pycoast Documentation

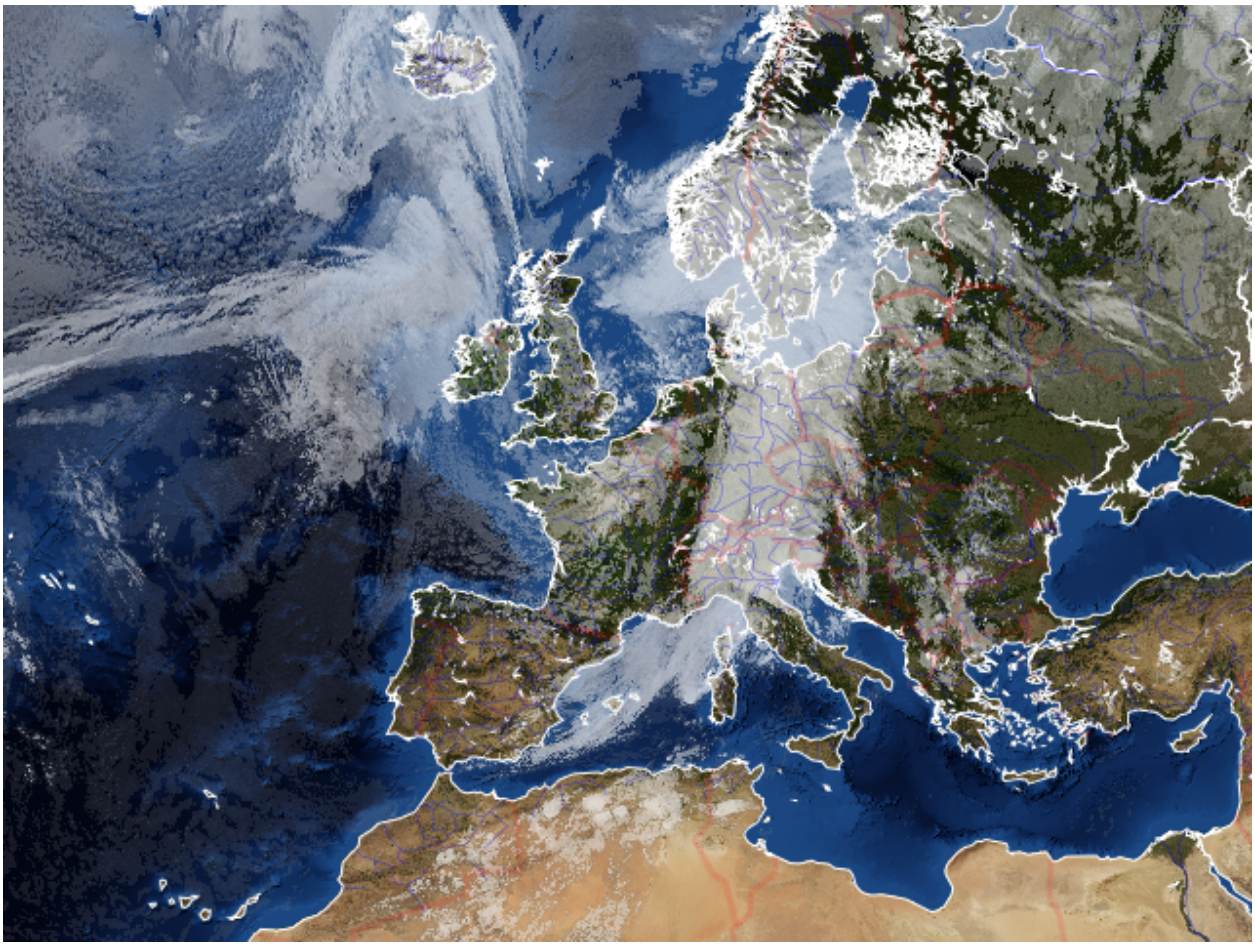
Release 0+unknown

Esben S. Nielsen

Apr 02, 2024

CONTENTS

Pycoast is a Python package to add coastlines, borders and rivers to raster images using data from the GSHHS and WDBII datasets



CHAPTER

ONE

INSTALLATION

The below sections describe how to install both the pycoast python library and additional data files that maybe required to use some features of pycoast.

If you have any trouble with the installation of the package or the data files described below, please file a bug report on GitHub:

https://github.com/pytroll/pycoast/

1.1 Package installation

Pycoast can be installed in an existing Python environment via pip or in a conda environment via `conda` using the conda-forge channel. To use pip:

```
pip install pycoast
```

Alternatively, with conda:

```
conda install -c conda-forge pycoast
```

1.2 Installation of shape files

To use the features of pycoast that draw country or other political borders, rivers, and lakes, shapefiles from the SOEST GSHHG website must be installed. Download the zipped GSHHS and WDBII shapefiles. At the time of writing the current zip file can be found at:

https://www.soest.hawaii.edu/pwessel/gshhg/gshhg-shp-2.3.7.zip

Unzip the files to a data directory (hereafter *DB_DATA_ROOT*). The absolute path/name of this directory is called *db_root_path* in the code examples used elsewhere in the documentation.

The structure of *DB_DATA_ROOT* should now be:

```
.
├── GSHHS_shp
│   ├── c
│   ├── f
│   ├── h
│   ├── i
│   └── l
```

(continues on next page)

(continued from previous page)

```
└── WDBII_shp
    ├── c
    ├── f
    ├── h
    ├── i
    └── l
```

Where each dir on the lowest level contains Shapefiles like *GSHHS_shp/c/GSHHS_c_L1.shp, WDBII_shp/WDBII_border_c_L1.shp*

1.3 Installation of city names

To use the features of Pycoast that depend on city locations or names, one or more files from GeoNames must be downloaded and made available in the same *DB_DATA_ROOT* directory created in the above GSHHG shapefile download. GeoNames releases multiple lists of city information available from their file archive:

https://download.geonames.org/export/dump/

There are files that contain city information for cities with a population larger than 500, 1000, 5000, and 15000. Only one of these files needs to be downloaded depending on your needs. At the time of writing the URLs for these files are:

- https://download.geonames.org/export/dump/cities500.zip
- https://download.geonames.org/export/dump/cities1000.zip
- https://download.geonames.org/export/dump/cities5000.zip
- https://download.geonames.org/export/dump/cities15000.zip

Once downloaded, extract the single cities .txt file inside and move it to a new `DB_DATA_ROOT/CITIES/` directory. Currently, Pycoast requires that the file be named "cities.txt". The structure of *DB_DATA_ROOT* should now be:

```
.
├── GSHHS_shp
│   ├── c
│   ├── f
│   ├── h
│   ├── i
│   └── l
├── WDBII_shp
│   ├── c
│   ├── f
│   ├── h
│   ├── i
│   └── l
└── CITIES
    └── cities.txt
```

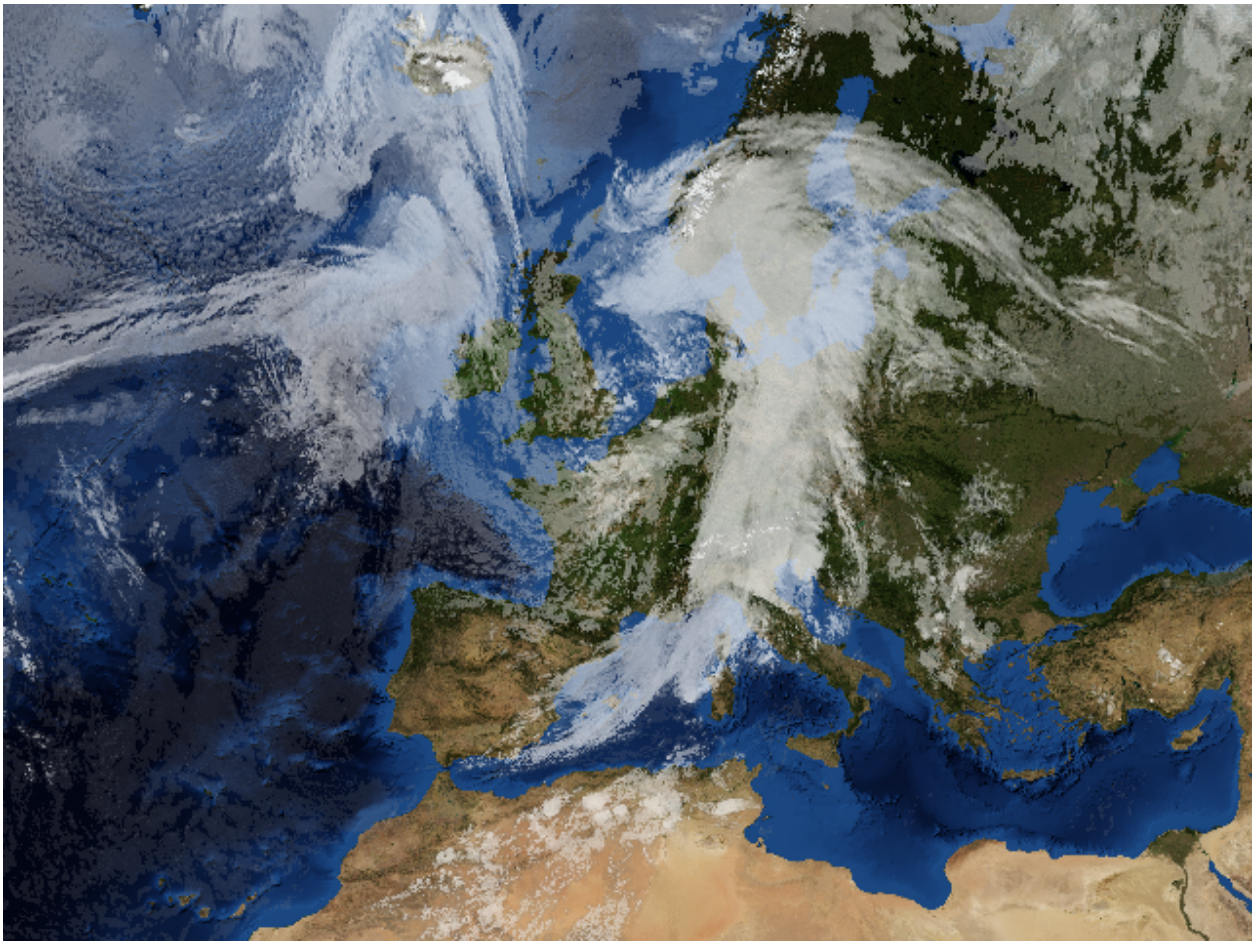
The PyCoast API documentation explains in detail how to use this city information via the *add_cities()* method.

CHAPTER

TWO

USAGE

Pycoast can be used to add coastlines, borders and rivers to a raster image if the geographic projection of the image and the image extent in projection coordinates are known



Pycoast can add contours to either a PIL image object:

```
>>> from PIL import Image
>>> from pycoast import ContourWriterAGG
>>> img = Image.open('BMNG_clouds_201109181715_areaT2.png')
>>> proj4_string = '+proj=stere +lon_0=8.00 +lat_0=50.00 +lat_ts=50.00 +ellps=WGS84'
>>> area_extent = (-3363403.31,-2291879.85,2630596.69,2203620.1)
>>> area_def = (proj4_string, area_extent)
```

(continues on next page)

(continued from previous page)

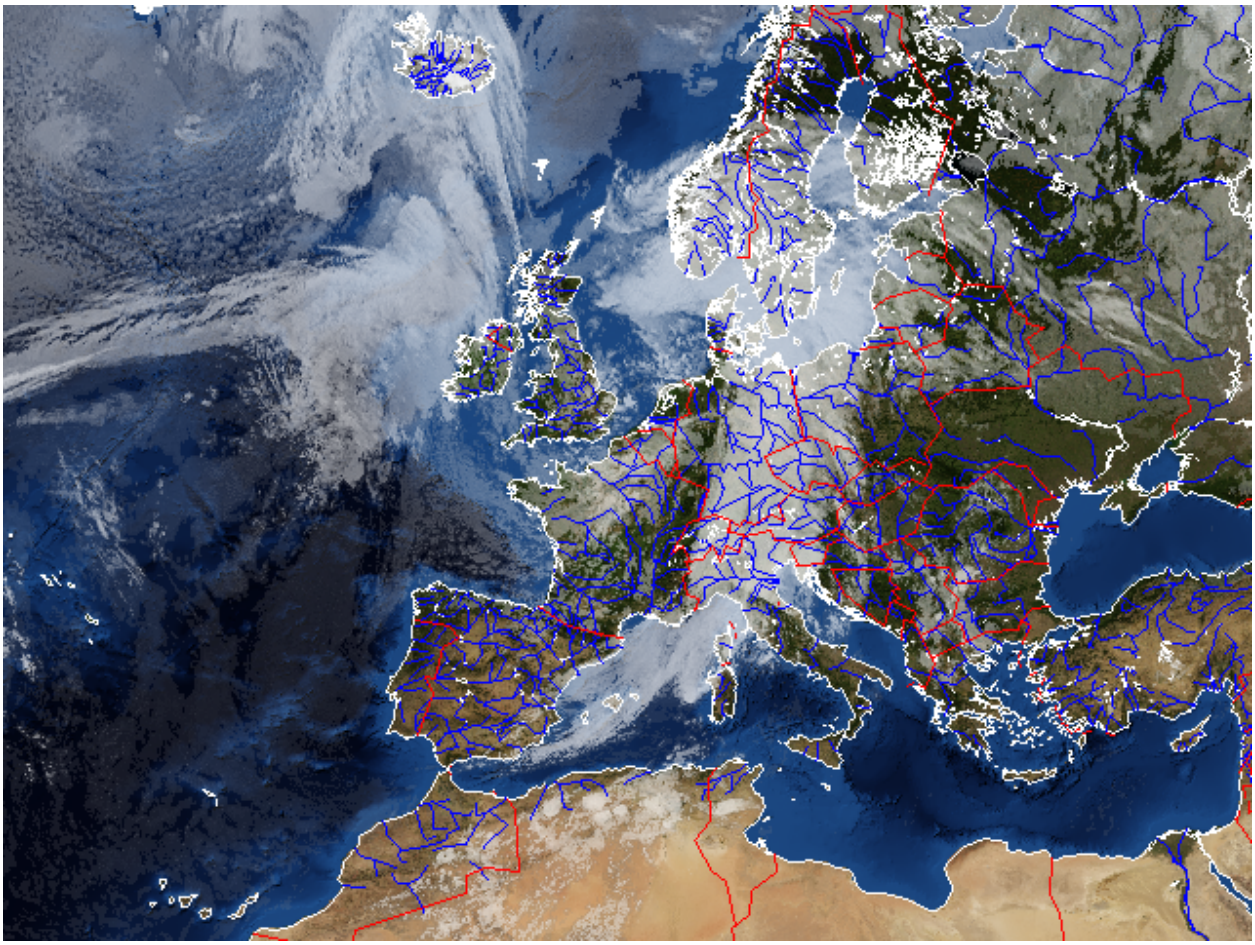
```
>>> cw = ContourWriterAGG('/home/esn/data/gshhs')
>>> cw.add_coastlines(img, area_def, resolution='l', level=4)
>>> cw.add_rivers(img, area_def, level=5, outline='blue')
>>> cw.add_borders(img, area_def, outline=(255, 0, 0))
>>> img.show()
```

