svgis

Release 0.5.1

Sep 25, 2021

Contents

Create SVG drawings from vector geodata files (SHP, geoJSON, etc).

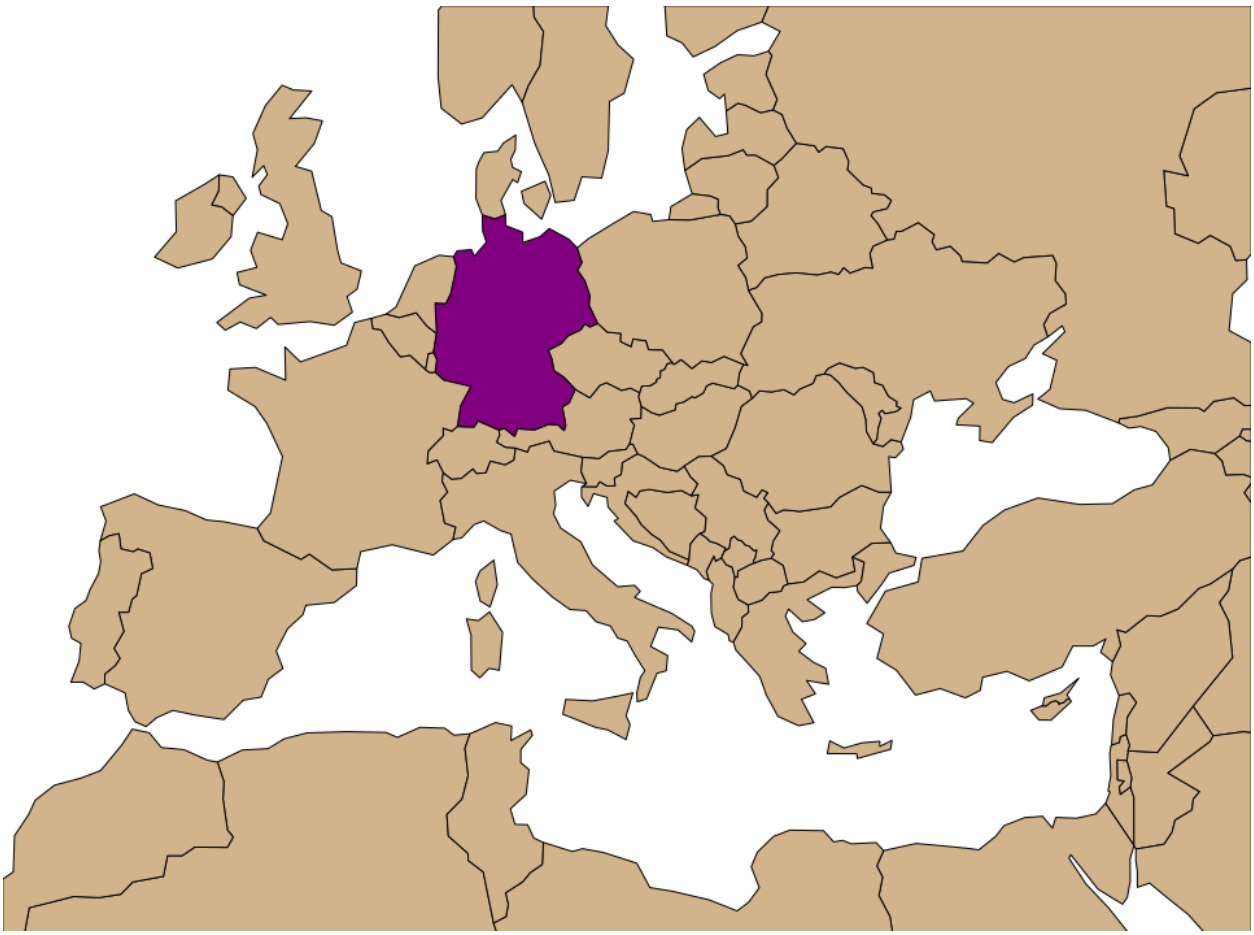
SVGIS is a command line tool for creating many small maps. It's designed to fit well with the Unix toolkit, and play nicely in a workflow, e.g. Makefile.

