AFEM Documentation

Release 1.0.0

Laughlin Research

Oct 21, 2018

CONTENTS

This documentation is for the Airframe Finite Element Modeler (AFEM) program developed by Laughlin Research, LLC.

CHAPTER ONE

ABOUT

AFEM is a “fit-for-purpose” engineering development toolkit designed to support the use of high-order structural analysis during the early phases of aircraft conceptual design. As a development toolkit, it provides the engineer with a flexible, modular, and extensible library of components and tools to rapidly build a useful structural model. It is **not** an end-user GUI application, but rather a library enabling engineers to rapidly build their own application specific tools and processes, encoding their own design rules and best practices along the way.

Although AFEM targets airframe design and analysis applications, really only the `afem.structure` and `afem.oml` packages directly support that cause. All the other packages provide a more general and “Pythonic” set of entities and tools that could potentially be used to develop applications in other disciplines and/or domains.

CHAPTER

TWO

TECHNOLOGY STACK

The AFEM core technology stack includes:

- Python[1]: The Python programming language enables rapid development and integration with other systems.
- OpenCASCADE[2]: This mature library provides advanced geometric modeling and CAD functionality and is under active development.
- Netgen[3]: This library enables advanced meshing capabilities including 3-D tetrahedral and 2-D unstructured quad-dominated surface meshing.
- Salome Platform[4]: The core meshing library from this open source application serves as the central component for AFEM's mesh generation capabilities.
- pyOCCT[5]: This open source project provides Python bindings to the OpenCASCADE and Salome Platform meshing libraries.

[1] https://www.python.org/
[2] https://www.opencascade.com
[3] https://sourceforge.net/projects/netgen-mesher
[4] http://www.salome-platform.org
[5] https://github.com/LaughlinResearch/pyOCCT

CHAPTER THREE

INSTALLATION

This section explains how to install AFEM and required dependencies.

Currently, only Python 3.5 and 3.6 Windows 64-bit is supported. Although AFEM is Python 2/3 compatible, some core dependencies have thus far only been built for Python 3.5 and 3.6 Windows 64-bit. More platforms will be supported in the future.

3.1 Installing Python

For new users, installing Anaconda[6] is **highly recommended**. Anaconda conveniently installs Python and many other commonly used packages for scientific computing. This installation documentation assumes the use of Anaconda Python. Otherwise, the official Python[7] distribution can be used.

Many of the packages required by AFEM will be included in the default Anaconda installation including NumPy and SciPy. See here[8] for a complete list and more information about the Anaconda installation.

A specific environment for AFEM should be created using an Anaconda Command Prompt:

```
conda create -n afem python=3.5
```

The `-n` flag signifies the name and in this example it is *afem*, but it can be anything. The `python=3.5` command tells Anaconda to create an environment based on Python 3.5.

Make sure this environment is active when installing and using AFEM. For Anaconda Python, activating this environment looks like:

```
activate afem
```

within an Anaconda Command Prompt.

3.2 Installing Dependencies

The pyOCCT[9] package developed by Laughlin Research should now be installed. For supported platforms, installing pyOCCT can be done by:

```
conda install -c laughlinresearch pyocct
```

[6] https://www.continuum.io/downloads
[7] https://www.python.org/downloads/
[8] https://docs.continuum.io/anaconda/pkg-docs
[9] https://github.com/LaughlinResearch/pyOCCT

Other dependencies such as NumPy and SciPy can be installed as needed using the conda package manager:

```
conda install numpy scipy
```

or pip:

```
pip install numpy scipy
```

A minimal graphical user interface requires the wxPython package which can be installed by:

```
conda install -c conda-forge wxpython
```

or with pip:

```
pip install wxpython
```

3.3 Installing AFEM

Be sure to activate the designed AFEM environment before installation. AFEM is a pure Python package and can be installed using the command:

```
python setup.py develop
```

within the AFEM root folder. The `develop` option links to the source code at runtime so changes in the source are reflected in any programs using AFEM. The regular installation command:

```
python setup.py install
```

can be used to actually install the AFEM package into the Python *site-packages* folder.

Installation files can be cleaned by:

```
conda clean -a
```

3.4 Building Documentation

The documentation can be built from sources using sphinx. Install sphinx and sphinx_rtd_theme in the desired conda environment by:

```
conda install sphinx sphinx_rtd_theme
```

Then navigate to the *docs/* folder and run:

```
make html
```

This should build html documentation in a *docs/build/html* folder. Open the index.html file with a web browser.

CHAPTER

FOUR

SETTINGS

Default settings for the AFEM program are provided in the `Settings` (page 9) class and accessed as class variables. These settings can be adjusted during use and are passed to many of the classes and methods. The `Settings` (page 9) class can be accessed by:

```
from afem.config import Settings
```

A logging utility is used to provide useful information during program execution. A file with the name *afem.log* will be automatically created wherever the main script is executed and whose contents will be dependent on the logging level. In order to output the logging content to the command window the following method should be called before the main script begins:

```
Settings.log_to_console()
```

The logging content will now be displayed in the command window as well as output to the log file.

Perhaps the setting with the most implication is what units are set for OpenCASCADE. This is especially critical during data exchange activities (e.g., STEP translation). The units can be set using the class method:

```
Settings.set_units('in')
```

Inches ('in'), feet ('ft'), meters ('m'), and millimeters ('mm') are available. OpenCASCADE uses millimeters by default, but AFEM should use inches as its default setting when units are relevant.

class `afem.config.`**`Settings`**

Bases: `object`

Settings.

Variables **`units`** (`str`) – The default units ('in', 'ft', 'm', 'mm'). The default value is inches ('INCH').

classmethod set_units(*units='in'*)

Set units.

Parameters **`units`** (`str`) – The units ('in', 'ft', 'm', 'mm').

Returns None.

Raises **`KeyError`** – If the given units are not supported.

static log_to_console()

Option to add a stream handler to the main file logger to log to stdout.

Returns None.

static set_loggging_level(*level='info'*)

Set the logging level.

Parameters `level` (*str*) – Logging level ('debug', 'info', 'warning', 'error', 'critical').

Returns None.

Raises `KeyError` – If the given level is not supported.

CHAPTER

FIVE

BASE

The `afem.base` package is a collection of simple entities that define base classes that are used throughout the library.

5.1 Entities

5.1.1 NamedItem

class `afem.base.entities.`**`NamedItem`**(*name='Item'*)

Bases: `object`

Base class for types that can be give a name.

Parameters **`name`** (*`str`*) – The name.

`name`

Getter The name.

Setter Set name.

Type str

`metadata`

Returns The metadata dictionary.

Return type afem.base.entities.Metadata

`set_name`(*name*)

Set name.

Parameters **`name`** (*`str`*) – The name.

Returns None.

5.1.2 ViewableItem

class `afem.base.entities.`**`ViewableItem`**

Bases: `object`

Base class for types that can be viewed.

`displayed_shape`

Returns The shape to be displayed.

Return type OCCT.TopoDS.TopoDS_Shape

`color`

Returns The color or *None* if not set.

Return type OCCT.Quantity.Quantity_Color or None

`transparency`

Returns The transparency.

Return type float

`set_color`(*r*, *g*, *b*)

Set color (0. <= r, g, b <= 1.).

Parameters

- **`r`** (*`float`*) – Red.
- **`g`** (*`float`*) – Green.
- **`b`** (*`float`*) – Blue.

Returns None.

`set_transparency`(*transparency*)

Set the opacity for graphics.

Parameters **`transparency`** (*`float`*) – Level of transparency (0 to 1).

Returns None.

`random_color`()

Set the color randomly.

Returns None.

CHAPTER

SIX

CORE

The `afem.core` package includes the *ShapeHolder* (page 13) class which is a base type meant to hold convenient properties and methods for the *Body* (page 228) and *Part* (page 253) types. A reference curve and surface can be set in this class as needed by the user and a number of streamlined methods are available for creating additional reference geometry. Included are methods to create points and planes along the reference curve as well as projecting points to either the reference curve or surface.

6.1 Entities

6.1.1 ShapeHolder

class afem.core.entities.**ShapeHolder**(*name*, *shape*, *cref=None*, *sref=None*, *expected_types=(<class 'afem.topology.entities.Shape'>,)*)

Bases: *afem.base.entities.NamedItem* (page 11), *afem.base.entities.ViewableItem* (page 11)

Core class that holds a shape plus reference geometry and common methods.

Parameters

- **name** (*str*) – The name.
- **cref** (afem.geometry.entities.Curve (page 43) *or None*) – The reference curve. If it is not a *TrimmedCurve* (page 48), then it will be converted to one.
- **sref** (afem.geometry.entities.Surface (page 49) *or None*) – The reference surface.
- **expected_types** (*Type(*afem.topology.entities.Shape (page 109)*) or collections.Sequence(Type(*afem.topology.entities.Shape (page 109)*)))* – The expected type(s).

type_name

Returns The class name.

Return type str

shape

Getter The shape.

Setter Set the shape.

Type *afem.topology.entities.Shape* (page 109)

displayed_shape

Returns The shape to be displayed.

Return type OCCT.TopoDS.TopoDS_Shape

cref

Getter The reference curve.

Setter Set the reference curve.

Type *afem.geometry.entities.TrimmedCurve* (page 48)

has_cref

Returns *True* if reference curve is available, *False* if not.

Return type bool

sref

Getter The reference surface.

Setter Set the reference surface.

Type *afem.geometry.entities.Surface* (page 49)

has_sref

Returns *True* if reference surface is available, *False* if not.

Return type bool

plane

Returns The reference surface if it is a plane.

Return type *afem.geometry.entities.Plane* (page 52)

Raises **TypeError** – If the reference surface is not a plane.

sref_shape

Returns The reference shape. This should be the same as the reference surface except it is processed (e.g., dividing closed surfaces and C0 boundaries) for better robustness in some operations.

Return type *afem.topology.entities.Shape* (page 109)

edge_compound

Returns A compound containing the edges.

Return type *afem.topology.entities.Compound* (page 119)

face_compound

Returns A compound containing the faces.

Return type *afem.topology.entities.Compound* (page 119)

set_shape(*shape*)

Set the shape.

Parameters **shape** (`afem.topology.entities.Shape` (page 109)) – The shape.

Returns None.

set_cref(*cref*)
Set the reference curve.

Parameters

- **cref** (`afem.geometry.entities.Curve` (page 43)) – The curve. If it is not a *`TrimmedCurve`* (page 48), then it will be converted to one. Access the original curve using the *basis_curve* property (i.e., part.cref.basis_curve).
- **cref** – The reference curve.

Returns None.

Raises **`TypeError`** – If *cref* is not a curve.

set_sref(*sref*)
Set the reference surface. This method also automatically creates a "reference surface shape" by converting the given surface into a face, dividing it if it is closed, and then dividing it at C0 boundaries. This shape is made available to improve robustness for certain topological operations and can be accessed via the *sref_shape* property.

Parameters **sref** (`afem.geometry.entities.Surface` (page 49)) – The surface.

Returns None.

Raises **`TypeError`** – If *sref* is not a surface.

set_u1(*u1*)
Set the first parameter of the reference curve.

Parameters **u1** (*`float`*) – The parameter.

Returns None.

Raises **`ValueError`** – If the *u1* is greater than or equal to *u2*.

set_u2(*u2*)
Set the last parameter of the reference curve.

Parameters **u2** (*`float`*) – The parameter.

Returns None.

Raises **`ValueError`** – If the *u2* is less than or equal to *u1*.

set_p1(*p1*)
Set the first parameter of the reference curve by inverting the point.

Parameters **p1** (*`point_like`*) – The point.

Returns None.

Raises **`RuntimeError`** – If no projection can be found.

set_p2(*p2*)
Set the last parameter of the reference curve by inverting the point.

Parameters **p2** (*`point_like`*) – The point.

Returns None.

Raises **`RuntimeError`** – If no projection can be found.

trim_u1(*entity*)
Trim the first parameter of the reference curve by interesting it with the entity.

Parameters **entity** (afem.geometry.entities.Geometry (page 34) *or* afem.topology.entities.Shape (page 109)) – The entity.

Returns None.

Raises **RuntimeError** – If an intersection with the reference curve cannot be found.

trim_u2 (*entity*)
Trim the last parameter of the reference curve by interesting it with the entity.

Parameters **entity** (afem.geometry.entities.Geometry (page 34) *or* afem.topology.entities.Shape (page 109)) – The entity.

Returns None.

Raises **RuntimeError** – If an intersection with the reference curve cannot be found.

point_from_parameter (*ds*, *u0=None*, *is_rel=False*)
Evaluate point on reference curve at a distance from a parameter.

Parameters

- **ds** (*float*) – The distance.
- **u0** (*float*) – The parameter. If not provided the first parameter of the reference curve will be used.
- **is_rel** (*bool*) – Option specifying if the distance is absolute or a relative to the length of the reference curve. If relative, then *ds* is multiplied by the curve length to get the absolute value for the *PointFromParameter* (page 58) method.

Returns The point.

Return type *afem.geometry.entities.Point* (page 35)

points_by_number (*n*, *d1=None*, *d2=None*, *shape1=None*, *shape2=None*)
Create a specified number of points along the reference curve.

Parameters

- **n** (*int*) – Number of points to create ($n > 0$).
- **d1** (*float*) – An offset distance for the first point. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last point. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **shape1** (afem.topology.entities.Shape (page 109)) – A shape to define the first point. This shape is intersected with the edge or wire.
- **shape2** (afem.topology.entities.Shape (page 109)) – A shape to define the last point. This shape is intersected with the edge or wire.

Returns The points.

Return type list(*afem.geometry.entities.Point* (page 35))

points_by_distance (*maxd*, *nmin=0*, *d1=None*, *d2=None*, *shape1=None*, *shape2=None*)
Create a points along the reference curve by distance.

Parameters

- **maxd** (*float*) – The maximum allowed spacing between points. The actual spacing will be adjusted to not to exceed this value.
- **nmin** (*int*) – Minimum number of points to create.

- **d1** (*float*) – An offset distance for the first point. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last point. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **shape1** (afem.topology.entities.Shape (page 109)) – A shape to define the first point. This shape is intersected with the edge or wire.
- **shape2** (afem.topology.entities.Shape (page 109)) – A shape to define the last point. This shape is intersected with the edge or wire.

Returns The points.

Return type list(*afem.geometry.entities.Point* (page 35))

point_to_cref(*pnt*, *direction=None*)
Project a point to reference curve.

Parameters

- **pnt** (afem.geometry.entities.Point (page 35)) – The point. Position will be updated.
- **direction** (*vector_like*) – Projection direction.

Returns *True* if projected, *False* if not.

Return type bool

points_to_cref(*pnts*, *direction=None*)
Project points to the reference curve.

Parameters

- **pnts** (*list (*afem.geometry.entities.Point (page 35)*)*) – The points. Position will be updated.
- **direction** (*vector_like*) – Projection direction.

Returns List of status for each point.

Return type list(bool)

point_to_sref(*pnt*, *direction=None*)
Project a point to reference surface.

Parameters

- **pnt** (afem.geometry.entities.Point (page 35)) – The point. Position will be updated.
- **direction** (*vector_like*) – Projection direction.

Returns *True* if projected, *False* if not.

Return type bool

points_to_sref(*pnts*, *direction=None*)
Project points to reference surface.

Parameters

- **pnts** (*list (*afem.geometry.entities.Point (page 35)*)*) – The points. Position will be updated.
- **direction** (*vector_like*) – Projection direction.

Returns List of status for each point.

Return type list(bool)

plane_from_parameter(*ds*, *u0=None*, *is_rel=False*, *ref_pln=None*, *tol=1e-07*)
Get a plane along the reference curve.

Parameters

- **ds** (*float*) – The distance.
- **u0** (*float*) – The parameter. If not provided the first parameter of the reference curve will be used.
- **is_rel** (*bool*) – Option specifying if the distance is absolute or a relative to the length of the reference curve. If relative, then *ds* is multiplied by the curve length to get the absolute value for the *PlaneFromParameter* (page 69) method.
- **ref_pln** (afem.geometry.entities.Plane (page 52)) – The normal of this plane will be used to define the normal of the new plane. If no plane is provided, then the first derivative of the curve will define the plane normal.
- **tol** (*float*) – Tolerance.

Returns The plane.

Return type *afem.geometry.entities.Plane* (page 52)

planes_by_number(*n*, *ref_pln=None*, *d1=None*, *d2=None*, *shape1=None*, *shape2=None*)
Create a specified number of planes along the reference curve.

Parameters

- **n** (*int*) – Number of points to create ($n > 0$).
- **ref_pln** (afem.geometry.entities.Plane (page 52)) – The normal of this plane will be used to define the normal of all planes along the curve. If no plane is provided, then the first derivative of the curve will define the plane normal.
- **d1** (*float*) – An offset distance for the first point. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last point. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **shape1** (afem.topology.entities.Shape (page 109)) – A shape to define the first point. This shape is intersected with the reference curve.
- **shape2** (afem.topology.entities.Shape (page 109)) – A shape to define the last point. This shape is intersected with the reference curve.

Returns List of planes along the curve.

Return type list(*afem.geometry.entities.Plane* (page 52))

Raises

- **TypeError** – If *shape* if not an edge or wire.
- **RuntimeError** – If OCC method fails.

planes_by_distance(*maxd*, *ref_pln=None*, *d1=None*, *d2=None*, *shape1=None*, *shape2=None*, *nmin=0*)
Create planes along the reference curve by distance between them.

Parameters

- **maxd** (*float*) – The maximum allowed spacing between planes. The actual spacing will be adjusted to not to exceed this value.
- **ref_pln** (`afem.geometry.entities.Plane` (page 52)) – The normal of this plane will be used to define the normal of all planes along the curve. If no plane is provided, then the first derivative of the curve will define the plane normal.
- **d1** (*float*) – An offset distance for the first point. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last point. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **shape1** (`afem.topology.entities.Shape` (page 109)) – A shape to define the first point. This shape is intersected with the reference curve.
- **shape2** (`afem.topology.entities.Shape` (page 109)) – A shape to define the last point. This shape is intersected with the reference curve.
- **nmin** (*int*) – Minimum number of planes to create.