or to an image file:

```
>>> from pycoast import ContourWriterAGG
>>> proj4_string = '+proj=stere +lon_0=8.00 +lat_0=50.00 +lat_ts=50.00 +ellps=WGS84'
>>> area_extent = (-3363403.31,-2291879.85,2630596.69,2203620.1)
>>> area_def = (proj4_string, area_extent)
>>> cw = ContourWriterAGG('/home/esn/data/gshhs')
>>> cw.add_coastlines_to_file('BMNG_clouds_201109181715_areaT2.png', area_def,␣
↪resolution='l', level=4)
>>> cw.add_rivers_to_file('BMNG_clouds_201109181715_areaT2.png', area_def, level=5,␣
↪outline='blue')
>>> cw.add_borders_to_file('BMNG_clouds_201109181715_areaT2.png', area_def, outline=(255,
↪ 0, 0))
```

Where the `area_extent` is the extent of the image in projection coordinates as (x_ll, y_ll, x_ur, x_ur) measured at pixel edges.

The argument to *ContourWriterAGG* must be replaced with your *GSHHS_DATA_ROOT*.

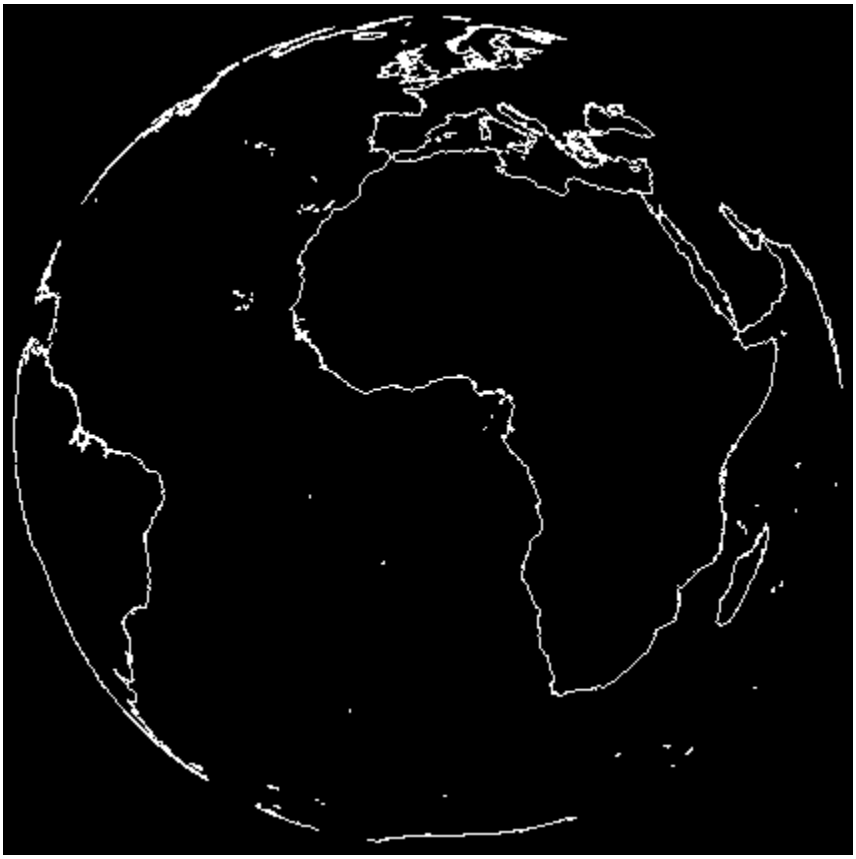


The resulting (not so pretty) image shows the effect of the various arguments. The `resolution` keyword argument controls the resolution of the dataset used. It defaults to 'c' for coarse. Increasing the resolution also increases the processing time. The `level` keyword argument controls the detail level of the dataset used. It defaults to *1* for the lowest detail level. See method docstrings for information about other possible argument values.

2.1 Example with tuple

Creating an image with coastlines only:

```
>>> from PIL import Image
>>> from pycoast import ContourWriterAGG
>>> img = Image.new('RGB', (425, 425))
>>> proj4_string = '+proj=geos +lon_0=0.0 +a=6378169.00 +b=6356583.80 +h=35785831.0'
>>> area_extent = (-5570248.4773392612, -5567248.074173444, 5567248.074173444, 5570248.
↪4773392612)
>>> area_def = (proj4_string, area_extent)
>>> cw = ContourWriterAGG('/home/esn/data/gshhs')
>>> cw.add_coastlines(img, area_def, resolution='l')
>>> img.show()
```



2.2 Example with AreaDefinition

Instead of a tuple for `area_def` a pyresample `AreaDefinition` object can be used. The code below produces the same image as above.

```
>>> from PIL import Image
>>> from pycoast import ContourWriterAGG
>>> from pyresample.geometry import AreaDefinition
>>> img = Image.new('RGB', (425, 425))
>>> area_def = AreaDefinition('my_area', 'Area Description', 'geos_proj',
...     {'proj': 'geos', 'lon_0': 0.0, 'a': 6378169.00, 'b': 6356583.80, 'h': 35785831.0}
↪,
...     425, 425,
...     (-5570248.4773392612, -5567248.074173444, 5567248.074173444, 5570248.4773392612))
>>> cw = ContourWriterAGG('/home/esn/data/gshhs')
>>> cw.add_coastlines(img, area_def, resolution='l')
>>> img.show()
```

CHAPTER

THREE

HIGH QUALITY CONTOURS USING AGG

The default plotting mode of pycoast uses PIL for rendering of contours. PIL does not support antialiasing and opacity. The AGG engine can be used for making high quality images using the aggdraw module.

First install the aggdraw module.

Tip: if the building of aggdraw fails with:

```
agg_array.h:523: error: cast from ‘agg::int8u*’ to ‘unsigned int’ loses precision
```

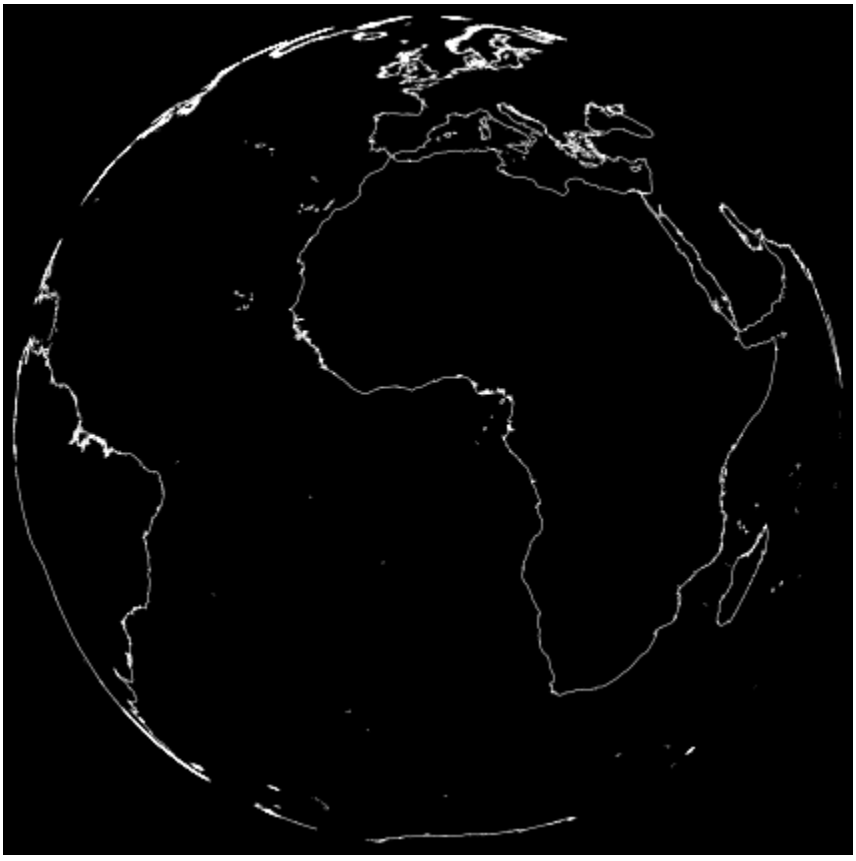
Try:

```
export CFLAGS="-fpermissive"
```

before building.

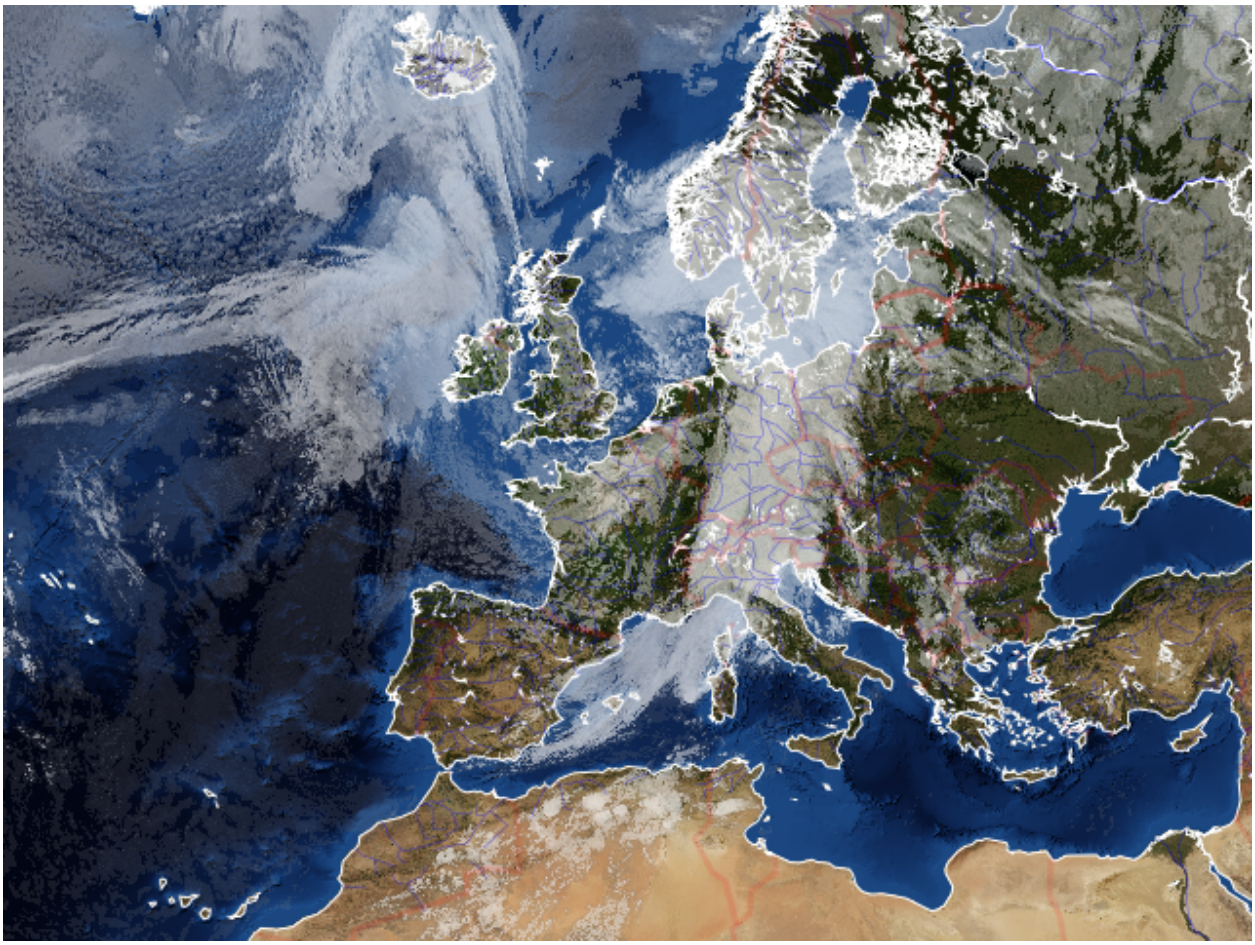
Using pycoast with AGG for making antialiased drawing:

```
>>> from PIL import Image
>>> from pycoast import ContourWriterAGG
>>> img = Image.new('RGB', (425, 425))
>>> proj4_string = '+proj=geos +lon_0=0.0 +a=6378169.00 +b=6356583.80 +h=35785831.0'
>>> area_extent = (-5570248.4773392612, -5567248.074173444, 5567248.074173444, 5570248.
↪4773392612)
>>> area_def = (proj4_string, area_extent)
>>> cw = ContourWriterAGG('/home/esn/data/gshhs')
>>> cw.add_coastlines(img, (proj4_string, area_extent), resolution='l', width=0.5)
>>> img.show()
```



and making the not-so-nice image from the first example nice:

```
>>> from PIL import Image
>>> from pycoast import ContourWriterAGG
>>> img = Image.open('BMNG_clouds_201109181715_areaT2.png')
>>> proj4_string = '+proj=stere +lon_0=8.00 +lat_0=50.00 +lat_ts=50.00 +ellps=WGS84'
>>> area_extent = (-3363403.31,-2291879.85,2630596.69,2203620.1)
>>> area_def = (proj4_string, area_extent)
>>> cw = ContourWriterAGG('/home/esn/data/gshhs')
>>> cw.add_coastlines(img, area_def, resolution='l', level=4)
>>> cw.add_rivers(img, area_def, level=5, outline='blue', width=0.5, outline_opacity=127)
>>> cw.add_borders(img, area_def, outline=(255, 0, 0), width=3, outline_opacity=32)
>>> img.show()
```



See docstrings of `ContourWriterAGG` methods for argument descriptions.