It also excels at basic maps that can be later elaborated in a drawing program (e.g. Illustrator).

With SVGIS, a command like this:

```
svgis draw ne_110m_admin_0_countries.shp \
--bounds -10 30 40 60 \
--project EPSG:3035 \
--scale 5000 \
--id-field name \
--style ".ne_110m_admin_0_countries {fill: tan;} #Germany { fill: purple }" \
```

Generates a map like this:



SVGIS is built on top of GDAL/OGR, so it can read just about any geodata file format you throw at it.

This documentation assumes some familiarity with CSS, how map projections work, assumes you have some geodata at your disposal. If you don't know where to find geodata, Natural Earth is a great place to start.

CHAPTER 1

Command-line usage

The main function of the SVGIS command line tool is to draw maps. SVGIS also comes with several helper tools for modifying maps and getting generating map projections.

1.1 svgis draw

Draw SVGs from input geodata.

```
Usage: svgis draw [OPTIONS] LAYER...

  Draw SVGs from input geodata

Options:
  -o, --output FILENAME           Defaults to stdout
  -b, --bounds minx miny maxx maxy
                                  In the same coordinate system as the first
                                  input layer
  -c, --style CSS                 CSS file or string
  -f, --scale INTEGER             Scale for the map (units are divided by this
                                  number)
  -p, --padding INTEGER           Buffer the map (in projection units)
  -i, --id-field FIELD            Geodata field to use as ID
  -a, --class-fields FIELDS       Geodata fields to use as class (comma-
                                  separated)
  -a, --data-fields FIELDS        Geodata fields to add as data-* attributes
                                  (comma-separated)
  -j, --crs KEYWORD               Specify a map projection. Accepts either an
                                  EPSG code (e.g. epsg:4456), a proj4 string,
                                  a file containing a proj4 string, "utm" (use
                                  local UTM), "file" (use existing), "local"
                                  (generate a local projection)
  -s, --simplify FACTOR           Simplify geometries, accepts an integer
                                  between 1 and 100, the percentage of each
```

(continues on next page)

(continued from previous page)

```
                                  geometry to retain.
  -P, --precision INTEGER         Rounding precision for coordinates (default:
                                  5)
  --clip / -n, --no-clip          Clip shapes to bounds. Slightly slower,
                                  produces smaller files (default: clip).
  -l, --inline / --no-inline      Inline CSS styles to each element. Slightly
                                  slower, but required by some clients (e.g.
                                  Adobe) (default: inline).
  --viewbox / -x, --no-viewbox    Draw SVG using a ViewBox (default: no
                                  ViewBox)
  -q, --quiet                     Ignore warnings
  -v, --verbose                   Talk a lot
  -h, --help                      Show this message and exit.
```

1.1.1 layers

Layers may be paths or an zip/gzip archive:

```
svgis draw data/cook.shp data/lake.shp
svgis draw zip://archive.zip/lorain.shp zip://archive.zip/cuyahoga.shp
svgis draw tar://archive.tar.gz/boston.geojson tar://archive.tar.gz/cambridge.geojson
```

1.1.2 bounds

Takes four arguments in min-lon, min-lat, max-lon, max-lat order.

This example draw the portion of the input file between latitudes 40 and 41 and longitudes -74 and -73 (roughly the New York City area).

```
svgis draw --bounds -74 40 -73 41 in.geojson out.svg
```

Note that coordinates are given in longitude, latitude order, since in the world of the computer, it's better to be consistent with things like (x, y) order.

1.1.3 scale

A integer scale. The map will be scaled by 1 over this number. In other words, for a map at a 1:1000 scale, use:

```
svgis draw --scale 1000 in.shp -o out.svg
```

This will produce a map where 1000 map units are scaled to one SVG unit. Clients will vary in how they represent a single unit (pixels, fractions of an inch). Additionally, clients may have trouble handling very large numbers, so using the scale option can be a handy way to produce usable drawings.

Scale won't alter geometries in any way other than scaling. To create smaller drawings, use the *simplify* option.

