Numpy stl Documentation

Release 1.8.0

Rick van Hattem

March 26, 2016

Contents

Contents:

CHAPTER 1

stl-numpy

Simple library to make working with STL files (and 3D objects in general) fast and easy.

Due to all operations heavily relying on *numpy* this is one of the fastest STL editing libraries for Python available.

1.1 Links

- The source: https://github.com/WoLpH/numpy-stl
- Project page: https://pypi.python.org/pypi/numpy-stl
- Reporting bugs: https://github.com/WoLpH/numpy-stl/issues
- Documentation: http://numpy-stl.readthedocs.org/en/latest/
- My blog: http://w.wol.ph/

1.2 Requirements for installing:

- numpy any recent version
- python-utils version 1.6 or greater

1.3 Installation:

pip install numpy-stl

1.4 Initial usage:

- *stl2bin your_ascii_stl_file.stl new_binary_stl_file.stl*
- *stl2ascii your_binary_stl_file.stl new_ascii_stl_file.stl*
- *stl your_ascii_stl_file.stl new_binary_stl_file.stl*

1.5 Quickstart

```
import numpy
from stl import mesh

# Using an existing stl file:
your_mesh = mesh.Mesh.from_file('some_file.stl')

# Or creating a new mesh (make sure not to overwrite the `mesh` import by
# naming it `mesh`):
VERTICE_COUNT = 100
data = numpy.zeros(VERTICE_COUNT, dtype=mesh.Mesh.dtype)
your_mesh = mesh.Mesh(data, remove_empty_areas=False)

# The mesh normals (calculated automatically)
your_mesh.normals
# The mesh vectors
your_mesh.v0, your_mesh.v1, your_mesh.v2
# Accessing individual points (concatenation of v0, v1 and v2 in triplets)
assert (your_mesh.points[0][0:3] == your_mesh.v0[0]).all()
assert (your_mesh.points[0][3:6] == your_mesh.v1[0]).all()
assert (your_mesh.points[0][6:9] == your_mesh.v2[0]).all()
assert (your_mesh.points[1][0:3] == your_mesh.v0[1]).all()

your_mesh.save('new_stl_file.stl')
```

Plotting using matplotlib is equally easy:

```
from stl import mesh
from mpl_toolkits import mplot3d
from matplotlib import pyplot

# Create a new plot
figure = pyplot.figure()
axes = mplot3d.Axes3D(figure)

# Load the STL files and add the vectors to the plot
your_mesh = mesh.Mesh.from_file('tests/stl_binary/HalfDonut.stl')
axes.add_collection3d(mplot3d.art3d.Poly3DCollection(your_mesh.vectors))

# Auto scale to the mesh size
scale = your_mesh.points.flatten(-1)
axes.auto_scale_xyz(scale, scale, scale)

# Show the plot to the screen
pyplot.show()
```

1.6 Modifying Mesh objects

```
from stl import mesh
import math
import numpy

# Create 3 faces of a cube
data = numpy.zeros(6, dtype=mesh.Mesh.dtype)
```

```
# Top of the cube
data['vectors'][0] = numpy.array([[0, 1, 1],
                                  [1, 0, 1],
                                  [0, 0, 1]])
data['vectors'][1] = numpy.array([[1, 0, 1],
                                  [0, 1, 1],
                                  [1, 1, 1]])
# Right face
data['vectors'][2] = numpy.array([[1, 0, 0],
                                  [1, 0, 1],
                                  [1, 1, 0]])
data['vectors'][3] = numpy.array([[1, 1, 1],
                                  [1, 0, 1],
                                  [1, 1, 0]])
# Left face
data['vectors'][4] = numpy.array([[0, 0, 0],
                                  [1, 0, 0],
                                  [1, 0, 1]])
data['vectors'][5] = numpy.array([[0, 0, 0],
                                  [0, 0, 1],
                                  [1, 0, 1]])

# Since the cube faces are from 0 to 1 we can move it to the middle by
# substracting .5
data['vectors'] -= .5

# Generate 4 different meshes so we can rotate them later
meshes = []
for _ in range(4):
    meshes.append(mesh.Mesh(data.copy()))

# Rotate 90 degrees over the Y axis
meshes[0].rotate([0.0, 0.5, 0.0], math.radians(90))

# Translate 2 points over the X axis
meshes[1].x += 2

# Rotate 90 degrees over the X axis
meshes[2].rotate([0.5, 0.0, 0.0], math.radians(90))
# Translate 2 points over the X and Y points
meshes[2].x += 2
meshes[2].y += 2

# Rotate 90 degrees over the X and Y axis
meshes[3].rotate([0.5, 0.0, 0.0], math.radians(90))
meshes[3].rotate([0.0, 0.5, 0.0], math.radians(90))
# Translate 2 points over the Y axis
meshes[3].y += 2


# Optionally render the rotated cube faces
from matplotlib import pyplot
from mpl_toolkits import mplot3d

# Create a new plot
figure = pyplot.figure()
axes = mplot3d.Axes3D(figure)
```

```
# Render the cube faces
for m in meshes:
    axes.add_collection3d(mplot3d.art3d.Poly3DCollection(m.vectors))

# Auto scale to the mesh size
scale = numpy.concatenate([m.points for m in meshes]).flatten(-1)
axes.auto_scale_xyz(scale, scale, scale)

# Show the plot to the screen
pyplot.show()
```

