Draft September 12, 2012

GalSim Library Quick Reference

1. Overview

BARNEY TODO: Tidy this whole thing up, make it look a lot less ugly, maybe use an entirely different document class.

The GalSim Library provides a number of Python classes and methods for simulating astronomical images. The fundamental work flow will normally be something like:

- Construct a representation of your desired astronomical object as a single GalSim `GSObject` instance or in combination using the special `Add` and `Convolve` compound-type `GSObjects` — see Section 2.
- *Optional*: Apply transformations such as shear or magnification using the methods of the resulting `GSObject` instance — see Section 3.
- Draw the object into a GalSim `Image` object representing a postage stamp image of your astronomical object. This can be done using the `draw()` or `drawShoot()` methods carried by all `GSObjects` for rendering images (`drawShoot` uses photon shooting) — see Section 3.
- *Optional*: Add noise to the `Image` using one of the GalSim random deviate classes — see Section 4.
- *Optional*: Add the postage stamp `Image` to a subsection of a larger `Image` instance, or to a larger structure containing multiple `Image` instances each derived from `GSObject`s as described above — see Section 5.
- Save the `Image`(s) to file in FITS (Flexible Image Transport System) format — see Section 5.

There are many examples of this workflow in the directory `GalSim/examples/`, showing most of the GalSim library in action, in the scripts named `demo1.py` – `demo8.py`.

We now provide a brief, reference description of the GalSim classes and methods which can be used in this workflow. Where possible this has been hyperlinked to the online GalSim documentation generated by *doxygen* where a more detailed description can generally be found.

2. The GSObjects

There are currently 12 types of `GSObject`. The first ten listed are 'simple' or 'atomic' `GSObjects` that can be initialized by providing values for their required or optional parameters; the last two are 'compound' classes used to represent combinations of `GSObjects`. They are summarized in the following hyperlinked list, in the order in which the classes appear in `GalSim/galsim/base.py`:

- `Gaussian` — *a 2D Gaussian light profile.*
- `Moffat` — *a Moffat profile, used to approximate PSFs.*
- `AtmosphericPSF` — *currently an image-based implementation of a Kolmogorov PSF (see below), but expected to evolve to use an image of a stochastically modelled atmospheric PSF in the near future.*
- `Airy` — *an Airy PSF for ideal diffraction through a circular aperture, supports central obscuration.*
- `Kolmogorov` — *the Kolmogorov PSF for long-exposure images through a turbulent atmosphere.*
- `OpticalPSF` — *a simple model for non-ideal (aberrated) propagation through circular or square apertures with obscuration.*
- `Pixel` — *used for integrating light onto square or rectangular pixels.*
- `Sersic` — *the Sérsic family of galaxy light profiles.*
- `Exponential` — *the Exponential disc, a Sérsic with index* $n = 1$.
- `DeVaucouleurs` — *commonly used to model galaxy bulge profiles, a Sérsic with index* $n = 4$.
- `RealGalaxy` — *models galaxies using real data, including a correction for the original PSF. Requires the download of external data for full functionality.*
- `Add` — *a* compound *object used for summing multiple* `GSObject`*s.*
- `Convolve` — *a* compound *object used for convolving multiple* `GSObject`*s.*

For more information and initialization details for each `GSObject`, the Python docstring for each class is available by typing

```
>>> print galsim.<GSObject_name>.__doc__
```

within the Python interpreter. Alternatively follow the hyperlinks on the class names above to view the *doxygen* documentation based on the Python docstrings.

