# svg-model Documentation

***Release 0.9.1.dev150602113***

**Christian Fobel**

**Jul 25, 2017**

# Contents

Contents:

CHAPTER 1

# Project Modules

## svg_model Package

### svg_model Package

svg_model.__init__.**compute_shape_centers**(*df_shapes*, *shape_i_column*, *inplace=False*)

Compute the center point of each polygon shape, and the offset of each vertex to the corresponding polygon center point.

**Parameters**

- **df_shapes** (*pandas.DataFrame*) – Table of polygon shape vertices (one row per vertex).

  **Must have at least the following columns:**

  - vertex_i: The index of the vertex within the corresponding shape.
  - x: The x-coordinate of the vertex.
  - y: The y-coordinate of the vertex.

- **shape_i_column** (*str or list, optional*) – Table rows with the same value in the shape_i_column column are grouped together as a shape.
- **in_place** (*bool, optional*) – If True, center coordinate columns are added directly to the input frame.

  Otherwise, center coordinate columns are added to copy of the input frame.

**Returns**

**Input frame with the following additional columns:**

- x_center/y_center: Absolute coordinates of shape center.
- **x_center_offset/y_center_offset:**
  - Coordinates of each vertex coordinate relative to shape center.

**Return type** pandas.DataFrame

svg_model.__init__.**fit_points_in_bounding_box**(*df_points*, *bounding_box*, *padding_fraction=0*)

Return data frame with `x`, `y` columns scaled to fit points from `df_points` to fill `bounding_box` while maintaining aspect ratio.

**Parameters**

- **df_points** (*pandas.DataFrame*) – A frame with at least the columns `x` and `y`, containing one row per point.
- **bounding_box** (*pandas.Series*) – A *pandas.Series* containing numeric *width* and *height* values.
- **padding_fraction** (*float*) – Fraction of padding to add around points.

**Returns** Input frame with the points with `x` and `y` values scaled to fill `bounding_box` while maintaining aspect ratio.

**Return type** pandas.DataFrame

svg_model.__init__.**fit_points_in_bounding_box_params**(*df_points*, *bounding_box*, *padding_fraction=0*)

Return offset and scale factor to scale `x`, `y` columns of `df_points` to fill `bounding_box` while maintaining aspect ratio.

**Parameters**

- **df_points** (*pandas.DataFrame*) – A frame with at least the columns `x` and `y`, containing one row per point.
- **bounding_box** (*pandas.Series*) – A *pandas.Series* containing numeric *width* and *height* values.
- **padding_fraction** (*float*) – Fraction of padding to add around points.

**Returns**

**(offset, scale)** – Offset translation and scale required to fit all points in `df_points` to fill `bounding_box` while maintaining aspect ratio.

`offset` contains `x` and `y` values for the offset.

**Return type** (pandas.Series, float)

svg_model.__init__.**get_scaled_svg_frame**(*svg_filepath*, ***kwargs*)

---

**Note:** Deprecated in `svg_model` 0.5 *`get_scaled_svg_frame()`* was removed in `svg_model` 0.5, it is replaced by `svg_model.scale_points()` and `svg_model.compute_shape_centers()`.

---

svg_model.__init__.**remove_layer**(*svg_source*, *layer_name*)

Remove layer(s) from SVG document.

**Parameters**

- **svg_source** (*str or file-like*) – A file path, URI, or file-like object.
- **layer_name** (*str or list*) – Layer name or list of layer names to remove from SVG document.

**Returns** File-like object containing XML document with layer(s) removed.

**Return type** StringIO.StringIO

svg_model.__init__.**scale_points**(*df_points*, *scale=3.5433070866141736*, *inplace=False*)

Translate points such that bounding box is anchored at (0, 0) and scale `x` and `y` columns of input frame by specified `scale`.

**Parameters**

- **df_points** (*pandas.DataFrame*) – Table of `x`/`y` point positions.

  **Must have at least the following columns:**

  - `x`: x-coordinate
  - `y`: y-coordinate

- **scale** (*float, optional*) – Factor to scale points by.

  By default, scale to millimeters based on Inkscape default of 90 pixels-per-inch.

- **scale** – Factor to scale points by.

- **in_place** (*bool, optional*) – If `True`, input frame will be modified.

  Otherwise, the scaled points are written to a new frame, leaving the input frame unmodified.

**Returns** Input frame with the points translated such that bounding box is anchored at (0, 0) and `x` and `y` values scaled by specified `scale`.

