GIMP plugin astronomy 0.9

Georg Hennig
georg.hennig@web.de

December 7, 2014

Contents

1 Installation

1.1 Compile

1.1.1 Unix

Usually, it is enough to run

```
tar -xvjf gimp-plugin-astronomy-VERSION.tar.bz2
cd gimp-plugin-astronomy-VERSION
./configure --prefix=/usr
make
su
<your root password>
make install
```

You can also skip the *make install* part, and copy the plugins (manually) to the

```
~/.gimp-2.4/plug-ins
```

folder, and the scripts to the

```
~/.gimp-2.4/scripts
```

folder.

GIMP plugin astronomy requires the development packages of GIMP and Gtk+
and their dependencies, and additionally the development packages of *fftw* (Fastest
Fourier Transform in the West, version 3, `http://www.fftw.org`) and *GSL* (GNU
scientific library, `http://www.gnu.org/software/gsl/`). Often these pack-
ages are called *gimp-VERSION-dev*, *gsl-VERSION-dev*. Of course, you'll need a C
development environment, which means *make* and *gcc* plus dependencies like *gtk*-
dev*, *glib*-dev*, etc.

1.1.2 Windows

For building GIMP plugin astronomy for Windows (using MinGW), you need

- A working *MinGW compiler*

- *make*

- *GIMP* development environment (headers and libraries), including its de-
  pendencies

- *GSL* library. You can compile it yourself, for example with these settings:

  ```
  CFLAGS="-D_X86_=1 -Dtry=__try -Dexcept=__except -DWIN32 -D_WIN32
          -DGSL_DLL -DDLL_EXPORT -O3 -W -Wall -I.. -I../cblas"
  LDFLAGS="-mwindows"
  ```

- *FFTW*, version 3. (*libfftw3-3.dll* and *fftw3.h*)

1.2 Windows binaries

To install gimp-plugin-astronomy on Windows systems, copy all files in the folder
plugins (all *astronomy-*.exe* and **.dll* files) to

```
c:\Documents and Settings\<User>\.gimp-2.4\plugins
```

and all files inside *scripts* (all *.scm* files) to

```
c:\Documents and Settings\<User>\.gimp-2.4\scripts
```

To have translated messages, copy the folder *lib* into

```
c:\Documents and Settings\<User>\.gimp-2.4\plugins
```

If building from sources, copy the file *language.gmo* to

```
lib/.../locale/<language>/LC_MESSAGES/gimp-plugin-astronomy.mo
```

The plugins are available in the GIMP image window under *Filters → Astronomy → plugin name.*

2 Scripts

2.1 Background gradient batch

To batch normalize the background of all images in a folder. Run on the command line something like

```
gimp -i -b '(background-gradient-batch "*.jpg" 200 200 2 1.0 200 0 0)'
  -b '(gimp-quit 0)'
```

Parameters are: image list, box width, box height, numbers of iteration of sigma clipping, sigma, minimum number of values inside a box, weighted values (0=all values,1=darkest,2=brightest), percentage of darkest or brightest values. Look at background gradient plugin description below to see, what these settings mean.

2.2 Brightness contrast batch

This script sets brightness and contrast to all layers; it doesn't provide a preview as the normal brightness and contrast plugin.

2.3 Set mode batch

A simple batch script to set overlay mode to all layers.

2.4 Border information

Draws a border (up to 2 different colors) around your images and on the bottom some information about your image, up to 2 lines at up to 3 positions (aligned left, right and centered).
Example:

2.5 Dark subtraction

Subtracts a dark frame from all layers; naturally the dark frame should have the same size as all layers.
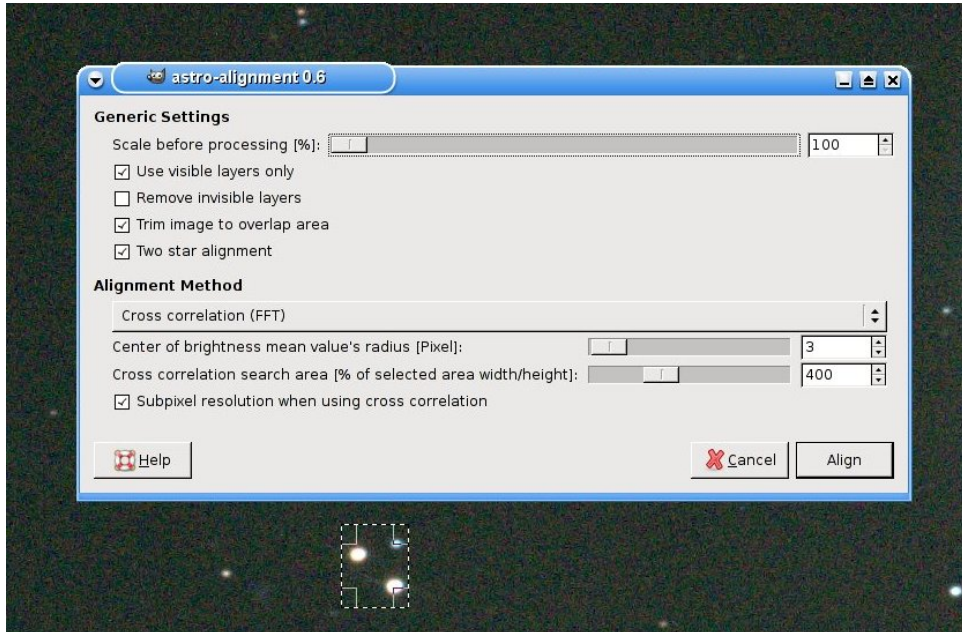
2.6 Flat division

Divides all layers by a flat field; size issue as above.

2.7 Normalize batch

Batch applies the normalize function (*Layer → Colors → Automatic → Normlize*) to all layers.
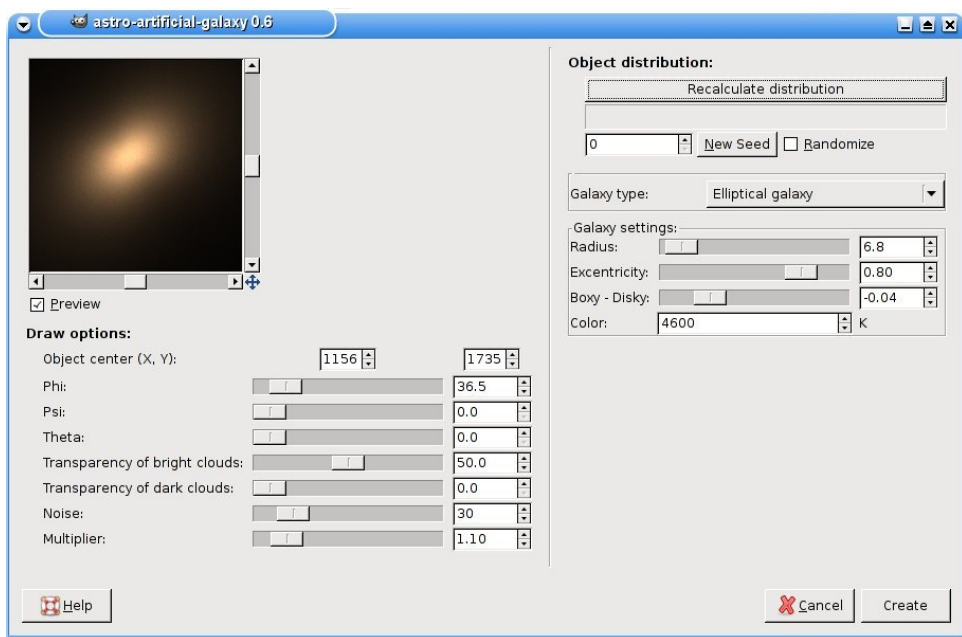
3 Plug-ins