3. Important GSObject methods

A number of methods are shared by all the `GSObjects` of Section 2, and are also to be found in `GalSim/galsim/base.py` within the definition of the `GSObject` base class. In what follows, we assume that a `GSObject` labelled `obj` has been instantiated using one of the calls described in the documentation linked above. For example,

```
>>> obj = galsim.Sersic(n=3.5, half_light_radius=1.743).
```

Some of the most important and commonly-used methods for such an instance are:

- `obj.copy()` — *return a copy of the* `GSObject`.
- `obj.getFlux()` — *get the flux of the* `GSObject`.
- `obj.scaleFlux(flux_ratio)` — *multiply the flux of the* `GSObject` *by* `flux_ratio`*.*
- `obj.setFlux(flux)` — *set the flux of the* `GSObject` *to* `flux`*.*
- `obj.applyDilation(scale)` — *apply a dilation of the linear size of the* `GSObject` *by a factor* `scale`*.*
- `obj.applyMagnification(scale)` — *dilate linear size by* `scale` *and* `GSObject` *flux by* `scale`^{2}*, conserving surface brightness.*
- `applyShear(*args, **kwargs)` — *apply a shear to the* `GSObject`*, handling a number of different input conventions.*
- `obj.applyRotation(theta)` — *apply a rotation of* `theta` *(positive direction anti-clockwise) to the* `GSObject`*, where* `theta` *is a* `galsim.Angle` *instance (see Section 6).*
- `applyShift(dx, dy)` — *apply a* (dx, dy) *shift to this object.*
- `obj.draw(...)` — *draw an image of the* `GSObject` *using Discrete Fourier Transforms and interpolation to perform the image rendering.*
- `obj.drawShoot(...)` — *draw an image of the* `GSObject` *by shooting a finite number of photons to perform the image rendering. The resulting image therefore contains stochastic noise, but the rendering is otherwise very close to exact.*

Once again, for more information regarding each `GSObject` method, the Python docstring is available

```
>>> print obj.<method_name>.__doc__
```

within the Python interpreter. Alternatively follow the hyperlinks on the class names above to view the *doxygen* documentation based on the Python docstrings. You will see that many of the `GSObject` instances also have their own specialized methods, often for retreiving parameter values. Examples are `obj.getSigma()` for the Gaussian or `obj.getHalfLightRadius()` for many of the `GSObjects`.

4. Random deviate classes and methods

A short summary of the 8 random deviates currently implemented in GalSim, with a short description of their distributions:

- `UniformDeviate` — *uniform distribution in the interval* $[0, 1)$.
- `GaussianDeviate` — *Gaussian distribution with* `mean` *and standard deviation* `sigma`.
- `BinomialDeviate` — *Binomial distribution for* `N` *trials each of probability* `p`.
- `PoissonDeviate` — *Poisson distribution with a single* `mean` *rate.*
- `CCDNoise` — *Distribution following a basic CCD noise model, depending on* `gain` *and* `read_noise`.
- `WeibullDeviate` — *Weibull distribution family (includes Rayleigh and Exponential) with shape parameters* `a` *and* `b`.
- `GammaDeviate` — *Gamma distribution for parameters* `alpha` *and* `beta`.
- `Chi2Deviate` — χ^2 *distribution for degrees of freedom parameter* `n`.

Unfortunately the random deviate classes are not yet fully integrated within the *doxygen* documentation, due to their being C++ with compiled Python wrappers. This means that the class names above are not hyperlinked.

However, the full docstrings are available in `galsim/random.py`, so please refer there for more information, or type

```
>>> print galsim.<RandomDeviate_name>.__doc__
```

within the Python interpreter.

We now illustrate the most commonly-used methods of the random deviates. If we assume that some random deviate instance has been instantiated as `dev`, for example

```
>>> dev = galsim.GaussianDeviate(sigma=3.9, mean=50.),
```

Some of the most important and commonly-used methods for such an instance are:

- `dev.applyTo(image)` — *adds a deviate distributed according to the distribution represented by* `dev` *to each element in in a supplied Image instance* `image` *(see Section 5).*
- `dev()` — *this direct call method returns a new random number drawn from the distribution represented by* `dev`.

5. Image classes and methods

- `ImageS`
- `ImageI`
- `ImageF`
- `ImageD`

- `img.read()`
- `img.addNoise()`
- `img.write()`

6. Miscellaneous classes and methods

- `Angle`
- `Ellipse`