(The shorthand option for `--scale` is `-f` as in factor.)

1.1.4 crs

The `crs` argument accepts a particular projection or a keyword that helps SVGIS pick a projection for you. It accepts:

- EPSG code
- Proj4 string
- A text file containing a Proj4 string
- Either the 'utm' or 'local' keyword

If this flag isn't given, SVGIS will check to see if the file is already in non lat-lng projection (e.g. a state plane system or the British National Grid). If the first input file is projected, that projection will be used for the output. If the first file is in lat-long coordinates, a local projection will be generated, just like if `--crs=local` was given.

This example will draw an svg with EPSG:2908, the New York Long Island state plane projection:

```
svgis draw --crs EPSG:2908 nyc.shp -o nyc.svg
```

This example uses a Proj.4 string to draw with the North America Albers Equal Area Conic projection, which doesn't have an EPSG code.

```
svgis draw in.shp -o out.svg \
    --crs "+proj=aea +lat_1=20 +lat_2=60 +lat_0=40 \
    +lon_0=-96 +x_0=0 +y_0=0 +datum=NAD83 +units=m +no_defs"
```

This is equivalent to the above, and uses a proj.4 file:

```
echo "+proj=aea +lat_1=20 +lat_2=60 +lat_0=40 \
+lon_0=-96 +x_0=0 +y_0=0 +datum=NAD83 +units=m +no_defs" > proj4.txt

svgis draw in.shp --crs proj4.txt -o out.svg
```

With the `utm` keyword, SVGIS attempts to draw coordinates in the local UTM projection. The centerpoint of the bounding box will be used to pick the zone. Expect poor results for input data that crosses several UTM boundaries.

```
svgis draw --crs utm in.shp -o out.svg
svgis draw -j utm in2.shp -o out2.svg
```

When the local argument is given, SVGIS will generate a Transverse Mercator projection that centers on the input bounding box. This generally gives good results for an region roughly the size of a large urban area.

```
svgis draw --crs local input.shp -o out.svg
svgis draw -j local input.shp -o out.svg
```

If, for some reason you want to draw an SVG in lat-long coordinates, use the `file` keyword to force the projection of the first passed file:

```
svgis draw --crs file input.shp -o out.svg
svgis draw -j file one.shp two.geojson -o out.svg
```

To properly convert the input coordinate, svgis needs to know your input projection. If the input file doesn't specify an internal projection, SVGIS will assume that the coordinates are given in WGS84.

(The shorthand option for `--crs` is `-j` as in ject.)

1.1.5 style

The style parameter takes either a CSS file or a CSS string.

```
svgis draw --style style.css in.shp -o out.svg
svgis draw --style "line { stroke: green; }" in.shp -o out.svg
```

SVGIS adds a `polygon` class to paths that drawn to represent multi-part polygons (polygons with holes).

This argument can be provided multiple times.

(The shorthand option for `--style` is `-c` as in CSS.)

1.1.6 padding

Adds a padding around the output image. Accepts an integer in svg units.

```
svgis draw --padding 100 in.shp -o out.svg
```

1.1.7 no-viewbox

By default, SVGIS uses a viewbox. If you have a problem opening the created svg file in your drawing program (e.g. Adobe Illustrator), try the '–no-viewbox' option, which will create an svg where the contents are translated into the frame.

```
svgis draw --no-viewbox in.shp -o out.svg
svgis draw -x in.shp -o out.svg
```

1.1.8 class-fields and id-field

Use these options to specify fields in the source geodata file to use to determine the class or id attributes of the output SVG features. In the output fields, whitespace is be replaced with underscores. See *Styling maps* for details.