1.7 Extending Mesh objects

```
from stl import mesh
import math
import numpy

# Create 3 faces of a cube
data = numpy.zeros(6, dtype=mesh.Mesh.dtype)

# Top of the cube
data['vectors'][0] = numpy.array([[0, 1, 1],
                                  [1, 0, 1],
                                  [0, 0, 1]])
data['vectors'][1] = numpy.array([[1, 0, 1],
                                  [0, 1, 1],
                                  [1, 1, 1]])
# Right face
data['vectors'][2] = numpy.array([[1, 0, 0],
                                  [1, 0, 1],
                                  [1, 1, 0]])
data['vectors'][3] = numpy.array([[1, 1, 1],
                                  [1, 0, 1],
                                  [1, 1, 0]])
# Left face
data['vectors'][4] = numpy.array([[0, 0, 0],
                                  [1, 0, 0],
                                  [1, 0, 1]])
data['vectors'][5] = numpy.array([[0, 0, 0],
                                  [0, 0, 1],
                                  [1, 0, 1]])

# Since the cube faces are from 0 to 1 we can move it to the middle by
# substracting .5
data['vectors'] -= .5

cube_back = mesh.Mesh(data.copy())
cube_front = mesh.Mesh(data.copy())

# Rotate 90 degrees over the X axis followed by the Y axis followed by the
# X axis
cube_back.rotate([0.5, 0.0, 0.0], math.radians(90))
cube_back.rotate([0.0, 0.5, 0.0], math.radians(90))
cube_back.rotate([0.5, 0.0, 0.0], math.radians(90))

cube = mesh.Mesh(numpy.concatenate([
```

```
    cube_back.data.copy(),
    cube_front.data.copy(),
]))

# Optionally render the rotated cube faces
from matplotlib import pyplot
from mpl_toolkits import mplot3d

# Create a new plot
figure = pyplot.figure()
axes = mplot3d.Axes3D(figure)

# Render the cube
axes.add_collection3d(mplot3d.art3d.Poly3DCollection(cube.vectors))

# Auto scale to the mesh size
scale = cube_back.points.flatten(-1)
axes.auto_scale_xyz(scale, scale, scale)

# Show the plot to the screen
pyplot.show()
```

1.8 Creating Mesh objects from a list of vertices and faces

```
import numpy as np
from stl import mesh

# Define the 8 vertices of the cube
vertices = np.array([\
    [-1, -1, -1],
    [+1, -1, -1],
    [+1, +1, -1],
    [-1, +1, -1],
    [-1, -1, +1],
    [+1, -1, +1],
    [+1, +1, +1],
    [-1, +1, +1]])
# Define the 12 triangles composing the cube
faces = np.array([\
    [0,3,1],
    [1,3,2],
    [0,4,7],
    [0,7,3],
    [4,5,6],
    [4,6,7],
    [5,1,2],
    [5,2,6],
    [2,3,6],
    [3,7,6],
    [0,1,5],
    [0,5,4]])

# Create the mesh
cube = mesh.Mesh(np.zeros(faces.shape[0], dtype=mesh.Mesh.dtype))
for i, f in enumerate(faces):
    for j in range(3):
```

```
        cube.vectors[i][j] = vertices[f[j],:]

# Write the mesh to file "cube.stl"
cube.save('cube.stl')
```

CHAPTER 2

stl package

2.1 Submodules

2.2 stl.main module

stl.main.**main**()

stl.main.**to_ascii**()

stl.main.**to_binary**()

2.3 stl.base module

stl.base.**AREA_SIZE_THRESHOLD = 0**
When removing empty areas, remove areas that are smaller than this

class stl.base.**BaseMesh**(*data*, *calculate_normals=True*, *remove_empty_areas=False*, *remove_duplicate_polygons=<RemoveDuplicates.NONE: 0>*, *name=u''*, ***kwargs*)
Bases: python_utils.logger.Logged, _abcoll.Mapping

Mesh object with easy access to the vectors through v0, v1 and v2. The normals, areas, min, max and units are calculated automatically.