Returns List of planes along the curve.

Return type list(*afem.geometry.entities.Plane* (page 52))

Raises

- **TypeError** – If *shape* if not an edge or wire.
- **RuntimeError** – If OCC method fails.

make_shell()

Attempt to make a shell from the faces of the part. This method only constructs the faces into a shell without checking for a valid shape.

Returns A new shell from the shape of the part.

Return type *afem.topology.entities.Shell* (page 117)

bbox(*tol=None*)

Return a bounding box of the body.

Parameters **tol** (*float or None*) – Optional tolerance to enlarge the bounding box.

Returns Bounding box of the body.

Return type *afem.topology.entities.BBox* (page 119)

extract_plane(*u1*, *v1*, *u2*, *v2*)

Extract a plane between parameters on the reference surface. The plane will be defined by three points. The first point is at (u1, v1), the second point is at (u2, v2), and the third point will be offset from the first point in the direction of the reference surface normal. The points should not be collinear.

Parameters

- **u1** (*float*) – First u-parameter.
- **v1** (*float*) – First v-parameter.
- **u2** (*float*) – Second u-parameter.
- **v2** (*float*) – Second v-parameter.

Returns The plane.

Return type *afem.geometry.entities.Plane* (page 52)

extract_curve(*u1*, *v1*, *u2*, *v2*, *basis_shape=None*)

Extract a trimmed curve within the reference surface between the parameters.

Parameters

- **u1** (*float*) – First u-parameter.
- **v1** (*float*) – First v-parameter.
- **u2** (*float*) – Second u-parameter.
- **v2** (*float*) – Second v-parameter.
- **basis_shape** (afem.geometry.entities.Surface (page 49) *or* afem.topology.entities.Shape (page 109)) – The shape that will be used to intersect with the reference shape. If not provided a plane will be created using the *extract_plane()* method. The parameters should create points that are on or very near the intersection between these two shapes. If they are not they will be projected to the intersection which could yield unanticipated results.

Returns The curve.

Return type *afem.geometry.entities.TrimmedCurve* (page 48)

Raises **RuntimeError** – If method fails.

CHAPTER

SEVEN

GEOMETRY

The `afem.geometry` package provides entities and tools for the creation and use of what is commonly referred to as "construction geometry" or "reference geometry" for both 2-D and 3-D domains. This package primarily wraps a number of OpenCASCADE native types and tools in order to provide a more "Pythonic" user interface. The entities and tools in the `geometry` package do not cover every OpenCASCADE type, but rather those frequently encountered during regular use. The entities and tools can be imported by:

```
from afem.geometry import *
```

Geometry entities cover general types like points, curves, planes, and surfaces. Tools exists for the creation, modification, intersection, and projection of the geometric entities. The simple example below demonstrates some of the key entities and tools of the `geometry` package:

```
from afem.geometry import *
from afem.graphics import Viewer

gui = Viewer()

# Create a point directly from the entity. Default is (0, 0, 0).
p1 = Point()

# Create a point by array-like
p2 = PointByArray([5, 0, 5]).point

# Create a point by x-, y-, and z-coordinates.
p3 = PointByXYZ(10, 0, 0).point

# Interpolate the points with a curve
c1 = NurbsCurveByInterp([p1, p2, p3]).curve

gui.add(p1, p2, p3, c1)
gui.start()

# Copy curve and translate
c2 = c1.copy()
c2.translate((0, 10, 0))

gui.add(c2)
gui.start()

# Copy and translate again
c3 = c2.copy()
c3.translate((0, 10, 10))
```

(continues on next page)

(continued from previous page)

```
gui.add(c3)
gui.start()

# Approximate a surface
s1 = NurbsSurfaceByApprox([c1, c2, c3]).surface

gui.add(s1)
gui.start()

# Extract an iso-curve
c4 = s1.u_iso(10.)

gui.add(c4)
gui.start()

# Create points along the curve
pnts = PointsAlongCurveByDistance(c4, 1.).points

gui.add(*pnts)
gui.start()

# Extract iso-curve
c5 = s1.v_iso(0.5)

gui.add(c5)
gui.start()

# Intersect two curves
cci = IntersectCurveCurve(c4, c5)

gui.clear()
gui.add(c4, c5, s1, *cci.points)
gui.start()

# Define a plane along a curve
pln = PlaneFromParameter(c4, 0., 2.).plane

# Intersect a surface and a plane
ssi = IntersectSurfaceSurface(s1, pln)

gui.add(s1, *ssi.curves)
gui.start()

# Project a point to a surface
p4 = pln.eval(5, 5)
proj = ProjectPointToSurface(p4, s1)
line = NurbsCurveByInterp([p4, proj.nearest_point]).curve

gui.add(p4, proj.nearest_point, line)
gui.start()
```

The entities, tool, and viewing tool are imported by:

```
from afem.geometry import *
from afem.graphics import Viewer
```

The first variable created is a *Point* (page 35) and is constructed directly from the entity itself. Since the *Point* (page 35) class is derived from the OpenCASCADE type `OCCT.gp.gp_Pnt`, one of the available constructor methods is using three floats to define an x-, y-, and z-coordinate. By default the location is (0., 0., 0.):

```
p1 = Point()
```

Points can also be created using a variety of tools including by an array or specified x-, y-, and z-coordinates. When tools are used the underlying entity must be retrieved from the tool by:

```
p2 = PointByArray([5, 0, 5]).point
p3 = PointByXYZ(10, 0, 0).point
```

A cubic curve can be created by interpolating points by:

```
c1 = NurbsCurveByInterp([p1, p2, p3]).curve
```

Most geometry types can be copied and a limited number of transformations can be directly applied to the entity depending on its type:

```
c2 = c1.copy()
c2.translate((0, 10, 0))
```

Here, a new *NurbsCurve* (page 47) was created and then translated in the y-direction by 10 units. This new curve is then copied again and translated in both the y- and z-directions by 10 units each:

```
c3 = c2.copy()
c3.translate((0, 10, 10))
```

These three curves are then used to approximate a *NurbsSurface* (page 53):

```
s1 = NurbsSurfaceByApprox([c1, c2, c3]).surface
```

Isoparametric curves (i.e., isocurve) can be extracted from surfaces in both the u- and v-directions. Below, an isocurve is extract at a constant u=10:

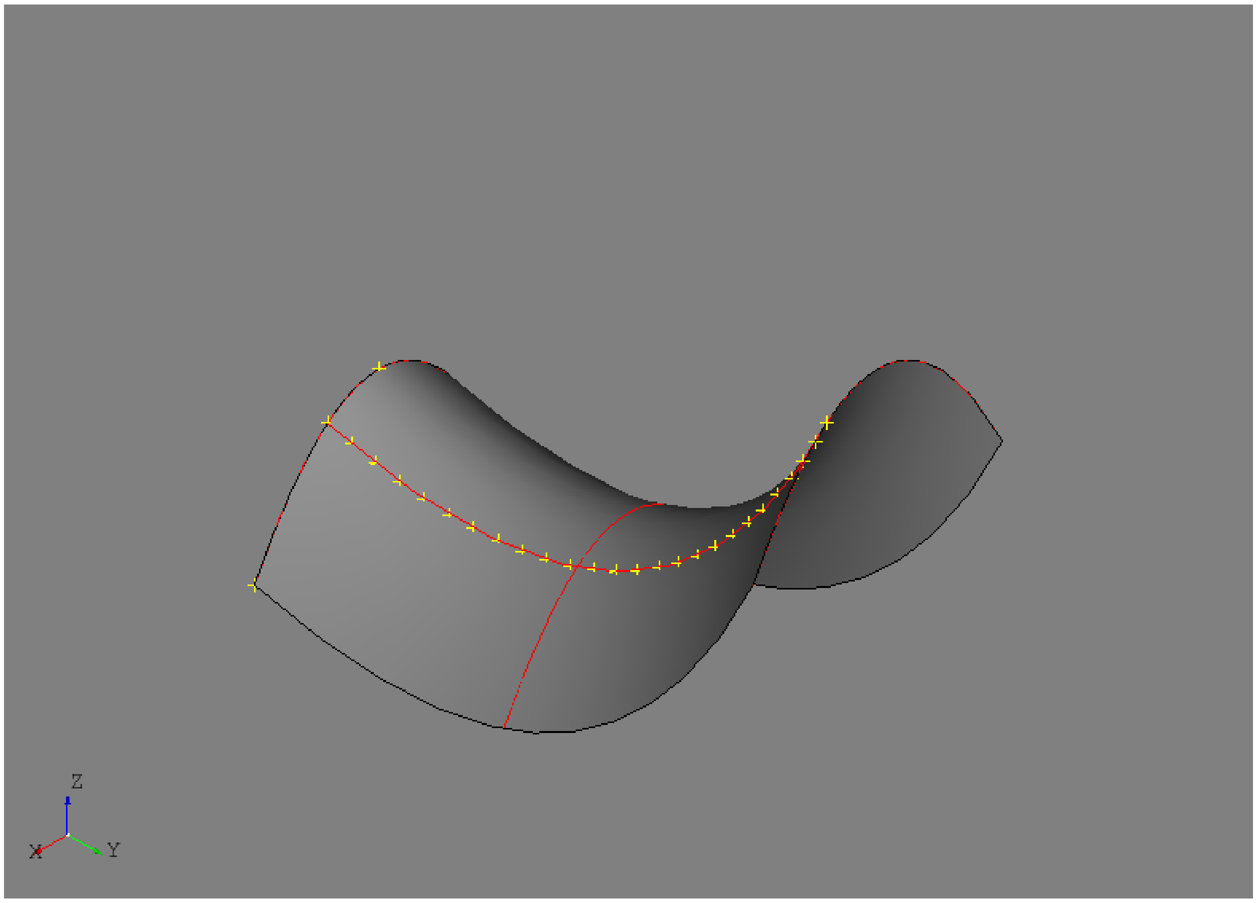
```
c4 = s1.u_iso(10.)
```

This isocurve is used to generate evenly spaced points using a target distance of 1 unit:

```
pnts = PointsAlongCurveByDistance(c4, 1.).points
```

The *PointsAlongCurveByDistance* (page 60) tool can also return the number of points created, the parameters on the curve for each point, the final spacing, and the interior points (i.e., exclude first and last). Others tools exist for creating points (and planes) along curves by a specified number rather than distance.

At this point the created geometry should look similar to the image below.



Another isocurve in the opposite direction is created and the intersection is found by:

```
c5 = s1.v_iso(0.5)
cci = IntersectCurveCurve(c4, c5)
```

The *IntersectCurveCurve* (page 80) tool provides intersection results including the point(s), parameter(s) on each curve, and number of intersections. Other tools exist for curve/surface and surface/surface intersections. Intersecting a plane and a surface is shown by:

```
pln = PlaneFromParameter(c4, 0., 2.).plane
ssi = IntersectSurfaceSurface(s1, pln)
```

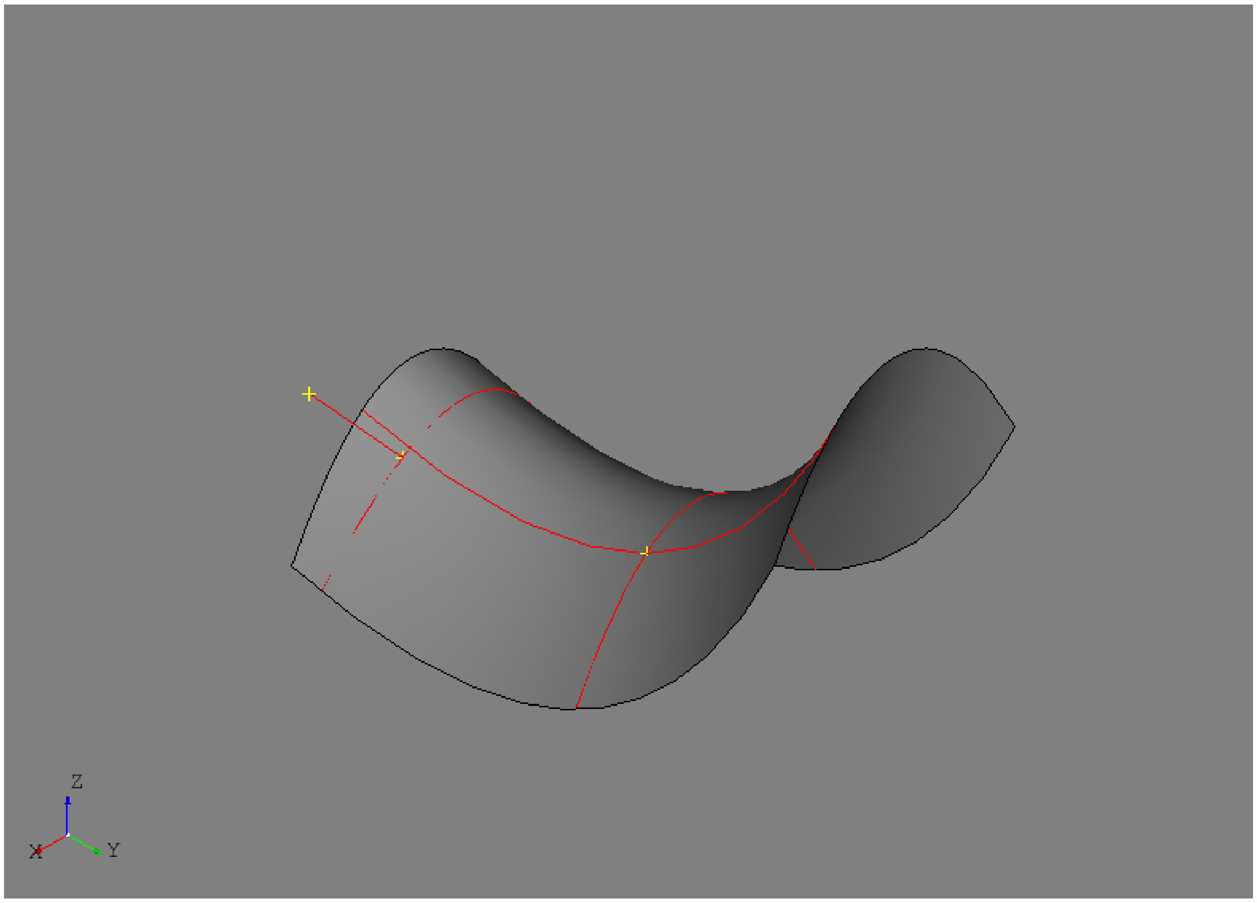
A *Plane* (page 52) is first created using the *PlaneFromParameter* (page 69) tool and then used in the *IntersectSurfaceSurface* (page 81) tool. The resulting intersection curves are approximated in 3-D space.

Projections to curves and surfaces are available using the projection tools. By default, projections are usually performed normal to the curve or surface, but some tools always projections along a specified direction. This operation actually becomes a curve intersection operation rather than a normal projection. The code below creates a point on a plane and then performs a normal projection to the surface:

```
p4 = pln.eval(5, 5)
proj = ProjectPointToSurface(p4, s1)
line = NurbsCurveByInterp([p4, proj.nearest_point]).curve
```

The `line` variable is created mostly for visualization purposes and to demonstrate some of the data that can be retrieved from the *ProjectPointToSurface* (page 78) tool. All point projection results are stored in the tool and sorted by minimum to maximum distance.

The intersection and projection results should look similar to the image below. Note that there are no renderings for infinite planes.



7.1 Entities

7.1.1 Geometry2D

class afem.geometry.entities.**Geometry2D**(*obj*)

Bases: *object* (page 25)

Base class for 2-D geometry.

Parameters **obj** (*OCCT.Geom2d.Geom2d_Geometry*) – The geometry object.

Raises **TypeError** – If the wrapped type of obj does not match the expected type.

object

Returns The underlying OpenCASCADE object.

Return type OCCT.Geom2d.Geom2d_Geometry

scale(*pnt*, *scale*)

Scale the geometry.

Parameters

- **pnt** (*point2d_like*) – The reference point.

- **`scale`** (*`float`*) – The scaling value.

Returns *True* if scaled.

Return type bool

`rotate`(*pnt*, *angle*)
Rotate the geometry about a point.

Parameters

- **`pnt`** (*`point2d_like`*) – The reference point.
- **`angle`** (*`float`*) – The angle in degrees.

Returns *True* if rotated.

Return type bool

7.1.2 Point2D

class `afem.geometry.entities.`**`Point2D`**(**args*)
Bases: `OCCT.gp.gp_Pnt2d`

A 2-D Cartesian point derived from `gp_Pnt2d`.

`xy`

Returns The point xy-location.

Return type numpy.ndarray

`y`
The point y-location.

Getter Returns the y-location.

Setter Sets the y-location.

Type float

`set_xy`(*xy*)
Set point coordinates.

Parameters **`xy`** (*`point2d_like`*) – Point coordinates.

Returns *True* if set, *False* if not.

Return type bool

`distance`(*other*)
Compute the distance between two points.

Parameters **`other`** (*`point2d_like`*) – The other point.

Returns Distance to the other point.

Return type float

`is_equal`(*other*, *tol=1e-07*)
Check for coincident points.

Parameters

- **`other`** (*`point_like`*) – The other point.
- **`tol`** (*`float`*) – Tolerance for coincidence.

Returns *True* if coincident, *False* if not.

Return type bool

scale(*pnt*, *scale*)

Scale the point.

Parameters

- **pnt** (*point2d_like*) – The reference point.
- **scale** (*float*) – The scaling value.

Returns *True* if scaled.

Return type bool

rotate(*pnt*, *angle*)

Rotate the point about a point.

Parameters

- **pnt** (*point2d_like*) – The reference point.
- **angle** (*float*) – The angle in degrees.

Returns *True* if rotated.

Return type bool

copy()

Return a new copy of the point.

Returns New point.