For example, the Natural Earth admin_0 file contains nation polygons, and includes `continent`, `income_grp` and `name` fields:

```
svgis draw --class-fields continent,income_grp --id-field name \
  ne_110m_admin_0_countries.shp -o out.svg
```

The result will include something like:

```
<g id="ne_110m_admin_0_countries">
    <g id="Afghanistan" class="continent_Asia income_grp_5_Low_income">/* Afghanistan␣
↪*/</g>
    <g id="Angola" class="continent_Africa income_grp_3_Upper_middle_income">/*␣
↪Angola */</g>
    /* ... */
    <g id="Zimbabwe" class="continent_Africa income_grp_5_Low_income">/* Zimbabwe */</
↪g>
</g>
```

Note that each layer is always wrapped in a group with `id` set to the its name, and `class` set to the names of the its fields.

The `id` (`ne_110m_admin_0_countries`) is repeated as a classes in each element of a layer. This makes writing CSS that addresses particular layers easier, given that some implementations of SVG don't properly implement css rules with ids (e.g. Adobe Illustrator, ImageMagick).

Note that the `income\_grp` field contains values like "5. Low income", which resultes in a class like `income_grp_5_Low_income`. Whitespace is replaced with underscores, periods and number signs (#) are removed. Missing values will be represented with the Pythonic "None".

```
.income_grp_5_Low_income {
    fill: teal;
    stroke: none;
}
.income_grp_None {
    fill: gray;
}
```

The `class-fields` argument can be provided multiple times.

1.1.9 data fields

Attributes with the `data-` prefix are useful for storing values in front-end Javascript. Convert specific fields in your geodata to attributes:

```
svgis draw --data-fields continent ne_110m_admin_0_countries.shp -o out.svg
```

The result will include something like:

```
<g id="ne_110m_admin_0_countries">
    <g id="Afghanistan" data-continent="Asia">/* Afghanistan */</g>
    <g id="Angola" data-continent="Africa">/* Angola */</g>
    /* ... */
    <g id="Zimbabwe" data-continent="Africa">/* Zimbabwe */</g>
</g>
```

1.1.10 simplify

Requires numpy. Install with `pip install svgis[simplify]` to make this available.

This option uses the Visvalingam algorithm to draw smaller shapes. The ideal setting will depend on source data and the requirements of the map.

The `--simplify` option takes a number between 1 and 100, which is the percentage of points to retain. Numbers above 80 usually produce output with few visible changes. Inputs below 20 often produce highly abstracted results.

```
svgis draw --simplify 75 in.shp -o out.svg
svgis draw -s 25 in.shp -o out.svg
```

1.1.11 precision

By default, the svg coordinates in drawings are rounded to five decimal places. The `precision` option allows for control over this rounding. Large values will create clunky and hard-to-use files. Small values (0 is the minimum) will yield smaller but possibly distorted files.

```
svgis draw --precision 10 in.geojson -o out.svg
svgis draw --precision 0 in.geojson -o out.svg
```

This will produce output like this (respectively):

```
<polyline points="2.7182818284,3.1415926535 1.0000000000,1.0000000000">
```

```
<polyline points="3,3 1,1">
```

1.1.12 no-inline

Run with this option to prevent svgis from adding style attributes onto each element. This will be quicker than the default, which requires parsing the CSS and examining each element. Some SVG clients (Adobe Illustrator) prefer inline styles.

Without `--no-inline` (or with `--inline`), SVG elements will look like:

```
<polyline style="stroke: green; ..." points="0,0 1,1">
```

```
svgis draw --style example.css --inline in.geojson -o out.svg
svgis draw -s example.css -l in.geojson -o out.svg
```

1.1.13 clip/no-clip

Install with `pip install svgis[clip]` to make this available. This requires additional libraries, see the shapely installation notes.

When installed with the clip option, SVGIS will try to clip output shapes to just outside of the bounding box. Use this option to disable that behavior.

:: svgis draw –bounds -170 40 -160 30 –no-clip in.shp –o out.shp svgis draw -b 125 30 150 50 -n in.shp –o out.shp

SVGIS always omits features that fall completely outside the bounding box, clipping edits the shapes so that the parts that lie outside of the bounding box are omitted.

Clipping won't occur when no bounding box is given.

1.2 Helpers

1.2.1 svgis bounds

Get the bounds of a layer. The `--crs` option will transform the bounds into the given projection, otherwise the native coordinates are returned.