Parameters

- **data** (*numpy.array*) – The data for this mesh
- **calculate_normals** (*bool*) – Whether to calculate the normals
- **remove_empty_areas** (*bool*) – Whether to remove triangles with 0 area (due to rounding errors for example)

Variables

- **name** (*str*) – Name of the solid, only exists in ASCII files
- **data** (*numpy.array*) – Data as *BaseMesh.dtype()*
- **points** (*numpy.array*) – All points (Nx9)
- **normals** (*numpy.array*) – Normals for this mesh, calculated automatically by default (Nx3)

- **vectors** (*numpy.array*) – Vectors in the mesh (Nx3x3)
- **attr** (*numpy.array*) – Attributes per vector (used by binary STL)
- **x** (*numpy.array*) – Points on the X axis by vertex (Nx3)
- **y** (*numpy.array*) – Points on the Y axis by vertex (Nx3)
- **z** (*numpy.array*) – Points on the Z axis by vertex (Nx3)
- **v0** (*numpy.array*) – Points in vector 0 (Nx3)
- **v1** (*numpy.array*) – Points in vector 1 (Nx3)
- **v2** (*numpy.array*) – Points in vector 2 (Nx3)

```
>>> data = numpy.zeros(10, dtype=BaseMesh.dtype)
>>> mesh = BaseMesh(data, remove_empty_areas=False)
>>> # Increment vector 0 item 0
>>> mesh.v0[0] += 1
>>> mesh.v1[0] += 2
```

```
>>> # Check item 0 (contains v0, v1 and v2)
>>> mesh[0]
array([ 1.,  1.,  1.,  2.,  2.,  2.,  0.,  0.,  0.], dtype=float32)
>>> mesh.vectors[0]
array([[ 1.,  1.,  1.],
       [ 2.,  2.,  2.],
       [ 0.,  0.,  0.]], dtype=float32)
>>> mesh.v0[0]
array([ 1.,  1.,  1.], dtype=float32)
>>> mesh.points[0]
array([ 1.,  1.,  1.,  2.,  2.,  2.,  0.,  0.,  0.], dtype=float32)
>>> mesh.data[0]
([0.0, 0.0, 0.0],
[[1.0, 1.0, 1.0], [2.0, 2.0, 2.0], [0.0, 0.0, 0.0]],
[0])
>>> mesh.x[0]
array([ 1.,  2.,  0.], dtype=float32)
```

```
>>> mesh[0] = 3
>>> mesh[0]
array([ 3.,  3.,  3.,  3.,  3.,  3.,  3.,  3.,  3.], dtype=float32)
```

```
>>> len(mesh) == len(list(mesh))
True
>>> (mesh.min_ < mesh.max_).all()
True
>>> mesh.update_normals()
>>> mesh.units.sum()
0.0
>>> mesh.v0[:] = mesh.v1[:] = mesh.v2[:] = 0
>>> mesh.points.sum()
0.0
```

areas
Mesh areas

dtype = dtype([('normals', '<f4', (3,)), ('vectors', '<f4', (3, 3)), ('attr', '<u2', (1,))])

- normals: `numpy.float32()`, *(3,)*
- vectors: `numpy.float32()`, *(3, 3)*

•attr: `numpy.uint16()`, *(1,)*

max_
Mesh maximum value

min_
Mesh minimum value

classmethod remove_duplicate_polygons(*data*, *value=<RemoveDuplicates.SINGLE: 1>*)

classmethod remove_empty_areas(*data*)

rotate(*axis*, *theta*, *point=None*)
Rotate the matrix over the given axis by the given theta (angle)

Uses the **'rotation_matrix'_** in the background.

Parameters

- **axis** (*numpy.array*) – Axis to rotate over (x, y, z)
- **theta** (*float*) – Rotation angle in radians, use *math.radians* to

convert degrees to radians if needed. :param numpy.array point: Rotation point so manual translation is not required

classmethod rotation_matrix(*axis*, *theta*)
Generate a rotation matrix to Rotate the matrix over the given axis by the given theta (angle)

Uses the Euler-Rodrigues formula for fast rotations: **'https://en.wikipedia.org/wiki/Euler%E2%80%93Rodrigues_formu**

Parameters

- **axis** (*numpy.array*) – Axis to rotate over (x, y, z)
- **theta** (*float*) – Rotation angle in radians, use *math.radians* to

convert degrees to radians if needed.

units
Mesh unit vectors

update_areas()

update_max()

update_min()

update_normals()
Update the normals for all points

update_units()

stl.base.**DIMENSIONS = 3**
Dimensions used in a vector

class stl.base.**Dimension**
Bases: enum.IntEnum

X = <Dimension.X: 0>
X index (for example, *mesh.v0[0][X]*)

Y = <Dimension.Y: 1>
Y index (for example, *mesh.v0[0][Y]*)

Z = <Dimension.Z: 2>
Z index (for example, *mesh.v0[0][Z]*)

class stl.base.**RemoveDuplicates**
Bases: enum.Enum

Choose whether to remove no duplicates, leave only a single of the duplicates or remove all duplicates (leaving holes).

