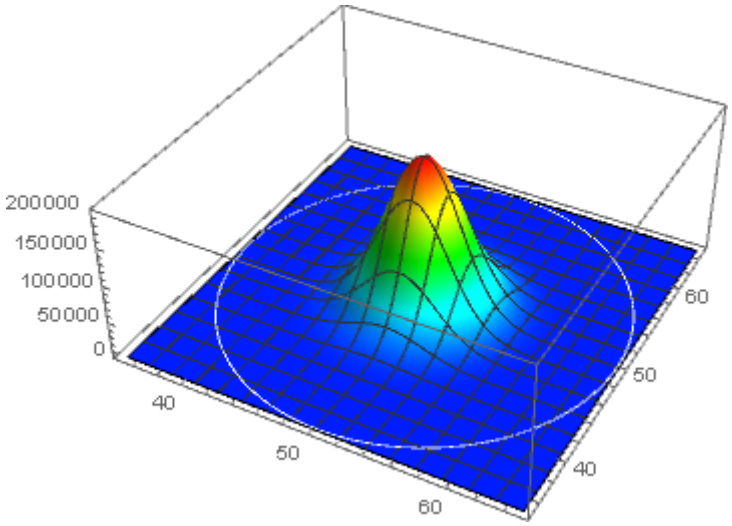


It is instructive to compare the theoretical count obtained from the PSF to the randomly drawn result including saturation:

In [379…

```
PLOT3 =  Plot3D [CCDIntensity[x, y],{x,xmin,xmax},{y,ymin,ymax},PlotRange->{All,A
 ColorFunction -> Function[{x, y, z}, Hue[.65 (1 - z)]],PlotPoints->{200,200}]
```

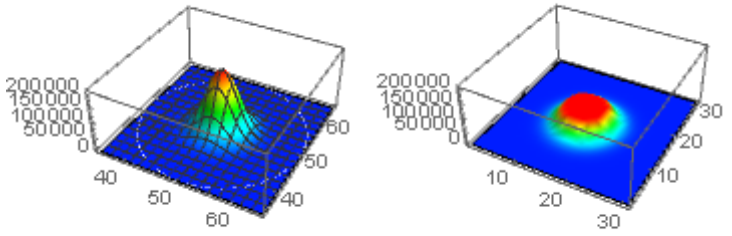
Out[379…



In [380…

```
GraphicsRow[{PLOT3,OBDspotxy3D}]
```

Out[380…



5.2 Example 2

Back to Table of Contents

It is useful to repeat all steps above for a total flux that avoids saturation. For instance:

In [448…

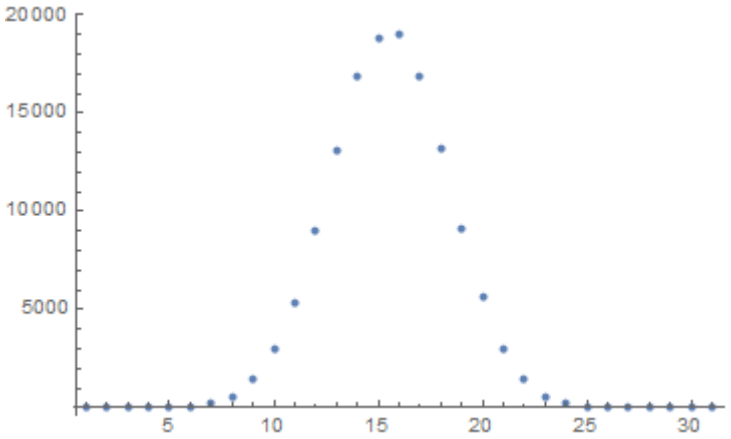
```
NTotalPhotons = 10^(6);

Intensity0 [xcmth01_, ycmth01_, Gammax_, Gammay_]:= NTotalPhotons/PSFInt[xcmth01,

Gammax0 = 2.0;
Gammay0 = 2.0;

Int0 = Intensity0 [xcmth, ycmth,Gammax0, Gammay0]

CCDIntensity [x_, y_]:= Int0 PSF[xcmth, ycmth, x, y, Gammax0, Gammay0]

NIntegrate[
 CCDIntensity[x, y], {x, -\[Infinity], \[Infinity]}, {y, -\[Infinity], \[Infinity]
 Method -> {"GlobalAdaptive", "MaxErrorIncreases" -> 10000,
   Method -> "GaussKronrodRule"}, MaxRecursion -> 20,
 WorkingPrecision -> 18]

RandomPixelIntensity [i_, j_] := If[Abs[Round[NIntegrate[
 CCDIntensity[xpr, ypr], {xpr, i, i+1}, {ypr, j, j+1}]]] > 0,
RandomVariate[PoissonDistribution[Abs[Round[NIntegrate[
 CCDIntensity[xpr, ypr], {xpr, i, i+1}, {ypr, j, j+1}]]]],1][[1]], 0]

Clear[OBDspotSyntheticPixels1]

Deltax = 15;
Deltay = 15;

xmin = IntegerPart[xcmth - Deltax]
xmax = IntegerPart[xcmth + Deltax]
ymin = IntegerPart[ycmth - Deltay]
ymax = IntegerPart[ycmth + Deltay]

GrayLevels = 2^(16) - 1

OBDspotSyntheticPixels2 = Timing[Table[CCDRV = RandomPixelIntensity [i, j]; If [CC
OBDspotSyntheticPixels2[[1]]

OBDspotSyntheticImage2 = Image[OBDspotSyntheticPixels2[[2]]]

Export["C:/DataFiles/Simulations/fabrizio-pinto-centroid-2.fits"
 , OBDspotSyntheticImage2]

xminImage = 1
xmaxImage = ImageDimensions[OBDspotSyntheticImage2][[1]]
yminImage = 1
ymaxImage = ImageDimensions[OBDspotSyntheticImage2][[2]]

Max[ImageData[OBDspotSyntheticImage2][[yminImage ;; ymaxImage,
     xminImage ;; xmaxImage ]]]

Min[ImageData[OBDspotSyntheticImage2][[yminImage ;; ymaxImage,
     xminImage ;; xmaxImage ]]]

OBDspotxCross =
 ListPlot[ImageData[OBDspotSyntheticImage2][[15, xminImage ;; xmaxImage ]],
  PlotRange -> All]

OBDspotxy3D =
 ListPlot3D[ImageData[OBDspotSyntheticImage2][[yminImage ;; ymaxImage, xminImage
  PlotRange -> {All,All, {0,20000}},Mesh->None,
 ColorFunction -> Function[{x, y, z}, Hue[.65 (1 - z)]]]

PLOT3 = Plot3D [CCDIntensity[x, y] {x,xmin,xmax} {y,ymin,ymax} PlotRange->{All,Al
```

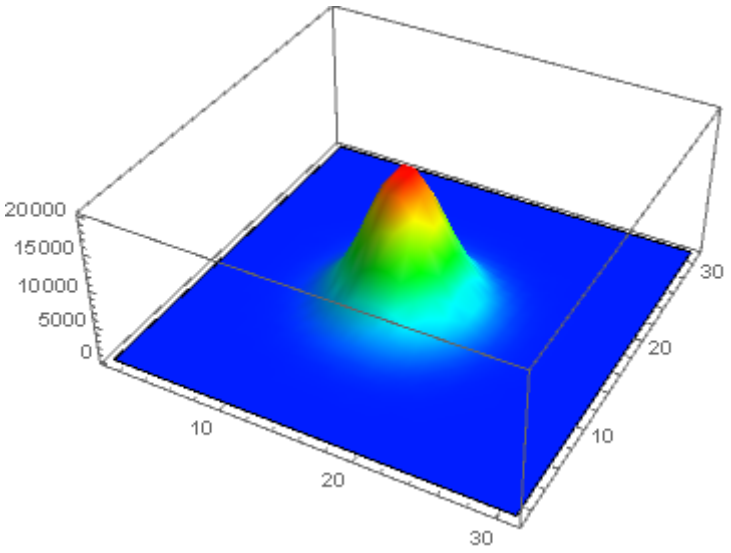
```
PLOT3 = Plot3D [ccDintensity[x, y],{x,xmin,xmax},{y,ymin,ymax},PlotRange->{All,A
 ColorFunction -> Function[{x, y, z}, Hue[.65 (1 - z)]],PlotPoints->{200,200}]

GraphicsRow[{PLOT3,OBDspotxy3D}]
```