Return type *afem.geometry.entities.Point2D* (page 26)

classmethod is_point2d_like(*entity*)

Check if the entity is point2d_like.

Parameters **entity** – An entity.

Returns *True* if the entity is point2d_like, *False* if not.

Return type bool

classmethod to_point2d(*entity*)

Convert entity to a point if possible.

Parameters **entity** – An entity.

Returns The entity if already a point, or a new point if it is “point like”.

Return type *afem.geometry.entities.Point2D* (page 26)

Raises **TypeError** – If entity cannot be converted to a Point.

classmethod by_xy(*x*, *y*)

Create a 2-D point by *x* and *y* locations.

Parameters

- **x** (*float*) – The x-location.
- **y** (*float*) – The y-location.

Returns The new 2-D point.

Return type *afem.geometry.entities.Point2D* (page 26)

7.1.3 Direction2D

class `afem.geometry.entities.`**`Direction2D`**(**args*)

Bases: `OCCT.gp.gp_Dir2d`

Unit vector in 2-D space derived from `gp_Dir2d`.

`i`

The direction i-component.

Getter Returns the i-component.

Setter Sets the i-component.

Type float

`j`

The direction j-component.

Getter Returns the j-component.

Setter Sets the j-component.

Type float

`ij`

Returns The direction ij-components.

Return type numpy.ndarray

`xy`

Returns The direction ij-components (Same as ij property, for compatibility only).

Return type numpy.ndarray

`mag`

Returns Direction magnitude is always 1.

Return type float

`rotate`(*pnt*, *angle*)

Rotate the direction about a point.

Parameters

- **pnt** (*`point2d_like`*) – The reference point.
- **angle** (*`float`*) – The angle in degrees.

Returns *True* if rotated.

Return type bool

classmethod `by_xy`(*x*, *y*)

Create a 2-D direction by *x* and *y* components.

Parameters

- **x** (*`float`*) – The x-component.
- **y** (*`float`*) – The y-component.

Returns The new 2-D direction.

Return type *afem.geometry.entities.Direction2D* (page 28)

classmethod by_vector(*v*)

Create a 2-D direction from a 2-D vector.

Parameters **v** (`afem.geometry.entities.Vector2D` (page 29)) – The 2-D vector.

Returns The new 2-D direction.

Return type *afem.geometry.entities.Direction2D* (page 28)

7.1.4 Vector2D

class `afem.geometry.entities.`**Vector2D**(**args*)

Bases: `OCCT.gp.gp_Vec2d`

Vector in 2-D space derived from `gp_Vec2d`.

x

The vector x-component.

Getter Returns the x-component.

Setter Sets the x-component.

Type float

y

The vector y-component.

Getter Returns the y-component.

Setter Sets the y-component.

Type float

xy

Returns The vector xy-components.

Return type numpy.ndarray

mag

Returns Vector magnitude.

Return type float

ij

Returns Normalized vector xy-components.

Return type numpy.ndarray

reverse()

Reverse the direction of the vector.

Returns None.

normalize()

Normalize the vector.

Returns None.

scale(*scale*)

Scale the vector.

Parameters **scale** (*`float`*) – Scaling value.

Returns None.

rotate(*pnt*, *angle*)
Rotate the vector about a point.

Parameters

- **pnt** (*point2d_like*) – The reference point.
- **angle** (*float*) – The angle in degrees.

Returns *True* if rotated.

Return type bool

classmethod to_vector2d(*entity*)
Convert entity to a 2-D vector if possible.

Parameters **entity** (*vector2d_like*) – An entity.

Returns The entity if already a Vector2D, or a new Vector2D if it is vector2d_like.

Return type *afem.geometry.entities.Vector2D* (page 29)

Raises **TypeError** – If entity cannot be converted to a Vector2D.

classmethod by_xy(*x*, *y*)
Create a 2-D vector by *x* and *y* components.

Parameters

- **x** (*float*) – The x-component.
- **y** (*float*) – The y-component.

Returns The new 2-D vector.

Return type *afem.geometry.entities.Vector2D* (page 29)

classmethod by_direction(*d*)
Create a 2-D vector from a 2-D direction.

Parameters **d** (afem.geometry.entities.Direction2D (page 28)) – The 2-D direction.

Returns The new 2-D vector.

Return type *afem.geometry.entities.Vector2D* (page 29)

classmethod by_points(*p1*, *p2*)
Create a 2-D vector by two 2-D points.

Parameters

- **p1** (afem.geometry.entities.Point2D (page 26)) – The first point.
- **p2** (afem.geometry.entities.Point2D (page 26)) – The second point.

Returns The new 2-D vector.

Return type *afem.geometry.entities.Vector2D* (page 29)

7.1.5 Curve2D

class afem.geometry.entities.**Curve2D**(*obj*)
Bases: *afem.geometry.entities.Geometry2D* (page 25)

Base class for 2-D curves around `Geom2d_Curve`.

`u1`

Returns The first parameter.

Return type float

`u2`

Returns The last parameter.

Return type float

`is_closed`

Returns *True* if curve is closed, *False* if not.

Return type bool

`is_periodic`

Returns *True* if curve is periodic, *False* if not.

Return type bool

`p1`

Returns The first point.

Return type *afem.geometry.entities.Point2D* (page 26)

`p2`

Returns The last point.

Return type *afem.geometry.entities.Point2D* (page 26)

`length`

Returns Curve length.

Return type float

`copy`()

Return a new copy of the curve.

Returns New curve.

Return type *afem.geometry.entities.Curve* (page 43)

`local_to_global_u`(**args*)

Convert parameter(s) from local domain 0. <= u <= 1. to global domain a <= u <= b.

Parameters **args** (*`float`*) – Local parameter(s).

Returns Global parameter(s).

Return type float or list(float)

`global_to_local_u`(**args*)

Convert parameter(s) from global domain a <= u <= b to local domain 0. <= u <= 1.

Parameters **args** (*`float`*) – Global parameter(s).

Returns Local parameter(s).

Return type float or list(float)

eval(*u*)
Evaluate a point on the curve.

Parameters **u** (*float*) – Curve parameter.

Returns Curve point.

Return type *afem.geometry.entities.Point2D* (page 26)

deriv(*u*, *d=1*)
Evaluate a derivative on the curve.

Parameters

- **u** (*float*) – Curve parameter.
- **d** (*int*) – Derivative to evaluate.

Returns Curve derivative.

Return type *afem.geometry.entities.Vector2D* (page 29)

reverse()
Reverse curve direction.

Returns None.

reversed_u(*u*)
Calculate the parameter on the reversed curve.

Parameters **u** (*float*) – Curve parameter.

Returns Reversed parameter.

Return type float

arc_length(*u1*, *u2*, *tol=1e-07*)
Calculate the curve length between the parameters.

Parameters

- **u1** (*float*) – First parameter.
- **u2** (*float*) – Last parameter.
- **tol** (*float*) – The tolerance.

Returns Curve length.

Return type float

to_3d(*pln*)
Convert this 2-D curve in the plane.

Parameters **pln** (afem.geometry.entities.Plane (page 52)) – The plane.

Returns The 3-D curve.

Return type *afem.geometry.entities.Curve2D* (page 30)

static wrap(*curve*)
Wrap the OpenCASCADE curve based on its type.

Parameters **curve** (*OCCT.Geom2d.Geom2d_Curve*) – The curve.

Returns The wrapped curve.

Return type *afem.geometry.entities.Curve2D* (page 30)

7.1.6 NurbsCurve2D

class afem.geometry.entities.**NurbsCurve2D**(*obj*)

Bases: *afem.geometry.entities.Curve2D* (page 30)

NURBS curve in 2-D space around `Geom2d_BSplineCurve`.

p

Returns Degree of curve.

Return type int

n

Returns Number of control points - 1.

Return type int

knots

Returns Knot vector.

Return type numpy.ndarray

mult

Returns Multiplicity of knot vector.

Return type numpy.ndarray

uk

Returns Knot sequence.

Return type numpy.ndarray

set_domain(*u1=0.0*, *u2=1.0*)

Reparameterize the knot vector between *u1* and *u2*.

Parameters

- **u1** (*float*) – First parameter.
- **u2** (*float*) – Last parameter.

Returns *True* if successful, *False* if not.

Return type bool

segment(*u1*, *u2*)

Segment the curve between parameters.

Parameters

- **u1** (*float*) – First parameter.
- **u2** (*float*) – Last parameter.

Returns *True* if segmented, *False* if not.

Return type bool

7.1.7 Geometry

class afem.geometry.entities.**Geometry**(*obj*)

Bases: *afem.base.entities.ViewableItem* (page 11)

Base class for geometry.

Parameters obj – The geometry object.

Variables

- **C0** (*OCCT.GeomAbs.GeomAbs_Shape.GeomAbs_C0*) – Only geometric continuity.
- **C1** (*OCCT.GeomAbs.GeomAbs_Shape.GeomAbs_C1*) – Continuity of the first derivative.
- **C2** (*OCCT.GeomAbs.GeomAbs_Shape.GeomAbs_C2*) – Continuity of the second derivative.
- **C3** (*OCCT.GeomAbs.GeomAbs_Shape.GeomAbs_C3*) – Continuity of the third derivative.
- **CN** (*OCCT.GeomAbs.GeomAbs_Shape.GeomAbs_CN*) – Continuity of the n-th derivative.
- **G1** (*OCCT.GeomAbs.GeomAbs_Shape.GeomAbs_G1*) – Tangent vectors on either side of a point on a curve are collinear with the same orientation.
- **G2** (*OCCT.GeomAbs.GeomAbs_Shape.GeomAbs_G2*) – Normalized vectors on either side of a point on a curve are equal.
- **ARC** (*OCCT.GeomAbs.GeomAbs_JoinType.GeomAbs_Arc*) – Arc join type.
- **TANGENT** (*OCCT.GeomAbs.GeomAbs_JoinType.GeomAbs_Tangent*) – Tangent join type.
- **INTERSECT** (*OCCT.GeomAbs.GeomAbs_JoinType.GeomAbs_Intersection*) – Intersection join type.

Raises TypeError – If the wrapped type of obj does not match the expected type.

object

Returns The underlying OpenCASCADE object.

Return type OCCT.Geom.Geom_Geometry

translate(*v*)

Translate the geometry along the vector.

Parameters v (*vector_like*) – The translation vector.

Returns *True* if translated, *False* if not.

Return type bool

Raises TypeError – If *v* cannot be converted to a vector.

mirror(*pln*)

Mirror the geometry using a plane.

Parameters pln (afem.geometry.entities.Plane (page 52)) – The plane.

Returns *True* if mirrored.

Return type bool

scale(*pnt*, *s*)

Scale the geometry.

Parameters

- **pnt** (*point_like*) – The reference point.
- **s** (*float*) – The scaling value.

Returns *True* if scaled.

Return type bool

rotate(*ax1*, *angle*)

Rotate the geometry about an axis.

Parameters

- **ax1** (afem.geometry.entities.Axis1 (page 41)) – The axis of rotation.
- **angle** (*float*) – The angle in degrees.

Returns *True* if rotated.

Return type bool

7.1.8 Point

class afem.geometry.entities.**Point**(**args*)

Bases: OCCT.gp.gp_Pnt, *afem.base.entities.ViewableItem* (page 11)

A 3-D Cartesian point derived from gp_Pnt.

displayed_shape

Returns The shape to be displayed.

Return type OCCT.TopoDS.TopoDS_Vertex

x

The point x-location.

Getter Returns the x-location.

Setter Sets the x-location.

Type float

y

The point y-location.

Getter Returns the y-location.

Setter Sets the y-location.

Type float

z

The point z-location.

Getter Returns the z-location.

Setter Sets the z-location.

Type float

xyz

Returns The point xyz-location.

Return type numpy.ndarray

set_xyz (*xyz*)

Set point coordinates.

Parameters **xyz** (*point_like*) – Point coordinates.

Returns *True* if set, *False* if not.

Return type bool

distance (*other*)

Compute the distance between two points.

Parameters **other** (*point_like*) – The other point.

Returns Distance to the other point.

Return type float

is_equal (*other*, *tol=1e-07*)

Check for coincident points.

Parameters

- **other** (*point_like*) – The other point.
- **tol** (*float*) – Tolerance for coincidence.

Returns *True* if coincident, *False* if not.

Return type bool

translate (*v*)

Translate the geometry along the vector.

Parameters **v** (*vector_like*) – The translation vector.

Returns *True* if translated, *False* if not.

Return type bool

Raises **TypeError** – If *v* cannot be converted to a vector.

mirror (*pln*)

Mirror the geometry using a plane.

Parameters **pln** (afem.geometry.entities.Plane (page 52)) – The plane.

Returns *True* if mirrored.

Return type bool

scale (*pnt*, *s*)

Scale the geometry.

Parameters

- **pnt** (*point_like*) – The reference point.
- **s** (*float*) – The scaling value.

Returns *True* if scaled.

Return type bool

rotate (*ax1*, *angle*)
Rotate the geometry about an axis.

Parameters

- **ax1** (`afem.geometry.entities.Axis1` (page 41)) – The axis of rotation.
- **angle** (*float*) – The angle in degrees.

Returns *True* if rotated.

Return type bool

rotate_xyz (*origin*, *x*, *y*, *z*)
Rotate the point about the global x-, y-, and z-axes using *origin* as the point of rotation if *origin* is a point. Otherwise, if *origin* is an *Axis3* (page 42), rotate the point about the axes of the coordinate system. Rotations follow the right-hand rule for each axis.

Parameters

- **origin** (*point_like or* `afem.geometry.entities.Axis3` (page 42)) – The origin of rotation.
- **x** (*float*) – Rotation about x-axis in degrees.
- **y** (*float*) – Rotation about y-axis in degrees.
- **z** (*float*) – Rotation about z-axis in degrees.

Returns None.

copy ()
Return a new copy of the point.

Returns New point.

Return type *afem.geometry.entities.Point* (page 35)

classmethod by_xyz (*x*, *y*, *z*)
Create a point by *x*, *y*, and *z* locations.

Parameters

- **x** (*float*) – The x-location.
- **y** (*float*) – The y-location.
- **z** (*float*) – The z-location.

Returns The new point.

Return type *afem.geometry.entities.Point* (page 35)

classmethod is_point_like (*entity*)
Check if the entity is point_like.

Parameters **entity** – An entity.

Returns *True* if the entity is point_like, *False* if not.

Return type bool

classmethod to_point (*entity*)
Convert entity to a point if possible.

Parameters **entity** – An entity.

Returns The entity if already a point, or a new point if it is “point like”.

Return type *afem.geometry.entities.Point* (page 35)

Raises **`TypeError`** – If entity cannot be converted to a Point.

7.1.9 Direction

class `afem.geometry.entities.`**`Direction`**(**args*)

Bases: `OCCT.gp.gp_Dir`

Unit vector in 3-D space derived from `gp_Dir`.

`i`

The direction i-component.

Getter Returns the i-component.

Setter Sets the i-component.

Type float

`j`

The direction j-component.

Getter Returns the j-component.

Setter Sets the j-component.

Type float

`k`

The direction k-component.

Getter Returns the k-component.

Setter Sets the k-component.

Type float

`ijk`

Returns The direction ijk-components.

Return type numpy.ndarray

`xyz`

Returns The direction ijk-components (Same as ijk property, for compatibility only).

Return type numpy.ndarray

`mag`

Returns Direction magnitude is always 1.

Return type float

`mirror`(*pln*)

Mirror the geometry using a plane.

Parameters **`pln`** (`afem.geometry.entities.Plane` (page 52)) – The plane.

Returns *True* if mirrored.

Return type bool

`rotate`(*ax1*, *angle*)

Rotate the geometry about an axis.

Parameters

- **ax1** (afem.geometry.entities.Axis1 (page 41)) – The axis of rotation.
- **angle** (*float*) – The angle in degrees.

Returns *True* if rotated.

Return type bool

classmethod to_direction(*entity*)

Convert entity to a Direction if possible.

Parameters **entity** (*vector_like*) – An entity.

Returns The entity if already a Direction, or a new Direction if it is vector_like.

Return type *afem.geometry.entities.Direction* (page 38)

Raises **TypeError** – If entity cannot be converted to a Direction.

classmethod by_xyz(*x, y, z*)

Create a direction by *x*, *y*, and *z* components.

Parameters

- **x** (*float*) – The x-component.
- **y** (*float*) – The y-component.
- **z** (*float*) – The z-component.

Returns The new direction.

Return type *afem.geometry.entities.Direction* (page 38)

classmethod by_vector(*v*)

Create a direction from a vector.

Parameters **v** (afem.geometry.entities.Vector (page 39)) – The vector.

Returns The new direction.

Return type *afem.geometry.entities.Direction* (page 38)

7.1.10 Vector

class afem.geometry.entities.**Vector**(**args*)

Bases: `OCCT.gp.gp_Vec`

Vector in 3-D space derived from `gp_Vec`.

x

The vector x-component.

Getter Returns the x-component.

Setter Sets the x-component.

Type float

y

The vector y-component.

Getter Returns the y-component.

Setter Sets the y-component.

Type float

z

The vector z-component.

Getter Returns the z-component.

Setter Sets the z-component.

Type float

xyz

Returns The vector xyz-components.

Return type numpy.ndarray

mag

Returns Vector magnitude.

Return type float

ijk

Returns Normalized vector xyz-components.

Return type numpy.ndarray

reverse()

Reverse the direction of the vector.

Returns None.

normalize()

Normalize the vector.

Returns None.

scale(*scale*)

Scale the vector.

Parameters **scale** (*float*) – Scaling value.

Returns None.

mirror(*pln*)

Mirror the geometry using a plane.

Parameters **pln** (`afem.geometry.entities.Plane` (page 52)) – The plane.

Returns *True* if mirrored.

Return type bool

rotate(*ax1*, *angle*)

Rotate the geometry about an axis.

Parameters

- **ax1** (`afem.geometry.entities.Axis1` (page 41)) – The axis of rotation.
- **angle** (*float*) – The angle in degrees.

Returns *True* if rotated.