ALL = <RemoveDuplicates.ALL: 2>

NONE = <RemoveDuplicates.NONE: 0>

SINGLE = <RemoveDuplicates.SINGLE: 1>

classmethod map(*value*)

stl.base.**VECTORS = 3**
Vectors in a point

2.4 stl.mesh module

class stl.mesh.**Mesh**(*data*, *calculate_normals=True*, *remove_empty_areas=False*, *remove_duplicate_polygons=<RemoveDuplicates.NONE: 0>*, *name=u''*, ***kwargs*)
Bases: *stl.stl.BaseStl*

2.5 stl.stl module

stl.stl.**ASCII = 1**
Force writing ASCII

stl.stl.**AUTOMATIC = 0**
Automatically detect whether the output is a TTY, if so, write ASCII otherwise write BINARY

stl.stl.**BINARY = 2**
Force writing BINARY

stl.stl.**BUFFER_SIZE = 4096**
Amount of bytes to read while using buffered reading

class stl.stl.**BaseStl**(*data*, *calculate_normals=True*, *remove_empty_areas=False*, *remove_duplicate_polygons=<RemoveDuplicates.NONE: 0>*, *name=u''*, ***kwargs*)
Bases: *stl.base.BaseMesh*

classmethod from_file(*filename*, *calculate_normals=True*, *fh=None*, *mode=0*, ***kwargs*)
Load a mesh from a STL file

Parameters

- **filename** (*str*) – The file to load
- **calculate_normals** (*bool*) – Whether to update the normals
- **fh** (*file*) – The file handle to open
- ****kwargs** (*dict*) – The same as for *stl.mesh.Mesh*

classmethod from_multi_file(*filename*, *calculate_normals=True*, *fh=None*, *mode=1*, ***kwargs*)
Load multiple meshes from a STL file

Parameters

- **filename** (*str*) – The file to load
- **calculate_normals** (*bool*) – Whether to update the normals
- **fh** (*file*) – The file handle to open
- ****kwargs** (*dict*) – The same as for *stl.mesh.Mesh*

classmethod load(*fh*, *mode=0*)
Load Mesh from STL file

Automatically detects binary versus ascii STL files.

Parameters

- **fh** (*file*) – The file handle to open
- **mode** (*int*) – Automatically detect the filetype or force binary

save(*filename*, *fh=None*, *mode=0*, *update_normals=True*)
Save the STL to a (binary) file

If mode is *AUTOMATIC* an *ASCII* file will be written if the output is a TTY and a *BINARY* file otherwise.

Parameters

- **filename** (*str*) – The file to load
- **fh** (*file*) – The file handle to open
- **mode** (*int*) – The mode to write, default is *AUTOMATIC*.
- **update_normals** (*bool*) – Whether to update the normals

stl.stl.**COUNT_SIZE = 4**
The amount of bytes in the count field

stl.stl.**HEADER_SIZE = 80**
The amount of bytes in the header field

stl.stl.**MAX_COUNT = 1000000.0**
The maximum amount of triangles we can read from binary files