3.1 Align

This plug-in moves (and rotates) the layers that they overlap correctly. For alignment by center of disk / brightness, select the area, where the star is, that you want to user for alignment. For cross correlation alignment, select the object you want to use for selection (from the first layer). Optionally you can crop the image to the area, where the layers overlap. De-rotation is done by selecting a second star during alignment process.
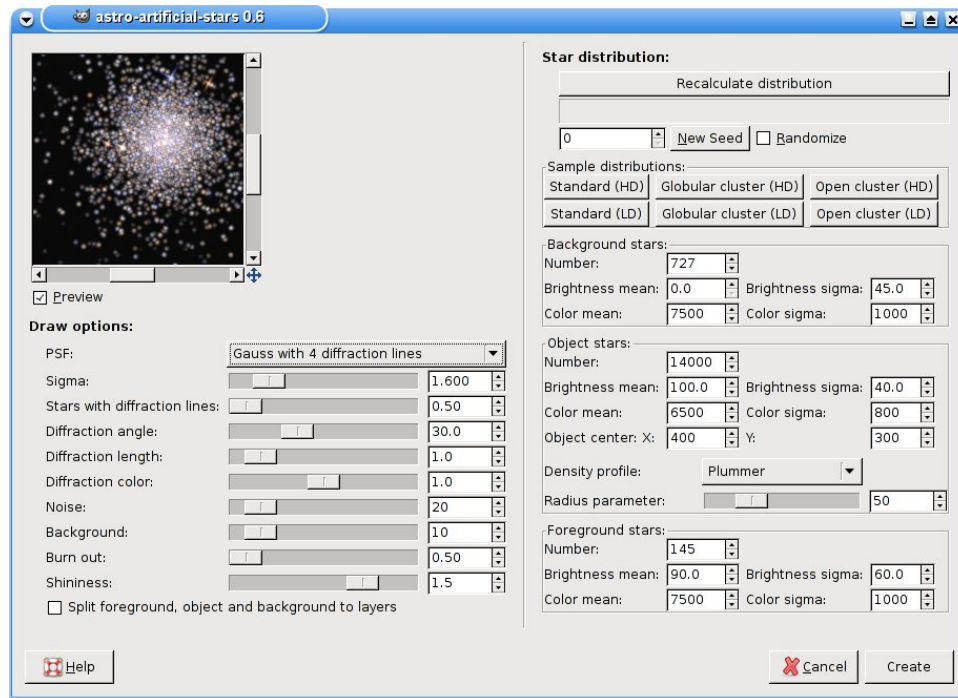
3.2 Artificial galaxy

3.2.1 Usage

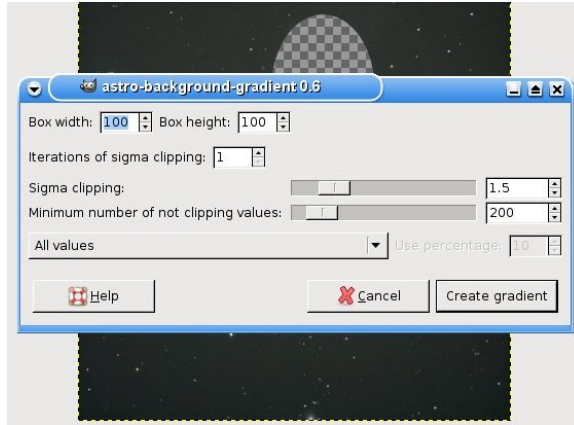Creates an artificial galaxy.

3.2.2 Explanation

Quite difficult to implement, only elliptical galaxies by now.

3.3 Artificial stars

Creates an artificial star distribution. The interface is splitted: On the right side, you can set up the options for the star distribution, on the right side how they will be drawn. To actually calculate a star distribution, you need to click on the button, as recalculation might take very long for many stars. Changes on the left side only affect the preview. The sample distribution buttons give you an idea, what you can do with this plug-in.

3.4 Background gradient

3.4.1 Usage

This plugin fills a layer with the normalized background of a layer, described by a polynomial of 4th order. To receive good results, copy your layer, cut out parts of the layer containing large objects with a color different than the background color (for example galaxies or nebulae) and then run the background gradient plugin on it. The background gradient layer is set to "divide", as this will remove the background gradient by amplifying dark parts. Of course, this will amplify noise, too.