**Return type** pandas.DataFrame

svg_model.__init__.**scale_to_fit_a_in_b**(*a_shape*, *b_shape*)

Return scale factor (scalar float) to fit *a_shape* into *b_shape* while maintaining aspect ratio.

**Parameters** **b_shape** (*a_shape,*) – Input shapes containing numeric *width* and *height* values.

**Returns** Scale factor to fit `a_shape` into `b_shape` while maintaining aspect ratio.

**Return type** float

svg_model.__init__.**shape_path_points**(*svg_path_d*)

**Parameters** **svg_path_d** (*str*) – `"d"` attribute of SVG `path` element.

**Returns**

List of coordinates of points found in SVG path.

Each point is represented by a dictionary with keys `x` and `y`.

**Return type** list

svg_model.__init__.**svg_polygons_to_df**(*svg_source*, *xpath='//svg:polygon'*, *namespaces={'svg': 'http://www.w3.org/2000/svg', 'inkscape': 'http://www.inkscape.org/namespaces/inkscape'}*)

Construct a data frame with one row per vertex for all shapes (e.g., `svg:path`, `svg:polygon`) in `svg_source`.

**Parameters**

- **svg_source** (*str or file-like*) – A file path, URI, or file-like object.
- **xpath** (*str, optional*) – XPath path expression to select shape nodes.
- **namespaces** (*dict, optional*) – Key/value mapping of XML namespaces.

**Returns**

Frame with one row per vertex for all shapes in `svg_source`, with the following columns:

- `path_id`: The `id` attribute of the corresponding shape.
- `vertex_i`: The index of the vertex within the corresponding shape.
- `x`: The x-coordinate of the vertex.
- `y`: The y-coordinate of the vertex.

**Return type** pandas.DataFrame

---

**Note:** Deprecated in `svg_model` 0.5.post10 *`svg_polygons_to_df()`* will be removed in `svg_model` 1.0, it is replaced by *`svg_shapes_to_df()`* because the latter is more general and works with `svg:path` and `svg:polygon` elements.

---

`svg_model.__init__.`**`svg_shapes_to_df`**(*svg_source*, *xpath='//svg:path | //svg:polygon'*, *namespaces={'svg': 'http://www.w3.org/2000/svg', 'inkscape': 'http://www.inkscape.org/namespaces/inkscape'}*)

Construct a data frame with one row per vertex for all shapes in `svg_source`.

**Parameters**

- **`svg_source`** (*`str or file-like`*) – A file path, URI, or file-like object.
- **`xpath`** (*`str, optional`*) – XPath path expression to select shape nodes.

  By default, all `svg:path` and `svg:polygon` elements are selected.
- **`namespaces`** (*`dict, optional`*) – Key/value mapping of XML namespaces.

**Returns**

Frame with one row per vertex for all shapes in `svg_source`, with the following columns:

- `vertex_i`: The index of the vertex within the corresponding shape.
- `x`: The x-coordinate of the vertex.
- `y`: The y-coordinate of the vertex.
- **other: attributes of the SVG shape element (e.g., `id`, `fill`,** etc.)

**Return type** pandas.DataFrame

## `body_group` Module

**class** svg_model.body_group.**BodyGroup**(*paths=None*)

Bases: object

#### Methods

| |
|---|
| *add_path*(name, geo_path) |
| *get_body*(name) |
| *get_name*(body) |

**add_path**(*name*, *geo_path*)

**get_body**(*name*)

**get_name**(*body*)

## color Module

svg_model.color.**hex_color_to_rgba**(*hex_color*, *normalize_to=255*)

Convert a hex-formatted number (i.e., *"#RGB[A]"* or *"#RRGGBB[AA]"*) to an RGBA tuple (i.e., *(<r>, <g>, <b>, <a>)*).

**Parameters**

- **hex_color** (*str*) – hex-formatted number (e.g., *"#2fc"*, *"#3c2f8611"*)
- **normalize_to** (*int, float*) – Factor to normalize each channel by

**Returns**

**RGBA tuple (i.e., *(<r>, <g>, <b>, <a>)*), where range of** each channel in tuple is *[0, normalize_to]*.

**Return type** (tuple)

## connections Module

svg_model.connections.**draw_lines_svg_layer**(*df_endpoints*, *layer_name='Connections'*)

svg_model.connections.**extend_shapes**(*df_shapes*, *axis*, *distance*)

Extend shape/polygon outline away from polygon center point by absolute distance.

svg_model.connections.**extract_adjacent_shapes**(*df_shapes*, *shape_i_column*, *extend=0.5*)

Generate list of connections between "adjacent" polygon shapes based on geometrical "closeness".