CHAPTER 3

tests and examples

3.1 tests.stl_corruption module

```
from __future__ import print_function
import pytest
import struct

from stl import mesh

_STL_FILE = '''
solid test.stl
facet normal -0.014565 0.073223 -0.002897
  outer loop
    vertex 0.399344 0.461940 1.044090
    vertex 0.500000 0.500000 1.500000
    vertex 0.576120 0.500000 1.117320
  endloop
endfacet
endsolid test.stl
'''.lstrip()


def test_valid_ascii(tmpdir):
    tmp_file = tmpdir.join('tmp.stl')
    with tmp_file.open('w+') as fh:
        fh.write(_STL_FILE)
        fh.seek(0)
        mesh.Mesh.from_file(str(tmp_file), fh=fh)


def test_ascii_with_missing_name(tmpdir):
    tmp_file = tmpdir.join('tmp.stl')
    with tmp_file.open('w+') as fh:
        # Split the file into lines
        lines = _STL_FILE.splitlines()

        # Remove everything except solid
        lines[0] = lines[0].split()[0]

        # Join the lines to test files that start with solid without space
        fh.write('\n'.join(lines))
        fh.seek(0)
```

```
        mesh.Mesh.from_file(str(tmp_file), fh=fh)


def test_ascii_with_blank_lines(tmpdir):
    _stl_file = '''
    solid test.stl


      facet normal -0.014565 0.073223 -0.002897

        outer loop

          vertex 0.399344 0.461940 1.044090
          vertex 0.500000 0.500000 1.500000

          vertex 0.576120 0.500000 1.117320

        endloop

      endfacet

    endsolid test.stl
    '''.lstrip()

    tmp_file = tmpdir.join('tmp.stl')
    with tmp_file.open('w+') as fh:
        fh.write(_stl_file)
        fh.seek(0)
        mesh.Mesh.from_file(str(tmp_file), fh=fh)


def test_incomplete_ascii_file(tmpdir):
    tmp_file = tmpdir.join('tmp.stl')
    with tmp_file.open('w+') as fh:
        fh.write('solid some_file.stl')
        fh.seek(0)
        with pytest.raises(struct.error):
            mesh.Mesh.from_file(str(tmp_file), fh=fh)

    with tmp_file.open('w+') as fh:
        fh.write(_STL_FILE[:-20])
        fh.seek(0)
        with pytest.raises(AssertionError):
            mesh.Mesh.from_file(str(tmp_file), fh=fh)

    with tmp_file.open('w+') as fh:
        fh.write(_STL_FILE[:82])
        fh.seek(0)
        with pytest.raises(struct.error):
            mesh.Mesh.from_file(str(tmp_file), fh=fh)

    with tmp_file.open('w+') as fh:
        fh.write(_STL_FILE[:100])
        fh.seek(0)
        with pytest.raises(AssertionError):
            mesh.Mesh.from_file(str(tmp_file), fh=fh)
```

```
def test_corrupt_ascii_file(tmpdir):
    tmp_file = tmpdir.join('tmp.stl')
    with tmp_file.open('w+') as fh:
        fh.write(_STL_FILE)
        fh.seek(40)
        print('####\n' * 100, file=fh)
        fh.seek(0)
        with pytest.raises(AssertionError):
            mesh.Mesh.from_file(str(tmp_file), fh=fh)

    with tmp_file.open('w+') as fh:
        fh.write(_STL_FILE)
        fh.seek(40)
        print(' ' * 100, file=fh)
        fh.seek(80)
        fh.write(struct.pack('@i', 10).decode('utf-8'))
        fh.seek(0)
        with pytest.raises(AssertionError):
            mesh.Mesh.from_file(str(tmp_file), fh=fh)


def test_corrupt_binary_file(tmpdir):
    tmp_file = tmpdir.join('tmp.stl')
    with tmp_file.open('w+') as fh:
        fh.write('#########\n' * 8)
        fh.write('#\0\0\0')
        fh.seek(0)
        mesh.Mesh.from_file(str(tmp_file), fh=fh)

    with tmp_file.open('w+') as fh:
        fh.write('#########\n' * 9)
        fh.seek(0)
        with pytest.raises(AssertionError):
            mesh.Mesh.from_file(str(tmp_file), fh=fh)

    with tmp_file.open('w+') as fh:
        fh.write('#########\n' * 8)
        fh.write('#\0\0\0')
        fh.seek(0)
        fh.write('solid test.stl')
        fh.seek(0)
        mesh.Mesh.from_file(str(tmp_file), fh=fh)
```

3.2 tests.test_commandline module

```
import sys

from stl import main

ascii_file = 'tests/stl_ascii/HalfDonut.stl'
binary_file = 'tests/stl_binary/HalfDonut.stl'


def test_main(tmpdir):
    original_argv = sys.argv[:]
```

```
    try:
        sys.argv[:] = ['stl', ascii_file, str(tmpdir.join('binary.stl'))]
        main.main()
        sys.argv[:] = ['stl', '-r', ascii_file, str(tmpdir.join('binary.stl'))]
        main.main()
        sys.argv[:] = ['stl', '-a', binary_file, str(tmpdir.join('ascii.stl'))]
        main.main()
        sys.argv[:] = ['stl', '-b', ascii_file, str(tmpdir.join('binary.stl'))]
        main.main()
    finally:
        sys.argv[:] = original_argv


def test_args(tmpdir):
    parser = main._get_parser('')

    def _get_name(*args):
        return main._get_name(parser.parse_args(list(map(str, args))))

    assert _get_name('--name', 'foobar') == 'foobar'
    assert _get_name('-', tmpdir.join('binary.stl')).endswith('binary.stl')
    assert _get_name(ascii_file, '-').endswith('HalfDonut.stl')
    assert _get_name('-', '-')


def test_ascii(tmpdir):
    original_argv = sys.argv[:]
    try:
        print(str(tmpdir.join('ascii.stl')))
        sys.argv[:] = ['stl', binary_file, str(tmpdir.join('ascii.stl'))]
        try:
            main.to_ascii()
        except SystemExit:
            pass
    finally:
        sys.argv[:] = original_argv


def test_binary(tmpdir):
    original_argv = sys.argv[:]
    try:
        sys.argv[:] = ['stl', ascii_file, str(tmpdir.join('binary.stl'))]
        try:
            main.to_binary()
        except SystemExit:
            pass
    finally:
        sys.argv[:] = original_argv
```

