3D Visualization toolbox for easy display of complex data from Matlab or Python

Emmanuel Cledat

May 31, 2019

Available at: `https://github.com/ManuCledat/Matlab_Python_3D_toolbox`

This 3D Visualization toolbox permits to create visualization for debugging purposes, data analysis and results rendering. It permits to export usual 3D object used in research in a CAD software. This eases the navigation in the 3D model, its modification and its rendering (e.g. background, shadows, fog could be easily modified). Finally, it permits to visualize, by a simple click the meta-data of the objects, inputted by the user (i.e. object name or number) and calculated by the CAD software, as traditional GIS functionalities.

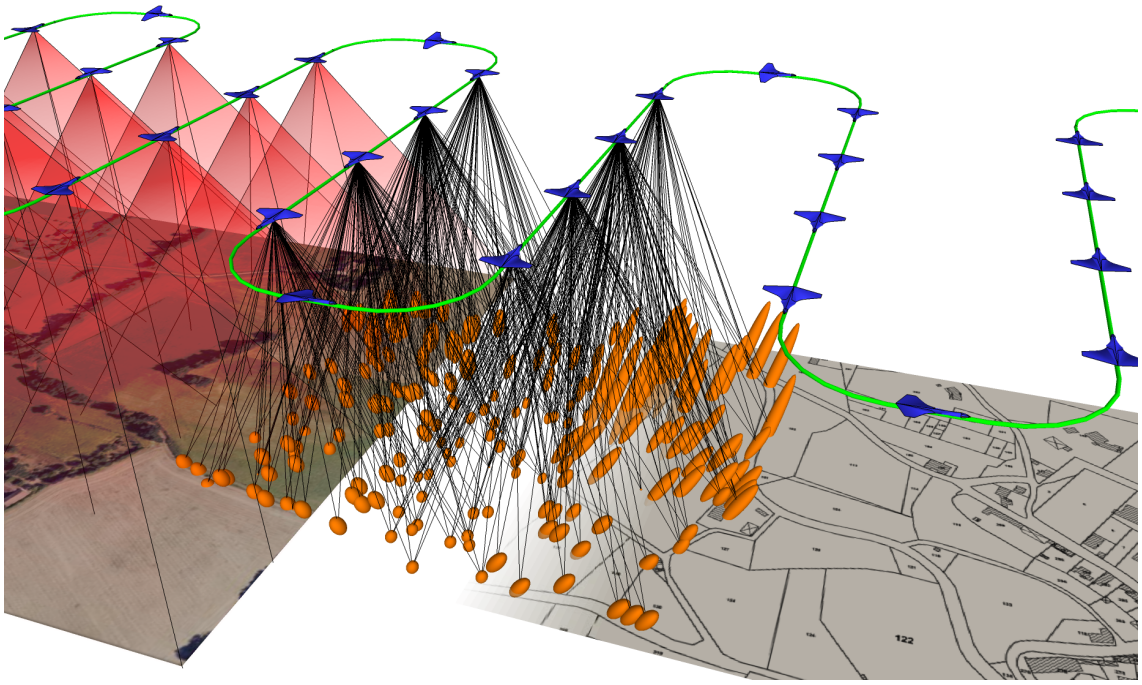


Figure 1: Example of figure created with the toolbox

Introduction

This toolbox is implemented both in Matlab and in Python. It permits to export 3D data (such as results of research data) in the free software sketchUp [1]. This CAD software where chosen for its user-friendliness and its diversity of use. This software could be used for various applications, such as mechanical engineering, architecture, spatial design, and environmental modeling. Moreover, it is usually chosen for educational purposes as a first CAD software to learn (here, a course designed for 14-15 years-old students [2]). Previous versions of our toolbox (not released) were used for geodetic engineering publications: [3, 4, 5, 6, 7, 8, 9].

1 User Manual for MATLAB

1.1 ruby_create

1.1.1 Description:

Creates and initializes a ruby file and returns a struct which contains id, path and name of the file. Default file is 'script_ruby_3d.rb'in the local folder. File must be closed by `ruby_close`.

1.1.2 Function Declaration:

```
file = ruby_create()
file = ruby_create(nameOrPath)
file = ruby_create(nameOrPath, name)
```

- `nameOrPath` (optional): 'script_ruby_3d.rb'(default) — string

  It is either the name of the file or the path of the file.

- `name` (optional): string

  Name of the file

1.1.3 Examples:

```
% Create file in local folder with name 'script_ruby_3d.rb'.
file = ruby_create();

% Specify name, .rb appended, 'my_model.rb'.
file = ruby_create('my_model');

% Alternative specifying full path. .rb is appended, 'C:\Users\my_model.rb'.
file = ruby_create('C:\Users', 'my_model');
```

1.2 ruby_close

1.2.1 Description:

Completes and closes the ruby file.

1.2.2 Function Declaration:

```
ruby_close(file)
```

- `file`: file struct

  File structure created with `ruby_create`

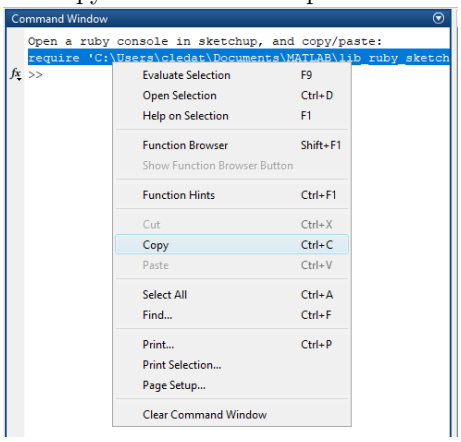
1.2.3 Examples:

```
ruby_close(file);
```

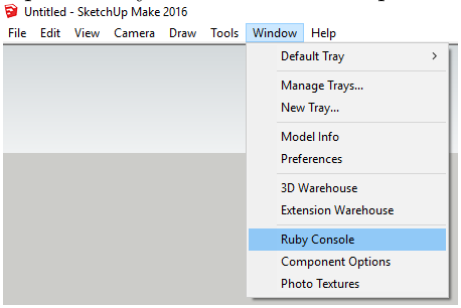
1.3 Input in SketchUp

If the file is closed correctly, it could be inputted in SketchUp as follow.