The result is four coordinates in minx, miny, maxx, maxy order:

```
svgis bounds in.shp
-87.8098 41.6444 -87.5209 42.0201
```

This is useful for setting the bounds of a drawing based on the bounds of a layer:

```
svgis bounds in.shp |
xargs -n 4 svgis draw --crs EPSG:3003 in.shp in2.json in3.shp --bounds > out.svg
```

Or, check what projection SVGIS will generate given a file:

```
svgis bounds in.shp |
xargs -n 4 svgis project --
```

Keep in mind that when converting between projections, `svgis bounds` is lazy. The returned bounding box will cover the geometry, but may include extra space.

```
Usage: svgis bounds [OPTIONS] [LAYER]

  Return the bounds for a given layer.

Options:
  -j, --crs KEYWORD  Specify a map projection. Accepts either an EPSG code
                     (e.g. epsg:4456), a proj4 string, a file containing a
                     proj4 string, "utm" (use local UTM), "file" (use
                     existing), "local" (generate a local projection)
  --latlon           Print bounds in latitude, longitude order
  -h, --help         Show this message and exit.
```

1.2.2 svgis graticule

Generate a graticule (grid) in a given projection. The output file is in geojson format, with WGS84 coordinates.

Specifying a projection will produce a grid in that projection, and the step must reflect projection units. However, the output file will always be in WGS84 lon/lat coordinates, but that shouldn't make a difference in it's use.

For coordinates with negative numbers, use the `--` argument separator to prevent the utility getting confused:

:: svgis graticule -o graticule.json – 16.3 -34.8 32.8 -22.0

```
Usage: svgis graticule [OPTIONS] minx miny maxx maxy

  Generate a GeoJSON containing a graticule. Accepts a bounding box in
  longitude and latitude (WGS84).

Options:
  -s, --step FLOAT       Step between lines (in projected units)  [required]
  -j, --crs TEXT         Specify a map projection. Accepts either an EPSG code
                         (e.g. epsg:4456), a proj4 string, a file containing a
                         proj4 string, "utm" (use local UTM), "local"
                         (generate a local projection)
  -o, --output FILENAME  Defaults to stdout.
  -h, --help             Show this message and exit.
```

1.2.3 svgis scale

Scale all coordinates in an SVG by a given factor.

```
Usage: svgis scale [OPTIONS] [INPUT] [OUTPUT]

Options:
  -f, --scale INTEGER
  -h, --help           Show this message and exit.
```

1.2.4 svgis style

Add or replace the CSS style in an SVG.

```
Usage: svgis style [OPTIONS] [INPUT] [OUTPUT]

Options:
  -s, --style TEXT    Style to append to SVG. Either a valid CSS string, a
                      file path (must end in '.css'). Use '-' for stdin.
  -r, --replace TEXT
  -h, --help          Show this message and exit.
```

1.2.5 svgis project

SVGIS can automatically generate local projections or pick the local UTM projection for input geodata. This utility gives the projection SVGIS would pick for a given boundary box in PROJ4 syntax. The input can be either a bounding box or a single coordinate pair.

By default, `svgis project` expects WGS84 coordinates. Specify another projection with the `--crs` argument.

The output projection will be a local Transverse Mercator projection. Use `--method utm` to return the local UTM projection.

For coordinates with negative numbers, use the `--` argument separator to prevent the utility getting confused:

:: svgis project -m utm – 16.3449768409 -34.8191663551 32.830120477 -22.0913127581 +proj=utm +zone=35 +south +datum=WGS84 +units=m +no_defs

svgis project -m local – -110.277906 35.450777 +ellps=GRS80 +lat_0 +lat_1=35.450777 +lat_2=35.450777 +lon_0=-111.0 +no_defs +proj=lcc +towgs84 +y_0=0

```
Usage: svgis project [OPTIONS] minx miny maxx maxy | x y

Options:
  -m, --method [utm|local]  Defaults to local
  -j, --crs TEXT            Projection of the bounding coordinates
  -h, --help                Show this message and exit.
```

CHAPTER 2

Styling maps

You can use any kind of CSS that you like to style maps made with SVGIS. However, when using the `inline` option, SVGIS supports a subset of CSS.