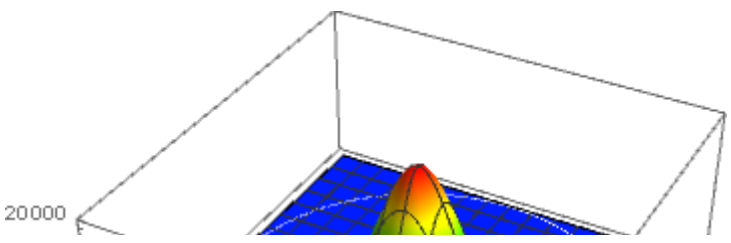
Out[448…

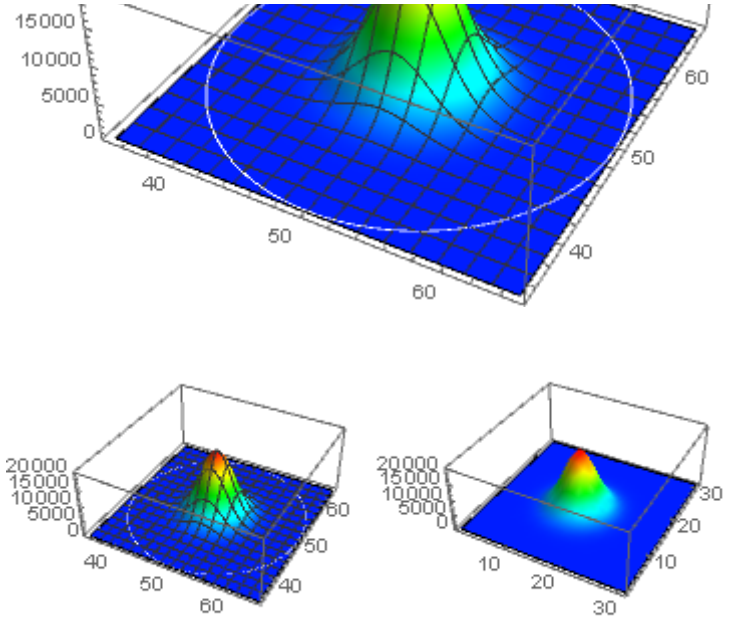
```
19894.3877774086897
1.0000000000000000×10^6
35
65
35
65
303.156
```

```
C:/DataFiles/Simulations/fabrizio-pinto-centroid-2.fits
1
31
1
31
19256.
0.
```









5.3 Example 3

Back to Table of Contents

As a final preparatory step for the activity below, let us generate a new centroid image in 8 bit format. This requires to again decrease the intensity of the source by a factor $\sim 2^{16}/2^8 = 256$, that is:

In [83]:

```
NTotalPhotons = 10^(4);

PSFCentral[xcmth01_, ycmth01_, x_, y_, Gammax_, Gammay_] = Exp[-(((x - xcmth01)^2)

PSFThreshold = 10^(-6);

PSF[xcmth01_, ycmth01_, x_, y_, Gammax_, Gammay_] = If[PSFCentral[xcmth01, ycmth01

PSFInt[xcmth01_, ycmth01_, Gammax_, Gammay_] := NIntegrate[
 PSF[xcmth01, ycmth01, x, y, Gammax, Gammay], {x, -\[Infinity], \[Infinity]}, {y,
 Method -> {"GlobalAdaptive", "MaxErrorIncreases" -> 10000,
   Method -> "GaussKronrodRule"}, MaxRecursion -> 20,
 WorkingPrecision -> 18]

Gammax0 = 2.0;
Gammay0 = 2.0;

PSFInt[xcmth, ycmth, Gammax0, Gammay0]
```

Out[83]:

$$e^{-\frac{(x-xcmth01)^2}{4\,Gammax^2}-\frac{(y-ycmth01)^2}{4\,Gammay^2}}$$

$$\text{If}\left[e^{-\frac{(x-xcmth01)^2}{4\,Gammax^2}-\frac{(y-ycmth01)^2}{4\,Gammay^2}} > \frac{1}{1\,000\,000},\ \text{PSFCentral}[xcmth01, ycmth01, x, y, Gammax, Gammay], 0\right]$$

50.2654322007114981

In [91]:

```
Intensity0 [xcmth01_, ycmth01_, Gammax_, Gammay_]:= NTotalPhotons/PSFInt[xcmth01,

Int0 = Intensity0 [xcmth, ycmth,Gammax0, Gammay0]

CCDIntensity [x_, y_]:= Int0 PSF[xcmth, ycmth, x, y, Gammax0, Gammay0]

NIntegrate[
 CCDIntensity[x, y], {x, -\[Infinity], \[Infinity]}, {y, -\[Infinity], \[Infinity]
 Method -> {"GlobalAdaptive", "MaxErrorIncreases" -> 10000,
   Method -> "GaussKronrodRule"}, MaxRecursion -> 20,
 WorkingPrecision -> 18]
```