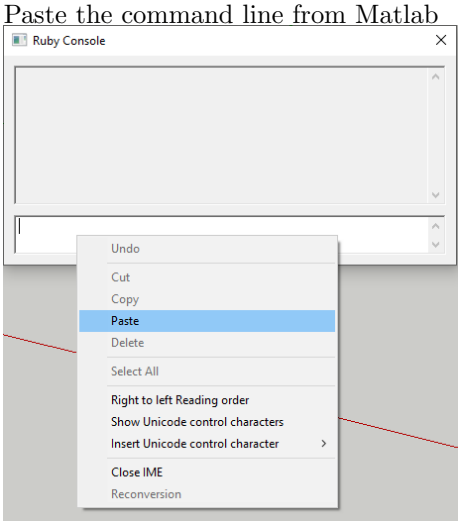
Copy the commanded printed on Matlab console.



Open a ruby console in SketchUp



Paste the command line from Matlab



1.4 ruby_point

1.4.1 Description:

Creates isolated points.

This function also permits to display point-clouds.

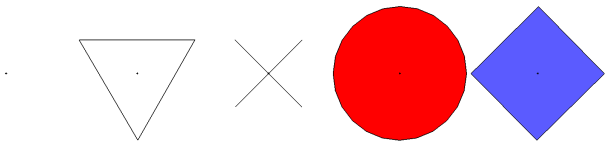


Figure 2: Different type of points

Figure 3: Display of a point-cloud of a church

1.4.2 Function Declaration:

```
ruby_point(file, XYZ, issymbolic, 'symbol', symbol, 'color' , color, ...
        'name', name)
```

- `file`: file struct

  File structure created with `ruby_create`.

- `XYZ`: (Nx3) matrix

  Matrix where N is the number of points. Each row corresponds to a point and the columns correspond to three coordinate axis.

- `issymbolic` (optional): 0 (default) — 1 — column vector containing 0 and 1 values.

  If it is a numeric value, all points are treated equally in terms of symbols. If it is 0, there is no symbol. If it is 1, a symbol is used for each and every point.

  If it is a column vector, the number of rows should be the same as the number of rows in XYZ matrix.

- `symbol` (optional): 'triangle'(default) — 'cross'— 'circle'— 'square'

  It defines how points are represented in 3D plane by assigning a particular symbol.

- `color` (optional): char — [R, G, B]

  Either one of the following colors or a set of RGB values.

  'n'(default): none

  'w': white

  'r': red

  'o': orange

  'y': yellow

  'g': green

  'b': blue

  'p': pink

  'k': black

- name (optional): string — column vector containing strings.

  If it is a column vector, the number of rows should be the same as the number of rows in XYZ matrix.

1.4.3 Examples:

```
% Construct some points (Here [2,2,2] and [3,3,3])
ruby_point(file, [2, 2, 2; 3, 2, 3]);

% To represent all points with a symbol, set the third argument to 1
ruby_point(file, [2, 3, 2; 3, 3, 3], 1);

% Symbol representation can be activated point by point
ruby_point(file, [2, 4, 2; 3, 4, 3], [1; 0]);

% The symbol is selected with a name-value pair. All points have the same
% symbol
ruby_point(file, [2, 5, 2; 3, 5, 3], 1, 'symbol', 'cross');

% With red cicrle
ruby_point(file, [2, 6, 2; 3, 6, 3], 1, 'symbol', 'circle', 'color', 'r');

% With blue square
ruby_point(file, [2, 7, 2; 3, 7, 3], 1, 'symbol', 'square',...
        'color', [0, 0, 255]);

% Points can be named globally or individually using a single string or an
% array of strings.
ruby_point(file, [2, 2, 2; 3, 3, 3], 1, 'name', string('Pts'));
ruby_point(file, [2, 2, 2; 3, 3, 3], 1, 'name',...
        [string('pt1'); string('pt2')]);
```

1.5 ruby_line

1.5.1 Description:

Draws a line or multiple lines.

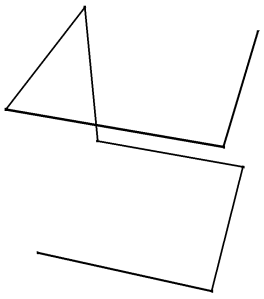


Figure 4: Example of line drawn by the library

1.5.2 Function Declaration:

```
ruby_line(file, XYZ, 'name', name);
```

- `file`: file struct

  File structure created with `ruby_create`.

- `XYZ`: (Nx3) matrix

  Matrix where N is the number of points and n should be at least 2. Each row corresponds to a point and the columns correspond to three coordinate axis. The points are connected by a line one by one according to the given order. In other words, first point is connected with the second point, second point is connected with third point and so on.

- `name` (optional): string — column vector containing strings

  If it is a column vector, the number of rows should be the same as the number of rows in XYZ matrix.

1.5.3 Examples:

```
% Draw stylized cube with lines.
ruby_line(file, [0 0 0; 1 0 0; 1 1 0; 0 1 0; 0 1 1; 0 0 1; 1 0 1; 1 1 1]);

% Give the drawn line a name.
ruby_line(file, [0, 0, 5; 5, 0, 5], 'name', string('Line1'));
```

1.6 ruby_axis

1.6.1 Description:

Creates a reference coordinate axis defined by the orientation matrix and center point.

1.6.2 Function Declaration:

```
ruby_axis(file, origin, R, 'name', name)
```

- `file`: file struct

  File structure created with `ruby_create`.

- `origin`: (1x3) matrix

  It contains x, y, z coordinates of the new origin

- `R`: (3x3) matrix

  Orthogonal orientation matrix.