This applies both to the selectors, which are limited by Python's built-in XML support, and to the declarations, which are limited by the styling rules for SVG.

A few useful things to know about how SVGIS draws maps:

- SVGIS places all the features in a layer in a group. This group has an `id` equal to the layer's name, and a `class` equal to the column names of the layer.
- The `class-fields` and `id-field` options can be used to add a `class` and `id` to the the drawing's elements. SVGIS always adds the layer name as a class to each feature.
- Polygons with holes are drawn as `path` elements with the class `polygon`.
- SVGIS can set the id and class of features based on the input data.
- By default, SVGIS draws black lines and no fill on shapes.

2.1 Style the features in a layer

This works because every element of layer `cb_2014_us_nation_20m.shp` will have the class `cb_2014_us_nation_20m`.

```
.cb_2014_us_nation_20m {
    fill: blue;
    stroke: green;
    stroke-width: 1px;
    stroke-dasharray: 5, 3, 2;
}
```

2.2 Style certain layers

Say that we're combining geodata from the US Census with data from Natural Earth. All Census layers have a GEOID field, and we use this to draw these layers with a `opacity: 0.50`.

```
/* example.css */
.GEOID * {
    opacity: 0.50;
}
.tl_2015_us_aiannh {
    fill: orange;
}
.ne_10m_time_zones {
    stroke-width: 2px;
}
```

Use this style to create a map projected in North America Equidistant Conic.

```
svgis draw --style example.css \
    --project EPSG:102010 \
    tl_2015_us_state.shp \
    tl_2015_us_aiannh.shp \
    ne_10m_time_zones.shp \
    -o out.svg
```

2.3 Style all polygons in a drawing

Polygons with holes are drawn as paths, and multipolygons are drawn in groups. To style all polygons, use the `.polygon` class:

```
polygon,
.polygon {
    fill: orange;
    stroke: black;
    stroke-opacity: 0.8;
}
```

2.4 Styling with data

The SVGIS `class-fields` and `id-field` options can be used to add `class` and `id` attributes to the output SVG. These, in turn, can be used to style the map based on the data.

Note that the SVGIS has to do minor clean up on the data. Whitespace is replaced with underscores, and periods, numbers signs and double-quote characters (`.#"`) are removed. Null values are represented with the Pythonic "None".

Additionally, CSS classes and IDs technically must begin with ascii letters, underscores or dashes. Classes and IDs that begin with other characters (after stripping illegal characters) are prefixed with an underscore (_).

2.4.1 Style a specific element

To style just Germany in the Natural Earth countries layer, use the `id-field` option to set the ID of all features to their `name_long`. This example also includes lakes. Because lakes don't have a `name_long` atribute, the individual

polygons won't have an ID field.

```
svgis draw --style purple.css \
    --id-field name_long \
    ne_110m_admin_0_countries.shp \
    ne_110m_lakes.shp \
    -o out.svg
```

```
/* purple.css */
#Germany {
    fill: purple;
}

#ne_110m_admin_0_countries polygon,
#ne_110m_admin_0_countries .polygon {
    fill: tan;
}

#ne_110m_lakes polygon,
#ne_110m_lakes .polygon {
    fill: blue;
}
```

2.4.2 Styling groups of elements

Use the `class-fields` option to add classes to data based on their data. In this example, the `income_grp` field in the admin-0 data set it used. This is ideal of SVGIS, since the data is already broken into bins. These bins have names like "5. Low Income", which SVGIS is sanitized to `5_Low_Income`.

```
/* style.css */
.income_grp_5_Low_income {
    fill: blue;
}
.income_grp_3_Upper_middle_income {
    fill: green;
}
```

```
svgis draw --style style.css \
--class-fields income_grp \
ne_110m_admin_0_countries.shp \
-o out.svg
```

CHAPTER 3

Classes and methods

Svgis is primarily built as a command line tool, but it has full API customization.

3.1 SVGIS

3.2 svg

3.3 draw

3.4 style

3.5 bounding

- genindex
- search