**Parameters**

- **df_shapes** (*pandas.DataFrame*) – Table of polygon shape vertices (one row per vertex).

  Table rows with the same value in the `shape_i_column` column are grouped together as a polygon.

- **shape_i_column** (*str or list[str]*) – Column name(s) that identify the polygon each row belongs to.

- **extend** (*float, optional*) – Extend `x`/`y` coords by the specified number of absolute units from the center point of each polygon. Each polygon is stretched independently in the `x` and `y` direction. In each direction, a polygon considered adjacent to all polygons that are overlapped by the extended shape.

**Returns**

Adjacency list as a frame containing the columns `source` and `target`.

The `source` and `target` of each adjacency connection is ordered such that the `source` is less than the `target`.

**Return type** pandas.DataFrame

svg_model.connections.**extract_connections**(*svg_source*, *shapes_canvas*, *line_layer='Connections'*, *line_xpath=None*, *path_xpath=None*, *namespaces=None*)

Load all <svg:line> elements and <svg:path> elements from a layer of an SVG source. For each element, if endpoints overlap distinct shapes in `shapes_canvas`, add connection between overlapped shapes.

Changed in version 0.6.post1: Allow both <svg:line> *and* <svg:path> instances to denote connected/adjacent shapes.

New in version 0.6.post1: `path_xpath`

**Parameters**

- **svg_source** (*filepath*) – Input SVG file containing connection lines.
- **shapes_canvas** (shapes_canvas.ShapesCanvas) – Shapes canvas containing shapes to compare against connection endpoints.
- **line_layer** (*str*) – Name of layer in SVG containing connection lines.
- **line_xpath** (*str*) – XPath string to iterate through connection lines.
- **path_xpath** (*str*) – XPath string to iterate through connection paths.
- **namespaces** (*dict*) – SVG namespaces (compatible with `etree.parse()`).

**Returns** Each row corresponds to connection between two shapes in `shapes_canvas`, denoted `source` and `target`.

**Return type** pandas.DataFrame

svg_model.connections.**get_adjacency_matrix**(*df_connected*)

Return matrix where $a_{i,j} = 1$ indicates polygon $i$ is connected to polygon $j$.

Also, return mapping (and reverse mapping) from original keys in *df_connected* to zero-based integer index used for matrix rows and columns.

## data_frame Module

svg_model.data_frame.**close_paths**(*df_svg*)

svg_model.data_frame.**get_bounding_box**(*df_points*)
Calculate the bounding box of all points in a data frame.

svg_model.data_frame.**get_bounding_boxes**(*df_shapes*, *shape_i_columns*)
Return a *pandas.DataFrame* indexed by *shape_i_columns* (i.e., each row corresponds to a single shape/polygon), containing the following columns:

- *width*: The width of the widest part of the shape.
- *height*: The height of the tallest part of the shape.

svg_model.data_frame.**get_nearest_neighbours**(*path_centers*)

svg_model.data_frame.**get_shape_areas**(*df_shapes*, *shape_i_columns*, *signed=False*)
Return a *pandas.Series* indexed by *shape_i_columns* (i.e., each entry corresponds to a single shape/polygon), containing the following columns the area of each shape.

If *signed=True*, a positive area value corresponds to a clockwise loop, whereas a negative area value corresponds to a counter-clockwise loop.

svg_model.data_frame.**get_shape_infos**(*df_shapes*, *shape_i_columns*)
Return a *pandas.DataFrame* indexed by *shape_i_columns* (i.e., each row corresponds to a single shape/polygon), containing the following columns:

- *area*: The area of the shape.
- *width*: The width of the widest part of the shape.
- *height*: The height of the tallest part of the shape.

svg_model.data_frame.**get_svg_frame**(*svg_filepath*)

svg_model.data_frame.**get_svg_path_frame**(*svg_path*)

svg_model.data_frame.**triangulate_svg_frame**(*svg_frame*)

## detect_connections Module

svg_model.detect_connections.**auto_detect_adjacent_shapes**(*svg_source*, *shape_i_attr='id'*, *layer_name='Connections'*, *shapes_xpath='//svg:path | //svg:polygon'*, *extend=1.5*)
Attempt to automatically find "adjacent" shapes in a SVG layer.

In a layer within a new SVG document, draw each detected connection between the center points of the corresponding shapes.

**Parameters**

- **svg_source** (*str*) – Input SVG file as a filepath (or file-like object).
- **shape_i_attr** (*str, optional*) – Attribute of each shape SVG element that uniquely identifies the shape.
- **layer_name** (*str, optional*) – Name to use for the output layer where detected connections are drawn.