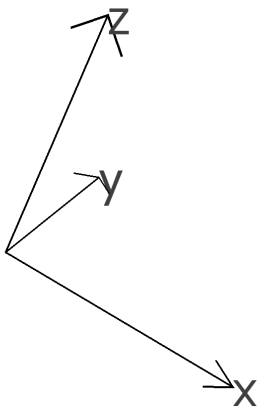


Figure 5: Axis definition

- `name` (optional): string

Examples:

```
R = [1, 0, 0; 0, cos(1), sin(1); 0, -sin(1), cos(1)];

% Create coordinate system at 4,4,4.
ruby_axis(file, [4, 4, 4], R);

% Create coordinate system at 4,5,4 named Origin2
ruby_axis(file, [4, 5, 4], R, 'name', string('Origin2'));
```

1.7 ruby_pose

1.7.1 Description:

Constructs a pose. The photo is taken in the -Z direction.

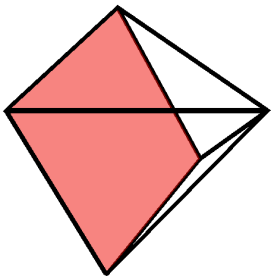


Figure 6: Camera pose

1.7.2 Function Declaration:

```
ruby_pose(file, P , R, 'focal', focal, 'width', width, ...
        'height', height, 'color', color, 'name', name)
```

- `file`: file struct

  File structure created with `ruby_create`.

- `P`: (1x3) matrix
  It contains x, y, z coordinates of the center of the projection of the pose.
- `R`: (3x3) matrix
  It is an orthogonal matrix.
- `focal` (optional): 0.2 (default) — Numeric value
  It is in meters.
- `width` (optional): 0.1 (default) — Numeric value
  It is in meters.
- `height` (optional): 0.1 (default) — Numeric value
  It is in meters.
- `color` (optional): char — [R, G, B]
  Either one of the following colors or a set of RGB values.
  'n'(default): none
  'w': white
  'r': red
  'o': orange
  'y': yellow
  'g': green
  'b': blue
  'p': pink
  'k': black
- `name` (optional): string.

1.7.3 Examples:

```
R = [1, 0, 0; 0, cos(1), sin(1); 0, -sin(1), cos(1)];

% Basic default pose.
ruby_pose(file, [5,5,5], R);

% Set custom focal length and image size.
ruby_pose(file, [6,5,5], R, 'focal', 0.1, 'width', 0.3, 'height', 0.2);

% Labeled.
ruby_pose(file, [7,5,5], R, 'focal', 0.1, 'width', 0.3, ...
        'height', 0.2, 'name', string('f0.1w0.3h0.2'));

% Set the image plane from undefined to blue.
ruby_pose(file, [8,5,5], R, 'focal', 0.1, 'width', 0.3, ...
        'height', 0.2, 'color', 'b');

% Set the image plane from undefined to red with hint of blue.
ruby_pose(file, [9,5,5], R, 'focal', 0.1, 'width', 0.3, ...
        'height', 0.2, 'color', [255, 0, 63]);
```

1.8 ruby_ellipsoid

1.8.1 Description:

Creates an ellipsoid.

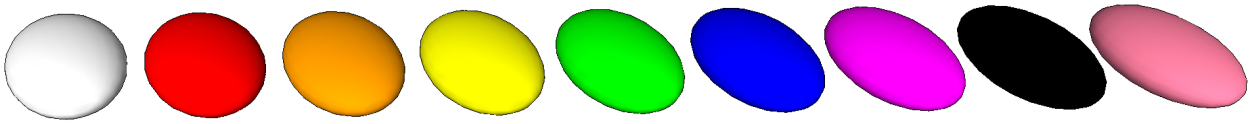


Figure 7: Ellipsoids

An ellipsoid is a surface defined by a center point P and a covariance matrix (also called tensor) K.

$$v^T \; K \; v = 1 \tag{1}$$

v is the vector between a point of the 3D space and the center of the ellipsoid: P.

1.8.2 Function Declaration:

```
ruby_ellipsoid(file, P , K , 'color', color, ...
        'name', name)
```

- `file`: file struct

  File structure created with `ruby_create`.

- `P`: (1x3) matrix

  It contains x, y, z coordinates of the center P of the ellipsoid.

- `K`: (3x3) matrix

  It contains the variance-covariance matrix of the ellipsoid.

- `color` (optional): char — [R, G, B]

  Either one of the following colors or a set of RGB values.

  'n'(default): none

  'w': white

  'r': red

  'o': orange

  'y': yellow

  'g': green

  'b': blue

  'p': pink

  'k': black

- `name` (optional): string.

1.8.3 Examples:

```
K = [1.666 -0.424 0.244; -0.424 2.961 -0.558; 0.244 -0.558 4.121];

% Default ellipse
ruby_ellipsoid(file, [0,0,5], K);

% With preset color
ruby_ellipsoid(file, [0,0,10], K, 'color', 'r');

% With RGB color
ruby_ellipsoid(file, [0,0,15], K, 'color', [248,123,156]);

% With label
ruby_ellipsoid(file, [0,0, 20], K, 'name', string('Pos1'));
```

1.9 ruby_plane

1.9.1 Description:

Constructs a polygone.

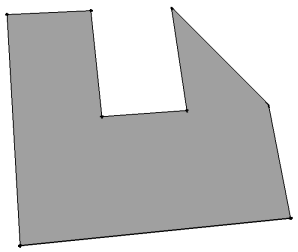


Figure 8: Example of polygone drawn by the function ruby_plane

1.9.2 Function Declaration:

```
ruby_plane(file, XYZ, 'texture', texture, 'color', color, 'name', name)
```

- `file`: file struct

  File structure created with `ruby_create`.

- `XYZ`: (Nx3) matrix

  It is a numeric matrix consisting of points that define a plane. Points must not be colinear, but must be coplanar.