3.4.2 Explanation

First, the plugin divides the layer into boxes with width and height defined by the user. The boxes at the edges can be smaller, depending on the layer and the box sizes. Then it loops through all these boxes.
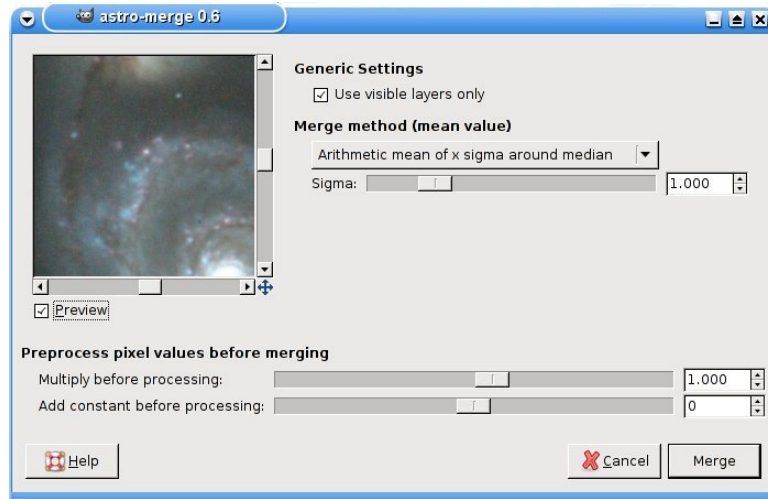
For each box, all pixel values (colors are treated seperately!) are sorted (pixels with alpha channel 0 are omitted). If the user decided to consider only darkest (brightest) z percent values, all other values are thrown away. Then, the median and the standard deviation σ of the remaining values is calculated. If sigma clipping iteration is $c > 1$, all values $> \text{median} + n \cdot \sigma$ and $< \text{median} - n \cdot \sigma$ are thrown away and this step is repeated $c - 1$ times.

Finally, if the number of remaining pixel values is big enough (compared to the minimum number requested by user), the arithmetic mean of the remaining values is calculated.

As this is done for each box, we have many mean pixel values, one at the position of each box. These mean pixel values are the pivotal points for a linear regression fit to a polynomial of 4th order.

Then the absolute maximum of the polynomial is calculated, and normalized values are written to the background gradient layer (maximum corresponds to 255).

3.5 Merge

3.5.1 Usage

Calculates the mean value of overlapping layers and saves it into a new image. You can choose, which method for the mean value you want: Arithmetic or geometric mean value, median, median / arithmetic mean or sigma median (1 or 2 passes). Each pixel value can be modified by a multiplier and an additive constant. For an explanation, what these settings mean, look below.

3.5.2 Explanation

First, this plugin resizes all layers to the image size (invisible to user). This can lead to invisible areas (alpha channel is 0) in a layer, where the layers don't overlap, except for layers without alpha channel: the areas won't be invisible, but have background color. Then the plugin loops through all pixels of the image reading the pixel value of all visible layers at this position. Then the pixel values are sorted, multiplied by the given constant, a given constant is added, and finally merged as described below, and the result is written to a new image. All colors are treated seperately.

Possible merge methods are (example: input pixel values $[3, 7, 10, 22, 212]$, greyscale values or brightness of one color)

- **Arithmetic mean**
$$\text{Result} = \frac{\sum(\text{pixel value})}{\text{number of pixels}}$$

In our example:

$$\text{Result} = \frac{3 + 7 + 10 + 22 + 212}{5} = 50.8 \approx 51$$

If the pixels are partially visible (alpha channel between 1 and 254), the numbers are weighted by this value (alpha channel of 255 corresponds to 1.0):

$$\text{Result} = \frac{\sum(\text{alpha channel} \cdot \text{pixel value})}{\sum(\text{alpha channel})}$$

- **Geometric mean**

$$\text{Result} = \sqrt[\text{number of pixels}]{\prod(\text{pixel value})}$$

In our example:

$$\text{Result} = \sqrt[5]{3 \cdot 7 \cdot 10 \cdot 22 \cdot 212} \approx 16$$

The pixel values are weighted by alpha channel values as above.

- **Median**
  The median is simply the pixel value, that is in the sorted pixel value list in the middle. If it contains an even number of pixel values, the arithmetic mean of both pixel values in the middle is calculated and rounded.
  In our example $[3, 7, 10, 22, 212]$:

$$\text{Result} = 10$$

For this and the following merge methods, no alpha channel dependent weighting is done. They are simply assumed as fully visible, if alpha channel > 0, and ignored if alpha channel $= 0$.