**Note:** Any existing layer with the same name will be overwritten.

- **shapes_xpath** (*str, optional*) – XPath path expression to select shape nodes.

  By default, all `svg:path` and `svg:polygon` elements are selected.

- **extend** (*float, optional*) – Extend `x`/`y` coords by the specified number of absolute units from the center point of each shape.

  Each shape is stretched independently in the `x` and `y` direction. In each direction, a shape is considered adjacent to all other shapes that are overlapped by the extended shape.

**Returns** File-like object containing SVG document with layer named according to `layer_name` with the detected connections drawn as `svg:line` instances.

**Return type** StringIO.StringIO

## `draw` Module

svg_model.draw.**draw_lines_svg_layer**(*df_endpoints*, *layer_name*, *layer_number=1*)

Draw lines defined by endpoint coordinates as a layer in a SVG file.

**Parameters**

- **df_endpoints** (*pandas.DataFrame*) – Each row corresponds to the endpoints of a single line, encoded through the columns: `x_source`, `y_source`, `x_target`, and `y_target`.
- **layer_name** (*str*) – Name of Inkscape layer.
- **layer_number** (*int, optional*) – Z-order index of Inkscape layer.

**Returns**

A file-like object containing SVG XML source.

The XML contains a layer named `"Connections"`, which in turn contains one line per row in the input `df_endpoints` table.

**Return type** StringIO.StringIO

svg_model.draw.**draw_shapes_svg_layer**(*df_shapes*, *shape_i_columns*, *layer_name*, *layer_number=1*, *use_svg_path=True*)

Draw shapes as a layer in a SVG file.

**Parameters**

- **df_shapes** (*pandas.DataFrame*) – Table of shape vertices (one row per vertex).
- **shape_i_columns** (*str or list*) – Either a single column name as a string or a list of column names in `df_shapes`. Rows in `df_shapes` with the same value in the `shape_i_columns` column(s) are grouped together as a shape.
- **layer_name** (*str*) – Name of Inkscape layer.
- **layer_number** (*int, optional*) – Z-order index of Inkscape layer.
- **use_svg_path** (*bool, optional*) – If `True`, electrodes are drawn as `svg:path` elements. Otherwise, electrodes are drawn as `svg:polygon` elements.

**Returns**

A file-like object containing SVG XML source.

The XML contains a layer named according to `layer_name`, which in turn contains `svg:polygon` or `svg:path` elements corresponding to the shapes in the input `df_shapes` table.

**Return type** StringIO.StringIO

## geo_path Module

This is a New BSD License. http://www.opensource.org/licenses/bsd-license.php

Copyright (c) 2008-2009, Jonathan Hartley (tartley@tartley.com) Copyright (c) 2012, Christian Fobel (christian@fobel.net) All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Jonathan Hartley nor the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

**class** svg_model.geo_path.**ColoredPath**(*loops*)
Bases: *svg_model.geo_path.Path*

### Methods

| |
|---|
| get_area() |
| get_bounding_box() |
| get_center() |
| get_centroid() |
| get_mass() |
| get_moment() |
| offset(x, y) |
| offset_to_origin() |

**class** svg_model.geo_path.**Path**(*loops*)
Bases: object

A Path is a list of loops.

### Methods

| |
|---|
| *get_area*() |
| *get_bounding_box*() |
| *get_center*() |
| *get_centroid*() |
| *get_mass*() |
| *get_moment*() |
| *offset*(x, y) |
| *offset_to_origin*() |

**get_area**()

**get_bounding_box**()

**get_center**()

**get_centroid**()

**get_mass**()

**get_moment**()

**offset**(*x*, *y*)

**offset_to_origin**()

## gui_demo Module

## loop Module

This is a New BSD License. http://www.opensource.org/licenses/bsd-license.php

Copyright (c) 2008-2009, Jonathan Hartley (tartley@tartley.com) Copyright (c) 2012, Christian Fobel (christian@fobel.net) All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Jonathan Hartley nor the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

**class** svg_model.loop.**Loop**(*verts=None*)
Bases: object

### Methods

| | |
|---|---|
| *get_area*() | Always returns a positive number. |
| *get_centroid*() | |
| *get_mass*() | |
| *get_signed_area*() | Return area of a simple (ie. |
| *is_clockwise*() | Assume y-axis points up |
| *offset*(x, y) | |

**density = 1**

**get_area**()
Always returns a positive number. If poly is self-intersecting, the actual area will be smaller than this.