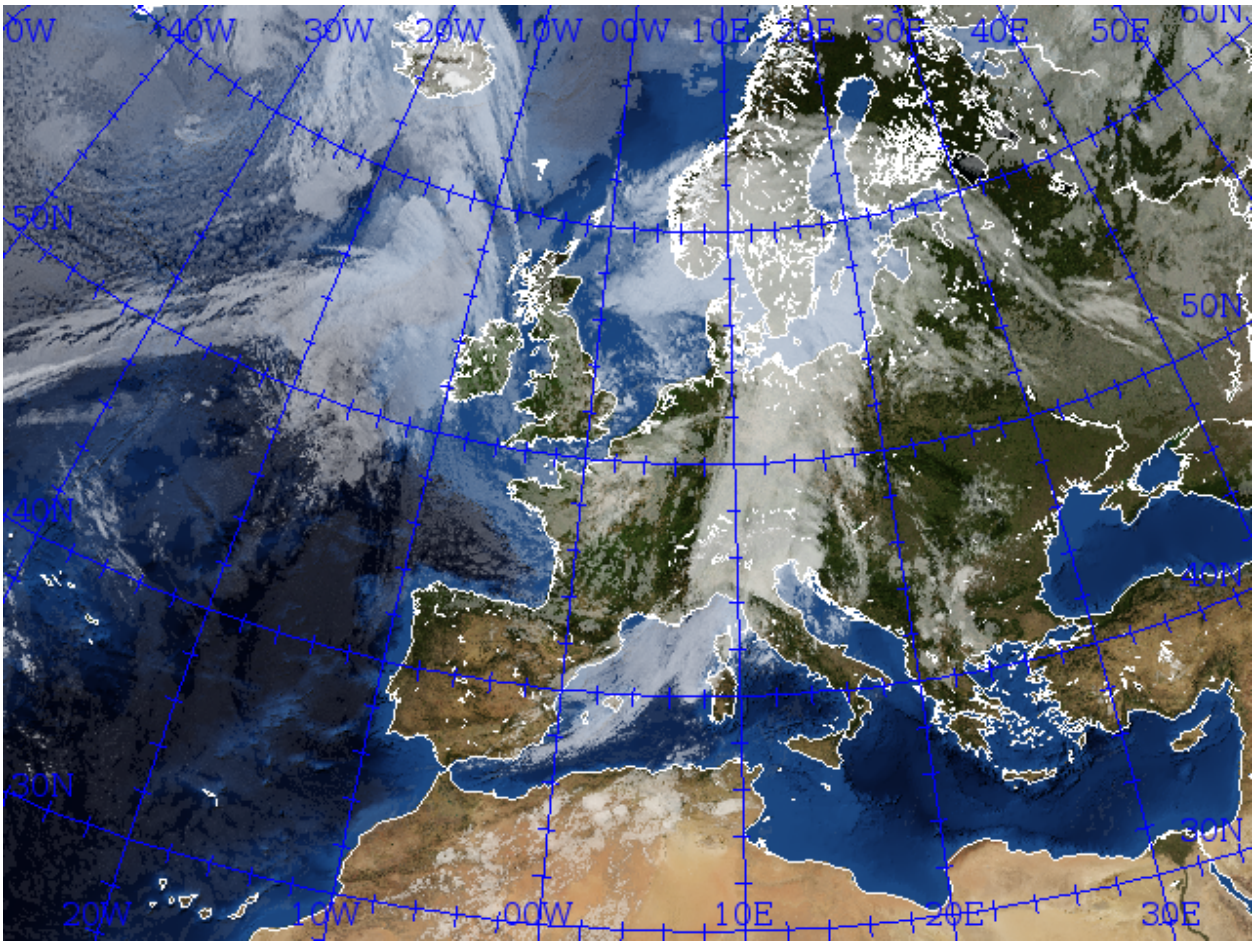
CHAPTER

FOUR

ADDING GRATICULE TO IMAGES

Pycoast can be used to add graticule to images. For PIL:

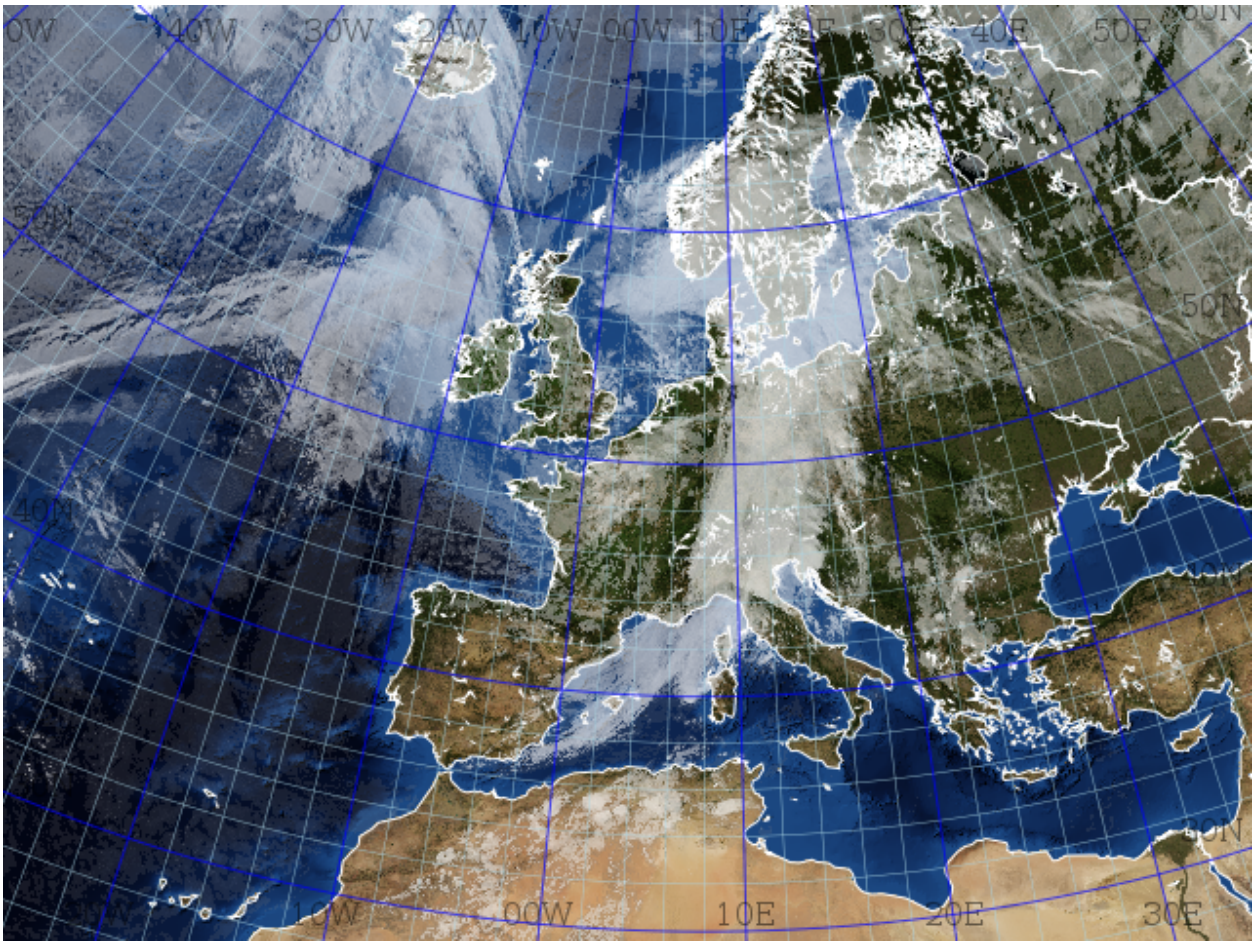
```
>>> from PIL import Image, ImageFont
>>> from pycoast import ContourWriterAGG
>>> proj4_string = '+proj=stere +lon_0=8.00 +lat_0=50.00 +lat_ts=50.00 +ellps=WGS84'
>>> area_extent = (-3363403.31,-2291879.85,2630596.69,2203620.1)
>>> area_def = (proj4_string, area_extent)
>>> cw = ContourWriterAGG('/home/esn/data/gshhs')
>>> font = ImageFont.truetype("/usr/share/fonts/truetype/ttf-dejavu/DejaVuSerif.ttf",16)
>>> img = Image.open('BMNG_clouds_201109181715_areaT2.png')
>>> cw.add_coastlines(img, area_def, resolution='l', level=4)
>>> cw.add_grid(img, area_def, (10.0,10.0),(2.0,2.0), font,fill='blue',
...             outline='blue', minor_outline='blue')
>>> img.show()
```



The font argument is optional for PIL if it is not given a default font will be used.

and for AGG:

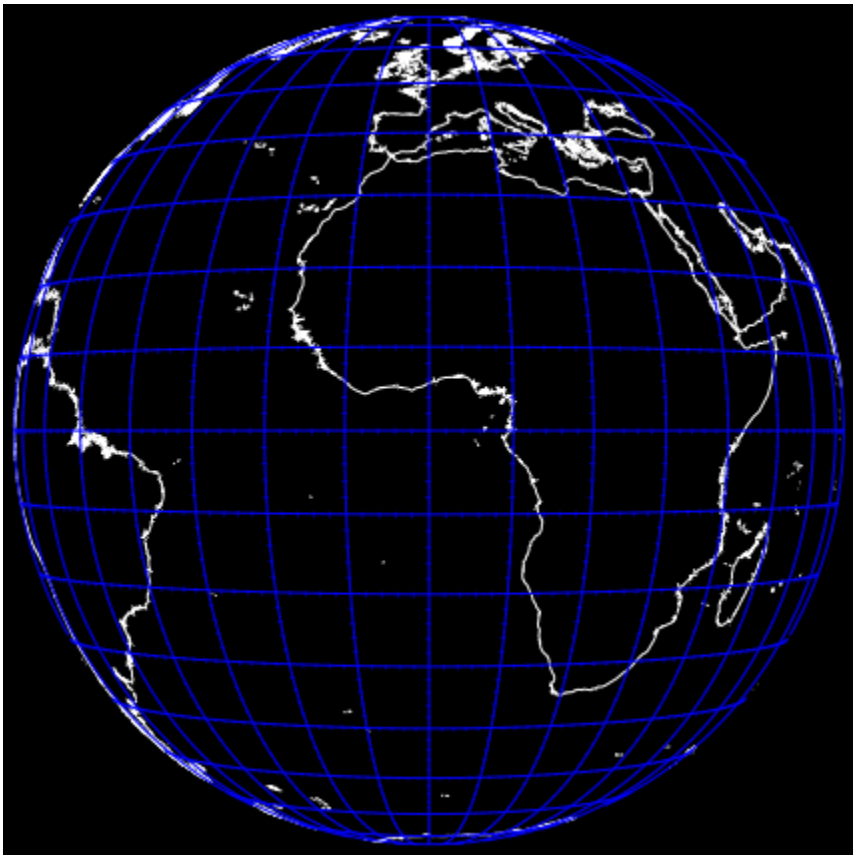
```
>>> from PIL import Image, ImageFont
>>> from pycoast import ContourWriterAGG
>>> import aggdraw
>>> proj4_string = '+proj=stere +lon_0=8.00 +lat_0=50.00 +lat_ts=50.00 +ellps=WGS84'
>>> area_extent = (-3363403.31,-2291879.85,2630596.69,2203620.1)
>>> area_def = (proj4_string, area_extent)
>>> cw = ContourWriterAGG('/home/esn/data/gshhs')
>>> font = aggdraw.Font('black', '/usr/share/fonts/truetype/ttf-dejavu/DejaVuSerif.ttf',
...                     opacity=127, size=16)
>>> img = Image.open('BMNG_clouds_201109181715_areaT2.png')
>>> cw.add_coastlines(img, area_def, resolution='l', level=4)
>>> cw.add_grid(img, area_def, (10.0,10.0),(2.0,2.0),font,
...             outline='blue',outline_opacity=175,width=1.0,
...             minor_outline='lightblue',minor_outline_opacity=200,minor_width=0.5,
...             minor_is_tick=False)
>>> img.show()
```



Note the difference in the optional font argument for PIL and AGG. With AGG the font argument is mandatory unless the keyword argument `write_text=False` is used.