3.3 tests.test_convert module

```
import os
import pytest
import tempfile

from stl import stl
```

```
ascii_file = 'tests/stl_ascii/HalfDonut.stl'
binary_file = 'tests/stl_binary/HalfDonut.stl'


@pytest.fixture
def current_path():
    return os.path.dirname(os.path.abspath(__file__))


@pytest.fixture
def ascii_path(current_path):
    return os.path.join(current_path, 'stl_ascii')


@pytest.fixture
def binary_path(current_path):
    return os.path.join(current_path, 'stl_binary')


def _test_conversion(from_, to, mode):
    for name in os.listdir(from_):
        source_file = os.path.join(from_, name)
        expected_file = os.path.join(to, name)

        mesh = stl.StlMesh(source_file)
        with open(expected_file, 'rb') as expected_fh:
            expected = expected_fh.read()
            # For binary files, skip the header
            if mode is stl.BINARY:
                expected = expected[80:]

            with tempfile.TemporaryFile() as dest_fh:
                mesh.save(name, dest_fh, mode)
                # Go back to the beginning to read
                dest_fh.seek(0)
                dest = dest_fh.read()
                # For binary files, skip the header
                if mode is stl.BINARY:
                    dest = dest[80:]

                assert dest.strip() == expected.strip()


def test_ascii_to_binary(ascii_path, binary_path):
    _test_conversion(ascii_path, binary_path, mode=stl.BINARY)


def test_binary_to_ascii(ascii_path, binary_path):
    _test_conversion(binary_path, ascii_path, mode=stl.ASCII)


def test_stl_mesh(tmpdir):
    tmp_file = tmpdir.join('tmp.stl')

    mesh = stl.StlMesh(ascii_file)
    with pytest.raises(ValueError):
        mesh.save(filename=str(tmp_file), mode='test')
```

```
    mesh.save(str(tmp_file))
    mesh.save(str(tmp_file), update_normals=False)
```