- `texture` (optional): ” (default)

  An image file path with extension ‘.png’, ‘jpg’or‘.jpeg’

- `color` (optional): char — [R, G, B]

  Either one of the following colors or a set of RGB values.

  ‘n’(default): none

  ‘w’: white

  ‘r’: red

'o': orange

'y': yellow

'g': green

'b': blue

'p': pink

'k': black

- `name` (optional): string

1.9.3 Examples:

```
% Inclined plane
ruby_plane(file_id, [ 5 , 0 , 0 ;...
                      5 , 2 , 2 ;...
                      6 , 2 , 2 ;...
                      6 , 1 , 1 ;...
                      7 , 1 , 1 ;...
                      7 , 2 , 2 ;...
                      8 , 1 , 1 ;...
                      8 , 0 , 0 ])

% Triangle with texture and name.
ruby_plane(file, [0, 0, 3; 0, 3, 0; 2, 0, 0],...
        'texture', '/images/rainbow.jpeg', 'name', string('myplane'));

% Rectangle with red color.
ruby_plane(file, [0, 1, 3; 0, 1, 5; 1, 2, 5; 1, 2, 3], 'color', 'r');

% Triangle with blue color.
ruby_plane(file, [0, 4, 3; 0, 7, 0; 2, 4, 0], 'color', [0, 0, 255]);
```

1.10 ruby_tin

1.10.1 Description:

Constructs a triangulated irregular network (tin).

1.10.2 Function Declaration:

```
ruby_tin(file, XYZ , triangles, 'texture', texture, 'color', color, ...
        'name', name)
```

- `file`: file struct

  File structure created with `ruby_create`.

- `XYZ`: (Nx3) matrix

  It is a numeric matrix consists of position of points. N is the number of points.

- `triangles`: (Mx3) matrix

  It is a numeric matrix which consists of indexes of triangle points. M is the number of triangles. The triangles are best created using the `dalaunay` function.

- `texture` (optional): '' (default)

  An image file path with extension '.png', 'jpg'or'.jpeg'

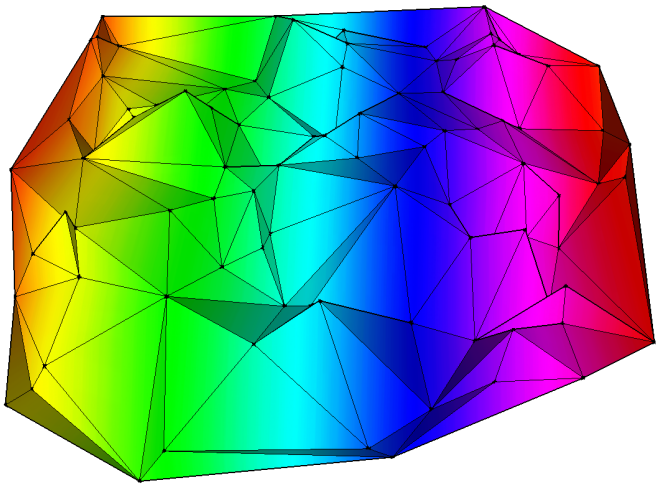


Figure 9: Example of Digital Elevation Model created by ruby_tin with a gradient texture overlaid

- `color` (optional): char — [R, G, B]

  Either one of the following colors or a set of RGB values.

  'n'(default): none

  'w': white

  'r': red

  'o': orange

  'y': yellow

  'g': green

  'b': blue

  'p': pink

  'k': black

- `name` (optional): string.

Examples:

```
Points = [ 10 * rand(100,1) + 10 , 10*rand(100,1) ,  randn(100,1) ];
triangles = delaunay(Points(:, 1:2));

ruby_tin(file, Points, triangles, 'texture', '/images/rainbow.jpeg',...
    'name', string('Elevation'));
```

1.11 ruby_antenna

1.11.1 Description:

Constructs an antenna or multiple antennas.

1.11.2 Function Declaration:

```
ruby_antenna(file, P, 'color', color, 'name', name)
```

- `file`: file struct

  File structure created with `ruby_create`.

- `P`: (Nx3) matrix

  A numeric matrix consisting of antenna positions. N is the number of antennas.

- `color` (optional): char — [R, G, B]

  Either one of the following colors or a set of RGB values.

  'r'(default): red

  'w': white

  'o': orange

  'y': yellow

  'g': green

  'b': blue

  'p': pink

  'k': black

- name (optional): string — column vector containing strings

1.11.3 Examples:

```
% Create default (red) antenna at 2,2,2
ruby_antenna(file, [2, 2, 2]);

% Green Antenna
ruby_antenna(file, [2, 2, 3], 'color', 'g');

%Blue Antenna
ruby_antenna(file, [2, 2, 4], 'color', [0, 0, 255]);

% Add global antenna name
ruby_antenna(file, [2, 2, 5], 'name', string('Ant1'));
```

1.12 ruby_arrow

1.12.1 Description:

Constructs an arrow or quiver plot.

1.12.2 Function Declaration:

```
ruby_arrow(file, P , v, 'color', color, 'name', name)
```

- `file`: file struct

  File structure created with `ruby_create`.

- `P`: (3x1) matrix

  Is a vector with the coordinates.

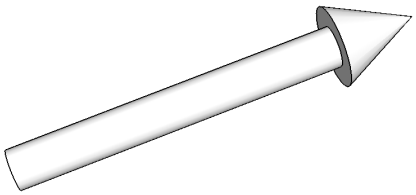


Figure 10: Example of single arrow

- `v`: (3x1) matrix
  Is a vector representing the direction of the arrow.
- `color` (optional): char — [R, G, B]
  Either one of the following colors or a set of RGB values.
  'n'(default): none
  'w': white
  'r': red
  'o': orange
  'y': yellow
  'g': green
  'b': blue
  'p': pink
  'k': black
- name (optional): string

1.12.3 Examples:

```
% Draw arrow at 5,5,5 pointing along x axis
ruby_arrow(file, [5; 5; 5], [1; 0; 0]);

% Red arrow
ruby_arrow(file, [5; 5; 5], [0; 1; 0], 'color', 'r'));

% Labeled blue arrow
ruby_arrow(file, [5; 5; 5], [0; 0; 1], 'color', [0, 0, 255], ...
        'name', string('Arrow1'));
```

2 User Manual for Python

2.1 ruby_create

2.1.1 Description:

Creates and initializes a ruby file. Returns file object **`file`**.