**get_centroid**()

**get_mass**()

**get_signed_area**()
Return area of a simple (ie. non-self-intersecting) polygon. If verts wind anti-clockwise, this returns a negative number. Assume y-axis points up.

**is_clockwise**()
Assume y-axis points up

**offset**(*x*, *y*)

## merge Module

svg_model.merge.**get_svg_layers**(*svg_sources*)
Collect layers from input svg sources.

**Parameters** **svg_sources** (*list*) – A list of file-like objects, each containing one or more XML layers.

**Returns** **(width, height), layers** – The first item in the tuple is the shape of the largest layer, and the second item is a list of Element objects (from lxml.etree module), one per SVG layer.

**Return type** (int, int), list

svg_model.merge.**merge_svg_layers**(*svg_sources*, *share_transform=True*)
Merge layers from input svg sources into a single XML document.

**Parameters**

- **svg_sources** (*list*) – A list of file-like objects, each containing one or more XML layers.
- **share_transform** (*bool*) – If exactly one layer has a transform, apply it to *all* other layers as well.

**Returns** File-like object containing merge XML document.

**Return type** StringIO.StringIO

## path_group Module



**class** svg_model.path_group.**PathGroup**(*paths*, *boundary*)

Bases: object

### Methods

| |
|---|
| *get_bounding_box*() |
| *load_svg*(svg_path[, on_error]) |

**get_bounding_box**()

**classmethod load_svg**(*svg_path*, *on_error=None*)

## plot Module

## point_query Module

svg_model.point_query.**get_shapes_pymunk_space**(*df_convex_shapes*, *shape_i_columns*)

Return two-ple containing:

- A *pymunk.Space* instance.
- A *pandas.Series* mapping each *pymunk.Body* object in the *Space* to a shape index.

The *Body* to shape index mapping makes it possible to, for example, look up the index of the convex shape associated with a *Body* returned by a *pymunk* point query in the *Space*.

## seidel Module

**class** svg_model.seidel.**Edge**(*p*, *q*)
Bases: object

### Methods

| |
|---|
| *add_mpoint*(point) |
| *is_above*(point) |
| *is_below*(point) |

**add_mpoint**(*point*)

**is_above**(*point*)

**is_below**(*point*)

**class** svg_model.seidel.**MonotoneMountain**

### Methods

| |
|---|
| *add*(point) |
| *angle*(p) |
| *angle_sign*() |
| *gen_mono_poly*() |
| *is_convex*(p) |
| *process*() |
| *remove*(point) |
| *triangulate*() |
| *valid*(p) |

**add**(*point*)

**angle**(*p*)

**angle_sign**()

**gen_mono_poly**()

**is_convex**(*p*)

**process**()

**remove**(*point*)

**triangulate**()

**valid**(*p*)

**class** svg_model.seidel.**Node**(*lchild*, *rchild*)
Bases: object

### Methods

| |
|---|
| *replace*(node) |

**replace**(*node*)

**class** svg_model.seidel.**Point**(*x*, *y*)

Bases: object

### Methods

| |
|---|
| *clone*() |
| *cross*(p) |
| *dot*(p) |
| *length*() |
| *less*(p) |
| *neq*(other) |
| *normalize*() |

**clone**()

**cross**(*p*)

**dot**(*p*)

**length**()

**less**(*p*)

**neq**(*other*)

**normalize**()

**class** svg_model.seidel.**QueryGraph**(*head*)

### Methods

| |
|---|
| *case1*(sink, edge, tlist) |
| *case2*(sink, edge, tlist) |
| *case3*(sink, edge, tlist) |
| *case4*(sink, edge, tlist) |
| *follow_edge*(edge) |
| *locate*(edge) |
| *replace*(sink, node) |

**case1**(*sink*, *edge*, *tlist*)

**case2**(*sink*, *edge*, *tlist*)

**case3**(*sink*, *edge*, *tlist*)

**case4**(*sink*, *edge*, *tlist*)

**follow_edge**(*edge*)

**locate**(*edge*)

**replace**(*sink*, *node*)

**class** svg_model.seidel.**Sink**(*trapezoid*)
Bases: *svg_model.seidel.Node*

### Methods

| |
|---|
| *locate*(edge) |
| replace(node) |

**locate**(*edge*)

**class** svg_model.seidel.**Trapezoid**(*left_point*, *right_point*, *top*, *bottom*)
Bases: object

### Methods

| |
|---|
| *add_points*() |
| *area*() |
| *contains*(point) |
| *segments*(p) |
| *trim_neighbors*() |
| *update_left*(ul, ll) |
| *update_left_right*(ul, ll, ur, lr) |
| *update_right*(ur, lr) |
| *vertices*() |