Return type bool

classmethod to_vector(*entity*)
Convert entity to a vector if possible.

Parameters **entity** (*vector_like*) – An entity.

Returns The entity if already a vector, or a new vector if it is vector_like.

Return type *afem.geometry.entities.Vector* (page 39)

Raises **TypeError** – If entity cannot be converted to a Vector.

classmethod by_xyz(*x*, *y*, *z*)
Create a vector by *x*, *y*, and *z* components.

Parameters

- **x** (*float*) – The x-component.
- **y** (*float*) – The y-component.
- **z** (*float*) – The yzcomponent.

Returns The new vector.

Return type *afem.geometry.entities.Vector* (page 39)

classmethod by_direction(*d*)
Create a vector from a direction.

Parameters **d** (afem.geometry.entities.Direction (page 38)) – The direction.

Returns The new vector.

Return type *afem.geometry.entities.Vector* (page 39)

classmethod by_points(*p1*, *p2*)
Create a vector by two points.

Parameters

- **p1** (afem.geometry.entities.Point (page 35)) – The first point.
- **p2** (afem.geometry.entities.Point (page 35)) – The second point.

Returns The new vector.

Return type *afem.geometry.entities.Vector* (page 39)

7.1.11 Axis1

class afem.geometry.entities.**Axis1**(**args*)
Bases: OCCT.gp.gp_Ax1

Axis in 3-D space derived from gp_Ax1.

origin

Returns The origin of the axis.

Return type *afem.geometry.entities.Point* (page 35)

classmethod by_direction(*p*, *d*)
Create an axis by an origin point and direction.

Parameters

- **p** (afem.geometry.entities.Point (page 35)) – The origin.

- **d** (`afem.geometry.entities.Direction` (page 38)) – The direction.

Returns The new axis.

Return type afem.geometry.entities.Axis

7.1.12 Axis3

class `afem.geometry.entities.`**`Axis3`**(**args*)

Bases: `OCCT.gp.gp_Ax3`

Coordinate system in 3-D space derived from `gp_Ax3`.

origin

Returns The origin of the coordinate system.

Return type *afem.geometry.entities.Point* (page 35)

x_dir

Returns The x-direction.

Return type *afem.geometry.entities.Direction* (page 38)

y_dir

Returns The y-direction.

Return type *afem.geometry.entities.Direction* (page 38)

z_dir

Returns The z-direction.

Return type *afem.geometry.entities.Direction* (page 38)

x_axis

Returns The x-axis.

Return type *afem.geometry.entities.Axis1* (page 41)

y_axis

Returns The y-axis.

Return type *afem.geometry.entities.Axis1* (page 41)

z_axis

Returns The z-axis.

Return type *afem.geometry.entities.Axis1* (page 41)

classmethod by_normal(*p*, *n*)

Create a coordinate system by an origin point and normal direction.

Parameters

- **p** (`afem.geometry.entities.Point` (page 35)) – The origin.
- **n** (`afem.geometry.entities.Direction` (page 38)) – The normal direction.

Returns The new coordinate system.

Return type *afem.geometry.entities.Axis3* (page 42)

classmethod by_xdirection (*p*, *n*, *x*)

Create a coordinate system by an origin, a normal direction, and its x-direction.

Parameters

- **p** (`afem.geometry.entities.Point` (page 35)) – The origin.
- **n** (`afem.geometry.entities.Direction` (page 38)) – The normal direction.
- **x** (`afem.geometry.entities.Direction` (page 38)) – The direction of the x-axis.

Returns The new coordinate system.

Return type *afem.geometry.entities.Axis3* (page 42)

7.1.13 Curve

class `afem.geometry.entities.`**Curve** (*obj*)

Bases: *`afem.geometry.entities.Geometry`* (page 34)

Base class for curves around `Geom_Curve`.

Variables

- **LINE** (*`OCCT.GeomAbs.GeomAbs_CurveType.GeomAbs_Line`*) – Line type.
- **CIRCLE** (*`OCCT.GeomAbs.GeomAbs_CurveType.GeomAbs_Circle`*) – Circle type.
- **ELLIPSE** (*`OCCT.GeomAbs.GeomAbs_CurveType.GeomAbs_Ellipse`*) – Ellipse type.
- **HYPERBOLA** (*`OCCT.GeomAbs.GeomAbs_CurveType.GeomAbs_Hyperbola`*) – Hyperbola type.
- **PARABOLA** (*`OCCT.GeomAbs.GeomAbs_CurveType.GeomAbs_Parabola`*) – Parabola type.
- **BEZIER** (*`OCCT.GeomAbs.GeomAbs_CurveType.GeomAbs_BezierCurve`*) – Bezier curve type.
- **BSPLINE** (*`OCCT.GeomAbs.GeomAbs_CurveType.GeomAbs_BSplineCurve`*) – BSpline curve type.
- **OFFSET** (*`OCCT.GeomAbs.GeomAbs_CurveType.GeomAbs_OffsetCurve`*) – Offset curve type.
- **OTHER** (*`OCCT.GeomAbs.GeomAbs_CurveType.GeomAbs_OtherCurve`*) – Other curve type.

displayed_shape

Returns The shape to be displayed.

Return type OCCT.TopoDS.TopoDS_Edge

u1

Returns The first parameter.

Return type float

u2

Returns The last parameter.

Return type float

is_closed

Returns *True* if curve is closed, *False* if not.

Return type bool

is_periodic

Returns *True* if curve is periodic, *False* if not.

Return type bool

p1

Returns The first point.

Return type *afem.geometry.entities.Point* (page 35)

p2

Returns The last point.

Return type *afem.geometry.entities.Point* (page 35)

length

Returns Curve length.

Return type float

copy()

Return a new copy of the curve.

Returns New curve.

Return type *afem.geometry.entities.Curve* (page 43)

local_to_global_u(**args*)

Convert parameter(s) from local domain 0. <= u <= 1. to global domain a <= u <= b.

Parameters **args** (`float`) – Local parameter(s).

Returns Global parameter(s).

Return type float or list(float)

global_to_local_u(**args*)

Convert parameter(s) from global domain a <= u <= b to local domain 0. <= u <= 1.

Parameters **args** (`float`) – Global parameter(s).

Returns Local parameter(s).

Return type float or list(float)

eval(*u*)

Evaluate a point on the curve.

Parameters **u** (`float`) – Curve parameter.

Returns Curve point.

Return type *afem.geometry.entities.Point* (page 35)

deriv(*u, d=1*)

Evaluate a derivative on the curve.

Parameters

- **u** (*float*) – Curve parameter.
- **d** (*int*) – Derivative to evaluate.

Returns Curve derivative.

Return type *afem.geometry.entities.Vector* (page 39)

reverse()

Reverse curve direction.

Returns None.

reversed_u(*u*)

Calculate the parameter on the reversed curve.

Parameters u (*float*) – Curve parameter.

Returns Reversed parameter.

Return type float

arc_length(*u1*, *u2*, *tol=1e-07*)

Calculate the curve length between the parameters.

Parameters

- **u1** (*float*) – First parameter.
- **u2** (*float*) – Last parameter.
- **tol** (*float*) – The tolerance.

Returns Curve length.

Return type float

invert(*p*)

Invert the point on the curve to find the parameter.

Parameters p (*point_like*) – The point.

Returns The nearest parameter on the curve.

Return type float

Raises RuntimeError – If algorithm fails.

static wrap(*curve*)

Wrap the OpenCASCADE curve based on its type.

Parameters curve (*OCCT.Geom.Geom_Curve*) – The curve.

Returns The wrapped curve.

Return type *afem.geometry.entities.Curve* (page 43)

7.1.14 Line

class afem.geometry.entities.**Line**(*obj*)

Bases: *afem.geometry.entities.Curve* (page 43)

Infinite line around `Geom_Line`.

classmethod by_axis(*ax1*)

Create a line from an axis.

Parameters **ax1** (afem.geometry.entities.Axis1 (page 41)) – The axis.

Returns The new line.

Return type *afem.geometry.entities.Line* (page 45)

classmethod by_direction(*p*, *d*)
Create a line passing through a point along a direction.

Parameters

- **p** (afem.geometry.entities.Point (page 35)) – The point.
- **d** (afem.geometry.entities.Direction (page 38)) – The direction.

Returns The new line.

Return type *afem.geometry.entities.Line* (page 45)

7.1.15 Circle

class afem.geometry.entities.**Circle**(*obj*)
Bases: *afem.geometry.entities.Curve* (page 43)

Circular curve around Geom_Circle.

radius

Returns The radius.

Return type float

center

Returns The center point of the circle.

Return type *afem.geometry.entities.Point* (page 35)

set_radius(*r*)
Set the radius.

Parameters **r** (*float*) – The radius.

Returns None.

7.1.16 Ellipse

class afem.geometry.entities.**Ellipse**(*obj*)
Bases: *afem.geometry.entities.Curve* (page 43)

Elliptical curve around Geom_Ellipse.

major_radius

Returns The major radius.

Return type float

minor_radius

Returns The minor radius.

Return type float

set_major_radius(*r*)
Set the major radius.

Parameters **r** (*float*) – The major radius.

Returns None.

set_minor_radius(*r*)
Set the minor radius.

Parameters **r** (*float*) – The minor radius.

Returns None.

7.1.17 NurbsCurve

class afem.geometry.entities.**NurbsCurve**(*obj*)
Bases: *afem.geometry.entities.Curve* (page 43)

NURBS curve around `Geom_BSplineCurve`.

p

Returns Degree of curve.

Return type int

n

Returns Number of control points.

Return type int

knots

Returns Knot vector.

Return type numpy.ndarray

mult

Returns Multiplicity of knot vector.

Return type numpy.ndarray

uk

Returns Knot sequence.

Return type numpy.ndarray

cp

Returns Control points.

Return type numpy.ndarray

w

Returns Weights of control points.

Return type numpy.ndarray

cpw

Returns Homogeneous control points.

Return type numpy.ndarray

set_domain(*u1=0.0*, *u2=1.0*)
Reparameterize the knot vector between *u1* and *u2*.

Parameters

- **u1** (*float*) – First parameter.
- **u2** (*float*) – Last parameter.

Returns *True* if successful, *False* if not.

Return type bool

segment(*u1*, *u2*)
Segment the curve between parameters.

Parameters

- **u1** (*float*) – First parameter.
- **u2** (*float*) – Last parameter.

Returns *True* if segmented, *False* if not.

Return type bool

set_cp(*i*, *cp*, *weight=None*)
Modify the curve by setting the specified control point and weight.

Parameters

- **i** (*int*) – The point index (1 <= *i* <= *n*).
- **cp** (afem.geometry.entities.Point (page 35)) – The point.
- **weight** (*float or None*) – The weight.

Returns None.

classmethod by_data(*cp*, *knots*, *mult*, *p*, *weights=None*, *is_periodic=False*)
Create a NURBS curve by data.

Parameters

- **cp** (*collections.Sequence(point_like)*) – Control points.
- **knots** (*collections.Sequence(float)*) – Knot vector.
- **mult** (*collections.Sequence(int)*) – Multiplicities of knot vector.
- **p** (*int*) – Degree.
- **weights** (*collections.Sequence(float)*) – Weights of control points.
- **is_periodic** (*bool*) – Flag for periodicity.

7.1.18 TrimmedCurve

class afem.geometry.entities.**TrimmedCurve**(*obj*)
Bases: *afem.geometry.entities.Curve* (page 43)

Trimmed curve around `Geom_TrimmedCurve`. This defines a basis curve limited by two parameter values.

basis_curve

Returns The basis curve.

Return type *afem.geometry.entities.Curve* (page 43)

set_trim(*u1*, *u2*, *sense=True*, *adjust_periodic=True*)

Set the trimming parameters on the basis curve.

Parameters

- **u1** (*float*) – The first parameter.
- **u2** (*float*) – The last parameter.
- **sense** (*bool*) – If the basis curve is periodic, the trimmed curve will have the same orientation as the basis curve if `True` or opposite if `False`.
- **adjust_periodic** (*bool*) – If the basis curve is periodic, the bounds of the trimmed curve may be different from *u1* and *u2* if `True`.

Returns None.

Raises

- **RuntimeError** – If *u1* == *u2*.
- **RuntimeError** – If *u1* or *u2* is outside the bounds of the basis curve.

classmethod by_parameters(*basis_curve*, *u1=None*, *u2=None*, *sense=True*, *adjust_periodic=True*)

Create a trimmed curve using a basis curve and limiting parameters.

Parameters

- **basis_curve** (`afem.geometry.entities.Curve` (page 43)) – The basis curve.
- **u1** (*float*) – The first parameter. If not provided then the first parameter of the basis curve will be used.
- **u2** (*float*) – The last parameter. If not provided then the last parameter of the basis curve will be used.
- **sense** (*bool*) – If the basis curve is periodic, the trimmed curve will have the same orientation as the basis curve if `True` or opposite if `False`.
- **adjust_periodic** (*bool*) – If the basis curve is periodic, the bounds of the trimmed curve may be different from *u1* and *u2* if `True`.

Raises

- **TypeError** – If the basis curve is not a valid curve type.
- **ValueError** – If *u1* >= *u2*.

7.1.19 Surface

class afem.geometry.entities.**Surface**(*obj*)

Bases: *afem.geometry.entities.Geometry* (page 34)

Base class for surfaces around `Geom_Surface`.

Variables

- **PLANE** (*OCCT.GeomAbs.GeomAbs_SurfaceType.GeomAbs_Plane*) – Plane type.
- **CYLINDER** (*OCCT.GeomAbs.GeomAbs_SurfaceType.GeomAbs_Cylinder*) – Cylinder type.
- **CONE** (*OCCT.GeomAbs.GeomAbs_SurfaceType.GeomAbs_Cone*) – Cone type.

- **SPHERE** (*OCCT.GeomAbs.GeomAbs_SurfaceType.GeomAbs_Sphere*) – Sphere type.
- **TORUS** (*OCCT.GeomAbs.GeomAbs_SurfaceType.GeomAbs_Torus*) – Torus type.
- **BEZIER** (*OCCT.GeomAbs.GeomAbs_SurfaceType.GeomAbs_BezierSurface*) – Bezier type.
- **BSPLINE** (*OCCT.GeomAbs.GeomAbs_SurfaceType.GeomAbs_BSplineSurface*) – BSpline type.
- **REVOLUTION** (*OCCT.GeomAbs.GeomAbs_SurfaceType.GeomAbs_SurfaceOfRevolution*) – Revolution type.
- **EXTRUSION** (*OCCT.GeomAbs.GeomAbs_SurfaceType.GeomAbs_SurfaceOfExtrusion*) – Extrusion type.
- **OFFSET** (*OCCT.GeomAbs.GeomAbs_SurfaceType.GeomAbs_OffsetSurface*) – Offset type.
- **OTHER** (*OCCT.GeomAbs.GeomAbs_SurfaceType.GeomAbs_OtherSurface*) – Other type.

displayed_shape

Returns The shape to be displayed.

Return type OCCT.TopoDS.TopoDS_Face

adaptor

Returns A surface adaptor.

Return type OCCT.GeomAdaptor.GeomAdaptor_Surface

u1

Returns The first parameter in u-direction.

Return type float

u2

Returns The last parameter in u-direction.

Return type float

v1

Returns The first parameter in v-direction.

Return type float

v2

Returns The last parameter in v-direction.

Return type float

area

Returns The surface area.

Return type float

copy()

Return a new copy of the surface.

Returns New surface.

Return type *afem.geometry.entities.Surface* (page 49)

is_planar(*tol=1e-07*)
Check if surface is planar.

Parameters **tol** (*float*) – The tolerance.

Returns *True* if planar, *False* if not.

Return type bool

as_plane(*tol=1e-07*)
Get a plane if the surface is planar.

Parameters **tol** (*float*) – The tolerance.

Returns Return this plane if it's already a plane, return *None* if the surface is not planar, or return a plane.

Return type *afem.geometry.entities.Surface* (page 49) or None

eval(*u=0.0*, *v=0.0*)
Evaluate a point on the surface.

Parameters

- **u** (*float*) – Surface u-parameter.
- **v** (*float*) – Surface v-parameter.

Returns Surface point.

Return type *afem.geometry.entities.Point* (page 35)

deriv(*u*, *v*, *nu*, *nv*)
Evaluate a derivative on the surface.

Parameters

- **u** (*float*) – Surface u-parameter.
- **v** (*float*) – Surface v-parameter.
- **nu** (*int*) – Derivative in u-direction.
- **nv** (*int*) – Derivative in v-direction.

Returns Surface derivative.

Return type *afem.geometry.entities.Vector* (page 39)

norm(*u*, *v*)
Evaluate a normal on the surface.

Parameters

- **u** (*float*) – Surface u-parameter.
- **v** (*float*) – Surface v-parameter.

Returns Surface normal.

Return type *afem.geometry.entities.Vector* (page 39)

surface_area(*u1*, *v1*, *u2*, *v2*, *tol=1e-07*)
Calculate the surface area between the parameters.

Parameters

- **u1** (*float*) – The first parameter in u-direction.
- **v1** (*float*) – The first parameter in v-direction.
- **u2** (*float*) – The last parameter in u-direction.
- **v2** (*float*) – The last parameter in v-direction.
- **tol** (*float*) – The tolerance.

Returns The area.

Return type float

u_iso(*u*)

Get a iso-parametric curve at a constant u-parameter.

Parameters **u** (*float*) – The u-parameter.

Returns The curve.

Return type *afem.geometry.entities.Curve* (page 43)

v_iso(*v*)

Get a iso-parametric curve at a constant v-parameter.

Parameters **v** (*float*) – The v-parameter.

Returns The curve.

Return type *afem.geometry.entities.Curve* (page 43)

invert(*p*)

Invert the point on the surface to find the parameters.

Parameters **p** (*point_like*) – The point.

Returns The nearest parameters on the curve.

Return type tuple(float, float)

Raises **RuntimeError** – If algorithm fails.

static wrap(*surface*)

Wrap the OpenCASCADE surface based on its type.