- **Arithmetic mean of x sigma around median**
  First, the median is calculated as above. Then the standard deviation is calculated:

$$\sigma = \sqrt{\frac{1}{\text{number of pixels} - 1} \sum(\text{pixel value} - \text{median})^2}$$

In our example:

$$\sigma = 101.25$$

Then all values outside median $\pm n \cdot \sigma$ are thrown out, in our case (with $n = 1.0$) all pixel values below $10 - 101.25 = -91.25$ or above 111.25. This means, the pixel value list is now reduced to $[3, 7, 10, 22]$. In a final step, the arithmetic mean of these pixel values is calculated as shown above.
  In our example:

$$\text{Result} = 10.5 \approx 11$$

- **Sigma median (1 pass)**
  Same as above, but the result is not the arithmetic mean of the remaining pixel values, but the median.
  In our example ($[3, 7, 10, 22]$):

$$\text{Result} = 8.5 \approx 9$$

- **Sigma median (2 passes)**
  Same as above, but the standard deviation is calculated from the remaining pixel values, and the pixel values outside median $\pm n \cdot \sigma_{\text{new}}$ are thrown away. The result is the median of these final remaining pixel values.
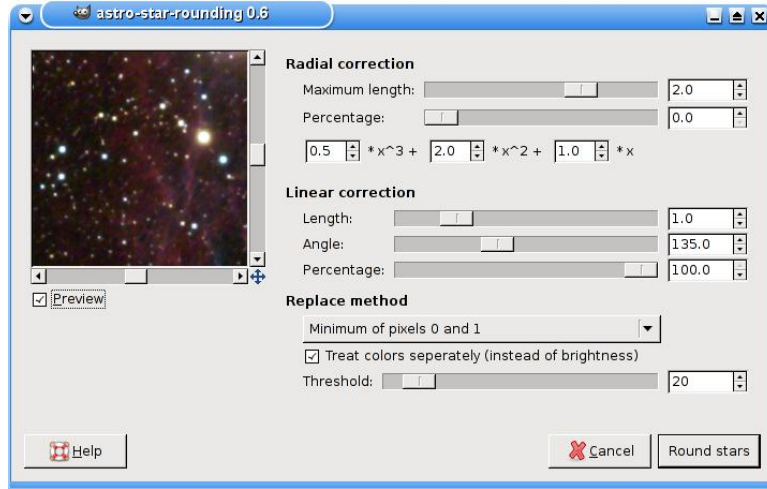 In our example ($[3, 7, 10, 22]$, $\sigma \approx 8.5$, final remaining pixel values $[3, 7, 10]$):

$$\text{Result} = 7$$

For few layers (less than 5 layers), the arithmetic mean seems to be a good choice, and for more layers, you should choose one of the median values, as they ignore (more or less) mavericks, such as hot/dark pixels.

On real images, I can't see a big difference between the different median values, but they are a lot better than the simple arithmetic mean, because hot pixels always remain visible, also when using many layers.
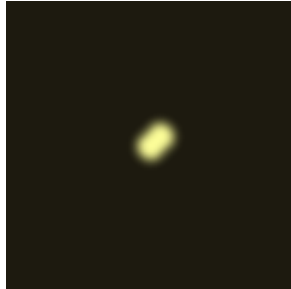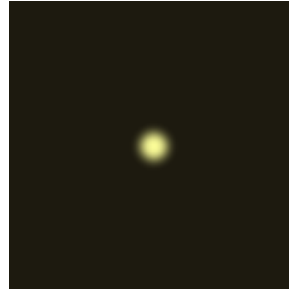
3.6 Star rounding

3.6.1 Usage

This plugin can round stars, that are longish due to guiding errors or due to an uneven field (longish in radial direction). The length values should match the elongation of the stars. The percentages determine, how much of the correction is applied.

Example:

before	after

3.6.2 Explanation

This plugin loops two times through all pixels of a drawable, either a complete layer or a selected area.

The first time, a radial elongation is corrected. For this purpose, the length is calculated at each pixel, depending on the distance r from the middle of the drawable, where the coefficients of the polynomial are given by the user:

$$P(r) = a_3 r^3 + a_2 r^2 + a_3 r$$

So in the middle, the length would be 0, and in the corners the given maximum length. The angle is calculated, depending on the position of the current pixel.