2.1.2 Function Declaration:

```
file = ruby_create(name_or_path = 'script_ruby_sketchup.rb')
```

- `nameOrPath` (optional): string

  File name of output file. .rb file extension is mandatory.

2.1.3 Examples:

```
# Open 'script_ruby_sketchup.rb' in local folder.
file = ruby_create()

# Open 'script3d.rb' in local folder.
file = ruby_create('script3d.rb')

# Open '/Users/myUser/Documents/script3d.rb'.
file = ruby_create('/Users/myUser/Documents/script3d.rb')
```

2.2 ruby_close

2.2.1 Description:

Completes and closes the ruby file.

2.2.2 Function Declaration:

```
ruby_close(file)
```

- **`file`**: file struct

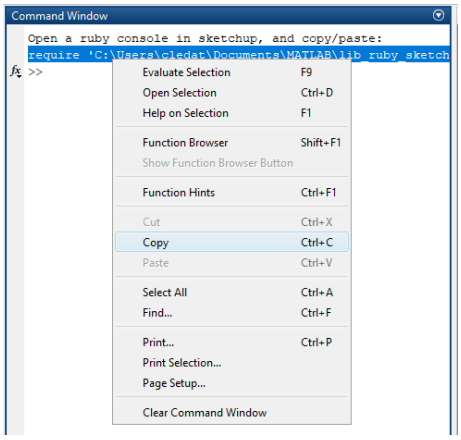
  File structure created with `ruby_create`.

2.2.3 Examples:

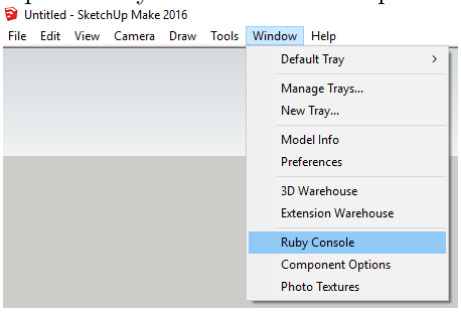
```
ruby_close(file);
```

2.3 Input in SketchUp

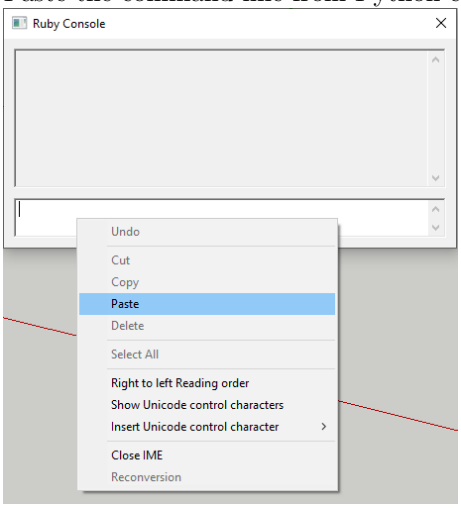
If the file is closed correctly, it could be inputted in SketchUp as follow.
Copy the commanded printed on Python console.



Open a ruby console in SketchUp



Paste the command line from Python console



2.4 ruby_point

2.4.1 Description:

Creates isolated points.

2.4.2 Function Declaration:

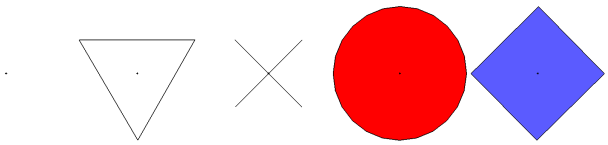


Figure 11: Different type of points

```
ruby_point(file, XYZ, issymbolic = 0, symbol = 'triangle', color = 'n', name = '');
```

- **file**: file struct

  File structure created with `ruby_create`.

- `XYZ`: (Nx3) numpy.array

  N is the number of points. Each row corresponds to a point and the columns correspond to the three coordinate axis.

- `issymbolic` (optional): int — (Nx1) numpy.array

  Single integer or column vector containing 0 and 1 values.

  If it is a numeric value, all points are treated equally in terms of symbols. If it is 0, there is no symbol. If it is 1, a symbol is used for each and every point.

  If it is a column vector, the number of rows should be the same as the number of rows in XYZ matrix.

- `symbol` (optional): 'triangle'(default) — 'cross'— 'circle'— 'square'

  It defines how points are represented in 3D plane by assigning a particular symbol.

- `color` (optional): char

  One of the following colors.

  'n'(default): none

  'w': white

  'r': red

  'o': orange

  'y': yellow

  'g': green

  'b': blue

  'p': pink

  'k': black

- `name` (optional): string — (Nx1) numpy.array of strings

  If it is a column vector, each point is given a separate name. Otherwise all points share the same name.

2.4.3 Examples:

```
# Draws a single point
ruby_point(file, np.array([[-5, 2, 2]]))

# Draws 2 points with default symbol(triangle) and default color(white)
ruby_point(file, np.array([[-4, 2, 2], [-3, 2, 2]]), issymbolic = 1)
```

```
# Draws 2 points with default symbol(triangle) and color black
ruby_point(file, np.array([[-2, 2, 2], [-1, 2, 2]]), issymbolic = 1,
    color = 'k')

# Draws 2 points with color red
# One with symbol(square) and other without symbols
# Both share the same name ('points')
ruby_point(file, np.array([[2, 2, 2], [3, 2, 2]]),
    issymbolic = np.array([[1], [0]]), symbol = 'square', color = 'r',
    name = "points")

# Draws 2 points with color green
# One without symbols and other with symbol(circle)
# They are named 'point1' and 'point2' respectively
ruby_point(file, np.array([[4, 2, 2], [5, 2, 2]]),
    issymbolic = np.array([[0], [1]]), symbol = 'circle', color = 'g',
    name = np.array([["point1"], ["point2"]]))
```

2.5 ruby_line

2.5.1 Description:

Draws a line or polyline.