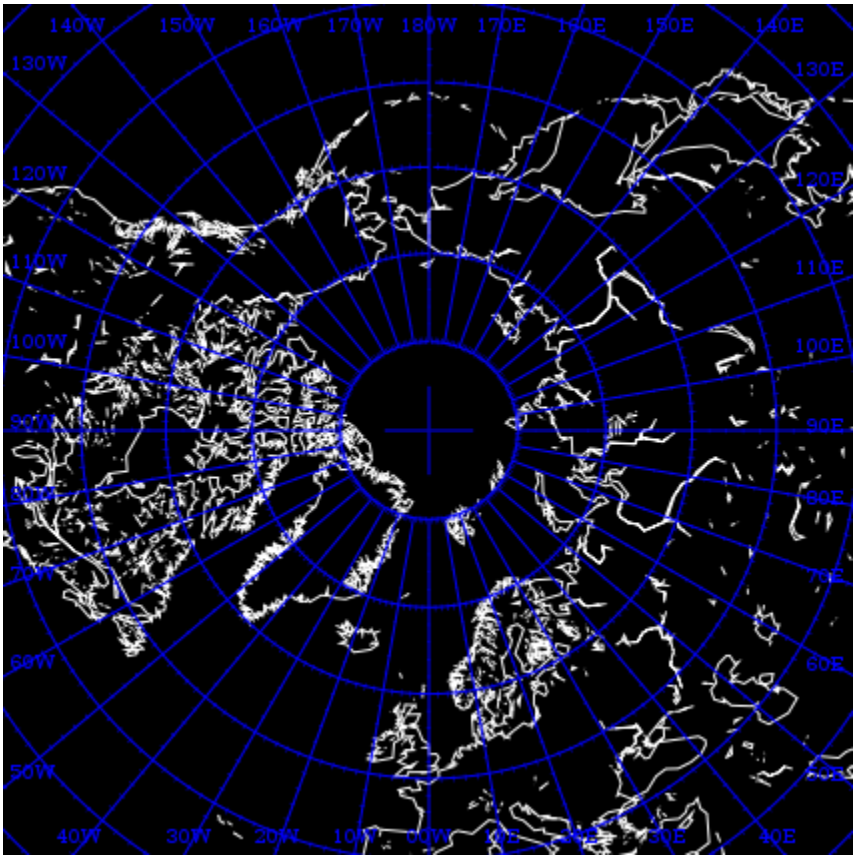
From v0.5.0 the graticule is also usable for globe projections:

```
>>> from PIL import Image
>>> from pycoast import ContourWriterAGG
>>> img = Image.new('RGB', (425, 425))
>>> proj4_string = '+proj=geos +lon_0=0.0 +a=6378169.00 +b=6356583.80 +h=35785831.0'
>>> area_extent = (-5570248.4773392612, -5567248.074173444, 5567248.074173444, 5570248.
↪4773392612)
>>> area_def = (proj4_string, area_extent)
>>> cw = ContourWriterAGG(gshhs_root_dir)
>>> cw.add_coastlines(img, area_def, resolution='l')
>>> cw.add_grid(img, area_def, (10.0,10.0),(2.0,2.0), fill='blue',
... outline='blue', minor_outline='blue', write_text=False)
>>> img.show()
```



The lon and lat labeling is shown where the lines intersect the image border. By default the lon intersections with top and bottom and the lat intersections with left and right border are displayed . The placement behaviour can be controlled with the `lon_placement` and `lat_placement` keyword variables. The placement specifier is a string containing the desired placement where 't' is top, 'b' bottom, 'l' left and 'r' right. E.g. `lon_placement='tl'` will make the lon labels display at the top and left border.

```
>>> from PIL import Image
>>> from pycoast import ContourWriterAGG
>>> import aggdraw
>>> img = Image.new('RGB', (425, 425))
>>> proj4_string = '+proj=laea +lat_0=90 +lon_0=0 +a=6371228.0 +units=m'
>>> area_extent = (-5326849.0625, -5326849.0625, 5326849.0625, 5326849.0625)
>>> area_def = (proj4_string, area_extent)
>>> cw = ContourWriterAGG('/home/esn/data/gshhs')
>>> cw.add_coastlines(img, area_def, resolution='c', level=4)
>>> font = aggdraw.Font('blue', '/usr/share/fonts/truetype/ttf-dejavu/DejaVuSerif.ttf',␣
↪size=10)
>>> cw.add_grid(img, area_def, (10.0,10.0),(2.0,2.0), font=font, fill='blue',
...             outline='blue', minor_outline='blue',
...             lon_placement='tblr', lat_placement='')
>>> img.show()
```



Tip: If the adding graticule with AGG fails with something like:

```
Traceback (most recent call last):
    File "grid_demo_AGG.py", line 13, in <module>
        font=aggdraw.Font("blue", "/usr/share/fonts/truetype/ttf-dejavu/DejaVuSerif.ttf",
↪ size=16)
IOError: cannot load font (no text renderer)
```

make sure the FREETYPE_ROOT in setup.py of aggdraw points to the correct location e.g. set *FREETYPE_ROOT = "/usr"*

CHAPTER

FIVE

PYCOAST FROM A CONFIGURATION FILE

If you want to run to avoid typing the same options over and over again, or if caching is an optimization you want, you can use a configuration file with the pycoast options you need:

```
[cache]
file=/var/run/satellit/white_overlay
regenerate=False

[coasts]
level=1
width=0.75
outline=white
fill=yellow

[borders]
outline=white
width=0.5
```

Then, you can just call:

```
>>> cw.add_overlay_from_config("my_config.cfg", area_def)
```

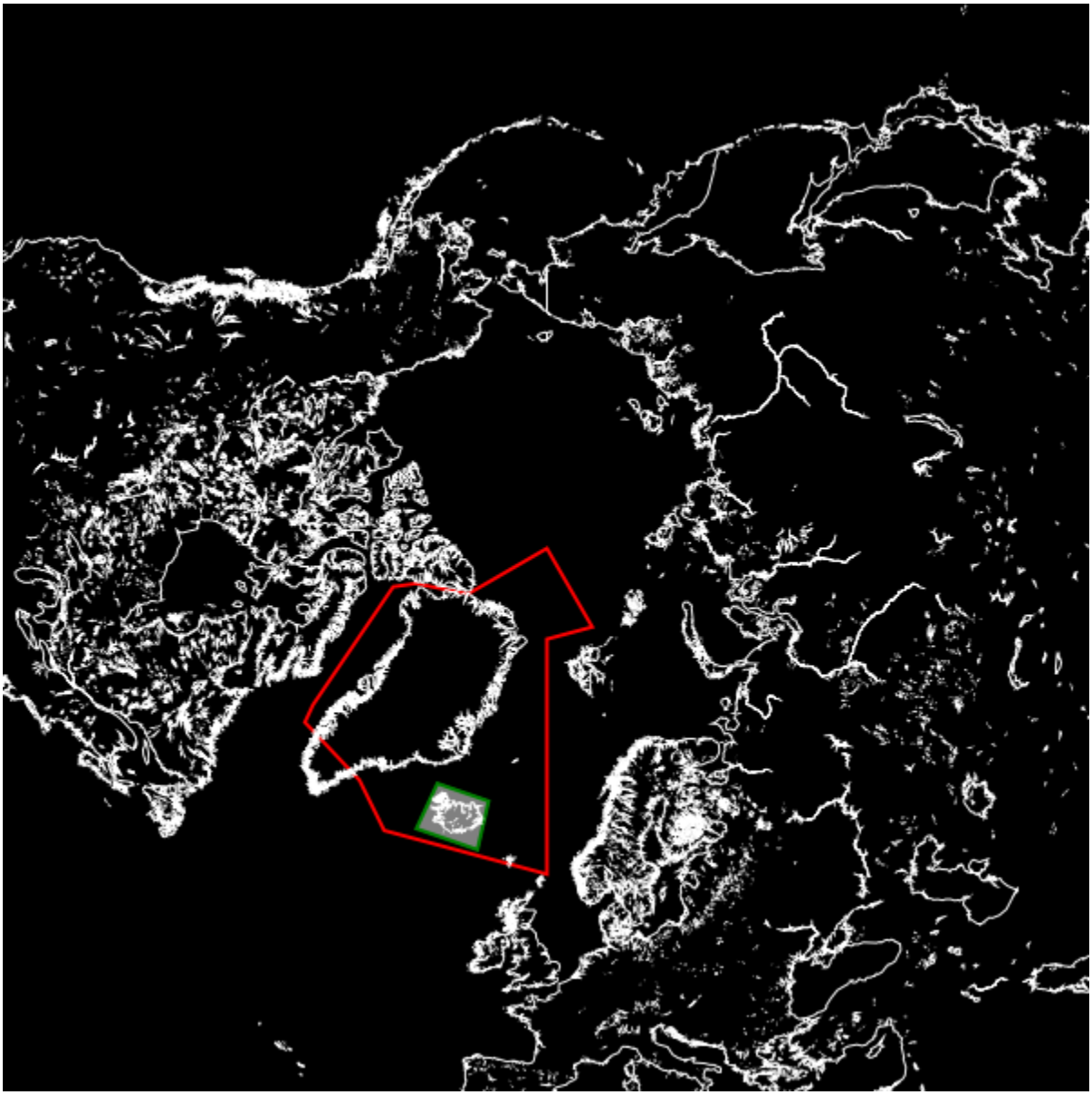
CHAPTER

SIX

CUSTOM SHAPES AND LINES

Pycoast can add custom polygons and lines, useful for outlining special target areas. The following example shows how we might use the `add_polygon` method to highlight the Reykjavik Air Traffic Control area and an additional filled box around Iceland.

```
>>> from PIL import Image
>>> from pycoast import ContourWriterAGG
>>> img = Image.new('RGB', (600, 600))
>>> proj4_string = '+proj=laea +lat_0=90 +lon_0=0 +a=6371228.0 +units=m'
>>> area_extent = (-5326849.0625, -5326849.0625, 5326849.0625, 5326849.0625)
>>> area_def = (proj4_string, area_extent)
>>> cw = ContourWriterAGG('/home/esn/data/gshhs')
...
>>> REYKJAVIK_ATC = [(0.0,73.0),(0.0,61.0),(-30.0,61.0),(-39,63.5),(-55+4/6.0,63.5),(-
↪57+45/60.0,65),(-76,76),(-75,78),(-60,82),(0,90),(30,82),(0,82)]
>>> ICELAND_BOX = [(-25,62.5),(-25,67),(-13,67),(-13,62.5)]
>>> cw.add_polygon(img, area_def, REYKJAVIK_ATC, outline='red',width=2)
>>> cw.add_polygon(img, area_def, ICELAND_BOX, outline='green', fill='gray', width=2)
>>> cw.add_coastlines(img, area_def, resolution='l', level=4)
>>> img.show()
```



The `add_polygon` method accepts a list of longitude, latitude pairs. An equivalent `add_line` method is also available which does not tie the first and last coordinates in the list.

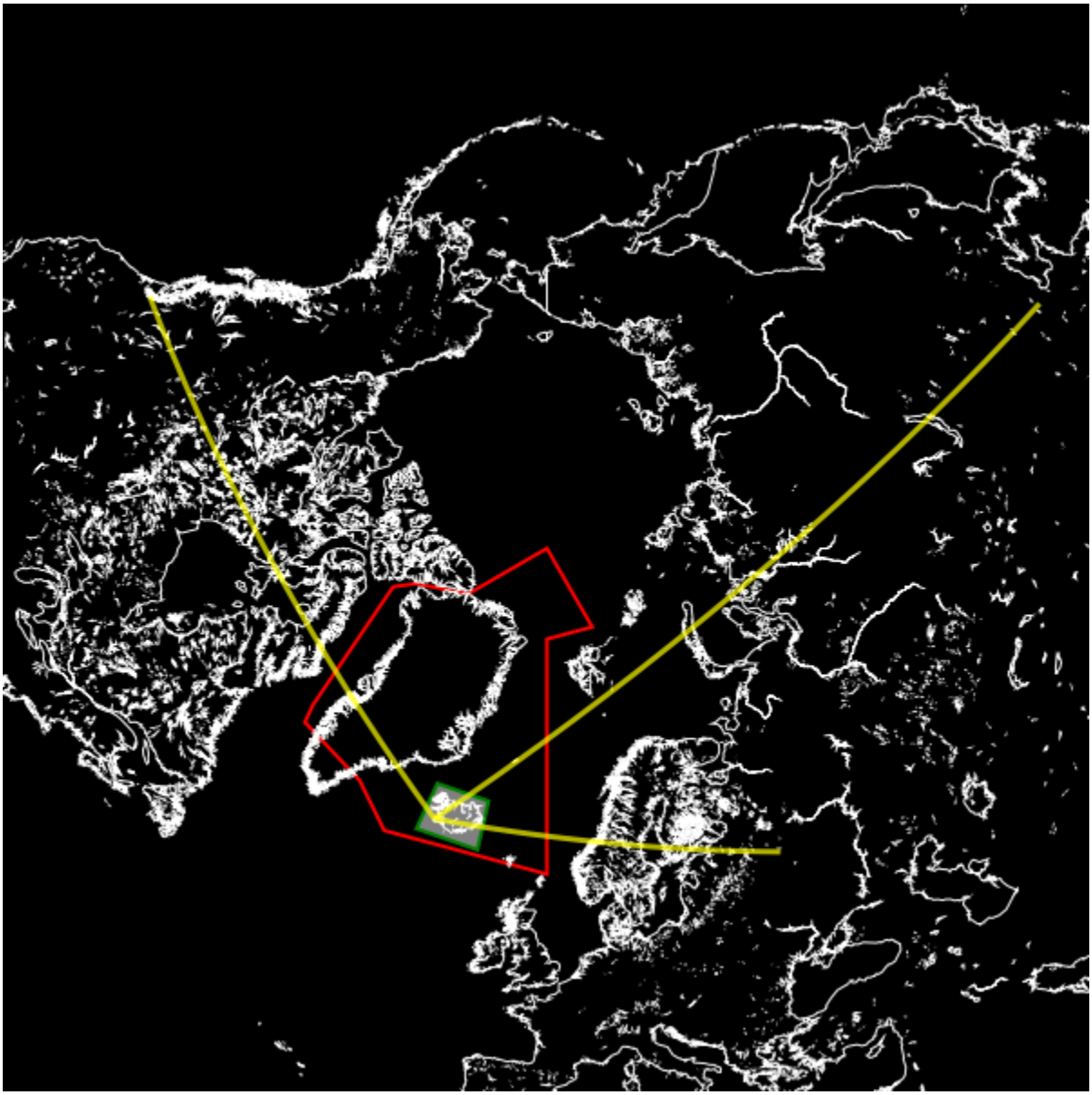
Now we can plot some air traffic routes from Keflavik to Seattle, Moscow and Beijing,

```
>>> ROUTE_KEF_MOS = [(-22.6056, 63.985), (-19.046655824698217, 64.2936159845089), (-15.
↪41883293246517, 64.51404924194419), (-11.744200494490052, 64.64399069686961), (-8.
↪046778033221322, 64.6820416591038), (-4.351563677581442, 64.62778714494442), (-0.
↪6834599011921236, 64.48181810544278), (2.9337905930008565, 64.24569983825512), (6.
↪478548138904879, 63.92189044240429), (9.932010650466118, 63.513618932636106), (13.
↪278688573156892, 63.02473642018875), (16.506610526365268, 62.459555054119136), (19.
↪607285620724404, 61.82268835291907), (22.575472462848946, 61.1189038062041 94), (25.
↪408815405909454, 60.352995069199515), (28.107407514323345, 59.52967751291583), (30.
↪673330797710015, 58.65350788682086), (33.110211639277665, 57.7288266642078), (35.
↪42281629953696, 56.75972029885026), (37.6167, 55.75)]
>>> ROUTE_KEF_SEA = [(-22.6056, 63.985), (-28.15308892820336, 65.36580325755281), (-34.
```