Out[91]:

```
198.943877774086897
10000.0000000000000
```

In [40]:

```
RandomPixelIntensity [i_, j_] := If[Abs[Round[NIntegrate[
 CCDIntensity[xpr, ypr], {xpr, i, i+1}, {ypr, j, j+1}]]] > 0,
RandomVariate[PoissonDistribution[Abs[Round[NIntegrate[
 CCDIntensity[xpr, ypr], {xpr, i, i+1}, {ypr, j, j+1}]]]],1][[1]], 0]

Clear[OBDspotSyntheticPixels1]

Deltax = 15;
Deltay = 15;

xmin = IntegerPart[xcmth - Deltax]
xmax = IntegerPart[xcmth + Deltax]
ymin = IntegerPart[ycmth - Deltay]
ymax = IntegerPart[ycmth + Deltay]

GrayLevels = 2^(8) - 1

OBDspotSyntheticPixels3 = Timing[Table[CCDRV = RandomPixelIntensity [i, j]; If [CC
OBDspotSyntheticPixels3[[1]]
```

Out[40]:

```
35
65
35
65
255
231.375
```

In [53]:

```
OBDspotSyntheticImage3 = Image[OBDspotSyntheticPixels3[[2]]]

Export["C:/DataFiles/Simulations/fabrizio-pinto-centroid-3.fits"
 , OBDspotSyntheticImage3]

xminImage = 1
xmaxImage = ImageDimensions[OBDspotSyntheticImage3][[1]]
yminImage = 1
ymaxImage = ImageDimensions[OBDspotSyntheticImage3][[2]]

Max[ImageData[OBDspotSyntheticImage3][[yminImage ;; ymaxImage,
     xminImage ;; xmaxImage ]]]

Min[ImageData[OBDspotSyntheticImage3][[yminImage ;; ymaxImage,
     xminImage ;; xmaxImage ]]]

OBDspotxCross =
 ListPlot[ImageData[OBDspotSyntheticImage3][[15, xminImage ;; xmaxImage ]],
  PlotRange -> All]

OBDspotxy3D =
 ListPlot3D[ImageData[OBDspotSyntheticImage3][[yminImage ;; ymaxImage, xminImage
  PlotRange -> {All,All, {0,300}},Mesh->None,
 ColorFunction -> Function[{x, y, z}, Hue[.65 (1 - z)]]]

PLOT4 =  Plot3D [CCDIntensity[x, y],{x,xmin,xmax},{y,ymin,ymax},PlotRange->{All,Al
 ColorFunction -> Function[{x, y, z}, Hue[.65 (1 - z)]],PlotPoints->{200,200}]

GraphicsRow[{PLOT4,OBDspotxy3D}]
```

Out[53]:

C:/DataFiles/Simulations/fabrizio-pinto-centroid-3.fits

1

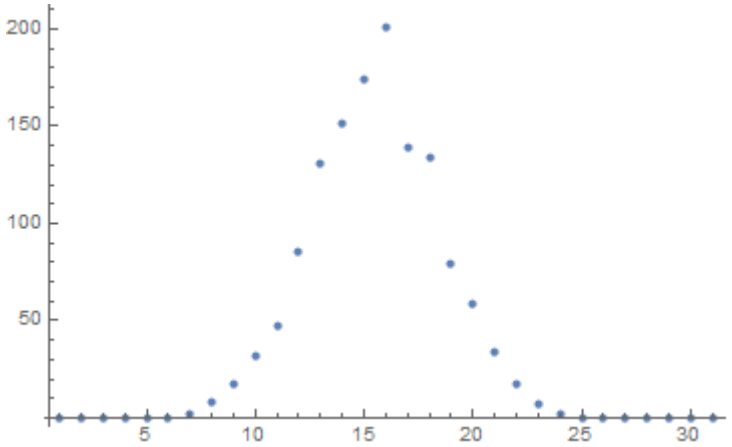
31

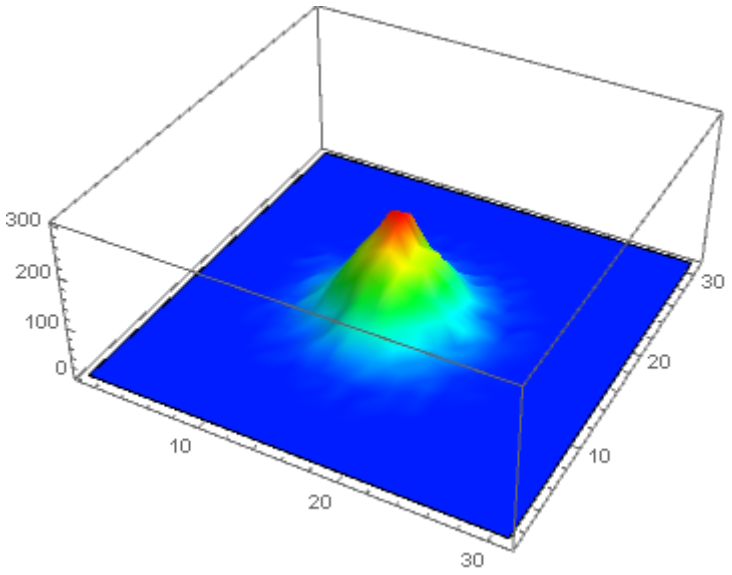
1

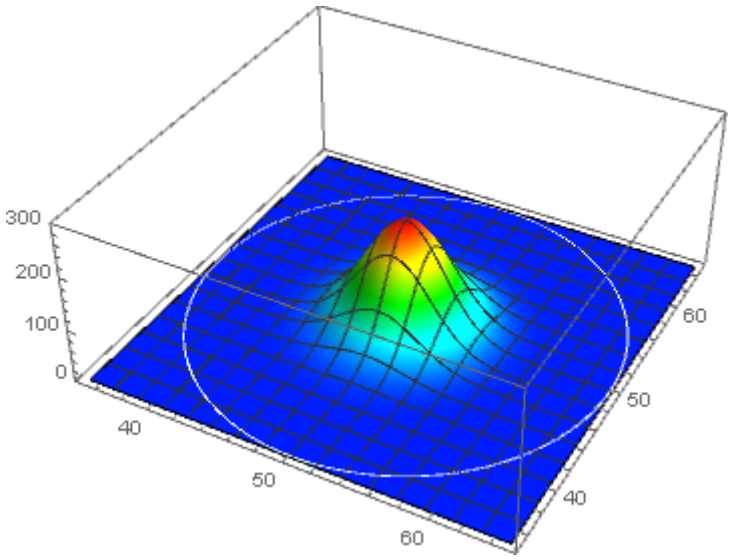
31

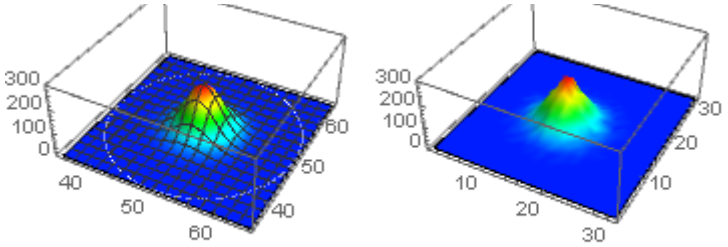
213.

0.









Due to the lower photon count/pixel, the source noise in this case is far more appreciable. In practice, one would acquire many images under the same conditions and *stack* them, for instance by means of **Deep Sky Stacker**. Here we shall not simulate this computing intensive process.

6. Overlaying the synthetic image onto a target image

Back to Table of Contents

Now carry out a similar procedure with the final goal of creating a synthetic image of an asteroid at a hypothetical as seen from the Earth. The plan is to locate the desired star field in **SIMBAD**, download that image, and place the *synthetic asteroid* at a specific position in that field. Note that we shall not concern ourselves with correctly simulating the magnitude of the object, which would require a photometric calibration, but only its position.

Important: We must always remember to clearly label all images of the real sky that have been altered for the purposes of simulation by saving such images to FITS with a dedicated FITS header and comments. It is *crucial* that no astronomical image modified for testing purposes be circulated without a clear indication that real data were purposefully altered for simulation purposes. Doing otherwise is both unethical and counterproductive to the research process. Add a descriptive COMMENT in the FITS header or other metadata field if possible.

