sanpera Documentation

Release 0.1.0

Eevee

June 14, 2016

Contents

sanpera is an image editing and manipulation library for Python.

CHAPTER 1

Table of contents

1.1 Overview

1.1.1 Warning

ImageMagick is a complicated library with poor documentation and too many warts. Additionally, the author's C is a bit rusty. sanpera is thus completely unreliable and may destroy your images, eat all your memory, and/or burn your house down.

Patches and expertise are, of course, entirely welcome.

1.1.2 Compatibility

sanpera is known to work with Python 2.7. It *probably* works with earlier versions of Python 2, and *possibly* works with Python 3.

1.1.3 Installation

sanpera requires Cython.

You will also need ImageMagick and its headers installed if you plan to get very far.

Minimum versions are unclear; thusfar sanpera has only been built and used with the latest versions of everything.

1.1.4 Regarding ImageMagick

There are several Python libraries, in varying states of completion and decay, that wrap ImageMagick.

sanpera is explicitly *not* such a library. ImageMagick is considered an implementation detail, and its design influences sanpera's as little as possible. sanpera actually goes to considerable lengths to subvert ImageMagick features in many cases, where such features are awkward in Python, obscure and surprising, inappropriate for a general-use library, or otherwise deemed undesireable.

ImageMagick was chosen for its ubiquity, fairly broad feature set, and familiarity. GraphicsMagick was briefly evaluated, and even powered the initial prototype, but it has languished considerably since it was forked from ImageMagick. Other candidates for an underlying library were either too cumbersome, too underpowered, or tragically unknown to the author at the time.

1.1.5 Name

As it's a library and not a program, the name "sanpera" is written in lowercase.

"Sanpera" is the Hindi term for a snake charmer—i.e., one who might manipulate Python with magick.

1.1.6 Links

- Project homepage
- GitHub repository
- ImageMagick

1.2 Introduction

1.2.1 Usage

ImageMagick has "Usage" documentation which demonstrates hundreds of specific operations as performed by the `convert` utility. Part of sanpera's test suite is built around these demos: a `convert` command is rewritten into Python, both are executed, and the results are compared. If you're already familiar with `convert`, this is a fast way to get up to speed: just find a Usage test that does what you want and look at the equivalent Python. Usage tests are kept in `sanpera/tests/im_usage`.

1.2.2 Images versus frames

As far as sanpera is concerned, and unlike many other libraries, images and frames are separate concepts.

An *image* is a collection of metadata and a stack of zero or more frames. Each *frame* is a rectangular grid of actual pixel data. High-level operations such as converting between image formats tend to be done on an image; custom effects, drawing, and pixel inspection must be done on individual frames.

The distinction removes API ambiguity between single-frame and multi-frame images, and helps avoid some common pitfalls when programs written for single-frame images are used for multi-frame images.

Additionally, destructive image operations tend to return new image objects, whereas destructive frame operations cheerfully operate in-place.

Images are represented by the `Image` class. Frames are represented by the `ImageFrame` class. An `Image` acts as a sequence of frames, but the interface is somewhat hindered to prevent two images from claiming to own the same frame at the same time.

1.2.3 Geometry

sanpera has a small set of geometry-related utility classes. Properties of images and frames, such as size, return `Size` objects.

For convenience's sake, any method or function *anywhere* in sanpera that expects a geometry object will also accept a plain tuple; for example, you may say `img.resized((100, 100))` rather than `img.resized(Size(100, 100))`. Don't forget the extra pair of parentheses!

1.2.4 Reading and writing

Read from a file:

```
img = Image.read('foo.png')
```

Or from a string:

```
img = Image.from_buffer(pngdata)
```

Similarly, write to a file:

```
img.write('foo.png', format='png')

# If the image was read from a file or string, it "remembers" its original
# format, and the format can be omitted:
img.write('foo.png')
```

Or a string:

```
buf = img.to_buffer(format='png')

# Same thing applies
buf = img.to_buffer()
```

Images cannot be read from or written to arbitrary file-like objects; the underlying library simply doesn't support chunked i/o. The best sanpera could do is read everything into a single buffer and write it out all at once, which deceptively implies some optimization where there is none.

You may of course do this yourself:

```
img = Image.from_buffer(filelike.read())
```

Note that ImageMagick's special filename syntax (`miff:foobar[1]` and the like) is *not* supported by the above methods, as it leads to surprising behavior for particular filenames and leaves the developer to sort the mess out. You can still use it explicitly:

```
img = Image.from_magick('png:badly_named_file.gif')
```

If you just want to use the built-in patterns or gradients, there are easier ways.

1.2.5 Resizing

```
img = img.resized((100, 100))
```

1.2.6 Cropping

```
img = img.cropped(Size(40, 40).at((30, 30)))
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

CHAPTER 3

Links

- Homepage
- GitHub project page
- ImageMagick