(continues on next page)

(continued from previous page)

```
↪26244035327647, 66.52172028653052), (-40.896187287785146, 67.41807846160079), (-47.
↪960443294166176, 68.02301075853937), (-55.302469834902446, 68.31206181696378), (-62.
↪72513195737088, 68.27259499211274), (-70.01742497152813, 67.90637421611629), (-76.
↪99054572503543, 67.22919099479928), (-83.50520476774184, 66.26770704836584), (-89.
↪48175180569157, 65.05485573003652), (-94.89452260904564, 63.62539374850556), (-99.
↪75771059724035, 62.012611982850714), (-104.1099689970044, 60.24644267746881), (-108.
↪00184199066507, 58.352707879886715), (-111.48717146239099, 56.3531052759957), (-114.
↪61800147728289, 54.26558085318135), (-117.4419933502085, 52.104852107803715), (-120.
↪00142613885524, 49.88294778482337), (-122.3331, 47.6097)]
>>> ROUTE_KEF_BEI = [(-22.6056, 63.985), (-17.489150553128045, 67.07686353046147), (-10.
↪93541135202904, 69.95803521761742), (-2.422591560170639, 72.52376059083646), (8.
↪601530816977142, 74.6151942209109), (22.350514164676376, 76.01770036199035), (38.
↪03768523094268, 76.51449133498859), (53.7147372147881, 76.00872266593849), (67.
↪44042282956654, 74.598879606615), (78.43970951791597, 72.50222030140003), (86.
↪9320528199369, 69.93299364768527), (93.47049967796295, 67.04949777818322), (98.
↪57586637530908, 63.95606630048991), (102.64426083795271, 60.71933633909033), (105.
↪95716114438707, 57.38212969906091), (108.71149093382456, 53.97256160920469), (111.
↪04582088648519, 50.509589240989264), (113.05910256207024, 47.00634823698568), (114.
↪82328673157406, 43.472181706860376), (116.3917, 39.9139)]
>>> cw.add_line(img, area_def, ROUTE_KEF_MOS, outline='yellow',outline_opacity=180,
↪width=3)
>>> cw.add_line(img, area_def, ROUTE_KEF_SEA, outline='yellow',outline_opacity=180,
↪width=3)
>>> cw.add_line(img, area_def, ROUTE_KEF_BEI, outline='yellow',outline_opacity=180,
↪width=3)
```



CHAPTER

SEVEN

ADD CUSTOM POINTS WITH DESCRIPTIONS

Pycoast can add a symbol to points of interest on an image. The following examples show how we might use the `add_points()` method to annotate the points on an image.

First of all, we setup a PIL image with an area definition, then we add coastlines and borders for reference.

```
>>> from PIL import Image
>>> from pycoast import ContourWriterAGG
>>> img = Image.new('RGB', (1024, 1024), (255, 255, 255))
>>> proj4_string = '+proj=laea +lat_0=90 +lon_0=0 +a=6371228.0 +units=m'
>>> area_extent = (-5326849.0625, -5326849.0625, 5326849.0625, 5326849.0625)
>>> area_def = AreaDefinition('nh', 'nh', 'nh', proj4_string, 1024, 1024, area_extent)
>>> cw = ContourWriterAGG('/home/esn/data/gshhs')
>>> cw.add_coastlines(img, area_def, outline='black', resolution='l', level=4)
>>> cw.add_borders(img, area_def, outline='black', width=3, level=1, resolution='c')
```

Now we can add a circle, which is the default symbol, with default point size 6 at the location of Berlin, the name of the location will marked in a text box with black borders and the default text size is 12.

```
>>> points_list = [((13.4050, 52.5200), 'Berlin')]
>>> cw.add_points(pil_img, area, points_list=points_list, font_file=font_file)
```

We can also annotate the image with text only by setting the ptsize to 0. The example below will add 'Rome' at the given location without a symbol.

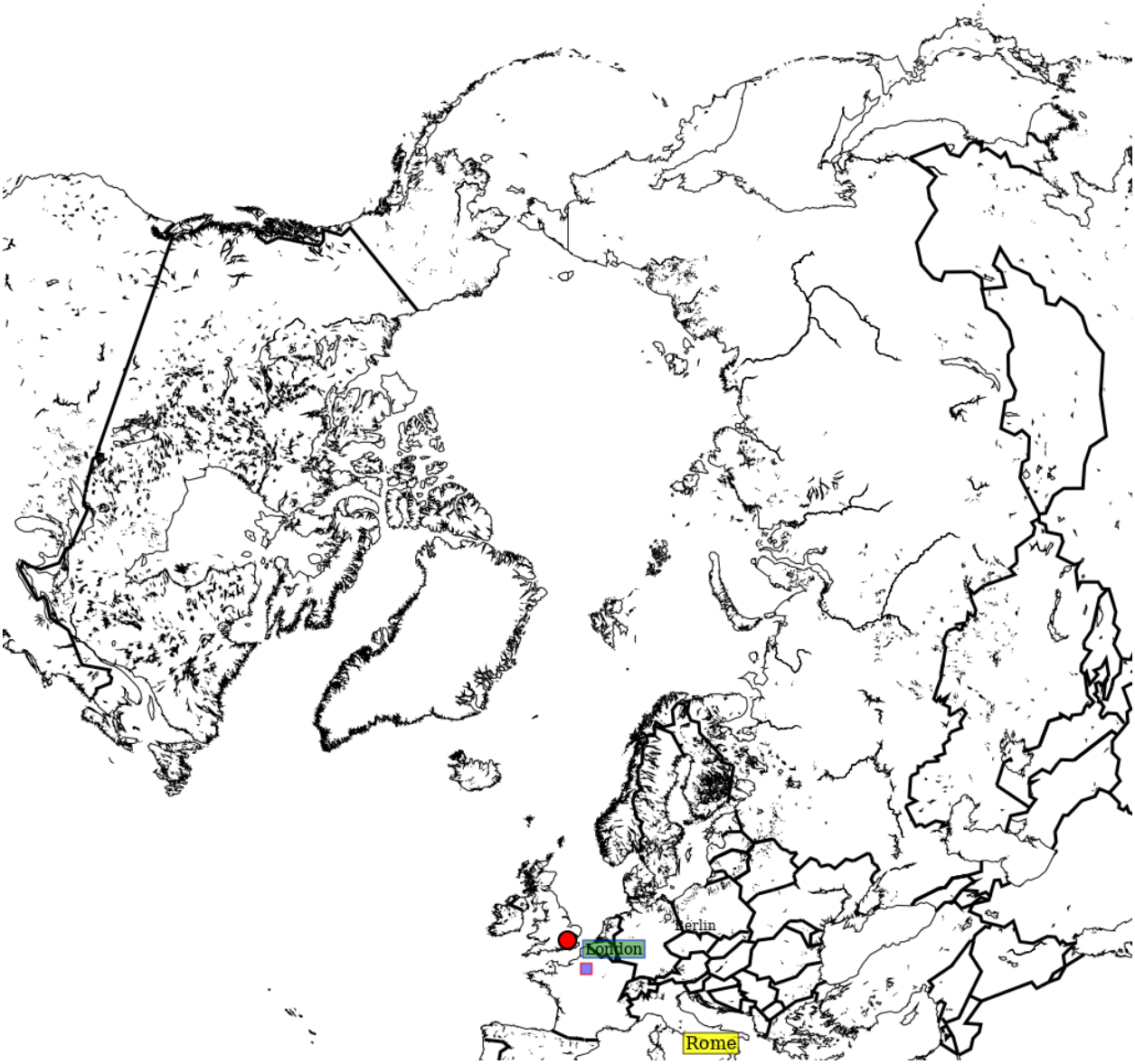
```
>>> points_list = [((12.4964, 41.9028), 'Rome')]
>>> cw.add_points(pil_img, area, points_list=points_list,
...               font_file=font_file, font_size=16,
...               symbol='circle', ptsize=0,
...               box_outline='black', text_linewidth=1,
...               box_fill='yellow', box_opacity=200)
```

Similarly, assign the description as an empty string will only draw the symbol on the image. The example below will draw a square symbol at the location of Paris.

```
>>> points_list = [((2.3522, 48.8566), '')]
>>> cw.add_points(pil_img, area, points_list=points_list,
...               font_file=font_file,
...               symbol='square', ptsize=10,
...               outline='red', width=1,
...               fill='blue', fill_opacity=128)
```

Finally, we can fully customize the annotation as the example below, which will add a circle in black with line width set to 2 and filled in red color with opacity equals 255; the description will be 'London' in a textbox with blue borders and filled with green color with opacity set to 128.

```
>>> points_list = [((0.1278, 51.5074), 'London')]
>>> cw.add_points(img, area_def, points_list=points_list,
...               font_file=font_file, font_size=14,
...               symbol='circle', ptsize=14,
...               outline='black', width=2,
...               fill='red', fill_opacity=255,
...               box_outline='blue', box_linewidth=1.5,
...               box_fill='green', box_opacity=128)
>>> img.show()
```



Please check out the docstrings of the `add_points()` function for the full description of the parameters.

Moreover, we can organize the overlay parameters in a dictionary and use `add_overlay_from_dict()` to apply the overlays in one shot.

```
>>> points = {'points_list': [((2.3522, 48.8566), 'Paris'), ((0.1278, 51.5074), 'London
↪')],
...           'font': font_file,
...           'symbol': 'circle', 'ptsize': 16,
...           'outline': 'black', 'width': 3,
...           'fill': 'red', 'fill_opacity': 128,
...           'box_outline': 'blue', 'box_linewidth': 0.5,
...           'box_fill': 'yellow', 'box_opacity': 200}
```

```
>>> overlays = {'coasts': {'outline': 'black', 'level': 4, 'resolution': 'l'},
...             'borders': {'outline': 'black', 'width': 3, 'level': 1, 'resolution': 'c
↪'},
...             'points': points}
```

```
>>> img = cw.add_overlay_from_dict(overlays, area_def)
```

```
>>> img.save('./add_overlays_from_dict.png')
```

Here we have used the *points* key in the *overlays* dict to pass a list of points. We could also use the *text* key in the *overlays* dict, which is just an alias, they both behave the same. This allows for two different lists of points to be plotted with their own rendering styles. Setting *symbol:None* and *ptsize:0* will render just text without a symbol, but this is not automatically implied by the use of *text*.

The coordinates of the point are specified in geographic degrees longitude (N), latitude (E) by default but can be changed to projected image coordinates in pixels (right, down). Use the *coord_ref* key in your *points* dict and with a value *'lonlat'* or *'image'*. Negative image coordinates will go left/up from the opposite side of the image.

Thus the dict *overlays = { 'text': { 'symbol':None, 'ptsize':0, 'coord_ref':'image'* can be used to plot text strings at fixed points on the image regardless of projection.