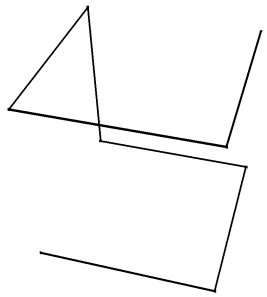


Figure 12: Example of line drawn by the library

2.5.2 Function Declaration:

```
ruby_line(file, XYZ, name = 'name');
```

- **file**: file struct

  File structure created with `ruby_create`.

- `XYZ`: (Nx3) numpy.array N is the number of points and should be at least 2. Each row corresponds to a point and the columns correspond to three coordinate axis. The points are connected by a line one by one according to the given order. In other words, the first point is connected with the second point, the second point is connected with the third point and so on.

- `name` (optional): string — ((N-1)x1) numpy.array of strings

  If it is a column vector, each segment is given a separate name. Otherwise all segments share the same name.

2.5.3 Examples:

```
# Draws 3 connected lines
ruby_line(file, np.array([[0, 3, 2], [0, 3, 4], [0, 5, 4], [0, 5, 2]]))

# Draws 2 connected lines with the same name ('lines')
ruby_line(file, np.array([[0, 3, 4], [2, 3, 4], [2, 3, 2]]), name = "lines")

# Draws 2 connected lines with different names ('line1', 'line2')
ruby_line(file, np.array([[0, 5, 4], [2, 5, 4], [2, 5, 2]]),
    name = np.array([["line1"], ["line2"]]))
```

2.6 ruby_axis

2.6.1 Description:

Creates a reference coordinate axis by changing the orientation and center point of the original axis.

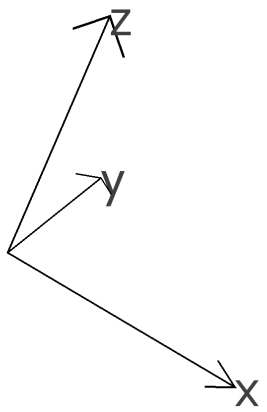


Figure 13: Axis definition

2.6.2 Function Declaration:

`ruby_axis(file, P, R, name = '')`

- **file**: file struct
  File structure created with `ruby_create`.
- `P`: (1x3) numpy.array
  Contains x, y, z coordinates of the axis origin.
- `R`: (3x3) numpy.array
  Orthogonal rotation matrix representing the orientation of the axis.
- `name` (optional): string

2.6.3 Examples:

```
# Orthogonal orientation matrix
R = numpy.array([[1, 0, 0], [0, cos(1), sin(1)], [0, -sin(1), cos(1)]])
# Draws an axis with name ('reference_axis')
ruby_axis(file, np.array([[4, 4, 4]]), R, name = "reference_axis")
```

2.7 ruby_pose

2.7.1 Description:

Constructs a pose.

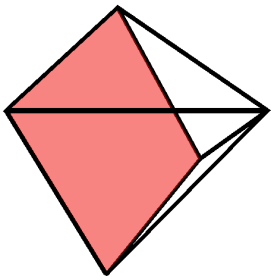


Figure 14: Camera pose

2.7.2 Function Declaration:

`ruby_pose(`**`file`**`, P, R, focal = 0.2, width = 0.1, height = 0.1, color = 'n', name = '')`

- **`file`**: file struct
  File structure created with `ruby_create`.
- `P`: (1x3) numpy.array
  Contains x, y, z coordinates of the center of the projection of the pose.
- `R`: (3x3) numpy.array
  Orthogonal rotation matrix representing the orientation of the pose.
- `focal` (optional): float
  In meters.
- `width` (optional): float
  In meters.
- `height` (optional): float
  In meters.
- `color` (optional): char
  One of the following colors.
  'n'(default): none
  'w': white
  'r': red
  'o': orange
  'y': yellow
  'g': green
  'b': blue
  'p': pink
  'k': black
- `name` (optional): string

2.7.3 Examples:

```
# Orthogonal Matrix
R = np.array([[1, 0, 0], [0,  np.cos(1), np.sin(1)], [0, -np.sin(1), np.cos(1)]])

# Draws a pose with default focal length (0.2), width (0.1), height (0.1)
# and color (weight) options
ruby_pose(file, np.array([[5, 5, 5]]), R)

# Draws a pose with a name('pose1') and focal length (0.1)
ruby_pose(file, np.array([[6, 5, 5]]), R,  focal = 0.1, name = "pose1")

# Draws a pose with the given focal length (0.6) and color (orange)
ruby_pose(file, np.array([[7, 5, 5]]), R,  focal = 0.6, color = 'o')

# Draws a pose with the given focal length (0.8) and width (0.3)
ruby_pose(file, np.array([[8, 5, 5]]), R, focal = 0.8, width = 0.3, color = 'p')

# Draws a pose with the given focal length (1), width (0.3), height (0.2)
# and color (red)
ruby_pose(file, np.array([[9, 5, 5]]), R, focal = 1, width = 0.3, height = 0.2,
    color = 'r')
```

2.8 ruby_ellipsoid

2.8.1 Description:

Creates an ellipsoid.