**add_points**()

**area**()

**contains**(*point*)

**segments**(*p*)

**trim_neighbors**()

**update_left**(*ul*, *ll*)

**update_left_right**(*ul*, *ll*, *ur*, *lr*)

**update_right**(*ur*, *lr*)

**vertices**()

**class** svg_model.seidel.**TrapezoidalMap**
Bases: object

### Methods

| |
|---|
| *bounding_box*(edges) |
| *case1*(t, e) |

Continued on next page

Table 1.13 – continued from previous page

| |
|---|
| *case2*(t, e) |
| *case3*(t, e) |
| *case4*(t, e) |
| *clear*() |

**bounding_box**(*edges*)

**case1**(*t*, *e*)

**case2**(*t*, *e*)

**case3**(*t*, *e*)

**case4**(*t*, *e*)

**clear**()

**class** svg_model.seidel.**Triangulator**(*poly_line*)

Bases: object

### Methods

| |
|---|
| *create_mountains*() |
| *init_edges*(points) |
| *mark_outside*(t) |
| *mono_polies*() |
| *order_edges*(edge_list) |
| *process*() |
| *trapezoid_map*() |
| *triangles*() |

**create_mountains**()

**init_edges**(*points*)

**mark_outside**(*t*)

**mono_polies**()

**order_edges**(*edge_list*)

**process**()

**trapezoid_map**()

**triangles**()

**class** svg_model.seidel.**XNode**(*point*, *lchild*, *rchild*)

Bases: *svg_model.seidel.Node*

### Methods

| |
|---|
| *locate*(edge) |
| replace(node) |

**locate**(*edge*)

**class** svg_model.seidel.**YNode**(*edge*, *lchild*, *rchild*)
Bases: *svg_model.seidel.Node*

#### Methods

| | |
|---|---|
| *locate*(edge) | |
| replace(node) | |

**locate**(*edge*)

svg_model.seidel.**isink**(*trapezoid*)

svg_model.seidel.**line_intersect**(*edge*, *x*)

svg_model.seidel.**merge_sort**(*l*)

svg_model.seidel.**orient2d**(*pa*, *pb*, *pc*)

svg_model.seidel.**shear_transform**(*point*)

## shapes_canvas Module

**class** svg_model.shapes_canvas.**ShapesCanvas**(*df_shapes*, *shape_i_columns*, *canvas_shape=None*, *padding_fraction=0*)
Bases: object

The *ShapesCanvas* class fits all shapes defined by vertices in a *pandas.DataFrame* (one vertex per row) into a specified canvas shape (with optional padding), while maintaining aspect ratio.

The *ShapesCanvas.find_shape* method returns the shape located at the specified *canvas* coordinates (or *None*, if no shape intersects with specified point).

#### Methods

| | |
|---|---|
| *find_shape*(canvas_x, canvas_y) | Look up shape based on canvas coordinates. |
| *from_svg*(svg_filepath, *args, **kwargs) | |
| *reset_shape*([canvas_shape, padding_fraction]) | |

**find_shape**(*canvas_x*, *canvas_y*)
Look up shape based on canvas coordinates.

**classmethod from_svg**(*svg_filepath*, **args*, ***kwargs*)

**reset_shape**(*canvas_shape=None*, *padding_fraction=None*)

svg_model.shapes_canvas.**get_transform**(*offset*, *scale*)

## tesselate Module

svg_model.tesselate.**tesselate_shapes_frame**(*df_shapes*, *shape_i_columns*)
Tesselate each shape path into one or more triangles.

**Parameters**

- **df_shapes** (*pandas.DataFrame*) – Table containing vertices of shapes, one row per vertex, with the *at least* the following columns:
  - `x`: The x-coordinate of the vertex.
  - `y`: The y-coordinate of the vertex.
- **shape_i_columns** (*str or list*) – Column(s) forming key to differentiate rows/vertices for each distinct shape.

**Returns**

- *pandas.DataFrame*
- *Table where each row corresponds to a triangle vertex, with the following*
- *columns –*
  - `shape_i_columns[]`: The shape path index column(s).
  - `triangle_i`: The integer triangle index within each electrode path.
  - `vertex_i`: The integer vertex index within each triangle.

## Subpackages

### bin Package

#### detect_connections Module

svg_model.bin.detect_connections.**parse_args**(*args=None*)
Parses arguments, returns (options, args).