Parameters **surface** (*OCCT.Geom.Geom_Surface*) – The curve.

Returns The wrapped surface.

Return type *afem.geometry.entities.Surface* (page 49)

7.1.20 Plane

class afem.geometry.entities.**Plane**(*obj*)

Bases: *afem.geometry.entities.Surface* (page 49)

Infinite plane around Geom_Plane.

origin

Returns The origin of the plane. This simply evaluates the plane at u=0 and v=0.

Return type *afem.geometry.entities.Point* (page 35)

axis

Returns The main axis (normal) of the plane.

Return type *afem.geometry.entities.Axis1* (page 41)

gp_pln

Returns The underlying gp_Pln.

Return type OCCT.gp.gp_Pln

distance (*pnt*)

Compute the distance between a point and this plane.

Parameters **pnt** (*point_like*) – The point.

Returns The distance.

Return type float

Raises **TypeError** – If *pnt* cannot be converted to a point.

rotate_x (*angle*)

Rotate the plane about its x-axis.

Parameters **angle** (*float*) – The rotation angle in degrees.

Returns None.

rotate_y (*angle*)

Rotate the plane about its y-axis.

Parameters **angle** (*float*) – The rotation angle in degrees.

Returns None.

classmethod by_system (*ax3*)

Create a plane by a coordinate system.

Parameters **ax3** (afem.geometry.entities.Axis3 (page 42)) – The system.

Returns The new plane.

Return type *afem.geometry.entities.Plane* (page 52)

classmethod by_normal (*p*, *n*)

Create a plane by an origin and a normal.

Parameters

- **p** (afem.geometry.entities.Point (page 35)) – The origin.
- **n** (afem.geometry.entities.Direction (page 38)) – The normal direction.

Returns The new plane.

Return type *afem.geometry.entities.Plane* (page 52)

7.1.21 NurbsSurface

class afem.geometry.entities.**NurbsSurface** (*obj*)

Bases: *afem.geometry.entities.Surface* (page 49)

NURBS surface around Geom_BSplineSurface.

p

Returns Degree in u-direction.

Return type int

q

Returns Degree in v-direction.

Return type int

n

Returns Number of control points in u-direction.

Return type int

m

Returns Number of control points in v-direction.

Return type int

uknots

Returns Knot vector in u-direction.

Return type numpy.ndarray

umult

Returns Multiplicity of knot vector in u-direction.

Return type numpy.ndarray

uk

Returns Knot sequence in u-direction.

Return type numpy.ndarray

vknots

Returns Knot vector in v-direction.

Return type numpy.ndarray

vmult

Returns Multiplicity of knot vector in v-direction.

Return type numpy.ndarray

vk

Returns Knot sequence in v-direction.

Return type numpy.ndarray

cp

Returns Control points.

Return type numpy.ndarray

w

Returns Weights of control points.

Return type numpy.ndarray

cpw

Returns Homogeneous control points.

Return type numpy.ndarray

set_udomain(*u1=0.0*, *u2=1.0*)
Reparameterize the knot vector between *u1* and *u2*.

Parameters

- **u1** (*float*) – First parameter.
- **u2** (*float*) – Last parameter.

Returns *True* if successful, *False* if not.

Return type bool

set_vdomain(*v1=0.0*, *v2=1.0*)
Reparameterize the knot vector between *v1* and *v2*.

Parameters

- **v1** (*float*) – First parameter.
- **v2** (*float*) – Last parameter.

Returns *True* if successful, *False* if not.

Return type bool

local_to_global_param(*d*, **args*)
Convert parameter(s) from local domain 0. <= u,v <= 1. to global domain a <= u,v <= b.

Parameters

- **d** (*str*) – Parameter direction ('u' or 'v').
- **args** (*float*) – Local parameter(s).

Returns Global parameter(s).

Return type float or list(float)

global_to_local_param(*d*, **args*)
Convert surface parameter(s) from global domain a <= u,v <= b to local domain 0. <= u,v <= 1.

Parameters

- **d** (*str*) – Parameter direction ('u' or 'v').
- **args** (*float*) – Global parameter(s).

Returns Local parameter(s).

Return type float or list(float)

segment(*u1*, *u2*, *v1*, *v2*)
Segment the surface between parameters.

Parameters

- **u1** (*float*) – First parameter in u-direction.
- **u2** (*float*) – Last parameter in u-direction.
- **v1** (*float*) – First parameter in v-direction.
- **v2** (*float*) – Last parameter in v-direction.

Returns *True* if segmented, *False* if not.

Return type bool

locate_u (*u*, *tol2d=1e-09*, *with_knot_repetition=False*)
Locate the u-value in the knot sequence.

Parameters

- **u** (*float*) – The parameter.
- **tol2d** (*int*) – The parametric tolerance. Used to determine if *u* is at an existing knot.
- **with_knot_repetition** (*bool*) – Considers location of knot value with repetition of multiple knot value if `True`.

Returns Bounding knot locations (i1, i2).

Return type tuple(int)

locate_v (*v*, *tol2d=1e-09*, *with_knot_repetition=False*)
Locate the v-value in the knot sequence.

Parameters

- **v** (*float*) – The parameter.
- **tol2d** (*int*) – The parametric tolerance. Used to determine if *v* is at an existing knot.
- **with_knot_repetition** (*bool*) – Considers location of knot value with repetition of multiple knot value if `True`.

Returns Bounding knot locations (i1, i2).

Return type tuple(int)

insert_uknot (*u*, *m=1*, *tol2d=1e-09*)
Insert a u-value in the knot sequence.

Parameters

- **u** (*float*) – The knot value.
- **m** (*int*) – If the knot already exists, the multiplicity of the knot is increased if the previous multiplicity is lower than *m*. Otherwise it does nothing.
- **tol2d** (*float*) – Parametric tolerance for comparing knot values.

Returns None.

insert_vknot (*v*, *m=1*, *tol2d=1e-09*)
Insert a v-value in the knot sequence.

Parameters

- **v** (*float*) – The knot value.
- **m** (*int*) – If the knot already exists, the multiplicity of the knot is increased if the previous multiplicity is lower than *m*. Otherwise it does nothing.
- **tol2d** (*float*) – Parametric tolerance for comparing knot values.

Returns None.

set_uknots (*uknots*)
Set the knots of the surface in the u-direction.

Parameters **uknots** (*collections.Sequence(float)*) – Knot values.

Returns None.

Raises **`ValueError`** – If the number of the given knots does not equal the number of the existing knots.

`set_vknots`(*vknots*)
Set the knots of the surface in the u-direction.

Parameters **`vknots`** (*`collections.Sequence(float)`*) – Knot values.

Returns None.

Raises **`ValueError`** – If the number of the given knots does not equal the number of the existing knots.

`set_cp`(*i*, *j*, *cp*, *weight=None*)
Modify the surface by setting the specified control point and weight.

Parameters

- **`i`** (*`int`*) – The point index in u-direction (1 <= *i* <= *n*).
- **`j`** (*`int`*) – The point index in v-direction (1 <= *j* <= *m*).
- **`cp`** (`afem.geometry.entities.Point` (page 35)) – The point.
- **`weight`** (*`float or None`*) – The weight.

Returns None.

`set_cp_row`(*u_index*, *cp*, *weights=None*)
Modify the surface by setting the specified row of control points and weights.

Parameters

- **`u_index`** (*`int`*) – The row index (1 <= *u_index* <= *n*).
- **`cp`** (*`list`*(`afem.geometry.entities.Point` (page 35))) – The points.
- **`weights`** (*`list(float) or None`*) – The weights.

Returns None.

`set_cp_col`(*v_index*, *cp*, *weights=None*)
Modify the surface by setting the specified column of control points and weights.

Parameters

- **`v_index`** (*`int`*) – The column index (1 <= *v_index* <= *m*).
- **`cp`** (*`list`*(`afem.geometry.entities.Point` (page 35))) – The points.
- **`weights`** (*`list(float) or None`*) – The weights.

Returns None.

`classmethod by_data`(*cp*, *uknots*, *vknots*, *umult*, *vmult*, *p*, *q*, *weights=None*, *is_u_periodic=False*, *is_v_periodic=False*)
Create a NURBS surface by data.

Parameters

- **`cp`** (*`list(list(point_like))`*) – Two-dimensional list of control points.
- **`uknots`** (*`list(float)`*) – Knot vector for u-direction.
- **`vknots`** (*`list(float)`*) – Knot vector for v-direction.
- **`umult`** (*`list(int)`*) – Multiplicities of knot vector in u-direction.
- **`vmult`** (*`list(int)`*) – Multiplicities of knot vector in v-direction.

- **p** (*int*) – Degree in u-direction.
- **q** (*int*) – Degree in v-direction.
- **weights** (*list (list (float))*) – Two-dimensional list of control point weights.
- **is_u_periodic** (*bool*) – Flag for periodicity in u-direction.
- **is_v_periodic** (*bool*) – Flag for periodicity in v-direction.

7.2 Create

7.2.1 PointByXYZ

class afem.geometry.create.**PointByXYZ**(*x=0.0*, *y=0.0*, *z=0.0*)

Bases: `object`

Create a point by x, y, and z location.

Parameters

- **x** (*float*) – x-location.
- **y** (*float*) – y-location.
- **z** (*float*) – z-location.

point

Returns The created point.

Return type *afem.geometry.entities.Point* (page 35)

7.2.2 PointByArray

class afem.geometry.create.**PointByArray**(*xyz=(0.0, 0.0, 0.0)*)

Bases: `afem.geometry.create.PointByXYZ` (page 58)

Create a point from an array.

Parameters **xyz** (*array_like*) – Array describing point location.

Raises **ValueError** – If the length of the array is not equal to three.

7.2.3 PointFromParameter

class afem.geometry.create.**PointFromParameter**(*c*, *u0*, *ds*, *tol=1e-07*)

Bases: `object`

Create a point along a curve at a specified distance from a parameter.

Parameters

- **c** (`afem.adaptor.entities.AdaptorCurve` (page 176) *or* `afem.geometry.entities.Curve` (page 43) *or* `afem.topology.entities.Edge` (page 115) *or* `afem.topology.entities.Wire` (page 116)) – The curve.
- **u0** (*float*) – The initial parameter.
- **ds** (*float*) – The distance along the curve from the given parameter.

- **tol** (*float*) – Tolerance.

is_done

Returns *True* if done, *False* otherwise.

Return type bool

point

Returns The created point.

Return type *afem.geometry.entities.Point* (page 35)

parameter

Returns The parameter on the curve.

Return type float

7.2.4 PointsAlongCurveByNumber

class afem.geometry.create.**PointsAlongCurveByNumber**(*c*, *n*, *u1=None*, *u2=None*, *d1=None*, *d2=None*, *tol=1e-07*)

Bases: `object`

Create a specified number of points along a curve. The points will be equidistant.

Parameters

- **c** (`afem.adaptor.entities.AdaptorCurve` (page 176) *or* `afem.geometry.entities.Curve` (page 43) *or* `afem.topology.entities.Edge` (page 115) *or* `afem.topology.entities.Wire` (page 116)) – The curve.
- **n** (*int*) – Number of points to create.
- **u1** (*float*) – The parameter of the first point (default=*c.u1*).
- **u2** (*float*) – The parameter of the last point (default=*c.u2*).
- **d1** (*float*) – An offset distance for the first point. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last point. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **tol** (*float*) – Tolerance.

Raises **RuntimeError** – If `GCPnts_UniformAbscissa` fails.

npts

Returns The number of points.

Return type int

points

Returns The points.

Return type list(*afem.geometry.entities.Point* (page 35))

parameters

Returns The parameters.

Return type list(float)

spacing

Returns The spacing between the first and second points if there are more than one point. Otherwise *None*.

Return type float or None

interior_points

Returns The points between the first and last points.

Return type list(*afem.geometry.entities.Point* (page 35))

7.2.5 PointsAlongCurveByDistance

class `afem.geometry.create.`**PointsAlongCurveByDistance**(*c*, *maxd*, *u1=None*, *u2=None*, *d1=None*, *d2=None*, *nmin=0*, *tol=1e-07*)

Bases: `object`

Create points along a curve by distance between points. The points will be equidistant. This method calculates the number of points given the curve length and then uses *`PointsAlongCurveByNumber`* (page 59).

Parameters

- **c** (`afem.adaptor.entities.AdaptorCurve` (page 176) *or* `afem.geometry.entities.Curve` (page 43) *or* `afem.topology.entities.Edge` (page 115) *or* `afem.topology.entities.Wire` (page 116)) – The curve.
- **maxd** (*float*) – The maximum allowed spacing between points. The actual spacing will be adjusted to not to exceed this value.
- **u1** (*float*) – The parameter of the first point (default=c.u1).
- **u2** (*float*) – The parameter of the last point (default=c.u2).
- **d1** (*float*) – An offset distance for the first point. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last point. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **nmin** (*int*) – Minimum number of points to create.
- **tol** (*float*) – Tolerance.

Raises **RuntimeError** – If OCC method fails.

npts

Returns The number of points.

Return type int

points

Returns The points.

Return type list(*afem.geometry.entities.Point* (page 35))

parameters

Returns The parameters.

Return type list(float)

spacing

Returns The spacing between the first and second points if there are more than one point. Otherwise *None*.

Return type float or None

interior_points

Returns The points between the first and last points.

Return type list(*afem.geometry.entities.Point* (page 35))

7.2.6 DirectionByXYZ

class afem.geometry.create.**DirectionByXYZ**(*x=1.0*, *y=0.0*, *z=0.0*)

Bases: object

Create a direction (i.e., unit vector) by x-, y-, and z-components.

Parameters

- **x** (*float*) – x-component.
- **y** (*float*) – y-component.
- **z** (*float*) – z-component.

direction

Returns The direction.

Return type *afem.geometry.entities.Direction* (page 38)

7.2.7 DirectionByArray

class afem.geometry.create.**DirectionByArray**(*xyz=(1.0, 0.0, 0.0)*)

Bases: *afem.geometry.create.DirectionByXYZ* (page 61)

Create a direction (i.e., unit vector) from an array-like object.

Parameters **xyz** (*array_like*) – Array-like object defining xyz-components.

Raises **ValueError** – If *len(xyz) != 3*.

7.2.8 DirectionByPoints

class afem.geometry.create.**DirectionByPoints**(*p1*, *p2*)

Bases: *afem.geometry.create.DirectionByArray* (page 61)

Create a direction (i.e., unit vector) between two points.

Parameters

- **p1** (*point_like*) – The first point.
- **p2** (*point_like*) – The last point.

7.2.9 VectorByXYZ

class afem.geometry.create.**VectorByXYZ**(*x=1.0, y=0.0, z=0.0*)

Bases: `object`

Create a vector by x-, y-, and z-components.

Parameters

- **x** (*float*) – x-component.
- **y** (*float*) – y-component.
- **z** (*float*) – z-component.

vector

Returns The vector.

Return type *afem.geometry.entities.Vector* (page 39)

7.2.10 VectorByArray

class afem.geometry.create.**VectorByArray**(*xyz=(1.0, 0.0, 0.0)*)

Bases: *afem.geometry.create.VectorByXYZ* (page 62)

Create a vector from an array-like object.

Parameters **xyz** (*array_like*) – Array-like object defining xyz-components.

Raises **ValueError** – If *len(xyz) != 3*.

7.2.11 VectorByPoints

class afem.geometry.create.**VectorByPoints**(*p1, p2*)

Bases: `object`

Create a vector between two points.

Parameters

- **p1** (*point_like*) – The first point.
- **p2** (*point_like*) – The last point.

Raises **TypeError** – If *p1* or *p2* cannot be converted to a *Point* (page 35).

vector

Returns The vector.

Return type *afem.geometry.entities.Vector* (page 39)

7.2.12 LineByVector

class afem.geometry.create.**LineByVector**(*p, d*)

Bases: `object`

Create a line by an origin and a vector.

Parameters

- **p** (*point_like*) – Origin of line.
- **d** (*vector_like*) – Direction of line.

Raises

- **TypeError** – If *p* cannot be converted to a *Point* (page 35)
- **TypeError** – If *d* cannot be converted to a *Direction* (page 38)

line

Returns The line.

Return type *afem.geometry.entities.Line* (page 45)

7.2.13 LineByPoints

class afem.geometry.create.**LineByPoints**(*p1*, *p2*)
Bases: object

Create a line through two points.

Parameters

- **p1** (*point_like*) – The first point.
- **p2** (*point_like*) – The last point.

Raises **TypeError** – If *p1* or *p2* cannot be converted to a *Point* (page 35).

line

Returns The line.

Return type *afem.geometry.entities.Line* (page 45)

7.2.14 CircleByNormal

class afem.geometry.create.**CircleByNormal**(*center*, *normal*, *radius*)
Bases: object

Create a circle using a center, normal, and radius.

Parameters

- **center** (*point_like*) – The center point.
- **normal** (*vector_like*) – The normal of the plane.
- **radius** (*float*) – The radius.

circle

Returns The circle.

Return type *afem.geometry.entities.Circle* (page 46)

7.2.15 CircleByPlane

class `afem.geometry.create.`**`CircleByPlane`**(*center*, *plane*, *radius*)

Bases: `object`

Create a circle using a center, plane, and radius.

Parameters

- **`center`** (*`point_like`*) – The center point.
- **`plane`** (`afem.geometry.entities.Plane` (page 52)) – The plane.
- **`radius`** (*`float`*) – The radius.

`circle`

Returns The circle.

Return type *afem.geometry.entities.Circle* (page 46)

7.2.16 CircleBy3Points

class `afem.geometry.create.`**`CircleBy3Points`**(*p1*, *p2*, *p3*)

Bases: `object`

Create a circle using three points.

Parameters

- **`p1`** (*`point_like`*) – The first point.
- **`p2`** (*`point_like`*) – The second point.
- **`p3`** (*`point_like`*) – The third point.

`circle`

Returns The circle.