The test field chosen is centered upon the star **HIP 42840**:

$\alpha_{\text{SIMBAD}} = 08\text{h}\,43\text{m}\,45.5623596864\text{s}$

$\delta_{\text{SIMBAD}} = +22^\circ\,42'\,47.881040724''$ (Optical)

A star field was downloaded from **SIMBAD**, which provides 8 bit images. This image was uploaded to nova.astrometry.net in order to be solved. Astrometry.net quickly reported *Success*, identifying the center of the star field as located at:

$\alpha_{\text{AS}} = 08\text{h}\,43\text{m}\,54.629\text{s}$

$\delta_{AS} = +22^\circ\ 42'\ 26.207''$ (Optical)

and producing a `new-image` FITS image (`BITPIX  = 8 / bits per pixel`) to be opened in **ASTAP**. Finally, that image was made monochrome and re-saved to FITS as shown below:

In [69]:
```
RealImage1 =
Import["C:/DataFiles/Simulations/fabrizio-pinto-simbad-HIP-42840-mono.fits", "FITS
```

Out[69]:

Let us extract the pixel size of the image so as to be able to properly position the synthetic asteroid:

In [70]:
```
ImageDimensions[RealImage1]
```

Out[70]:
```
{1349, 657}
```

In order to better view the process, we shall resize the syntetic image to a much larger size:

In [71]:
```
OBDSI3 = ImageResize[OBDspotSyntheticImage3, {500,500}]
```

Out[71]:

In [72]:
```
ImageDimensions[OBDSI3]
```

Out[72]:
```
{500, 500}
```

Now we can overlay this image onto the full frame at specific pixel coordinates. Let us first place the synthetic asteroid at the center of the frame. The `Floor` function provides the integer equal to or smaller than its argument, so as to avoid addressing a decimal pixel position.

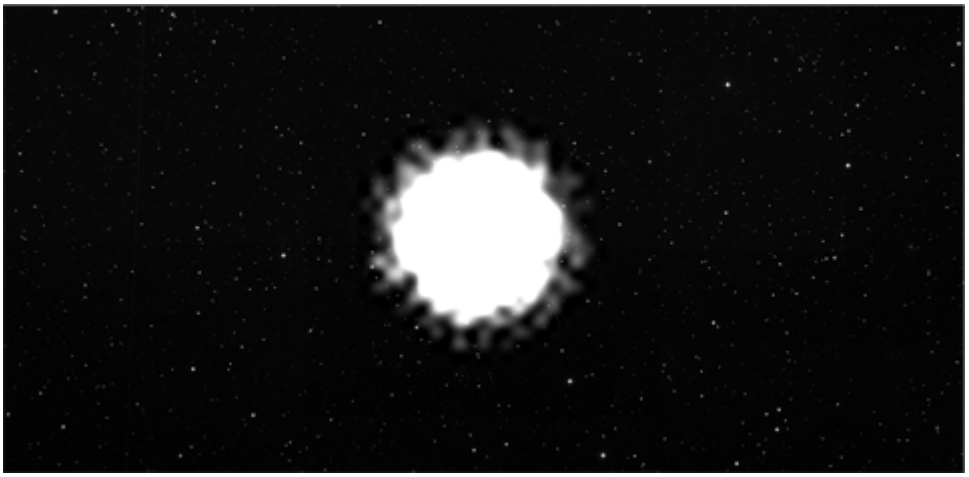
In [73]:
```
SynthAstX = Floor[(ImageDimensions[RealImage1][[1]] / 2)]
SynthAstY = Floor[(ImageDimensions[RealImage1][[2]] / 2)]
```

Out[73]:
```
674
328
```

We can now lay the asteroid synthetic image onto the SIMBAD star field:

In [75]:
```
SynthRealImageFull = ImageCompose[RealImage1, {OBDSI3, 0.1},{SynthAstX,SynthAstY}]
```

Out[75]:



As a proof of feasibility, we can now export this to TIFF:

In [76]:
```
Export["C:/DataFiles/Simulations/fabrizio-pinto-SynthRealImageFull-1.tif", SynthRe
```

Out[76]:
```
C:/DataFiles/Simulations/fabrizio-pinto-SynthRealImageFull-1.tif
```

Note: At this time, the output file must be reopened (for instance, in **GIMP**) and simply resaved in order to be used by **ASTAP**.

We can now repeat this example with a much smaller synthetic asteroid and better blending parameter. Obviously, scaling the entire subframe retains the same shape of the centroid whereas the parameters $\sigma_{X,Y}$ could be changed while keeping the same subframe size. This issue is connected to the process of reproduction of the PSF in the main star field frame, which may even include reproducing *spike* features and other artifacts that appear in SIMBAD images (see References).