### svgload Package

#### path_parser Module

SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

**class** svg_model.svgload.path_parser.**LoopTracer**
Bases: object

### Methods

| | |
|---|---|
| *get_point*(command) | |
| *onBadCommand*(action) | |
| *onClose*(command) | |
| *onHorizontalMove*(command) | |
| *onLine*(command) | |
| *onMove*(command) | |
| *onVerticalMove*(command) | |
| *to_loops*(commands) | commands : list of tuples, as output from to_tuples() method, eg: |

**get_point**(*command*)

**onBadCommand**(*action*)

**onClose**(*command*)

**onHorizontalMove**(*command*)

**onLine**(*command*)

**onMove**(*command*)

**onVerticalMove**(*command*)

**to_loops**(*commands*)

**commands** [list of tuples, as output from to_tuples() method, eg:] [('M', 1, 2), ('L', 3, 4), ('L', 5, 6), ('z')]

Interprets the command characters at the start of each tuple to return a list of loops, where each loop is a closed list of verts, and each vert is a pair of ints or floats, eg:

[[1, 2, 3, 4, 5, 6]]

Note that the final point of each loop is eliminated if it is equal to the first. SVG defines commands:

M x,y: move, start a new loop L x,y: line, draw boundary H x: move horizontal V y: move vertical Z: close current loop - join to start point

Lower-case command letters (eg 'm') indicate a relative offset. See http://www.w3.org/TR/SVG11/paths.html

**exception** svg_model.svgload.path_parser.**ParseError**
Bases: exceptions.Exception

**class** svg_model.svgload.path_parser.**PathDataParser**
Bases: object

### Methods

| | |
|---|---|
| *get_char*(allowed) | |
| *get_chars*(allowed) | |
| *get_number*() | |
| *to_tuples*(data) | path_data : string, from an svg path tag's 'd' attribute, eg: |

**get_char**(*allowed*)

**get_chars**(*allowed*)

**get_number**()

**to_tuples**(*data*)

**path_data** [string, from an svg path tag's 'd' attribute, eg:] 'M 46,74 L 35,12 l 53,-13 z'

**returns the same data collected in a list of tuples, eg:** [ ('M', 46, 74), ('L', 35, 12), ('l', 53, -13), ('z') ],

The input data may have floats instead of ints, this will be reflected in the output. The input may have its whitespace stripped out, or its commas replaced by whitespace.

**class** svg_model.svgload.path_parser.**PathParser**
Bases: object

### Methods

| | |
|---|---|
| *get_id*(attributes) | |
| *parse*(tag) | returns (id, path) |
| *parse_color*(color) | color : string, eg: '#rrggbb' or 'none' |
| *parse_style*(style) | style : string, eg: |

**get_id**(*attributes*)

**next_id = 1**

**parse**(*tag*)
returns (id, path) where: 'id' is the path tag's id attribute

'path' is a populated instance of SvgPath

```
>>> from lxml import etree
>>> from lxml.builder import E
>>> path_tag = etree.XML("""
...     <path id="path0"
...         style="fill:#0000ff;stroke:#000000;stroke-width:0.
↪10000000000000001;stroke-miterlimit:4;stroke-dasharray:none"
...         d="M 525.93385,261.47322 L 525.933 85,269.65826 L 534.07239,269.
↪65826 L 534.07239,261.47322 L 525.93385,261.47322" />
... """)
```

```
>>> path_parser = PathParser()
>>> id, svg_path = path_parser.parse(path_tag)
>>> id
'path0'
>>> svg_path.color
(0, 0, 255)
>>> len(svg_path.loops)
1
>>> svg_path.loops[0].verts
[(534.07239, 261.47322), (534.07239, 269.65826), (525.933, 85), (525.93385,␣
↪261.47322)]
```

Note that only absolute commands (i.e., uppercase) are currently supported. For example: paths will throw a ParseError exception. For example:

```
>>> path_tag = E.path(id="path0", d="M 636.0331,256.9345 l 636.0331,256.9345
↪")
>>> print etree.tostring(path_tag)
<path d="M 636.0331,256.9345 l 636.0331,256.9345" id="path0"/>
>>> path_parser.parse(path_tag)
Traceback (most recent call last):
...
ParseError: unsupported svg path command: l
>>>
```

**parse_color**(*color*)

color : string, eg: '#rrggbb' or 'none' (where rr, gg, bb are hex digits from 00 to ff) returns a triple of unsigned bytes, eg: (0, 128, 255)

**parse_style**(*style*)

**style** [string, eg:] fill:#ff2a2a;fill-rule:evenodd;stroke:none;stroke-width:1px; stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1

returns color as a triple of unsigned bytes: (r, g, b), or None

### svg_parser Module

SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

**class** svg_model.svgload.svg_parser.**Svg**
Bases: object

Maintains an ordered list of paths, each one corresponding to a path tag from an SVG file. Creates a pylget Batch containing all these paths, for rendering as a single OpenGL GL_TRIANGLES indexed vert primitive.