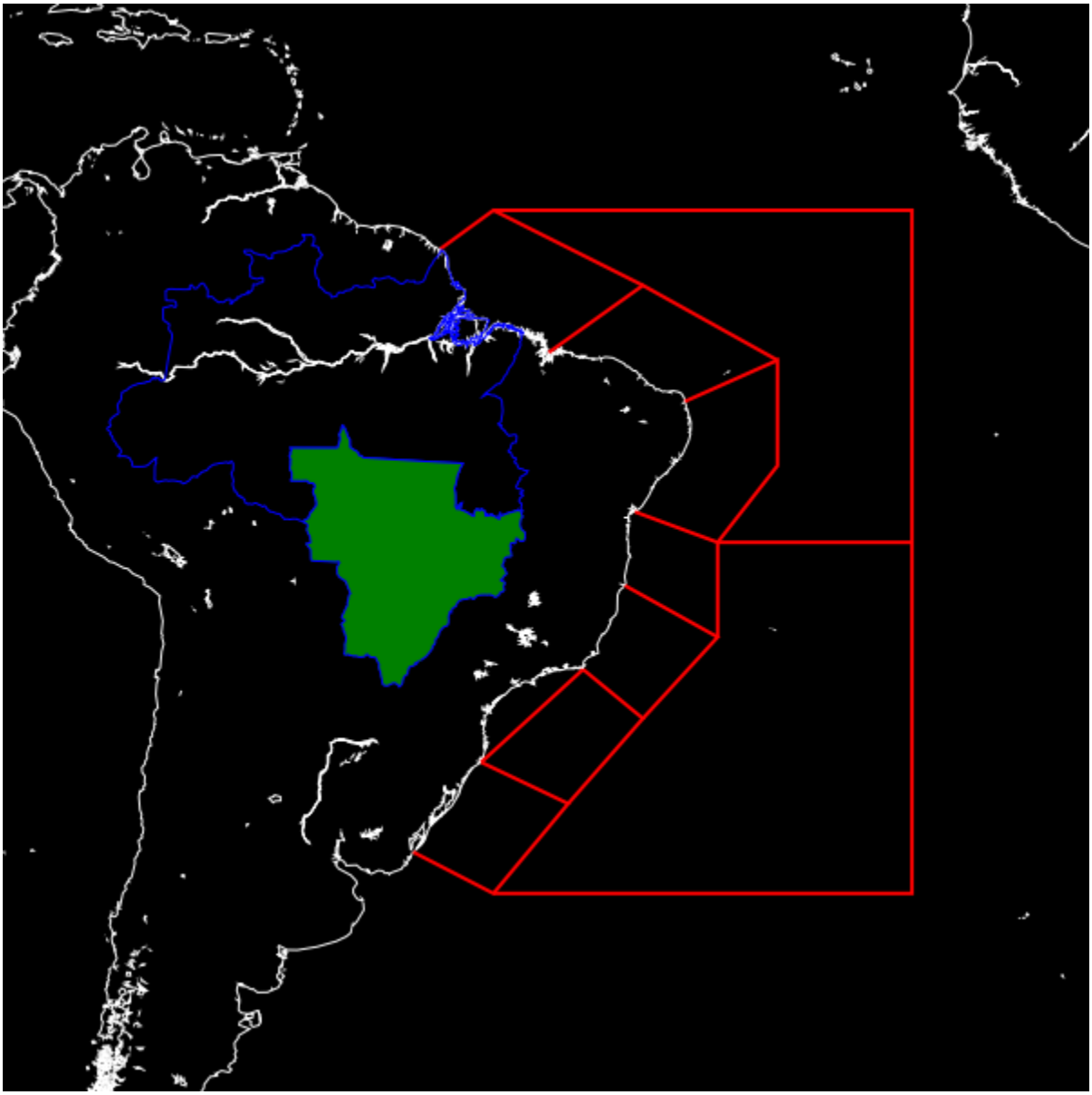
CHAPTER

EIGHT

SHAPES FROM ESRI SHAPE FILES

Pycoast supports plotting of polygons and polylines from ESRI shapefiles, currently only in lonlat coordinates format. In the following example we use `add_shapefile_shapes` method to plot all line shapes found in the file Metareas.shp. We then use the `add_shapefile_shape` (notice the singular) to plot only the 3rd and 4th shape_id within the file BR_Regioes.shp.

```
>>> from pycoast import ContourWriterAGG
>>> from PIL import Image
>>> img = Image.new('RGB', (600, 600))
>>> proj4_string = '+proj=merc +lon_0=-60 +lat_ts=-30.0 +a=6371228.0 +units=m'
>>> area_extent = (-2000000.0, -5000000.0, 5000000.0, 2000000.0)
>>> area_def = (proj4_string, area_extent)
>>> cw = ContourWriterAGG(gshhs_root_dir)
...
>>> cw.add_coastlines(img, area_def, resolution='l', level=4)
>>> cw.add_shapefile_shapes(img, area_def,
                            'tests/test_data/shapes/Metareas.shp',
                            outline='red',width=2)
>>> cw.add_shapefile_shape(img, area_def,
                           'tests/test_data/shapes/divisao_politica/BR_Regioes.shp',
                           3, outline='blue')
>>> cw.add_shapefile_shape(img, area_def,
                           'tests/test_data/shapes/divisao_politica/BR_Regioes.shp',
                           4, outline='blue', fill='green')
```



8.1 Reproject unsupported shapefiles

If your shapefile is not in lonlat coordinates, then you can re-project your shape file using `ogr2ogr` (part of GDAL tools), e.g.

```
$ ogr2ogr original.shp  lonlat.shp  -t_srs "+proj=latlong"
```

This should work if you have all the extra meta-files original.* included with your original.shp. Please refer to the OGR documentation for more information.

8.2 Complex shape drawing

To further customize how shapes are drawn the `add_shapes()` can be used. This is the low-level version of the `add_shapefile_shape` method described above. This method takes an iterable of shape objects to draw with optional drawing parameters. In combination with python generators this can provide a high performance method for drawing multiple shapes with per-shape customization. In the below example a custom generator function is defined to open a shapefile and specify that each polygon should be blue and any other shape should be red.

```
>>> import shapefile
>>> def my_shapes_generator():
...     sf = shapefile.Reader(filename)
...     for shape in sf.shapes():
...         if shape.shapeType == shapefile.POLYGON:
...             kwargs = {'fill': (0, 0, 255)}
...         else:
...             kwargs = {'fill': (255, 0, 0)}
...         yield (shape, kwargs)
... cw.add_shapes(img, area_def, my_shapes_generator())
```

CHAPTER

NINE

TESTING

The tests can be run using pytest:

```
$ pytest pycoast/tests/
```

CHAPTER

TEN

PYCOAST

10.1 pycoast package

10.1.1 Subpackages

pycoast.tests package

Submodules

pycoast.tests.test_pycoast module

Main unit tests for pycoast.

class pycoast.tests.test_pycoast.**FakeAreaDef**(*proj4_string*, *area_extent*, *x_size*, *y_size*)

Bases: object

A fake area definition object.

class pycoast.tests.test_pycoast.**TestContourWriterAGG**

Bases: _ContourWriterTestBase

Test AGG contour writer.

test_add_cities_agg()

test_add_cities_cfg_agg()

test_add_cities_from_dict_agg()

test_add_grid_from_dict_agg()

test_add_one_shapefile_from_cfg_agg()

test_add_points_agg()

test_add_polygon_agg()

test_add_shapefile_shapes_agg()

test_add_shapefiles_from_dict_agg()

test_coastlines_convert_to_rgba_agg()

test_config_file_coasts_and_grid()

test_config_file_points_and_borders_agg()

test_europe_agg()

test_europe_agg_file()

test_geos_agg()

test_grid_agg()

test_grid_agg_file()

test_grid_agg_txt()

test_grid_geos_agg()

test_grid_nh_agg()

test_lonlat_pm_change()

Test that a longlat projection with a non-0 prime meridian is handled correctly.

class pycoast.tests.test_pycoast.**TestContourWriterPIL**

Bases: object

Test PIL-based contour writer.

test_add_cities_cfg_pil(*cw_pil*, *new_test_image*)

test_add_cities_from_dict_pil(*cw_pil*, *new_test_image*)

test_add_cities_pil(*cw_pil*, *new_test_image*)

test_add_grid_from_dict_pil(*cw_pil*, *new_test_image*)

test_add_one_shapefile_from_cfg_pil(*cw_pil*, *new_test_image*)

test_add_points_bad_coord_ref(*cw_pil*)

test_add_points_bad_image_coords(*cw_pil*)

test_add_points_coordinate_conversion(*cw_pil*, *new_test_image*)

Check that a point added with lonlat coordinates matches the same point in pixel coordinates.

test_add_points_pil(*cw_pil*, *new_test_image*)

test_add_polygon(*cw_pil*, *new_test_image*)

test_add_shapefile_shapes(*cw_pil*, *new_test_image*)

test_add_shapefiles_from_dict_pil(*cw_pil*, *new_test_image*)

test_config_file_coasts_and_grid(*cw_pil*, *new_test_image*)

test_config_file_points_and_borders_pil(*cw_pil*, *new_test_image*)

test_europe_coastlines_rivers_and_borders(*cw_pil*, *new_test_image*)

Test coastlines, rivers and borders over Europe.

test_europe_coastlines_rivers_and_borders_on_file(*cw_pil*, *test_file_path*)

Test coastlines, rivers and borders over Europe on a file.

test_geos(*cw_pil*, *new_test_image*)

test_grid(*cw_pil*, *new_test_image*, *filename*, *shape*, *area_def*, *level*, *grid_kwargs*)

test_grid_file(*cw_pil*, *grid_file_path*)

test_grid_nh(*cw_pil*, *new_test_image*)

class pycoast.tests.test_pycoast.**TestFromConfig**

Bases: object

Test burning overlays from a config file.

test_cache_generation_reuse(*tmpdir*)

Test generating a transparent foreground and cache it.

test_cache_nocache_consistency(*tmp_path*, *include_background_pattern*, *background_mode*, *upper_right_opacity*)

Test that an image generated with an image looks the same when using a cached foreground.

test_caching_with_param_changes(*tmpdir*)

Testing caching when changing parameters.

test_foreground()

Test generating a transparent foreground.

test_get_resolution(*crs_input*, *extents*, *size*, *exp_resolution*)

Get the automagical resolution computation.

pycoast.tests.test_pycoast.**bering_straight**()

Create an area covering the Bering straight.

pycoast.tests.test_pycoast.**brazil**()

Create a Brazil area.

pycoast.tests.test_pycoast.**cd_test_dir**(*monkeypatch*)

Change directory to the pycoast/tests directory.

pycoast.tests.test_pycoast.**cw_agg**()

Create a PIL ContourWriter.

pycoast.tests.test_pycoast.**cw_pil**()

Create a PIL ContourWriter.

pycoast.tests.test_pycoast.**dateline_1**()

Create an area covering the dateline.

pycoast.tests.test_pycoast.**dateline_2**()

Create another area covering the dateline.

pycoast.tests.test_pycoast.**eurasia**()

Create a Eurasia area.

pycoast.tests.test_pycoast.**europe**()

Create a Europe area.

pycoast.tests.test_pycoast.**europe_1024**()

Create a Europe area in 1024 pixels size.

pycoast.tests.test_pycoast.**fft_metric**(*data1*, *data2*, *max_value=0.1*, *plot_failure=False*)

Execute FFT metric.

pycoast.tests.test_pycoast.**fft_proj_rms**(*a1*, *a2*)

Compute the RMS of differences between FFT vectors of a1 and projection of FFT vectors of a2.

This metric is sensitive to large scale changes and image noise but insensitive to small rendering differences.

pycoast.tests.test_pycoast.**geos**()

Create a geos area.

pycoast.tests.test_pycoast.**germ**()

Create an area covering Germany.

pycoast.tests.test_pycoast.**grid_file_path**(*tmp_path*)

Create a test grid image file on disk.

pycoast.tests.test_pycoast.**hawaii**()

Create an area covering Hawai.

pycoast.tests.test_pycoast.**images_match**(*ref_image*, *test_image*)

Check is images match.

pycoast.tests.test_pycoast.**lonlat_0**(*pm=0*)

Create longlat projection over Cuba.

pycoast.tests.test_pycoast.**new_test_image**(*request*, *tmp_path*)

Create a new test image, and save it to tmp_path if the test fails.

pycoast.tests.test_pycoast.**nh**()

Create the nh area.

pycoast.tests.test_pycoast.**nh_1024**()

Create the nh area in 1024x1024 pixels.

pycoast.tests.test_pycoast.**nh_425**()

Create the nh area in 425 pixels size.

pycoast.tests.test_pycoast.**nh_def**(*shape*)

Create the nh area definition with custom shape.

pycoast.tests.test_pycoast.**north_atlantic**()

Create a North Atlantic area.

pycoast.tests.test_pycoast.**south_america**()

Create an area covering south America.

pycoast.tests.test_pycoast.**test_file_path**(*tmp_path*)

Create a test image file on disk.

pycoast.tests.test_pycoast.**test_no_scratch**(*new_test_image*, *cw*, *filename*, *shape*, *area_def*, *specific_kwargs*)

Test no scratches are visible.

pycoast.tests.test_pycoast.**test_shapes**(*new_test_image*, *cw*, *filename*, *area_def*, *specific_kwargs*)

Test western/eastern shapes.

pycoast.tests.test_pycoast.**uk_and_ireland**()

Create an area covering Ireland and the UK.

Module contents

Pycoast unit tests.

10.1.2 Submodules

pycoast.conftest module

The conftest file.

pycoast.conftest.**pytest_runtest_makereport**(*item*, *call*)

Add test status in the report for fixtures to use.

pycoast.cw_agg module

ContourWriter based on the aggdraw library.

class pycoast.cw_agg.**ContourWriterAGG**(*db_root_path=None*)

Bases: *ContourWriterBase*

Adds countours from GSHHS and WDBII to images using the AGG engine for high quality images.

Parameters

db_root_path
[str] Path to root dir of GSHHS and WDBII shapefiles

add_borders(*image*, *area_def*, *resolution='c'*, *level=1*, *outline='white'*, *width=1*, *outline_opacity=255*, *x_offset=0*, *y_offset=0*)

Add borders to a PIL image object.

Parameters

image
[object] PIL image object

proj4_string
[str] Projection of area as Proj.4 string

area_extent
[list] Area extent as a list (LL_x, LL_y, UR_x, UR_y)

resolution
[str, optional {'c', 'l', 'i', 'h', 'f'}] Dataset resolution to use

level
[int, optional {1, 2, 3}] Detail level of dataset

outline
[str or (R, G, B), optional] Border color

width
[float, optional] Width of coastline

outline_opacity
[int, optional {0; 255}] Opacity of coastline color

x_offset
[float, optional] Pixel offset in x direction

y_offset
[float, optional] Pixel offset in y direction

`add_borders_to_file`(*filename*, *area_def*, *resolution='c'*, *level=1*, *outline='white'*, *width=1*, *outline_opacity=255*, *x_offset=0*, *y_offset=0*)

Add borders to an image file. The resulting image is in 'RGBA' mode.

Parameters

image
[object] Image file

proj4_string
[str] Projection of area as Proj.4 string

area_extent
[list] Area extent as a list (LL_x, LL_y, UR_x, UR_y)

resolution
[str, optional {'c', 'l', 'i', 'h', 'f'}] Dataset resolution to use

level
[int, optional {1, 2, 3}] Detail level of dataset

outline
[str or (R, G, B), optional] Border color

width
[float, optional] Width of coastline

outline_opacity
[int, optional {0; 255}] Opacity of coastline color

x_offset
[float, optional] Pixel offset in x direction

y_offset
[float, optional] Pixel offset in y direction

`add_coastlines`(*image*, *area_def*, *resolution='c'*, *level=1*, *fill=None*, *fill_opacity=255*, *outline='white'*, *width=1*, *outline_opacity=255*, *x_offset=0*, *y_offset=0*)

Add coastlines to a PIL image object.