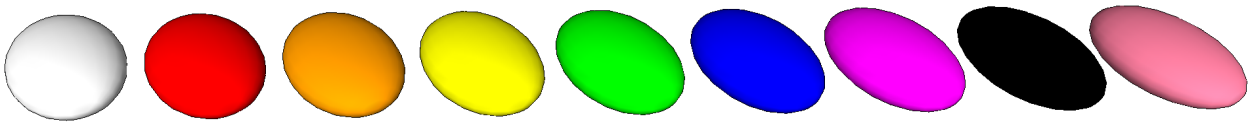


Figure 15: Ellipsoids

An ellipsoid is a surface defined by a center point P and a covariance matrix (also called tensor) K.

$$v^T \; K \; v = 1 \tag{2}$$

v is the vector between a point of the 3D space and the center of the ellipsoid: P.

2.8.2 Function Declaration:

```
ruby_ellipsoid(file, P , K , texture = '', color = 'n', name='')
```

- **file**: file struct

  File structure created with `ruby_create`.

- `P`: (1x3) numpy.array

  Contains x, y, z coordinates of the center of the ellipsoid.

- `K`: (3x3) numpy.array

  Contains the variance-covariance matrix of the ellipsoid.

- `texture` (optional): string

  Image file path with extension '.png', 'jpg'or'.jpeg'.

- `color` (optional): char

  One of the following colors.

  'n'(default): none

  'w': white

  'r': red

  'o': orange

  'y': yellow

  'g': green

  'b': blue

  'p': pink

  'k': black

- `name` (optional): string

2.8.3 Examples:

```
# Variance-covariance matrix
R = np.array([[1, 0, 0],
    [0,  np.cos(1), np.sin(1)],
    [0, -np.sin(1), np.cos(1)]])
K = np.linalg.inv(R) * np.array([[1, 0, 0], [0, 4, 0], [0, 0, 9]]) * R

# Draws an ellipsoid with color(red) given variance-covariance matrix R
ruby_ellipsoid(file, np.array([[0, 0, 10]]), K, color = 'r')

# Draws an ellipsoid with texture given variance-covariance matrix R
ruby_ellipsoid(file, np.array([[0, 0, 20]]), K, texture = '/images/color.jpg')

# Draws an ellipsoid with color(yellow) and name('error ellipsoid')
# given variance-covariance matrix K
ruby_ellipsoid(file, np.array([[0, 0, 30]]), K, color = 'y',
    name = "error ellipsoid")
```

2.9 ruby_plane

2.9.1 Description:

Constructs a polygone.

2.9.2 Function Declaration:

```
ruby_plane(file, XYZ, texture = '', color = 'n', name = '')
```

- **`file`**: file struct

  File structure created with `ruby_create`.

- `XYZ`: (Nx3) numpy.array

  Numeric matrix that consists of N points that define a plane, points must not be colinear, but must be coplanar. N must be larger than 3.

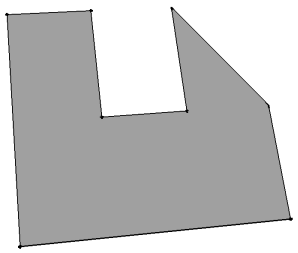


Figure 16: Example of polygone drawn by the function ruby_plane

- `texture` (optional): string
  Image file path with extension '.png', 'jpg'or'.jpeg'.
- `color` (optional): char
  One of the following colors.
  'n'(default): none
  'w': white
  'r': red
  'o': orange
  'y': yellow
  'g': green
  'b': blue
  'p': pink
  'k': black
- `name` (optional): string

Examples:

```
% Inclined plane
ruby_plane(file_id, [ 5 , 0 , 0 ;...
                      5 , 2 , 2 ;...
                      6 , 2 , 2 ;...
                      6 , 1 , 1 ;...
                      7 , 1 , 1 ;...
                      7 , 2 , 2 ;...
                      8 , 1 , 1 ;...
                      8 , 0 , 0 ])

# Draws a blue plane with the label 'myplane'
ruby_plane(file, np.array([[0, 1, 3], [0, 1, 5], [1, 2, 5], [1, 2, 3]]),
    color = 'b', name = 'myplane')
```

2.10 ruby_tin

Description: Constructs a triangulated irregular network (tin).
Function Declaration:

```
ruby_tin(file, XYZ , triangles, texture = '', color = 'n', name = '')
```

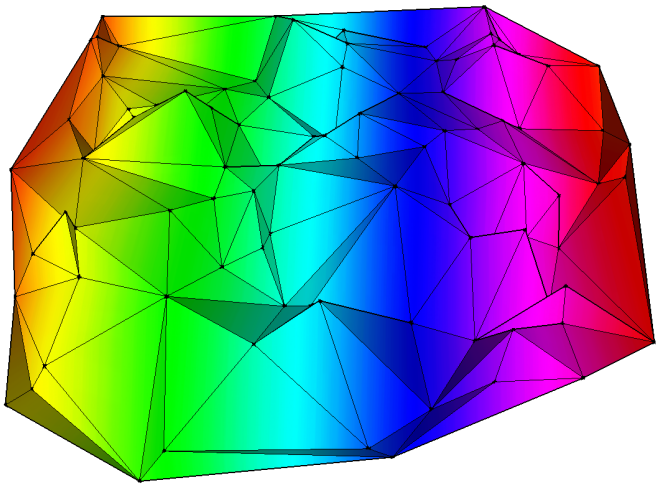


Figure 17: Example of Digital Elevation Model created by ruby_tin with a gradient texture overlaid

- **`file`**: file struct
  File structure created with `ruby_create`.
- `XYZ`: (Nx3) numpy.array
  Numeric matrix of point coordinates. N is the number of points.
- `triangles`: (Mx3) numpy.array
  Numeric matrix of point indexes as returned by delaunay.simplices. M is th number of triangles.
- `texture` (optional): string
  Image file path with extension '.png', 'jpg'or'.jpeg'.
- `color` (optional): char
  One of the following colors.
  'n'(default): none
  'w': white
  'r': red
  'o': orange
  'y': yellow
  'g': green
  'b': blue
  'p': pink
  'k': black
- `name` (optional): string

Examples:

```
# Create dataset
p1 = 10 * np.random.rand(100, 1) + 10
p2 = 10 * np.random.rand(100, 1)
p3 = np.random.rand(100, 1)

points = np.concatenate((p1, p2), axis = 1)
points = np.concatenate((points, p3), axis = 1)

# Extract triangles
triangles = Delaunay(points[:, 0:2])

# Create tin with texture
ruby_tin(file, points, triangles.simplices.copy(),
    texture = '/images/rainbow.jpeg')
```

2.11 ruby_theodolite

2.11.1 Description:

Constructs a theodolite.