Return type *afem.geometry.entities.Circle* (page 46)

7.2.17 NurbsCurve2DByInterp

class `afem.geometry.create.`**`NurbsCurve2DByInterp`**(*qp*, *is_periodic=False*, *v1=None*, *v2=None*, *tol=1e-07*)

Bases: `object`

Create a 2-D cubic curve by interpolating 2-D points.

Parameters

- **`qp`** (*`collections.Sequence(point2d_like)`*) – Points to interpolate.
- **`is_periodic`** (*`bool`*) – Flag for curve periodicity. If *True* the curve will be periodic and closed.
- **`v1`** (*`vector2d_like`*) – Tangent to match at first point.
- **`v2`** (*`vector2d_like`*) – Tangent to match at last point.
- **`tol`** (*`float`*) – Tolerance used to check for coincident points and the magnitude of end vectors.

Raises **`RuntimeError`** – If `Geom2dAPI_Interpolate` fails.

`curve`

Returns The 2-D NURBS curve.

Return type *afem.geometry.entities.NurbsCurve2D* (page 33)

7.2.18 NurbsCurve2DByApprox

class afem.geometry.create.**NurbsCurve2DByApprox**(*qp*, *dmin=3*, *dmax=8*, *continuity=GeomAbs_Shape.GeomAbs_C2*, *parm_type=Approx_ParametrizationType.Approx_ChordLength*, *tol=1e-06*)

Bases: `object`

Create a 2-D NURBS curve by approximating 2-D points.

Parameters

- **`qp`** (*`collections.Sequence(point2d_like)`*) – Points to approximate.
- **`dmin`** (*`int`*) – Minimum degree.
- **`dmax`** (*`int`*) – Maximum degree.
- **`continuity`** (*`OCCT.GeomAbs.GeomAbs_Shape`*) – Desired continuity of curve.
- **`parm_type`** (*`OCCT.Approx.Approx_ParametrizationType`*) – Parametrization type.
- **`tol`** (*`float`*) – The tolerance used for approximation. The distance from the points to the resulting curve should be lower than *tol*.

Raises **`RuntimeError`** – If OCC method fails to interpolate the points with a curve.

`curve`

Returns The 2-D NURBS curve.

Return type *afem.geometry.entities.NurbsCurve2D* (page 33)

7.2.19 NurbsCurve2DByPoints

class afem.geometry.create.**NurbsCurve2DByPoints**(*qp*)

Bases: *`afem.geometry.create.NurbsCurve2DByApprox`* (page 65)

Create a 2-D linear curve (i.e., a polyline) between points. This method uses *`NurbsCurve2DByApprox`* (page 65) to fit a linear curve.

Parameters **`qp`** (*`collections.Sequence(point2d_like)`*) – Points.

7.2.20 NurbsCurveByInterp

class afem.geometry.create.**NurbsCurveByInterp**(*qp*, *is_periodic=False*, *v1=None*, *v2=None*, *tol=1e-07*)

Bases: `object`

Create a cubic curve by interpolating points.

Parameters

- **qp** (*collections.Sequence(point_like)*) – Points to interpolate.
- **is_periodic** (*bool*) – Flag for curve periodicity. If *True* the curve will be periodic and closed.
- **v1** (*vector_like*) – Tangent to match at first point.
- **v2** (*vector_like*) – Tangent to match at last point.
- **tol** (*float*) – Tolerance used to check for coincident points and the magnitude of end vectors.

Raises **RuntimeError** – If OCC method fails.

curve

Returns The NURBS curve.

Return type *afem.geometry.entities.NurbsCurve* (page 47)

7.2.21 NurbsCurveByApprox

class afem.geometry.create.**NurbsCurveByApprox**(*qp*, *dmin=3*, *dmax=8*, *continuity=GeomAbs_Shape.GeomAbs_C2*, *parm_type=Approx_ParametrizationType.Approx_ChordLength*, *tol=0.001*)

Bases: object

Create a NURBS curve by approximating points.

Parameters

- **qp** (*collections.Sequence(point_like)*) – Points to approximate.
- **dmin** (*int*) – Minimum degree.
- **dmax** (*int*) – Maximum degree.
- **continuity** (*OCCT.GeomAbs.GeomAbs_Shape*) – Desired continuity of curve.
- **parm_type** (*OCCT.Approx.Approx_ParametrizationType*) – Parametrization type.
- **tol** (*float*) – The tolerance used for approximation. The distance from the points to the resulting curve should be lower than *tol*.

Raises **RuntimeError** – If OCC method fails to interpolate the points with a curve.

curve

Returns The NURBS curve.

Return type *afem.geometry.entities.NurbsCurve* (page 47)

7.2.22 NurbsCurveByPoints

class afem.geometry.create.**NurbsCurveByPoints**(*qp*)

Bases: *afem.geometry.create.NurbsCurveByApprox* (page 66)

Create a linear curve (i.e., a polyline) between points. This method uses *NurbsCurveByApprox* (page 66) to fit a linear curve.

Parameters **qp** (*collections.Sequence(point_like)*) – Points.

7.2.23 TrimmedCurveByPoints

class afem.geometry.create.**TrimmedCurveByPoints**(*basis_curve*, *p1*, *p2*, *sense=True*, *adjust_periodic=True*)

Bases: `object`

Create a trimmed curve using a basis curve and limiting points. The points are projected to the basis curve to find the limiting parameters.

Parameters

- **basis_curve** (`afem.geometry.entities.Curve` (page 43)) – The basis curve.
- **p1** (*point_like*) – The first point.
- **p2** (*point_like*) – The last point.
- **sense** (*bool*) – If the basis curve is periodic, the trimmed curve will have the same orientation as the basis curve if `True` or opposite if `False`.
- **adjust_periodic** (*bool*) – If the basis curve is periodic, the bounds of the trimmed curve may be different from *u1* and *u2* if `True`.

Raises

- **TypeError** – If the basis curve is not a valid curve type.
- **ValueError** – If *u1* >= *u2*.

curve

Returns The trimmed curve.

Return type *afem.geometry.entities.TrimmedCurve* (page 48)

7.2.24 PlaneByNormal

class afem.geometry.create.**PlaneByNormal**(*origin=(0.0, 0.0, 0.0)*, *vnorm=(0.0, 0.0, 1.0)*)

Bases: `object`

Create a plane by an origin and a normal vector.

Parameters

- **origin** (*point_like*) – The origin.
- **vnorm** (*vector_like*) – The normal vector.

Raises

- **TypeError** – If *origin* cannot be converted to `Point` (page 35).
- **TypeError** – If *vnorm* cannot be converted to `Direction` (page 38).

plane

Returns The plane.

Return type *afem.geometry.entities.Plane* (page 52)

7.2.25 PlaneByAxes

class `afem.geometry.create.`**`PlaneByAxes`**(*origin=(0.0, 0.0, 0.0), axes='xz'*)

Bases: `object`

Create a plane by an origin and basic axes.

Parameters

- **origin** (*point_like*) – The origin.
- **axes** (*str*) – The axes ('xy', 'xz', 'yz').

Raises

- **TypeError** – If *origin* cannot be converted to `Point` (page 35).
- **ValueError** – If *axes* is not a supported option.

plane

Returns The plane.

Return type *afem.geometry.entities.Plane* (page 52)

7.2.26 PlaneByPoints

class `afem.geometry.create.`**`PlaneByPoints`**(*p1, p2, p3*)

Bases: `object`

Create a plane by three points. The points must not be collinear. The plane will be defined by (*p2* - *p1*) x (*p3* - *p1*).

Parameters

- **p1** (*point_like*) – First point. This point will be used as the origin.
- **p2** (*point_like*) – Second point.
- **p3** (*point_like*) – Third point.

Raises

- **TypeError** – if *p1*, *p2*, or *p3* cannot be converted to a `Point` (page 35).
- **ValueError** – If the points are collinear.

plane

Returns The plane.

Return type *afem.geometry.entities.Plane* (page 52)

7.2.27 PlaneByApprox

class `afem.geometry.create.`**`PlaneByApprox`**(*pnts, tol=1e-07*)

Bases: `object`

Create a plane by fitting points. The points must not be collinear.

Parameters

- **pnts** (*list (point_like)*) – Points to fit plane. Should not be collinear.
- **tol** (*float*) – Tolerance used to check for collinear points.

Raises

- **ValueError** – If the number of points is less than three.
- **RuntimeError** – If a plane cannot be fit to the points.

plane

Returns The plane.

Return type *afem.geometry.entities.Plane* (page 52)

7.2.28 PlaneFromParameter

class afem.geometry.create.**PlaneFromParameter**(*c*, *u0*, *ds*, *ref_pln=None*, *tol=1e-07*)

Bases: object

Create a plane along a curve at a specified distance from a parameter.

Parameters

- **c** (afem.adaptor.entities.AdaptorCurve (page 176) *or* afem.geometry.entities.Curve (page 43) *or* afem.topology.entities.Edge (page 115) *or* afem.topology.entities.Wire (page 116)) – The curve.
- **u0** (*float*) – The initial parameter.
- **ds** (*float*) – The distance along the curve from the given parameter.
- **ref_pln** (afem.geometry.entities.Plane (page 52)) – The normal of this plane will be used to define the normal of the new plane. If no plane is provided, then the first derivative of the curve will define the plane normal.
- **tol** (*float*) – Tolerance.

Returns The plane.

Return type *afem.geometry.entities.Plane* (page 52)

plane

Returns The plane.

Return type *afem.geometry.entities.Plane* (page 52)

parameter

Returns The parameter on the curve.

Return type float

7.2.29 PlaneByOrientation

class afem.geometry.create.**PlaneByOrientation**(*origin=(0.0, 0.0, 0.0)*, *axes='xz'*, *alpha=0.0*, *beta=0.0*, *gamma=0.0*)

Bases: object

Create a plane by rotation angles.

Parameters

- **origin** (*point_like*) – The origin.
- **axes** (*str*) – The reference axes ('xy', 'xz', 'yz').

- **alpha** (*float*) – Rotation in degrees about global x-axis.
- **beta** (*float*) – Rotation in degrees about global y-axis.
- **gamma** (*float*) – Rotation in degrees about global z-axis.

plane

Returns The plane.

Return type *afem.geometry.entities.Plane* (page 52)

7.2.30 PlaneByCurveAndSurface

class afem.geometry.create.**PlaneByCurveAndSurface**(*crv*, *srf*, *u*)

Bases: object

Create a plane using a curve that lies on a surface. The x-axis of the plane is found by the cross product of the surface normal and the first derivative of the curve at the parameter. The normal of the plane is found by the cross product of the x-axis and the surface normal. The origin will be located at a point along the curve at the given parameter.

Parameters

- **crv** (afem.geometry.entities.Curve (page 43)) – The curve.
- **srf** (afem.geometry.entities.Surface (page 49)) – The surface.
- **u** (*float*) – The curve parameter.

plane

Returns The plane.

Return type *afem.geometry.entities.Plane* (page 52)

7.2.31 PlanesAlongCurveByNumber

class afem.geometry.create.**PlanesAlongCurveByNumber**(*c*, *n*, *ref_pln=None*, *u1=None*, *u2=None*, *d1=None*, *d2=None*, *tol=1e-07*)

Bases: object

Create planes along a curve using a specified number. The origin of the planes will be equidistant along the curve.

Parameters

- **c** (afem.adaptor.entities.AdaptorCurve (page 176) *or* afem.geometry.entities.Curve (page 43) *or* afem.topology.entities.Edge (page 115) *or* afem.topology.entities.Wire (page 116)) – The curve.
- **n** (*int*) – Number of planes to create ($n > 0$).
- **ref_pln** (afem.geometry.entities.Plane (page 52)) – The normal of this plane will be used to define the normal of all planes along the curve. If no plane is provided, then the first derivative of the curve will define the plane normal.
- **u1** (*float*) – The parameter of the first plane (default=c.u1).
- **u2** (*float*) – The parameter of the last plane (default=c.u2).

- **d1** (*float*) – An offset distance for the first plane. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last plane. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **tol** (*float*) – Tolerance.

Raises `RuntimeError` – If *PointsAlongCurveByNumber* (page 59) fails to generate points along the curve.

nplanes

Returns The number of planes.

Return type int

planes

Returns The planes.

Return type list(*afem.geometry.entities.Plane* (page 52))

parameters

Returns The parameters.

Return type list(float)

spacing

Returns The spacing between the first and second points if there are more than one point. Otherwise *None*.

Return type float or None

interior_planes

Returns The planes between the first and last planes.

Return type list(*afem.geometry.entities.Plane* (page 52))

7.2.32 PlanesAlongCurveByDistance

class afem.geometry.create.**PlanesAlongCurveByDistance**(*c, maxd, ref_pln=None, u1=None, u2=None, d1=None, d2=None, nmin=0, tol=1e-07*)

Bases: `object`

Create planes along a curve by distance between points. The origin of the planes will be equidistant along the curve. This method calculates the number of points given the curve length and then uses *PlanesAlongCurveByNumber* (page 70).

Parameters

- **c** (`afem.adaptor.entities.AdaptorCurve` (page 176) *or* `afem.geometry.entities.Curve` (page 43) *or* `afem.topology.entities.Edge` (page 115) *or* `afem.topology.entities.Wire` (page 116)) – The curve.
- **maxd** (*float*) – The maximum allowed spacing between planes. The actual spacing will be adjusted to not to exceed this value.

- **ref_pln** (afem.geometry.entities.Plane (page 52)) – The normal of this plane will be used to define the normal of all planes along the curve. If no plane is provided, then the first derivative of the curve will define the plane normal.
- **u1** (*float*) – The parameter of the first plane (default=c.u1).
- **u2** (*float*) – The parameter of the last plane (default=c.u2).
- **d1** (*float*) – An offset distance for the first plane. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last plane. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **nmin** (*int*) – Minimum number of planes to create.
- **tol** (*float*) – Tolerance.

Raises RuntimeError – If *PointsAlongCurveByDistance* (page 60) fails to generate points along the curve.

nplanes

Returns The number of planes.

Return type int

planes

Returns The planes.

Return type list(*afem.geometry.entities.Plane* (page 52))

parameters

Returns The parameters.

Return type list(float)

spacing

Returns The spacing between the first and second points if there are more than one point. Otherwise *None*.

Return type float or None

interior_planes

Returns The planes between the first and last planes.

Return type list(*afem.geometry.entities.Plane* (page 52))

7.2.33 PlanesBetweenPlanesByNumber

class afem.geometry.create.**PlanesBetweenPlanesByNumber**(*pln1*, *pln2*, *n*, *d1=None*, *d2=None*)

Bases: object

Create planes between two other planes. This method will create a line normal to the first plane at its origin and then intersect that line with the second plane. Planes are then created along this line using *PlanesAlongCurveByNumber* (page 70). Planes are not generated at the same location as the boundary planes.

Parameters

- **pln1** (`afem.geometry.entities.Plane` (page 52)) – The first plane. This will be the reference plane to define the orientation of the new planes.
- **pln2** (`afem.geometry.entities.Plane` (page 52)) – The last plane.
- **n** (*int*) – The number of planes.
- **d1** (*float*) – An offset distance for the first plane. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last plane. This is typically a negative number indicating a distance from *u2* towards *u1*.

Raises RuntimeError – If the line extending from the normal of the first plane cannot be intersected with the second plane.

nplanes

Returns The number of planes.

Return type int

planes

Returns The planes.

Return type list(*afem.geometry.entities.Plane* (page 52))

spacing

Returns The spacing between the first and second points if there are more than one point. Otherwise *None*.

Return type float or None

interior_planes

Returns The planes between the first and last planes.

Return type list(*afem.geometry.entities.Plane* (page 52))

7.2.34 PlanesBetweenPlanesByDistance

class afem.geometry.create.**PlanesBetweenPlanesByDistance**(*pln1*, *pln2*, *maxd*, *d1=None*, *d2=None*, *nmin=0*)

Bases: `object`

Create planes between two other planes by distance. This method will create a line normal to the first plane at its origin and then intersect that line with the second plane. Planes are then created along this line using *`PlanesAlongCurveByNumber`* (page 70). Planes are not generated at the same location as the boundary planes.

Parameters

- **pln1** (`afem.geometry.entities.Plane` (page 52)) – The first plane. This will be the reference plane to define the orientation of the new planes.
- **pln2** (`afem.geometry.entities.Plane` (page 52)) – The last plane.
- **maxd** (*float*) – The maximum allowed spacing between planes. The actual spacing will be adjusted to not to exceed this value.
- **d1** (*float*) – An offset distance for the first plane. This is typically a positive number indicating a distance from *u1* towards *u2*.

- **d2** (*float*) – An offset distance for the last plane. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **nmin** (*int*) – Minimum number of planes to create.

Raises RuntimeError – If the line extending from the normal of the first plane cannot be intersected with the second plane.

nplanes

Returns The number of planes.

Return type int

planes

Returns The planes.

Return type list(*afem.geometry.entities.Plane* (page 52))

spacing

Returns The spacing between the first and second points if there are more than one point. Otherwise *None*.

Return type float or None

interior_planes

Returns The planes between the first and last planes.

Return type list(*afem.geometry.entities.Plane* (page 52))

7.2.35 PlanesAlongCurveAndSurfaceByDistance

class afem.geometry.create.**PlanesAlongCurveAndSurfaceByDistance**(*c*, *s*, *maxd*, *u1=None*, *u2=None*, *d1=None*, *d2=None*, *nmin=0*, *tol=1e-07*)

Bases: object

Create planes along a curve and surface by distance between points. The origin of the planes will be equidistant along the curve. This method calculates the number of points given the curve length and then uses *PlaneByCurveAndSurface* (page 70) at each point.

Parameters

- **c** (afem.geometry.entities.Curve (page 43)) – The curve.
- **s** (afem.geometry.entities.Surface (page 49)) – The surface.
- **maxd** (*float*) – The maximum allowed spacing between planes. The actual spacing will be adjusted to not to exceed this value.
- **u1** (*float*) – The parameter of the first plane (default=c.u1).
- **u2** (*float*) – The parameter of the last plane (default=c.u2).
- **d1** (*float*) – An offset distance for the first plane. This is typically a positive number indicating a distance from *u1* towards *u2*.