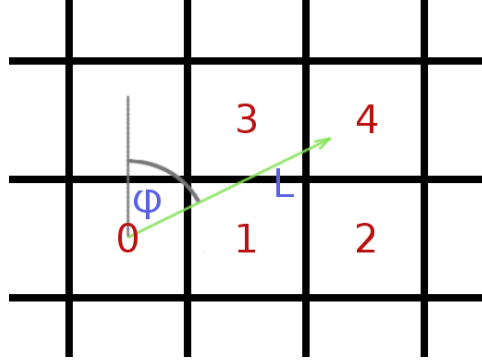The second loop corrects a linear elongation with given length L and angle ϕ.

For each pixel, a minimum (or maximum) value is calculated, depending on length L and angle ϕ, either calculated for each pixel (radial correction) or constant (linear elongation).

The minimum (or maximum) values are calculated in the following way:

Decide, whether the brightness of the current pixel value is bigger than the threshold value (for minimum) or smaller (for maximum).

If it isn't, don't modify the current pixel.

Otherwise, identify the pixels 0 to 4 (0 is the current pixel, 1-4 are the pixels where the vector (L, ϕ) is pointing to:

Now, it calculates the brightness for pixels 0-5, where brightness is the greyscale value or calculated from RGB by

$$I = 0.30 \cdot R + 0.59 \cdot G + 0.11 \cdot B$$

Then the distance from the point the vector is pointing to to the rows of pixels 1 and 2, and pixels 3 and 4, and to the columns of pixels 1 and 3, and pixels 2 and 4 is calculated. In this example:

$$D_{\text{row } 3,4} = 0.17$$

$$D_{\text{row } 1,2} = 0.83$$

$$D_{\text{column } 1,3} = 0.69$$

$$D_{\text{column } 2,4} = 0.31$$

And from these distances a weighting value

$$W = 1 - D$$

Now it finds the pixel closest to the vector, in this example $(L = 1.9, \phi = 64.2°)$ it is pixel 4.

The new pixel value is now calculated from all these values in the following way, weighted by the given percentage value. The result is either a color or greyscale value, or a brightness value not modifying the color:

- **Minimum of pixels 0 and 1**

  R = percentage $\cdot$ min{pixel 0, pixel closest to vector} + (100 − percentage) $\cdot$ pixel 0

  In our example:

  R = percentage $\cdot$ min{pixel 0, pixel 4} + (100 − percentage) $\cdot$ pixel 0

- **Minimum of pixels 0-4**

  R = percentage $\cdot$ min{pixel 0, pixel 1, pixel 2, pixel 3, pixel 4} + (100 − percentage) $\cdot$ pixel 0

- **Minimum of pixel 0 and mean value of pixels 1-4**

  $$R = \text{percentage} \cdot \min\{\text{pixel 0}, \sum_{i=1}^{4} W_{\text{row}} \cdot W_{\text{column}} \cdot \text{pixel i}\} + (100 - \text{percentage}) \cdot \text{pixel 0}$$

  In our example:

  R = percentage · min{pixel 0, 0.17 · 0.31 · pixel 1 + 0.17 · 0.69 · pixel 2
  + 0.83 · 0.31 · pixel 3 + 0.83 · 0.69 · pixel 4} + (100 − percentage) · pixel 0

- **Maximum of pixels 0 and 1**

  R = percentage · max{pixel 0, pixel closest to vector} + (100 − percentage) · pixel 0

  In our example:

  R = percentage · max{pixel 0, pixel 4} + (100 − percentage) · pixel 0

- **Maximum of pixels 0-4**

  R = percentage · max{pixel 0, pixel 1, pixel 2, pixel 3, pixel 4} + (100 − percentage) · pixel 0

- **Maximum of pixel 0 and mean value of pixels 1-4**

  $$R = \text{percentage} \cdot \max\{\text{pixel 0}, \sum_{i=1}^{4} W_{\text{row}} \cdot W_{\text{column}} \cdot \text{pixel i}\} + (100 - \text{percentage}) \cdot \text{pixel 0}$$

  In our example:

  R = percentage · max{pixel 0, 0.17 · 0.31 · pixel 1 + 0.17 · 0.69 · pixel 2
  + 0.83 · 0.31 · pixel 3 + 0.83 · 0.69 · pixel 4} + (100 − percentage) · pixel 0

The result is written back to the drawable. The alpha channel is not modified.