2.11.2 Function Declaration:

```
ruby\_theodolite(file, P, name = '')
```

- **file**: file struct

  File structure created with `ruby_create`.

- `P`: (1x3) numpy.array

  Numeric matrix consisting of theodolite coordinates.

- `name` (optional): string

2.11.3 Examples:

```
ruby_theodolite(file, numpy.array([[1, 2, 0]]), name = "theodolite")
```

2.12 ruby_antenna

2.12.1 Description:

Constructs an antenna or multiple antennas.

2.12.2 Function Declaration:

```
ruby\_antenna(file, P, name = '')
```

- **file**: file struct

  File structure created with `ruby_create`.

- `P`: (Nx3) numpy.array

  Numeric matrix consisting of antenna positions. N is the number of antennas.

- `name` (optional): string — (Nx1) numpy.array of strings

  If it is a column vector, each antenna is given a separate name. Otherwise all antennas share the same name.

2.12.3 Examples:

```
ruby_antenna(file, numpy.array([[5, 5, 3]]), name = "antenna")
```

2.13 ruby_resection

2.13.1 Description:

Constructs a resection object.

2.13.2 Function Declaration:

```
ruby_resection(file, P, pos_antennas, name = '')
```

- **file**: file struct
  File structure created with `ruby_create`.
- `P`: (1x3) numpy.array
  Numeric matrix consisting of theodolite coordinates.
- `pos_antennas`: (Nx3) numpy.array
  Numeric matrix consisting of antenna positions. N is the number of antennas.
- `name` (optional): string

2.13.3 Examples:

```
ruby_resection(file, [5, 5, 3], [[0, -10, 0], [0, -2, 0], [-3, 0, 0]],
    name = "resection")
```

2.14 ruby_arrow

2.14.1 Description:

Draws an arrow / quiver plot.

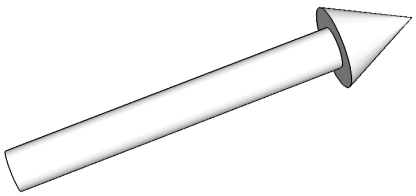


Figure 18: Example of single arrow

2.14.2 Function Declaration:

```
ruby_arrow(file, P, v, color = 'n', name = '')
```

- **file**: file struct
  File structure created with `ruby_create`.
- `P`: (3x1) numpy.array
  Coordinate vector.

- v: (3x1) numpy.array

  Orientation vector.

- color (optional): char

  One of the following colors.

  'n'(default): none

  'w': white

  'r': red

  'o': orange

  'y': yellow

  'g': green

  'b': blue

  'p': pink

  'k': black

- name (optional): string

2.14.3 Examples:

```
# Draws an arrow at 5,5,5, pointing along x axis
ruby_arrow(file, np.array([[5], [5], [5]]), np.array([[1], [0], [0]]))

# Draws a red arrow
ruby_arrow(file, np.array([[5], [5], [5]]), np.array([[0], [1], [0]]), \
    color = 'r')

# Draws a green arrow with label
ruby_arrow(file, np.array([[5], [5], [5]]), np.array([[0], [0], [1]]), \
    color = 'g', name = "arrow3")
```

Acknowledgements

The author would like to thank in particular Beat Geissmann for his technical contribution in formatting both Matlab and Python codes. Moreover, we would like to thank Pinar Ezgi Çöl and Simon Lütolf for their help, and the supervisors: Jan Skaloud, Jean-François Hullo and Bertrand Merminod who encouraged for the development of this toolbox.

References

[1] SketchUp, *3D Design Software, 3D Modeling on the Web.* https://www.sketchup.com/: Trimble, 2019.

[2] E. Thierry, "Technologie et DP3 - La technologie au collège," http://tkcollege.fr/informatique---sketchup---appartement.html.

[3] E. CLEDAT, "Conception d'une méthode de consolidation de grands réseaux lasergrammétriques." Master's thesis, INSA de Strasbourg http://eprints2.insa-strasbourg.fr/2060/, EDF: Electricité de France, Sep. 2015.

[4] M. Rehak, "Integrated sensor orientation on micro aerial vehicles," Ph.D. dissertation, Lausanne, 2017. [Online]. Available: http://infoscience.epfl.ch/record/225965

[5] J.-F. Hullo, "Fine registration of kilo-station networks - a modern procedure for terrestrial laser scanning data sets," in *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLI-B5. Copernicus GmbH, Jun. 2016, pp. 485–492. [Online]. Available: https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLI-B5/485/2016/

[6] E. Cledat, D. Cucci, and J. Skaloud, "Planning of UAV mission for precise terrain reconstruction in steep terrain without ground control points," Swiss Geoscience Meeting, Davos, Nov. 2017.

[7] E. Cledat and D. A. Cucci, "Mapping gnss restricted environments with a drone tandem and indirect position control," *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. IV-2/W3, pp. 1–7, 2017. [Online]. Available: https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-2-W3/1/2017/

[8] E. Cledat and M. Gebhard, "Prédiction de la précision d'un relevé photogrammétrique aéroporté par drone," Geosummit, Bern, Jun. 2018.

[9] E. Cledat, L. V. Jospin, A. S. Dubois, A. Weltermann, D. A. Cucci, and J. Skaloud, "Navigation de précision pour l'intervention de drones dans des zones difficiles," Geosummit, Bern, Jun. 2018.