- **d2** (*float*) – An offset distance for the last plane. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **nmin** (*int*) – Minimum number of planes to create.
- **tol** (*float*) – Tolerance.

Raises RuntimeError – If *PointsAlongCurveByDistance* (page 60) fails to generate points along the curve.

nplanes

Returns The number of planes.

Return type int

planes

Returns The planes.

Return type list(*afem.geometry.entities.Plane* (page 52))

parameters

Returns The parameters.

Return type list(float)

spacing

Returns The spacing between the first and second points if there are more than one point. Otherwise *None*.

Return type float or None

interior_planes

Returns The planes between the first and last planes.

Return type list(*afem.geometry.entities.Plane* (page 52))

rotate_x(*angle*)

Rotate each of the planes around their local x-axis.

Parameters angle (*float*) – The rotation angle in degrees.

Returns None.

rotate_y(*angle*)

Rotate each of the planes around their local y-axis.

Parameters angle (*float*) – The rotation angle in degrees.

Returns None.

7.2.36 NurbsSurfaceByInterp

class afem.geometry.create.**NurbsSurfaceByInterp**(*crvs*, *q=3*, *parm_type=Approx_ParametrizationType.Approx_ChordLengt* *tol2d=1e-09*)

Bases: `object`

Create a surface by interpolating curves. This method was developed from scratch using "The NURBS Book" since OpenCASCADE did not support interpolating surfaces with $q = 1$.

Parameters

- **crvs** (*list (curve_like)*) – List of curves to interpolate.
- **q** (*int*) – Degree. The parameter will be adjusted if the number of curves provided does not support the desired degree.
- **parm_type** (*OCCT.Approx.Approx_ParametrizationType*) – Parametrization type.
- **tol2d** (*float*) – 2-D tolerance.

surface

Returns The NURBS surface.

Return type *afem.geometry.entities.NurbsSurface* (page 53)

7.2.37 NurbsSurfaceByApprox

class `afem.geometry.create.`**NurbsSurfaceByApprox**(*crvs*, *dmin=3*, *dmax=8*, *tol3d=0.001*, *tol2d=1e-06*, *niter=5*, *continuity=GeomAbs_Shape.GeomAbs_C2*, *parm_type=Approx_ParametrizationType.Approx_ChordLengtl*

Bases: `object`

Create a NURBS surface by approximating curves.

Parameters

- **crvs** (*list (curve_like)*) – List of curves.
- **dmin** (*int*) – Minimum degree.
- **dmax** (*int*) – Maximum degree.
- **tol3d** (*float*) – 3-D tolerance.
- **tol2d** (*float*) – 2-D tolerance.
- **niter** (*int*) – Number of iterations.
- **continuity** (*OCCT.GeomAbs.GeomAbs_Shape*) – Desired continuity of curve.
- **parm_type** (*OCCT.Approx.Approx_ParametrizationType*) – Parametrization type.

Raises **RuntimeError** – If OCC method fails to approximate the curves with a surface.

surface

Returns The NURBS surface.

Return type *afem.geometry.entities.NurbsSurface* (page 53)

told3d_reached

Returns 3-D tolerance achieved.

Return type float

told2d_reached

Returns 2-D tolerance achieved.

Return type float

7.3 Project

7.3.1 PointProjector

class afem.geometry.project.**PointProjector**

Bases: `object`

Base class for point projections.

npts

Returns Number of projection results.

Return type int

success

Returns *True* if successful, *False* if not.

Return type bool

points

Returns Projected points.

Return type list(*afem.geometry.entities.Point* (page 35))

parameters

Returns Parameters of projected points.

Return type list(float)

nearest_point

Returns Nearest projection result to original point.

Return type *afem.geometry.entities.Point* (page 35)

nearest_param

Returns Parameter(s) of nearest point on curve(surface).

Return type float or tuple(float, float)

dmin

Returns Minimum distance of all projection results.

Return type float

point (*indx=1*)

Return the point result by index.

Parameters **indx** (*int*) – Index for point.

Returns Projected point.

Return type *afem.geometry.entities.Point* (page 35)

parameter (*indx=1*)

Return the parameter result by index.

Parameters **indx** (*int*) – Index for parameter.

Returns Parameter of point. For a curve projection a single float *u* will be returned. For a surface projection a tuple containing the *u* and *v* parameters will be returned (u, v).

Return type float or tuple(float, float)

distance (*indx=1*)

Return the projection distance by index.

Parameters **indx** (*int*) – Index for distance.

Returns Projection distance between original point and projection result.

Return type float

7.3.2 ProjectPointToCurve

class afem.geometry.project.**ProjectPointToCurve** (*pnt, crv, direction=None, update=False*)

Bases: *afem.geometry.project.PointProjector* (page 77)

Project a point to a curve.

Parameters

- **pnt** (*point_like*) – Point to project.
- **crv** (afem.adaptor.entities.AdaptorCurve (page 176) *or* afem.geometry.entities.Curve (page 43) *or* afem.topology.entities.Edge (page 115) *or* afem.topology.entities.Wire (page 116)) – Curve to project to.
- **direction** (*array_like*) – Direction of projection. If *None* then a normal projection will be performed. By providing a direction the tool actually performs a line-curve intersection. This is generally not recommended but provided by request.
- **update** (*bool*) – Option to update the point's location to match the nearest point.

7.3.3 ProjectPointToSurface

class afem.geometry.project.**ProjectPointToSurface** (*pnt, srf, direction=None, update=False, tol=1e-07*)

Bases: *afem.geometry.project.PointProjector* (page 77)

Project a point to a surface.

Parameters

- **pnt** (*point_like*) – Point to project.
- **srf** (afem.adaptor.entities.AdaptorSurface (page 179) *or* afem.geometry.entities.Surface (page 49) *or* afem.topology.entities.Face (page 117)) – Surface to project to.
- **direction** (*array_like*) – Direction of projection. If *None* then a normal projection will be performed. By providing a direction the tool actually performs a line-surface intersection. This is generally not recommended but provided by request.
- **update** (*bool*) – Option to update the point's location to match the nearest point.

7.3.4 CurveProjector

class afem.geometry.project.**CurveProjector**

Bases: object

Base class for curve projections.

success

Returns *True* if successful, *False* if not.

Return type bool

curve

Returns The projected curve.

Return type *afem.geometry.entities.Curve* (page 43)

7.3.5 ProjectCurveToPlane

class `afem.geometry.project.`**`ProjectCurveToPlane`**(*crv*, *pln*, *direction=None*, *keep_param=True*)

Bases: `afem.geometry.project.CurveProjector` (page 78)

Project a curve to a plane along a direction.

Parameters

- **crv** (`afem.geometry.entities.Curve` (page 43)) – Curve to project.
- **pln** (`afem.geometry.entities.Plane` (page 52)) – Plane to project to.
- **direction** (*array_like*) – Direction of projection. If *None* is provided, then the curve is projected normal to the plane.

Raises **RuntimeError** – If the OCC method fails to project the curve to the plane.

7.3.6 ProjectCurveToSurface

class `afem.geometry.project.`**`ProjectCurveToSurface`**(*crv*, *srf*)

Bases: `afem.geometry.project.CurveProjector` (page 78)

Project a curve to a surface. Only normal projections are supported.

Parameters

- **crv** (`afem.geometry.entities.Curve` (page 43)) – Curve to project.
- **srf** (`afem.geometry.entities.Surface` (page 49)) – Surface to project to.

Raises **RuntimeError** – If the OCC method fails to project the curve to the plane.

7.4 Intersect

7.4.1 CurveIntersector

class `afem.geometry.intersect.`**`CurveIntersector`**(*c1*, *c2*)

Bases: `object`

Base class for handling curve intersection methods and results.

npts

Returns Number of intersection points.

Return type int

success

Returns *True* if successful, *False* if not.

Return type bool

points

Returns List of intersection points.

Return type list(*afem.geometry.entities.Point* (page 35))

parameters

Returns List of intersection parameters. For a curve-curve intersection this will be a list of tuples containing the parameters of each curve [(u1, u2), (u1, u2), . . .]. For a curve-surface intersection this will be a list of tuples containing the parameters for the surface and then the curve [(u, v, t), (u, v, t), . . .].

Return type list(tuple(float))

point (*indx=1*)

Return the point result by index.

Parameters **indx** (*int*) – Index for point.

Returns Intersection point.

Return type *afem.geometry.entities.Point* (page 35)

query_point (*p0*, *distance_upper_bound=inf*)

Find the intersection result nearest to the provided point.

Parameters

- **p0** (*point_like*) – Point to search from.
- **distance_upper_bound** (*float*) – Return only results within this distance.

Returns Distance to nearest intersection result and its index (d, i). Returns (None, None) if no results are available.

Return type tuple

7.4.2 IntersectCurveCurve

class afem.geometry.intersect.**IntersectCurveCurve** (*crv1*, *crv2*, *itol=1e-07*)

Bases: *afem.geometry.intersect.CurveIntersector* (page 79)

Curve-curve intersection. This method converts the curves to edges and performs the intersections that way. This proved to be more robust than OpenCASCADE's native curve-curve intersection tool.

Parameters

- **crv1** (afem.geometry.entities.Curve (page 43)) – The first curve.
- **crv2** (afem.geometry.entities.Curve (page 43)) – The second curve.
- **itol** (*float*) – The intersection tolerance.

7.4.3 IntersectCurveSurface

class `afem.geometry.intersect.`**`IntersectCurveSurface`**(*crv*, *srf*)

Bases: `afem.geometry.intersect.CurveIntersector` (page 79)

Curve-surface intersection.

Parameters

- **`crv`** (`afem.geometry.entities.Curve` (page 43)) – The curve.
- **`srf`** (`afem.geometry.entities.Surface` (page 49)) – The surface.

7.4.4 SurfaceIntersector

class `afem.geometry.intersect.`**`SurfaceIntersector`**

Bases: `object`

Base class for handling surface intersection methods and results.

`ncrvs`

Returns Number of intersection curves.

Return type int

`success`

Returns *True* if successful, *False* if not.

Return type bool

`curves`

Returns The intersection curves.

Return type list(*afem.geometry.entities.Curve* (page 43))

`curve`(*indx=1*)

Generate an intersection curve.

Parameters **`indx`** (*`int`*) – Index of intersection curve.

Returns Intersection curve.

Return type *afem.geometry.entities.Curve* (page 43)

`curve_nearest_point`(*pnt*)

Get the index of the intersection curve that is nearest to the given reference point.

Parameters **`pnt`** (*`point_like`*) – Reference point.

Returns Index of curve nearest point.

Return type int

7.4.5 IntersectSurfaceSurface

class `afem.geometry.intersect.`**`IntersectSurfaceSurface`**(*srf1*, *srf2*, *itol=1e-07*, *approx=True*)

Bases: `afem.geometry.intersect.SurfaceIntersector` (page 81)

Surface-surface intersection.

Parameters

- **`srf1`** (`afem.geometry.entities.Surface` (page 49)) – The first surface.
- **`srf2`** (`afem.geometry.entities.Surface` (page 49)) – The second surface.
- **`itol`** (*`float`*) – Intersection tolerance.
- **`approx`** (*`bool`*) – Approximate intersection curves.

`tol3d`

Returns Tolerance reached for 3-D intersection curves.

Return type float

7.5 Distance

7.5.1 DistancePointToCurve

class `afem.geometry.distance.`**`DistancePointToCurve`**(*pnt*, *crv*, *tol=1e-10*)

Bases: `object`

Calculate the extrema between a point and a curve.

Parameters

- **`pnt`** (*`point_like`*) – The point.
- **`crv`** (`afem.adaptor.entities.AdaptorCurve` (page 176) *or* `afem.geometry.entities.Curve` (page 43) *or* `afem.topology.entities.Edge` (page 115) *or* `afem.topology.entities.Wire` (page 116)) – The curve.
- **`tol`** (*`float`*) – The tolerance.

Raises **`RuntimeError`** – If `Extrema_ExtPC` fails.

`nsol`

Returns The number of solutions.

Return type int

`dmin`

Returns The minimum distance.

Return type float

`dmax`

Returns The maximum distance.

Return type float

`distances`

Returns Sorted distances.

Return type list(float)

`parameters`

Returns Sorted parameters on curve.

Return type list(float)

points

Returns Sorted points on curve.

Return type list(*afem.geometry.entities.Point* (page 35))

7.5.2 DistancePointToSurface

class `afem.geometry.distance.`**DistancePointToSurface**(*pnt*, *srf*, *tol=1e-10*)

Bases: `object`

Calculate the extrema between a point and a surface.

Parameters

- **pnt** (*point_like*) – The point.
- **srf** (`afem.adaptor.entities.AdaptorSurface` (page 179) *or* `afem.geometry.entities.Surface` (page 49) *or* `afem.topology.entities.Face` (page 117)) – The surface.
- **tol** (*float*) – The tolerance.

Raises RuntimeError – If `Extrema_ExtPS` fails.

nsol

Returns The number of solutions.

Return type int

dmin

Returns The minimum distance.

Return type float

dmax

Returns The maximum distance.

Return type float

distances

Returns Sorted distances.

Return type list(float)

parameters

Returns Sorted parameters on surface. This is a list of tuples containing the (u, v) locations.

Return type list(tuple(float))

points

Returns Sorted points on surface.

Return type list(*afem.geometry.entities.Point* (page 35))

7.5.3 DistanceCurveToCurve

class afem.geometry.distance.**DistanceCurveToCurve**(*crv1*, *crv2*, *tol=1e-10*)
Bases: `object`

Calculate the extrema between two curves.

Parameters

- **crv1** – The first curve.
- **crv2** (`afem.adaptor.entities.AdaptorCurve` (page 176) *or* `afem.geometry.entities.Curve` (page 43) *or* `afem.topology.entities.Edge` (page 115) *or* `afem.topology.entities.Wire` (page 116)) – The second curve.
- **tol** (*float*) – The tolerance.

Raises RuntimeError – If `Extrema_ExtCC` fails.

nsol

Returns The number of solutions.

Return type int

is_parallel

Returns *True* if curves were parallel.

Return type bool

dmin

Returns The minimum distance.

Return type float

dmax

Returns The maximum distance.

Return type float

distances

Returns Sorted distances.

Return type list(float)

points1

Returns Sorted points on first curve.

Return type list(*afem.geometry.entities.Point* (page 35))

points2

Returns Sorted points on second curve.

Return type list(*afem.geometry.entities.Point* (page 35))

7.5.4 DistanceCurveToSurface

class afem.geometry.distance.**DistanceCurveToSurface**(*crv*, *srf*, *tol=1e-10*)
Bases: `object`

Calculate the extrema between a curve and surface.

Parameters

- **crv** (afem.adaptor.entities.AdaptorCurve (page 176) *or* afem.geometry.entities.Curve (page 43) *or* afem.topology.entities.Edge (page 115) *or* afem.topology.entities.Wire (page 116)) – The curve.
- **srf** (afem.adaptor.entities.AdaptorSurface (page 179) *or* afem.geometry.entities.Surface (page 49) *or* afem.topology.entities.Face (page 117)) – surface.
- **tol** (*float*) – The tolerance.

Raises **RuntimeError** – If the extrema algorithm fails.

nsol

Returns The number of solutions.

Return type int

is_parallel

Returns *True* if the curve and surface were parallel.

Return type bool

dmin

Returns The minimum distance.

Return type float

dmax

Returns The maximum distance.

Return type float

distances

Returns Sorted distances.

Return type list(float)

points1

Returns Sorted points on the curve.

Return type list(*afem.geometry.entities.Point* (page 35))

points2

Returns Sorted points on the surface.

Return type list(*afem.geometry.entities.Point* (page 35))

7.5.5 DistanceSurfaceToSurface

class afem.geometry.distance.**DistanceSurfaceToSurface**(*srf1*, *srf2*, *tol=1e-10*)

Bases: object

Calculate the extrema between two surfaces.

Parameters

- **srf1** (`afem.adaptor.entities.AdaptorSurface` (page 179) *or* `afem.geometry.entities.Surface` (page 49) *or* `afem.topology.entities.Face` (page 117)) – The first surface.
- **srf2** (`afem.adaptor.entities.AdaptorSurface` (page 179) *or* `afem.geometry.entities.Surface` (page 49) *or* `afem.topology.entities.Face` (page 117)) – The second surface.
- **tol** (*float*) – The tolerance.

Raises RuntimeError – If `Extrema_ExtSS` fails.

nsol

Returns The number of solutions.

Return type int

is_parallel

Returns *True* if the curve and surface were parallel.

Return type bool

dmin

Returns The minimum distance.

Return type float

dmax

Returns The maximum distance.

Return type float

distances

Returns Sorted distances.

Return type list(float)

points1

Returns Sorted points on the first surface.

Return type list(*afem.geometry.entities.Point* (page 35))

points2

Returns Sorted points on the second surface.

Return type list(*afem.geometry.entities.Point* (page 35))

7.6 Check

7.6.1 CheckGeom

class `afem.geometry.check.`**CheckGeom**

Bases: `object`

Geometry checker.

static is_point_like(*entity*)

Check if the entity is point_like.

Parameters **entity** – An entity.

Returns *True* if the entity is point_like, *False* if not.

Return type bool

static is_point (*geom*)
Check if the entity is a `Point` (page 35).

Parameters **geom** – An entity.

Returns *True* if the entity is a Point, *False* if not.

Return type bool

static to_point (*geom*)
Convert entity to a `Point` (page 35) if possible.

Parameters **geom** – An entity.

Returns The entity if already a Point, or a new Point if it is point_like.

Return type *afem.geometry.entities.Point* (page 35)

Raises **TypeError** – If entity cannot be converted to a Point.

static to_points (*geoms*)
Convert entities to points if possible.

Parameters **geoms** (*list (point_like)*) – List of entities.

Returns List of points.