Parameters

image
[object] PIL image object

proj4_string
[str] Projection of area as Proj.4 string

area_extent
[list] Area extent as a list (LL_x, LL_y, UR_x, UR_y)

resolution
[str, optional {'c', 'l', 'i', 'h', 'f'}] Dataset resolution to use

level
[int, optional {1, 2, 3, 4}] Detail level of dataset

fill
[str or (R, G, B), optional] Land color

fill_opacity
[int, optional {0; 255}] Opacity of land color

outline
[str or (R, G, B), optional] Coastline color

width
[float, optional] Width of coastline

outline_opacity
[int, optional {0; 255}] Opacity of coastline color

x_offset
[float, optional] Pixel offset in x direction

y_offset
[float, optional] Pixel offset in y direction

`add_coastlines_to_file`(*filename*, *area_def*, *resolution='c'*, *level=1*, *fill=None*, *fill_opacity=255*, *outline='white'*, *width=1*, *outline_opacity=255*, *x_offset=0*, *y_offset=0*)

Add coastlines to an image file. The resulting image is in 'RGBA' mode.

Parameters

filename
[str] Image file

proj4_string
[str] Projection of area as Proj.4 string

area_extent
[list] Area extent as a list (LL_x, LL_y, UR_x, UR_y)

resolution
[str, optional {'c', 'l', 'i', 'h', 'f'}] Dataset resolution to use

level
[int, optional {1, 2, 3, 4}] Detail level of dataset

fill
[str or (R, G, B), optional] Land color

fill_opacity
[int, optional {0; 255}] Opacity of land color

outline
[str or (R, G, B), optional] Coastline color

width
[float, optional] Width of coastline

outline_opacity
[int, optional {0; 255}] Opacity of coastline color

x_offset
[float, optional] Pixel offset in x direction

y_offset
[float, optional] Pixel offset in y direction

`add_grid`(*image*, *area_def*, *Dlonlat*, *dlonlat*, *font=None*, *write_text=True*, *fill=None*, *fill_opacity=255*, *outline='white'*, *width=1.0*, *outline_opacity=255*, *minor_outline='white'*, *minor_width=0.5*, *minor_outline_opacity=255*, *minor_is_tick=True*, *lon_placement='tb'*, *lat_placement='lr'*)

Add a lon-lat grid to a PIL image object.

Parameters

image
[object] PIL image object

proj4_string
[str] Projection of area as Proj.4 string

Dlonlat: (float, float)
Major grid line separation

dlonlat: (float, float)
Minor grid line separation

font: Aggdraw Font object, optional
Font for major line markings

write_text
[boolean, optional] Determine if line markings are enabled

fill_opacity
[int, optional {0; 255}] Opacity of text

outline
[str or (R, G, B), optional] Major line color

width
[float, optional] Major line width

outline_opacity
[int, optional {0; 255}] Opacity of major lines

minor_outline
[str or (R, G, B), optional] Minor line/tick color

minor_width
[float, optional] Minor line width

minor_outline_opacity
[int, optional {0; 255}] Opacity of minor lines/ticks

minor_is_tick
[boolean, optional] Use tick minor line style (True) or full minor line style (False)

`add_grid_to_file`(*filename*, *area_def*, *Dlonlat*, *dlonlat*, *font=None*, *write_text=True*, *fill=None*, *fill_opacity=255*, *outline='white'*, *width=1*, *outline_opacity=255*, *minor_outline='white'*, *minor_width=0.5*, *minor_outline_opacity=255*, *minor_is_tick=True*, *lon_placement='tb'*, *lat_placement='lr'*)

Add a lon-lat grid to an image. The resulting image is in 'RGBA' mode.

Parameters

image
[object] PIL image object

proj4_string
[str] Projection of area as Proj.4 string

Dlonlat: (float, float)
Major grid line separation

dlonlat: (float, float)
Minor grid line separation

font: Aggdraw Font object, optional
Font for major line markings

write_text
[boolean, optional] Determine if line markings are enabled

fill_opacity
[int, optional {0; 255}] Opacity of text

outline
[str or (R, G, B), optional] Major line color

width
[float, optional] Major line width

outline_opacity
[int, optional {0; 255}] Opacity of major lines

minor_outline
[str or (R, G, B), optional] Minor line/tick color

minor_width
[float, optional] Minor line width

minor_outline_opacity
[int, optional {0; 255}] Opacity of minor lines/ticks

minor_is_tick
[boolean, optional] Use tick minor line style (True) or full minor line style (False)

`add_line`(*image*, *area_def*, *lonlats*, *fill=None*, *fill_opacity=255*, *outline='white'*, *width=1*, *outline_opacity=255*, *x_offset=0*, *y_offset=0*)

Add a user defined poly-line from a list of (lon,lat) coordinates.

Parameters

image
[object] PIL image object

area_def
[list [proj4_string, area_extent]]

proj4_string : str
Projection of area as Proj.4 string

area_extent : list
Area extent as a list (LL_x, LL_y, UR_x, UR_y)

lonlats
[list of lon lat pairs] e.g. [(10,20),(20,30),...,(20,20)]

outline
[str or (R, G, B), optional] line color

width
[float, optional] line width

outline_opacity
[int, optional {0; 255}] Opacity of lines

x_offset
[float, optional] Pixel offset in x direction

y_offset
[float, optional] Pixel offset in y direction

`add_polygon`(*image*, *area_def*, *lonlats*, *fill=None*, *fill_opacity=255*, *outline='white'*, *width=1*, *outline_opacity=255*, *x_offset=0*, *y_offset=0*)

Add a user defined polygon from a list of (lon,lat) coordinates.

Parameters

image
[object] PIL image object

area_def
[list [proj4_string, area_extent]]

proj4_string : str
Projection of area as Proj.4 string

area_extent : list
Area extent as a list (LL_x, LL_y, UR_x, UR_y)

lonlats
[list of lon lat pairs] e.g. [(10,20),(20,30),...,(20,20)]

fill
[str or (R, G, B), optional] Polygon fill color

fill_opacity
[int, optional {0; 255}] Opacity of polygon fill

outline
[str or (R, G, B), optional] line color

width
[float, optional] line width

outline_opacity
[int, optional {0; 255}] Opacity of lines

x_offset
[float, optional] Pixel offset in x direction

y_offset
[float, optional] Pixel offset in y direction

`add_rivers`(*image*, *area_def*, *resolution='c'*, *level=1*, *outline='white'*, *width=1*, *outline_opacity=255*, *x_offset=0*, *y_offset=0*)

Add rivers to a PIL image object.

Parameters

image
[object] PIL image object

proj4_string
[str] Projection of area as Proj.4 string

area_extent
[list] Area extent as a list (LL_x, LL_y, UR_x, UR_y)

resolution
[str, optional {'c', 'l', 'i', 'h', 'f'}] Dataset resolution to use

level
[int, optional {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}] Detail level of dataset

outline
[str or (R, G, B), optional] River color

width
[float, optional] Width of coastline

outline_opacity
[int, optional {0; 255}] Opacity of coastline color

x_offset
[float, optional] Pixel offset in x direction

y_offset
[float, optional] Pixel offset in y direction

`add_rivers_to_file`(*filename*, *area_def*, *resolution='c'*, *level=1*, *outline='white'*, *width=1*, *outline_opacity=255*, *x_offset=0*, *y_offset=0*)

Add rivers to an image file. The resulting image is in 'RGBA' mode.

Parameters

image
[object] Image file

proj4_string
[str] Projection of area as Proj.4 string

area_extent
[list] Area extent as a list (LL_x, LL_y, UR_x, UR_y)

resolution
[str, optional {'c', 'l', 'i', 'h', 'f'}] Dataset resolution to use

level
[int, optional {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}] Detail level of dataset

outline
[str or (R, G, B), optional] River color

width
[float, optional] Width of coastline

outline_opacity
[int, optional {0; 255}] Opacity of coastline color

x_offset
[float, optional] Pixel offset in x direction

y_offset
[float, optional] Pixel offset in y direction

`add_shapefile_shape`(*image*, *area_def*, *filename*, *shape_id*, *feature_type=None*, *fill=None*, *fill_opacity=255*, *outline='white'*, *width=1*, *outline_opacity=255*, *x_offset=0*, *y_offset=0*)

Add a single shape file shape from an ESRI shapefile.

Note: To add all shapes in file use the 'add_shape_file_shapes' routine

Note: Currently only supports lon-lat formatted coordinates.

Parameters

image
[object] PIL image object

area_def
[list [proj4_string, area_extent]]

proj4_string : str
Projection of area as Proj.4 string

area_extent : list
Area extent as a list (LL_x, LL_y, UR_x, UR_y)

filename
[str] Path to ESRI shape file

shape_id
[int] integer id of shape in shape file {0; … }

feature_type
['polygon' or 'line',] only to override the shape type defined in shapefile, optional

fill
[str or (R, G, B), optional] Polygon fill color

fill_opacity
[int, optional {0; 255}] Opacity of polygon fill

outline
[str or (R, G, B), optional] line color

width
[float, optional] line width

outline_opacity
[int, optional {0; 255}] Opacity of lines

x_offset
[float, optional] Pixel offset in x direction

y_offset
[float, optional] Pixel offset in y direction

`add_shapefile_shapes`(*image*, *area_def*, *filename*, *feature_type=None*, *fill=None*, *fill_opacity=255*, *outline='white'*, *width=1*, *outline_opacity=255*, *x_offset=0*, *y_offset=0*)

Add shape file shapes from an ESRI shapefile.

Note: Currently only supports lon-lat formatted coordinates.

Parameters

image
[object] PIL image object

area_def
[list [proj4_string, area_extent]]

proj4_string : str
Projection of area as Proj.4 string

area_extent : list
Area extent as a list (LL_x, LL_y, UR_x, UR_y)

filename
[str] Path to ESRI shape file

feature_type
['polygon' or 'line',] only to override the shape type defined in shapefile, optional

fill
[str or (R, G, B), optional] Polygon fill color

fill_opacity
[int, optional {0; 255}] Opacity of polygon fill

outline
[str or (R, G, B), optional] line color

width
[float, optional] line width

outline_opacity
[int, optional {0; 255}] Opacity of lines

x_offset
[float, optional] Pixel offset in x direction

y_offset
[float, optional] Pixel offset in y direction

pycoast.cw_base module

Base class for contour writers.

class pycoast.cw_base.**ContourWriterBase**(*db_root_path=None*)

Bases: object

Base class for contourwriters. Do not instantiate.

Parameters

db_root_path
[str] Path to root dir of GSHHS and WDBII shapefiles

add_cities(*image*, *area_def*, *cities_list*, *font_file*, *font_size=12*, *symbol='circle'*, *ptsize=6*, *outline='black'*, *fill='white'*, *db_root_path=None*, ***kwargs*)

Add cities (symbol and UTF-8 names as description) to a PIL image object.

Parameters

image
[object] PIL image object

area_def
[object] Area Definition of the provided image

cities_list
[list of city names ['City1', 'City2', City3, …, 'CityN']]

a list of UTF-8 or ASCII strings. If either of these strings is found
in file db_root_path/CITIES/cities.red, longitude and latitude is read
and the city is added like a point with its UTF-8 name as description
e.g. cities_list = ['Zurich', 'Oslo'] will add cities 'Zürich', 'Oslo'.
Check the README_PyCoast.txt in archive cities2022.zip for more details.

font_file
[str] Path to font file

font_size
[int] Size of font

symbol
[string] type of symbol, one of the elelments from the list ['circle', 'hexagon', 'pentagon', 'square', 'triangle', 'star8', 'star7', 'star6', 'star5', 'asterisk']

ptsize
[int] Size of the point.

outline
[str or (R, G, B), optional] Line color of the symbol

fill
[str or (R, G, B), optional] Filling color of the symbol

Optional keyword arguments

width
[float] Line width of the symbol

outline_opacity
[int, optional {0; 255}] Opacity of the line of the symbol.

fill_opacity
[int, optional {0; 255}] Opacity of the filling of the symbol

box_outline
[str or (R, G, B), optional] Line color of the textbox borders.

box_linewidth
[float] Line width of the the borders of the textbox

box_fill
[str or (R, G, B), optional] Filling color of the background of the textbox.

box_opacity
[int, optional {0; 255}] Opacity of the background filling of the textbox.

`add_overlay_from_config`(*config_file*, *area_def*, *background=None*)

Create and return a transparent image adding all the overlays contained in a configuration file.

Parameters

config_file
[str] Configuration file name

area_def
[object] Area Definition of the creating image

`add_overlay_from_dict`(*overlays*, *area_def*, *cache_epoch=None*, *background=None*)

Create and return a transparent image adding all the overlays contained in the *overlays* dict.

Optionally caches overlay results for faster rendering of images with the same provided AreaDefinition and parameters. Cached results are identified by hashing the AreaDefinition and the overlays dictionary.

Note that if `background` is provided and caching is not used, the result will be the final result of applying the overlays onto the background. This is due to an optimization step avoiding creating a separate overlay image in memory when it isn't needed.

Warning: Font objects are ignored in parameter hashing as they can't be easily hashed. Therefore, font changes will not trigger a new rendering for the cache.

Parameters

overlays
[dict] overlays configuration

area_def
[object] Area Definition of the creating image

cache_epoch: seconds since epoch
The latest time allowed for cache the cache file. If the cache file is older than this (mtime), the cache should be regenerated. Defaults to 0 meaning always reuse the cached file if it exists. Requires "cache" to be configured in the provided dictionary (see below).

background: pillow image instance
The image on which to write the overlays on. If it's None (default), a new image is created, otherwise the provide background is used and changed *in place*.

The keys in *overlays* that will be taken into account are: cache, coasts, rivers, borders, shapefiles, grid, cities, points

For all of them except *cache*, the items are the same as the corresponding functions in pycoast, so refer to the docstrings of these functions (add_coastlines, add_rivers, add_borders, add_shapefile_shapes, add_grid, add_cities, add_points). For cache, two parameters are configurable:

- ***file*: specify the directory and the prefix**
  of the file to save the caches decoration to (for example /var/run/black_coasts_red_borders)
- ***regenerate*: True or False (default) to force the overwriting**
  of an already cached file.