### Methods

| | |
|---|---|
| *add_path*(id, path) | |
| *add_to_batch*(batch) | Adds paths to the given batch object. |
| *all_verts*() | |
| *get_boundary*() | |
| *get_bounding_box*() | |

**add_path**(*id*, *path*)

**add_to_batch**(*batch*)
Adds paths to the given batch object. They are all added as GL_TRIANGLES, so the batch will aggregate them all into a single OpenGL primitive.

**all_verts**()

**get_boundary**()

**get_bounding_box**()

**exception** svg_model.svgload.svg_parser.**SvgParseError**
Bases: exceptions.Exception

**class** svg_model.svgload.svg_parser.**SvgParser**
Bases: object

parse(filename) returns an Svg object, populated from the <path> tags in the file.

### Methods

| | |
|---|---|
| *parse*(xml_root[, on_error]) | Parse all <path> elements from xml_root. |
| *parse_file*(filename[, on_error]) | |

**parse**(*xml_root*, *on_error=None*)
Parse all <path> elements from xml_root.

Optional on_error arg specifies a callback function to be run when an error occurs during parsing. The specified on_error function must accept 3 arguments:

<svg filename>, <path_tag>, <error message>

An example on_error handler is provided as svg_load.svg_parser.parse_warning(), where all SvgParseErrors are converted to warning messages. See usage below: >>> import re >>> svg_parser = SvgParser() >>> path_tag = etree.XML(“”” ... <path ... xmlns=”http://www.w3.org/2000/svg” ... xmlns:dc=”http://purl.org/dc/elements/1.1/” ... xmlns:cc=”http://creativecommons.org/ns#” ... xmlns:rdf=”http://www.w3.org/1999/02/22-rdf-syntax-ns#” ... xmlns:svg=”http://www.w3.org/2000/svg” ... xmlns:sodipodi=”http://sodipodi.sourceforge.net/DTD/sodipodi-0.dtd” ... xmlns:inkscape=”http://www.inkscape.org/namespaces/inkscape” ... id=”path13231” ... d=”M8 4 l-4,4” ... linecap=”square” ... stroke=”#000000” ... stroke-width=”0.25” ... />”“”) >>> with warnings.catch_warnings(record=True) as w: ... svg = svg_parser.parse(path_tag, on_error=parse_warning) >>> print w[-1].category <type ‘exceptions.RuntimeWarning’> >>> match = re.search(r’^Error parsing None:d+, unsupported svg path command: l’, str(w[-1].message)) >>> print match is None False >>> path_tag = etree.XML(“”” ... <path ... xmlns=”http://www.w3.org/2000/svg” xmlns:inkscape=”http://www.inkscape.org/namespaces/inkscape” ... xmlns:dc=”http://purl.org/dc/elements/1.1/” xmlns:cc=”http://creativecommons.org/ns#” ... xmlns:rdf=”http://www.w3.org/1999/02/22-rdf-syntax-ns#” ... xmlns:svg=”http://www.w3.org/2000/svg” ... xmlns:sodipodi=”http://sodipodi.sourceforge.net/DTD/sodipodi-0.dtd” ... style=”fill:#0000ff;stroke:#ff0000;stroke-width:0.10000000000000001;stroke-miterlimit:4;stroke-dasharray:none” ... id=”path18327” ... d=”M 636.0331,256.9345 L 636.0331,256.9345” ... inkscape:connector-curvature=”0”/>”“”) >>> with warnings.catch_warnings(record=True) as w: ... svg = svg_parser.parse(path_tag, on_error=parse_warning) >>> print w[-1].category <type ‘exceptions.RuntimeWarning’> >>> match = re.search(r’^Error parsing None:d+, loop needs 3 or more verts’, str(w[-1].message)) >>> print match is None False

**parse_file**(*filename*, *on_error=None*)

svg_model.svgload.svg_parser.**parse_warning**(**args*)

CHAPTER 2

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## S

# Index

## H



## I



## L



## M



## N



## O

## X



## Y