3.4 tests.test_mesh module

```
import numpy

from stl.mesh import Mesh
from stl.base import RemoveDuplicates


def test_units_1d():
    data = numpy.zeros(1, dtype=Mesh.dtype)
    data['vectors'][0] = numpy.array([[0, 0, 0],
                                      [1, 0, 0],
                                      [2, 0, 0]])

    mesh = Mesh(data, remove_empty_areas=False)
    mesh.update_units()

    assert mesh.areas == 0
    assert (mesh.normals == [0, 0, 0]).all()
    assert (mesh.units == [0, 0, 0]).all()


def test_units_2d():
    data = numpy.zeros(2, dtype=Mesh.dtype)
    data['vectors'][0] = numpy.array([[0, 0, 0],
                                      [1, 0, 0],
                                      [0, 1, 0]])
    data['vectors'][1] = numpy.array([[1, 0, 0],
                                      [0, 1, 0],
                                      [1, 1, 0]])

    mesh = Mesh(data, remove_empty_areas=False)
    mesh.update_units()

    assert (mesh.areas == [.5, .5]).all()
    assert (mesh.normals == [[0, 0, 1.],
                             [0, 0, -1.]]).all()

    assert (mesh.units == [[0, 0, 1],
                           [0, 0, -1]]).all()


def test_units_3d():
    data = numpy.zeros(1, dtype=Mesh.dtype)
    data['vectors'][0] = numpy.array([[0, 0, 0],
                                      [1, 0, 0],
                                      [0, 1, 1.]])

    mesh = Mesh(data, remove_empty_areas=False)
    mesh.update_units()

    assert (mesh.areas - 2 ** .5) < 0.0001
    assert (mesh.normals == [0, -1, 1]).all()
```

```
    units = mesh.units[0]
    assert units[0] == 0
    # Due to floating point errors
    assert (units[1] + .5 * 2 ** .5) < 0.0001
    assert (units[2] - .5 * 2 ** .5) < 0.0001


def test_duplicate_polygons():
    data = numpy.zeros(6, dtype=Mesh.dtype)
    data['vectors'][0] = numpy.array([[1, 0, 0],
                                      [0, 0, 0],
                                      [0, 0, 0]])
    data['vectors'][1] = numpy.array([[2, 0, 0],
                                      [0, 0, 0],
                                      [0, 0, 0]])
    data['vectors'][2] = numpy.array([[0, 0, 0],
                                      [0, 0, 0],
                                      [0, 0, 0]])
    data['vectors'][3] = numpy.array([[2, 0, 0],
                                      [0, 0, 0],
                                      [0, 0, 0]])
    data['vectors'][4] = numpy.array([[1, 0, 0],
                                      [0, 0, 0],
                                      [0, 0, 0]])
    data['vectors'][5] = numpy.array([[0, 0, 0],
                                      [0, 0, 0],
                                      [0, 0, 0]])

    mesh = Mesh(data)
    assert mesh.data.size == 6

    mesh = Mesh(data, remove_duplicate_polygons=0)
    assert mesh.data.size == 6

    mesh = Mesh(data, remove_duplicate_polygons=False)
    assert mesh.data.size == 6

    mesh = Mesh(data, remove_duplicate_polygons=None)
    assert mesh.data.size == 6

    mesh = Mesh(data, remove_duplicate_polygons=RemoveDuplicates.NONE)
    assert mesh.data.size == 6

    mesh = Mesh(data, remove_duplicate_polygons=RemoveDuplicates.SINGLE)
    assert mesh.data.size == 3

    mesh = Mesh(data, remove_duplicate_polygons=True)
    assert mesh.data.size == 3

    assert (mesh.vectors[0] == numpy.array([[1, 0, 0],
                                            [0, 0, 0],
                                            [0, 0, 0]])).all()
    assert (mesh.vectors[1] == numpy.array([[2, 0, 0],
                                            [0, 0, 0],
                                            [0, 0, 0]])).all()
    assert (mesh.vectors[2] == numpy.array([[0, 0, 0],
                                            [0, 0, 0],
                                            [0, 0, 0]])).all()
```

```
    mesh = Mesh(data, remove_duplicate_polygons=RemoveDuplicates.ALL)
    assert mesh.data.size == 3

    assert (mesh.vectors[0] == numpy.array([[1, 0, 0],
                                            [0, 0, 0],
                                            [0, 0, 0]])).all()
    assert (mesh.vectors[1] == numpy.array([[2, 0, 0],
                                            [0, 0, 0],
                                            [0, 0, 0]])).all()
    assert (mesh.vectors[2] == numpy.array([[0, 0, 0],
                                            [0, 0, 0],
                                            [0, 0, 0]])).all()


def test_remove_all_duplicate_polygons():
    data = numpy.zeros(5, dtype=Mesh.dtype)
    data['vectors'][0] = numpy.array([[0, 0, 0],
                                      [0, 0, 0],
                                      [0, 0, 0]])
    data['vectors'][1] = numpy.array([[1, 0, 0],
                                      [0, 0, 0],
                                      [0, 0, 0]])
    data['vectors'][2] = numpy.array([[2, 0, 0],
                                      [0, 0, 0],
                                      [0, 0, 0]])
    data['vectors'][3] = numpy.array([[3, 0, 0],
                                      [0, 0, 0],
                                      [0, 0, 0]])
    data['vectors'][4] = numpy.array([[3, 0, 0],
                                      [0, 0, 0],
                                      [0, 0, 0]])

    mesh = Mesh(data, remove_duplicate_polygons=False)
    assert mesh.data.size == 5
    Mesh.remove_duplicate_polygons(mesh.data, RemoveDuplicates.NONE)

    mesh = Mesh(data, remove_duplicate_polygons=RemoveDuplicates.ALL)
    assert mesh.data.size == 3

    assert (mesh.vectors[0] == numpy.array([[0, 0, 0],
                                            [0, 0, 0],
                                            [0, 0, 0]])).all()
    assert (mesh.vectors[1] == numpy.array([[1, 0, 0],
                                            [0, 0, 0],
                                            [0, 0, 0]])).all()
    assert (mesh.vectors[2] == numpy.array([[2, 0, 0],
                                            [0, 0, 0],
                                            [0, 0, 0]])).all()


def test_empty_areas():
    data = numpy.zeros(3, dtype=Mesh.dtype)
    data['vectors'][0] = numpy.array([[0, 0, 0],
                                      [1, 0, 0],
                                      [0, 1, 0]])
    data['vectors'][1] = numpy.array([[1, 0, 0],
                                      [0, 1, 0],
                                      [1, 0, 0]])
```

```
    data['vectors'][2] = numpy.array([[1, 0, 0],
                                      [0, 1, 0],
                                      [1, 0, 0]])

    mesh = Mesh(data, remove_empty_areas=False)
    assert mesh.data.size == 3

    mesh = Mesh(data, remove_empty_areas=True)
    assert mesh.data.size == 1
```