In [77]:
```
OBDSI4 = ImageResize[OBDspotSyntheticImage3, {10,10}]
```

Out[77]:

In [78]:
```
ImageDimensions[OBDSI4]
```

Out[78]:
```
{10, 10}
```

In [79]:
```
SynthAstX = Floor[(ImageDimensions[RealImage1][[1]] / 2)]
SynthAstY = Floor[(ImageDimensions[RealImage1][[2]] / 2)]
```

Out[79]:
```
674
328
```

In [81]:
```
SynthRealImageFull2 = ImageCompose[RealImage1, {OBDSI4, 0.1},{SynthAstX,SynthAstY]
```

Out[81]:

In [82]:
```
Export["C:/DataFiles/Simulations/fabrizio-pinto-SynthRealImageFull-2.tif", SynthRe
```

Out[82]:
```
C:/DataFiles/Simulations/fabrizio-pinto-SynthRealImageFull-2.tif
```

This image can be used in all activities conducted in Pinto2022a, including obtaining an initial calibration by means of **Astrometry.net**.

Acknowledgments

Back to Table of Contents

It is my pleasure to thank my former student at the Izmir University of Economics, Ms. Hazal Karaaliler, for contributing to the validation and beta testing phases of the first version of this document and for reviewing the associated Wikis ahead of public release. An early version of this Jupyter Notebook was also employed as a computational tool in a final thesis project Ms. Karaaliler (https://osf.io/23jgn/) co-authored with her colleagues, Mr. Yagız Akan (https://osf.io/zt9jb/), Ms. Deniz Lena Demirbağ (https://osf.io/uwsqy/), and Ms. İlayda Macit (https://osf.io/kesr7/). That thesis, "Autonomous optical and inertial navigation of solar-sail propelled CubeSat class spacecraft during targeting missions to asteroids and minor moons," is also publicly available.

References

Back to Table of Contents

King1971
I. R. King, "The profile of a star image," *PASP*, **83**, 199-201 (1971).
https://iopscience.iop.org/article/10.1086/129100/pdf

Melchior2018
P. Melchior et al., "In the Crosshair: Astrometric Exoplanet Detection with WFIRST's Diffraction Spikes," *Astron. J.*, **155**, 102 (2018).
https://iopscience.iop.org/article/10.3847/1538-3881/aaa422/pdf

Merline1995
W. J. Merline and S. B. , "A realistic model for point-sources images on array detectors: The model and initial results," *Experimental Astronomy*, **6**, 163-210 (1995).

https://link.springer.com/article/10.1007/BF00421131

Moffat1969
A. F. J. Moffat, "A theoretical investigation of focal stellar images in the photographic emulsion and application to photographic photometry," *Astron. Astrophys.*, **3**, 455-461 (1969). https://articles.adsabs.harvard.edu/cgi-bin/nph-iarticle_query?1969A%26A.....3..455M&defaultprint=YES&filetype=.pdf

Pinto2016
Fabrizio Pinto, "Demonstration of biased membrane static figure mapping by optical beam subpixel centroid shift," *AIP Conf. Proc.*, **1742**, 030014 (2016). https://aip.scitation.org/doi/10.1063/1.4953135

Pinto2022a
Fabrizio Pinto, "Introduction to asteroid detection, centroid characterization, and orbit determination," Zenodo. https://doi.org/10.5281/zenodo.7006675 . See also https://osf.io/pwmtr/ and observational data at https://osf.io/pu5g9/.

Racine1996
R. Racine, "The Telescopic Point-Spread Function," *PASP*, **108**, 699-705 (1996). https://adsabs.harvard.edu/full/1996PASP..108..699R

Stallinga2010
S. Stallinga and B. Rieger, "Accuracy of the Gaussian Point Spread Function model in 2D localization microscopy," *Opt. Expres.*, **18**, 24461-24476 (2010). https://opg.optica.org/oe/fulltext.cfm?uri=oe-18-24-24461&id=207144

Trujillo2001b
I. Trujillo et al. "The effects of seeing on Sérsic profiles – II . The Moffat PSF," *MNRAS*, **328**, 977-985 (2001b). https://academic.oup.com/mnras/article/328/3/977/1247204

Back to Table of Contents