`add_points`(*image*, *area_def*, *points_list*, *font_file*, *font_size=12*, *symbol='circle'*, *ptsize=6*, *outline='black'*, *fill='white'*, *coord_ref='lonlat'*, ***kwargs*)

Add a symbol and/or text at the point(s) of interest to a PIL image object.

Parameters

image
[object] PIL image object

area_def
[object] Area Definition of the provided image

points_list
[list [((x, y), desc)]]

a list of points defined with (x, y) in float and a desc in string
[((x1, y1), desc1), ((x2, y2), desc2)]
See coord_ref (below) for the meaning of x, y.
x : float
longitude or pixel x of a point
y : float
latitude or pixel y of a point

desc : str
description of a point

font_file
[str] Path to font file

font_size
[int] Size of font

symbol
[string] type of symbol, one of the elelments from the list [‘circle’, ‘hexagon’, ‘pentagon’, ‘square’, ‘triangle’, ‘star8’, ‘star7’, ‘star6’, ‘star5, ‘asterisk’]

ptsize
[int] Size of the point (should be zero if symbol:None).

outline
[str or (R, G, B), optional] Line color of the symbol

fill
[str or (R, G, B), optional] Filling color of the symbol

Optional keyword arguments

coord_ref
[string] The interpretation of x,y in points_list: ‘lonlat’ (the default: x is degrees E, y is degrees N), or ‘image’ (x is pixels right, y is pixels down). If image coordinates are negative they are interpreted relative to the end of the dimension like standard Python indexing.

width
[float] Line width of the symbol

outline_opacity
[int, optional {0; 255}] Opacity of the line of the symbol.

fill_opacity
[int, optional {0; 255}] Opacity of the filling of the symbol

box_outline
[str or (R, G, B), optional] Line color of the textbox borders.

box_linewidth
[float] Line width of the the borders of the textbox

box_fill
[str or (R, G, B), optional] Filling color of the background of the textbox.

box_opacity
[int, optional {0; 255}] Opacity of the background filling of the textbox.

`add_shapes`(*image*, *area_def*, *shapes*, *feature_type=None*, *x_offset=0*, *y_offset=0*, ***kwargs*)

Draw shape objects to PIL image.

Parameters

image
[Image] PIL Image to draw shapes on

area_def
[(proj_str, area_extent) or AreaDefinition] Geolocation information for the provided image

shapes: iterable
Series of shape objects from pyshp. Can also be a series of 2-element tuples where the first element is the shape object and the second is a dictionary of additional drawing parameters for this shape.

feature_type: str
'polygon' or 'line' or None for what to draw shapes as. Default is to draw the shape with the type in the shapefile.

kwargs:
Extra drawing keyword arguments for all shapes

draw_hexagon(*draw*, *x*, *y*, *ptsize*, ***kwargs*)

draw_pentagon(*draw*, *x*, *y*, *ptsize*, ***kwargs*)

draw_star(*draw*, *symbol*, *x*, *y*, *ptsize*, ***kwargs*)

draw_triangle(*draw*, *x*, *y*, *ptsize*, ***kwargs*)

property is_agg: bool
Get if we are using the 'agg' backend.

class pycoast.cw_base.**GeoNamesCitiesParser**(*cities_filename: str*)
Bases: object

Helper for parsing citiesN.txt files from GeoNames.org.

iter_cities_names_lon_lat(*cities_list: list[str]*) → Generator[tuple[str, float, float], None, None]

pycoast.cw_base.**get_resolution_from_area**(*area_def*)
Get the best resolution for an area definition.

pycoast.cw_base.**hash_dict**(*dict_to_hash: dict*) → str
Hash dict object by serializing with json.

pycoast.cw_pil module

PIL-based ContourWriter.

class pycoast.cw_pil.**ContourWriterPIL**(*db_root_path=None*)
Bases: *ContourWriterBase*

Adds countours from GSHHS and WDBII to images.

Parameters

db_root_path
[str] Path to root dir of GSHHS and WDBII shapefiles

add_borders(*image*, *area_def*, *resolution='c'*, *level=1*, *outline='white'*, *x_offset=0*, *y_offset=0*)
Add borders to a PIL image object.

Parameters

image
[object] PIL image object

proj4_string
[str] Projection of area as Proj.4 string

area_extent
[list] Area extent as a list (LL_x, LL_y, UR_x, UR_y)

resolution
[str, optional {'c', 'l', 'i', 'h', 'f'}] Dataset resolution to use

level
[int, optional {1, 2, 3}] Detail level of dataset

outline
[str or (R, G, B), optional] Border color

x_offset
[float, optional] Pixel offset in x direction

y_offset
[float, optional] Pixel offset in y direction

`add_borders_to_file`(*filename*, *area_def*, *resolution='c'*, *level=1*, *outline='white'*, *x_offset=0*, *y_offset=0*)

Add borders to an image file.

Parameters

image
[object] Image file

proj4_string
[str] Projection of area as Proj.4 string

area_extent
[list] Area extent as a list (LL_x, LL_y, UR_x, UR_y)

resolution
[str, optional {'c', 'l', 'i', 'h', 'f'}] Dataset resolution to use

level
[int, optional {1, 2, 3}] Detail level of dataset

outline
[str or (R, G, B), optional] Border color

x_offset
[float, optional] Pixel offset in x direction

y_offset
[float, optional] Pixel offset in y direction

`add_coastlines`(*image*, *area_def*, *resolution='c'*, *level=1*, *fill=None*, *outline='white'*, *x_offset=0*, *y_offset=0*)

Add coastlines to a PIL image object.

Parameters

image
[object] PIL image object

proj4_string
[str] Projection of area as Proj.4 string

area_extent
[list] Area extent as a list (LL_x, LL_y, UR_x, UR_y)

resolution
[str, optional {'c', 'l', 'i', 'h', 'f'}] Dataset resolution to use

level
[int, optional {1, 2, 3, 4}] Detail level of dataset

fill
[str or (R, G, B), optional] Land color

outline
[str or (R, G, B), optional] Coastline color

x_offset
[float, optional] Pixel offset in x direction

y_offset
[float, optional] Pixel offset in y direction

`add_coastlines_to_file`(*filename*, *area_def*, *resolution='c'*, *level=1*, *fill=None*, *outline='white'*, *x_offset=0*, *y_offset=0*)

Add coastlines to an image file.

Parameters

filename
[str] Image file

proj4_string
[str] Projection of area as Proj.4 string

area_extent
[list] Area extent as a list (LL_x, LL_y, UR_x, UR_y)

resolution
[str, optional {'c', 'l', 'i', 'h', 'f'}] Dataset resolution to use

level
[int, optional {1, 2, 3, 4}] Detail level of dataset

fill
[str or (R, G, B)] Land color

outline
[str or (R, G, B), optional] Coastline color

x_offset
[float, optional] Pixel offset in x direction

y_offset
[float, optional] Pixel offset in y direction

`add_grid`(*image*, *area_def*, *Dlonlat*, *dlonlat*, *font=None*, *write_text=True*, *fill=None*, *outline='white'*, *minor_outline='white'*, *minor_is_tick=True*, *lon_placement='tb'*, *lat_placement='lr'*)

Add a lon-lat grid to a PIL image object.

Parameters

image
[object] PIL image object

proj4_string
[str] Projection of area as Proj.4 string

Dlonlat: (float, float)
Major grid line separation

dlonlat: (float, float)
Minor grid line separation

font: PIL ImageFont object, optional
Font for major line markings

write_text
[boolean, optional] Determine if line markings are enabled

fill
[str or (R, G, B), optional] Text color

outline
[str or (R, G, B), optional] Major line color

minor_outline
[str or (R, G, B), optional] Minor line/tick color

minor_is_tick
[boolean, optional] Use tick minor line style (True) or full minor line style (False)

`add_grid_to_file`(*filename*, *area_def*, *Dlonlat*, *dlonlat*, *font=None*, *write_text=True*, *fill=None*, *outline='white'*, *minor_outline='white'*, *minor_is_tick=True*, *lon_placement='tb'*, *lat_placement='lr'*)

Add a lon-lat grid to an image file.

Parameters

image
[object] PIL image object

proj4_string
[str] Projection of area as Proj.4 string

Dlonlat: (float, float)
Major grid line separation

dlonlat: (float, float)
Minor grid line separation

font: PIL ImageFont object, optional
Font for major line markings

write_text
[boolean, optional] Determine if line markings are enabled

fill
[str or (R, G, B), optional] Text color

outline
[str or (R, G, B), optional] Major line color

minor_outline
[str or (R, G, B), optional] Minor line/tick color

minor_is_tick
[boolean, optional] Use tick minor line style (True) or full minor line style (False)

`add_line`(*image*, *area_def*, *lonlats*, *fill=None*, *outline='white'*, *x_offset=0*, *y_offset=0*)

Add a user defined poly-line from a list of (lon,lat) coordinates.

Parameters

image
[object] PIL image object

area_def
[list [proj4_string, area_extent]]

proj4_string : str
Projection of area as Proj.4 string
area_extent : list
Area extent as a list (LL_x, LL_y, UR_x, UR_y)

lonlats
[list of lon lat pairs] e.g. [(10,20),(20,30),...,(20,20)]

outline
[str or (R, G, B), optional] line color

width
[float, optional] line width

x_offset
[float, optional] Pixel offset in x direction

y_offset
[float, optional] Pixel offset in y direction

`add_polygon`(*image*, *area_def*, *lonlats*, *fill=None*, *outline='white'*, *x_offset=0*, *y_offset=0*)

Add a user defined polygon from a list of (lon,lat) coordinates.

Parameters

image
[object] PIL image object

area_def
[list [proj4_string, area_extent]]

proj4_string : str
Projection of area as Proj.4 string
area_extent : list
Area extent as a list (LL_x, LL_y, UR_x, UR_y)

lonlats
[list of lon lat pairs] e.g. [(10,20),(20,30),...,(20,20)]

fill
[str or (R, G, B), optional] Polygon fill color

outline
[str or (R, G, B), optional] line color

x_offset
[float, optional] Pixel offset in x direction

y_offset
[float, optional] Pixel offset in y direction

`add_rivers`(*image*, *area_def*, *resolution='c'*, *level=2*, *outline='white'*, *x_offset=0*, *y_offset=0*)

Add rivers to a PIL image object.

Parameters

image
[object] PIL image object

proj4_string
[str] Projection of area as Proj.4 string

area_extent
[list] Area extent as a list (LL_x, LL_y, UR_x, UR_y)

resolution
[str, optional {'c', 'l', 'i', 'h', 'f'}] Dataset resolution to use

level
[int, optional {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}] Detail level of dataset

outline
[str or (R, G, B), optional] River color

x_offset
[float, optional] Pixel offset in x direction

y_offset
[float, optional] Pixel offset in y direction

`add_rivers_to_file`(*filename*, *area_def*, *resolution='c'*, *level=1*, *outline='white'*, *x_offset=0*, *y_offset=0*)

Add rivers to an image file.

Parameters

image
[object] Image file

proj4_string
[str] Projection of area as Proj.4 string

area_extent
[list] Area extent as a list (LL_x, LL_y, UR_x, UR_y)

resolution
[str, optional {'c', 'l', 'i', 'h', 'f'}] Dataset resolution to use

level
[int, optional {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}] Detail level of dataset

outline
[str or (R, G, B), optional] River color

x_offset
[float, optional] Pixel offset in x direction

y_offset
[float, optional] Pixel offset in y direction

`add_shapefile_shape`(*image*, *area_def*, *filename*, *shape_id*, *feature_type=None*, *fill=None*, *outline='white'*, *x_offset=0*, *y_offset=0*)

Add a single shape file shape from an ESRI shapefile.

Note: To add all shapes in file use the 'add_shape_file_shapes' routine. Note: Currently only supports lon-lat formatted coordinates.

Parameters

image
[object] PIL image object

area_def
[list [proj4_string, area_extent]]

proj4_string : str
Projection of area as Proj.4 string
area_extent : list
Area extent as a list (LL_x, LL_y, UR_x, UR_y)

filename
[str] Path to ESRI shape file

shape_id
[int] integer id of shape in shape file {0; … }

feature_type
['polygon' or 'line',] only to override the shape type defined in shapefile, optional

fill
[str or (R, G, B), optional] Polygon fill color

outline
[str or (R, G, B), optional] line color

x_offset
[float, optional] Pixel offset in x direction

y_offset
[float, optional] Pixel offset in y direction

`add_shapefile_shapes`(*image*, *area_def*, *filename*, *feature_type=None*, *fill=None*, *outline='white'*, *x_offset=0*, *y_offset=0*)

Add shape file shapes from an ESRI shapefile.

Note: Currently only supports lon-lat formatted coordinates.

Parameters

image
[object] PIL image object

area_def
[list [proj4_string, area_extent]]

proj4_string : str
Projection of area as Proj.4 string
area_extent : list
Area extent as a list (LL_x, LL_y, UR_x, UR_y)

filename
[str] Path to ESRI shape file

feature_type
['polygon' or 'line',] only to override the shape type defined in shapefile, optional

fill
[str or (R, G, B), optional] Polygon fill color

fill_opacity
[int, optional {0; 255}] Opacity of polygon fill

outline
[str or (R, G, B), optional] line color

outline_opacity
[int, optional {0; 255}] Opacity of lines

x_offset
[float, optional] Pixel offset in x direction

y_offset
[float, optional] Pixel offset in y direction

10.1.3 Module contents

Pycoast package for adding geographic-based decorations to images.

`class pycoast.ContourWriter(*args, **kwargs)`

Bases: *ContourWriterPIL*

Writer wrapper for deprecation warning.

Deprecated since version 1.2.0: Use *ContourWriterPIL* or *ContourWriterAGG* instead.

PYTHON MODULE INDEX

p

INDEX

D

E

F

G

H

I

L

M

N

P

S

T

U