3.5 tests.test_multiple module

```
from stl import mesh

_STL_FILE = '''
solid test.stl
facet normal -0.014565 0.073223 -0.002897
  outer loop
    vertex 0.399344 0.461940 1.044090
    vertex 0.500000 0.500000 1.500000
    vertex 0.576120 0.500000 1.117320
  endloop
endfacet
endsolid test.stl
'''.lstrip()


def test_single_stl(tmpdir):
    tmp_file = tmpdir.join('tmp.stl')
    with tmp_file.open('w+') as fh:
        fh.write(_STL_FILE)
        fh.seek(0)
        for m in mesh.Mesh.from_multi_file(str(tmp_file), fh=fh):
            pass


def test_multiple_stl(tmpdir):
    tmp_file = tmpdir.join('tmp.stl')
    with tmp_file.open('w+') as fh:
        fh.write(_STL_FILE)
        fh.write(_STL_FILE)
        fh.seek(0)
        for m in mesh.Mesh.from_multi_file(str(tmp_file), fh=fh):
            pass


def test_single_stl_file(tmpdir):
    tmp_file = tmpdir.join('tmp.stl')
    with tmp_file.open('w+') as fh:
        fh.write(_STL_FILE)
        fh.seek(0)
        for m in mesh.Mesh.from_multi_file(str(tmp_file)):
            pass
```

```
def test_multiple_stl_file(tmpdir):
    tmp_file = tmpdir.join('tmp.stl')
    with tmp_file.open('w+') as fh:
        fh.write(_STL_FILE)
        fh.write(_STL_FILE)
        fh.seek(0)
        for m in mesh.Mesh.from_multi_file(str(tmp_file)):
            pass
```

3.6 tests.test_rotate module

```
import math
import numpy

from stl.mesh import Mesh


def test_rotation():
    # Create 3 faces of a cube
    data = numpy.zeros(6, dtype=Mesh.dtype)

    # Top of the cube
    data['vectors'][0] = numpy.array([[0, 1, 1],
                                      [1, 0, 1],
                                      [0, 0, 1]])
    data['vectors'][1] = numpy.array([[1, 0, 1],
                                      [0, 1, 1],
                                      [1, 1, 1]])
    # Right face
    data['vectors'][2] = numpy.array([[1, 0, 0],
                                      [1, 0, 1],
                                      [1, 1, 0]])
    data['vectors'][3] = numpy.array([[1, 1, 1],
                                      [1, 0, 1],
                                      [1, 1, 0]])
    # Left face
    data['vectors'][4] = numpy.array([[0, 0, 0],
                                      [1, 0, 0],
                                      [1, 0, 1]])
    data['vectors'][5] = numpy.array([[0, 0, 0],
                                      [0, 0, 1],
                                      [1, 0, 1]])

    mesh = Mesh(data, remove_empty_areas=False)

    # Since the cube faces are from 0 to 1 we can move it to the middle by
    # substracting .5
    data['vectors'] -= .5

    # Rotate 90 degrees over the X axis followed by the Y axis followed by the
    # X axis
    mesh.rotate([0.5, 0.0, 0.0], math.radians(90))
    mesh.rotate([0.0, 0.5, 0.0], math.radians(90))
    mesh.rotate([0.5, 0.0, 0.0], math.radians(90))
```

```
    # Since the cube faces are from 0 to 1 we can move it to the middle by
    # substracting .5
    data['vectors'] += .5

    assert (mesh.vectors == numpy.array([
        [[1, 0, 0], [0, 1, 0], [0, 0, 0]],
        [[0, 1, 0], [1, 0, 0], [1, 1, 0]],
        [[0, 1, 1], [0, 1, 0], [1, 1, 1]],
        [[1, 1, 0], [0, 1, 0], [1, 1, 1]],
        [[0, 0, 1], [0, 1, 1], [0, 1, 0]],
        [[0, 0, 1], [0, 0, 0], [0, 1, 0]],
    ])).all()


def test_rotation_over_point():
    # Create 3 faces of a cube
    data = numpy.zeros(1, dtype=Mesh.dtype)

    data['vectors'][0] = numpy.array([[1, 0, 0],
                                      [0, 1, 0],
                                      [0, 0, 1]])

    mesh = Mesh(data, remove_empty_areas=False)

    mesh.rotate([1, 0, 0], math.radians(180), point=[1, 2, 3])
    assert (mesh.vectors == numpy.array([[1, -4, -6],
                                         [0, -5, -6],
                                         [0, -4, -7]])).all()


def test_no_rotation():
    # Create 3 faces of a cube
    data = numpy.zeros(3, dtype=Mesh.dtype)

    # Top of the cube
    data['vectors'][0] = numpy.array([[0, 1, 1],
                                      [1, 0, 1],
                                      [0, 0, 1]])

    mesh = Mesh(data, remove_empty_areas=False)

    # Rotate by 0 degrees
    mesh.rotate([0.5, 0.0, 0.0], math.radians(0))

    # Use a zero rotation matrix
    mesh.rotate([0.0, 0.0, 0.0], math.radians(90))
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

S

Z