Return type list(*afem.geometry.entities.Point* (page 35))

static is_point2d_like (*geom*)
Check if the entity is point2d_like.

Parameters **geom** – An entity.

Returns *True* if the entity is point2d_like, *False* if not.

Return type bool

static is_point2d (*geom*)
Check if the entity is a `Point2D` (page 26).

Parameters **geom** – An entity.

Returns *True* if the entity is a Point2D, *False* if not.

Return type bool

static to_point2d (*geom*)
Convert entity to a `Point2D` (page 26) if possible.

Parameters **geom** – An entity.

Returns The entity if already a Point2D, or a new Point2D if it is point2d_like.

Return type *afem.geometry.entities.Point2D* (page 26)

Raises **TypeError** – If entity cannot be converted to a Point2D.

static is_vector (*geom*)
Check if the entity is a `Vector` (page 39).

Parameters **geom** – An entity.

Returns *True* if the entity is a Vector, *False* if not.

Return type bool

static to_vector(*geom*)
Convert entity to a Vector if possible.

Parameters **geom** (*vector_like*) – An entity.

Returns The entity if already a Vector, or a new Vector if it is vector_like.

Return type *afem.geometry.entities.Vector* (page 39)

Raises **TypeError** – If entity cannot be converted to a Vector.

static to_vector2d(*geom*)
Convert entity to a Vector2D if possible.

Parameters **geom** (*vector2d_like*) – An entity.

Returns The entity if already a Vector2D, or a new Vector2D if it is vector2d_like.

Return type *afem.geometry.entities.Vector2D* (page 29)

Raises **TypeError** – If entity cannot be converted to a Vector2D.

static is_direction(*geom*)
Check if the entity is a `Direction` (page 38).

Parameters **geom** – An entity.

Returns *True* if the entity is a Direction, *False* if not.

Return type bool

static to_direction(*geom*)
Convert entity to a Direction if possible.

Parameters **geom** (*vector_like*) – An entity.

Returns The entity if already a Direction, or a new Direction if it is vector_like.

Return type *afem.geometry.entities.Direction* (page 38)

Raises **TypeError** – If entity cannot be converted to a Direction.

static is_plane(*geom*)
Check if the entity is a `Plane` (page 52).

Parameters **geom** – An entity.

Returns *True* if the entity is a Plane, *False* if not.

Return type bool

static is_curve(*geom*)
Check if the entity is a curve.

Parameters **geom** – An entity.

Returns *True* if the entity is a curve, *False* if not.

Return type bool

static is_curve2d(*geom*)
Check if the entity is a 2-D curve.

Parameters **geom** – An entity.

Returns *True* if the entity is a 2-D curve, *False* if not.

Return type bool

static is_line(*geom*)
Check if the entity is a `Line` (page 45).

Parameters **geom** – An entity.

Returns *True* if the entity is a Line, *False* if not.

Return type bool

static is_trimmed_curve(*geom*)
Check if the entity is a `TrimmedCurve` (page 48).

Parameters **geom** – An entity.

Returns *True* if the entity is a TrimmedCurve, *False* if not.

Return type bool

static is_surface(*geom*)
Check if the entity is a surface.

Parameters **geom** – An entity.

Returns *True* if the entity is a surface, *False* if not.

Return type bool

static is_axis3(*geom*)
Check if the entity is an `Axis3` (page 42).

Parameters **geom** – An entity.

Returns *True* if Axis3, *False* if not.

Return type bool

static nearest_point(*p*, *pnts*)
Find the point nearest to a given point.

Parameters

- **p** (*point_like*) – The point.
- **pnts** (*list (point_like)*) – List of points.

Returns The nearest point.

Return type *afem.geometry.entities.Point* (page 35)

7.7 Utilities

afem.geometry.utils.**uniform_parameters**(*n*, *a=0.0*, *b=1.0*)
Generate uniform parameters.

Parameters

- **n** (*int*) – Number of parameters.
- **a** (*float*) – Lower bound.
- **b** (*float*) – Upper bound.

Returns Parameters between [a, b].

Return type numpy.ndarray

`afem.geometry.utils.`**`chord_parameters`**(*pnts*, *a=0.0*, *b=1.0*)
Generate parameters using chord length method.

Parameters

- **pnts** (*list (point_like)*) – List of ordered points.
- **a** (*float*) – Lower bound.
- **b** (*float*) – Upper bound.

Returns Parameters between [a, b].

Return type numpy.ndarray

`afem.geometry.utils.`**`centripetal_parameters`**(*pnts*, *a=0.0*, *b=1.0*)
Generate parameters using centripetal method.

Parameters

- **pnts** (*list[(point_like)*) – List of ordered points.
- **a** (*float*) – Lower domain.
- **b** (*float*) – Upper domain.

Returns Parameters between [a, b].

Return type numpy.ndarray

`afem.geometry.utils.`**`reparameterize_knots`**(*u1*, *u2*, *tcol_knots*)
Reparameterize the knot values between *u1* and *u2*

`afem.geometry.utils.`**`find_span`**(*n*, *p*, *u*, *uk*)
Determine the knot span index.

Parameters

- **n** (*int*) – Number of control points - 1.
- **p** (*int*) – Degree.
- **u** (*float*) – Parameter.
- **uk** (*ndarray*) – Knot vector.

Returns Knot span.

Return type int

Reference: Algorithm A2.1 from “The NURBS Book”.

`afem.geometry.utils.`**`basis_funs`**(*i*, *u*, *p*, *uk*)
Compute the non-vanishing basis functions.

Parameters

- **i** (*int*) – Knot span index.
- **u** (*float*) – Parameter.
- **p** (*int*) – Degree.
- **uk** (*ndarray*) – Knot vector.

Returns Non-vanishing basis functions.

Return type ndarray

Reference: Algorithm A2.2 from “The NURBS Book”

CHAPTER

EIGHT

SKETCH

The afem.sketch package provides basic entities and tools for the creation of 2-D planar geometry. This is not a constraint-solving sketch module like ones found in most CAD systems, but rather a simplified way to define planar 2-D curves and convert them to 3-D for operations like lofting or extrusion. The entities and tools can be imported by:

```
from afem.sketch import *
```

The general process to use the sketch package is to define a reference plane in 3-D space, use that plane to define 2-D geometry, and then convert that 2-D geometry into 3-D space. The example below demonstrates this process:

```
from afem.geometry import *
from afem.graphics import Viewer
from afem.sketch import *
from afem.topology import *

# Create a new cross section
cs = Airfoil()

# Generate a 2-D profile by reading and approximating an airfoil file from
# the UIUC database. Close the trailing edge if necessary.
cs.read_uiuc('../models/clarky.dat', close=True)

# Define a plane at the root and scale
pln1 = PlaneByAxes(axes='xz').plane
cs.build(pln1, scale=5)
wire1 = cs.wires[0]

# Define plane at the tip and rotate
pln2 = PlaneByAxes((3, 15, 0), axes='xz').plane
cs.build(pln2, scale=1.5, rotate=3)
wire2 = cs.wires[0]

# Use the wires to loft a solid
shape = LoftShape([wire1, wire2], True).shape

gui = Viewer()
gui.add(wire1, wire2, shape)
gui.start()
```

The sketch entities and tools are imported by:

```
from afem.geometry import *
from afem.graphics import Viewer
from afem.sketch import *
from afem.topology import *
```

A special type of `CrossSection` (page 95) will be used to generate a section from an airfoil file:

```
cs = Airfoil()
```

where `cs` uses the xy-plane by default since no plane was provided. This default reference plane is the one used when defining 2-D geometry. The plane's origin corresponds to the (0, 0) location of the local coordinate system.

In this example, a 2-D curve is generated by reading an airfoil file from the UIUC[10] database:

```
cs.read_uiuc('../models/clarky.dat', close=True)
```

The option `close=True` is used to close the trailing edge of the airfoil if it is open (i.e., the points are not coincident). This methods uses 2-D approximation to fit the airfoil points.

The 2-D profile can now be converted to 3-D space by providing a new 3-D plane in the `build` method:

```
pln1 = PlaneByAxes(axes='xz').plane
cs.build(pln1, scale=5)
wire1 = cs.wires[0]
```

Note that the new plane can be oriented in any way, but the 2-D geometry will be relative to the plane's origin and local axes. The `build` method accepts a global scaling and rotation parameter which is relative to the plane's origin. When the cross section is built, 3-D topology will be available for use. Another section is built using a different reference plane as well as different scaling and rotation parameters:

```
pln2 = PlaneByAxes((3, 15, 0), axes='xz').plane
cs.build(pln2, scale=1.5, rotate=3)
wire2 = cs.wires[0]
```

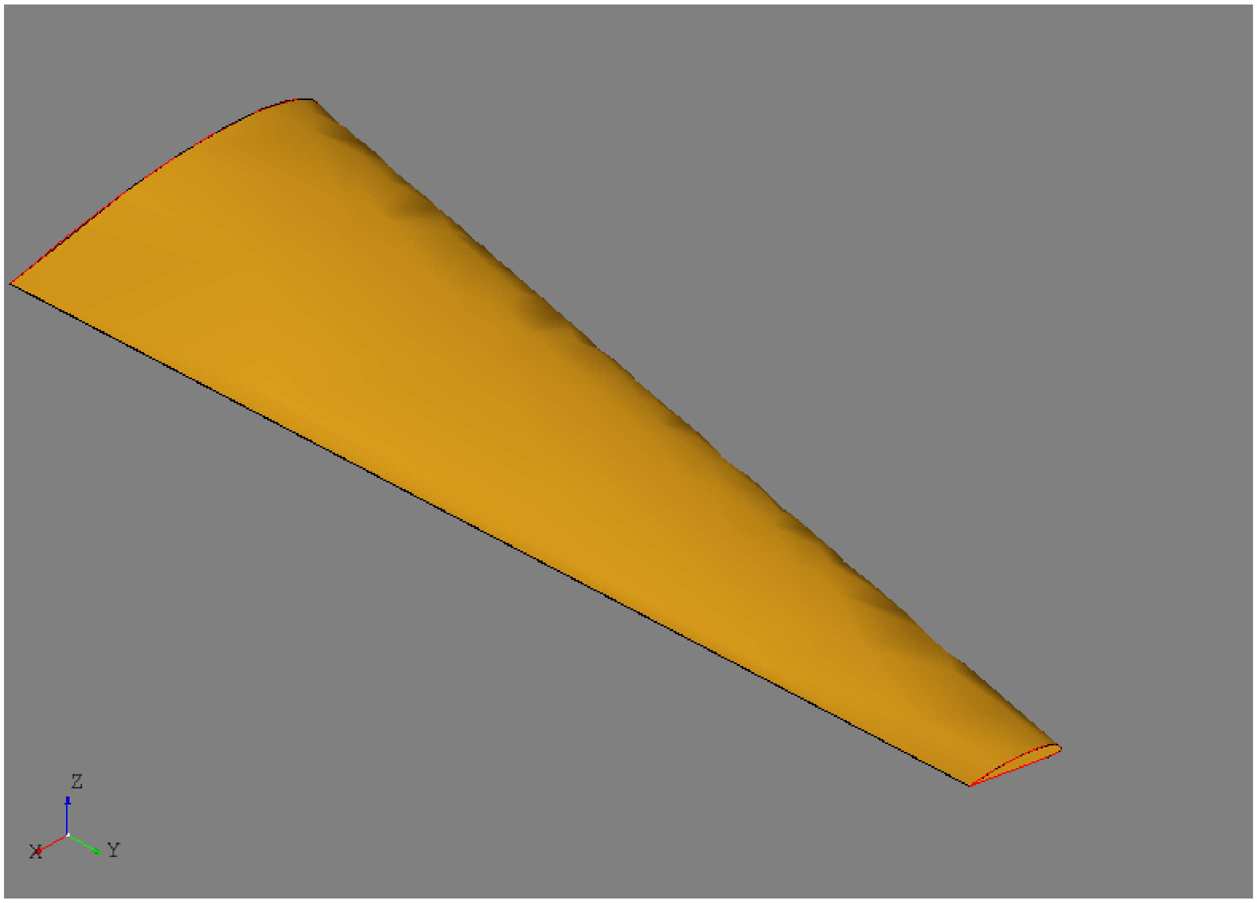
Note that the same 2-D profile is used to generate different 3-D shapes. The two wires created by this process are closed and used to generate a lofted solid:

```
shape = LoftShape([wire1, wire2], True).shape
```

The results can be visualized using the viewing tool and should look like the wing shape shown below:

```
gui = Viewer()
gui.add(wire1, wire2, shape)
gui.start()
```

[10] http://m-selig.ae.illinois.edu/ads/coord_database.html



8.1 Entities

8.1.1 CrossSection

class afem.sketch.entities.**CrossSection**(*pln=None*)

Bases: `object`

Planar cross section.

Parameters **pln** (`afem.geometry.entities.Plane` (page 52)) – The default construction plane. If *None* is provided, then the xy-plane is used.

shape

Returns The 3-D shape after building. If only one edge was available then an edge is returned, otherwise the edges are fused together and the resulting shape is returned.

Return type *afem.topology.entities.Edge* (page 115) or *afem.topology.entities.Shape* (page 109)

nwires

Returns Number of wires after building.

Return type int

wires

Returns The wires after building.

Return type list(*afem.topology.entities.Wire* (page 116))

has_face

Returns Check if face is available.

Return type bool

face

Returns The face if available.

Return type *afem.topology.entities.Face* (page 117)

clear()

Remove all 2-D curves from the cross section.

Returns None.

add_segment(*p1*, *p2*)

Add linear segment between the two points.

Parameters

- **p1** (`point2d_like`) – The first point.
- **p2** (`point2d_like`) – The second point.

Returns The 2-D curve.

Return type *afem.geometry.entities.NurbsCurve2D* (page 33)

add_approx(*pnts*, *close=False*)

Add a curve by approximating the 2-D points.

Parameters

- **pnts** (`collections.Sequence(point2d_like)`) – The points.
- **close** (`bool`) – Option to add a segment to close the 2-D curve.

Returns The 2-D curve.

Return type *afem.geometry.entities.NurbsCurve2D* (page 33)

add_interp(*pnts*, *close=False*)

Add a curve by interpolating the 2-D points.

Parameters

- **pnts** (`collections.Sequence(point2d_like)`) – The points.
- **close** (`bool`) – Option to add a segment to close the 2-D curve.

Returns The 2-D curve.

Return type *afem.geometry.entities.NurbsCurve2D* (page 33)

rotate(*angle*, *pnt=(0.0, 0.0)*)

Rotate each 2-D curve in the cross section.

Parameters

- **angle** (`float`) – The angle in degrees.
- **pnt** (`point2d_like`) – The reference point for rotation.

Returns None.

scale (*scale*, *pnt=(0.0, 0.0)*)
Scale each 2-D curve in the cross section.

Parameters

- **scale** (*float*) – The scaling parameter.
- **pnt** (*point2d_like*) – The reference point for scaling.

Returns None.

copy (*pln=None*)
Copy the cross section and underlying 2-D curves.

Parameters **pln** (`afem.geometry.entities.Plane` (page 52)) – The default plane for the new cross section. If *None*, then the default plane for this cross section is used.

Returns A new cross section.

Return type *afem.sketch.entities.CrossSection* (page 95)

build (*pln=None*, *scale=None*, *rotate=None*)
Build a shape in 3-D using the 2-D cross section curves. This method collects the 2-D curves, converts them into 3-D edges on the plane, fuses them together, and attempts to build wires and a face if applicable.

Parameters

- **pln** – The plane to build on. If *None*, then the default plane is used.
- **scale** (*float*) – Scale the 3-D curve after construction on the plane. The reference point is the plane origin.
- **rotate** (*float*) – Rotate the 3-D curve after construction on the plane. The reference point is the plane origin.

Returns *True* if successful, *False* otherwise.

Return type bool

8.1.2 Airfoil

class afem.sketch.entities.**Airfoil** (*pln=None*)
Bases: *afem.sketch.entities.CrossSection* (page 95)

Simple airfoil cross section. This class is similar to a generic `CrossSection` but assumes that the section curves are approximated by a distinct set of upper and lower points. This allows the class to store the leading and trailing edge points and build a chord line.

approx_points (*upr*, *lwr*, *close=True*)
Approximate a 2-D airfoil curve using a collection of upper and lower points. All other curves will be cleared from the cross section before this method is called.

Parameters

- **upr** (*collections.Sequence(point2d_like)*) – The upper points to approximate. The sequence should start at the leading edge and move towards the trailing edge.
- **lwr** (*collections.Sequence(point2d_like)*) – The lower points to approximate. The sequence should start at the leading edge and move towards the trailing edge.
- **close** (*bool*) – Option to close the airfoil by adding a segment.

Returns The 2-D curve.

Return type *afem.geometry.entities.NurbsCurve2D* (page 33)

Note: Before approximating the 2-D curve the upper and lower points are combined to form a single curve. It is important that the points are organized as described above so they are sorted correctly for the approximation.

read_uiuc(*fn*, *close=True*)

Generate 2-D curves by reading an airfoil file from the UIUC database.

Parameters

- **fn** (*str*) – The filename.
- **close** (*bool*) – Option to close the airfoil by adding a segment.

Returns The 2-D curve.

Return type *afem.geometry.entities.NurbsCurve2D* (page 33)

Note: The UIUC airfoil is assumed to be in a particular format and this method may fail if it's not. The points for the upper and lower surfaces from the file are combined to form a single curve that starts at the trailing edge, moves along the upper surface towards the leading edge, and then proceeds back to the trailing edge along the lower surface.

build_chord(*pln=None*, *scale=None*, *rotate=None*)

Build a chord line using the leading and trailing edge points of the airfoil.

Parameters

- **pln** – The plane to build on. If *None*, then the default plane is used.
- **scale** (*float*) – Scale the 3-D curve after construction on the plane. The reference point is the plane origin.
- **rotate** (*float*) – Rotate the 3-D curve after construction on the plane. The reference point is the plane origin.

Returns The 3-D chord line.

Return type *afem.geometry.entities.NurbsCurve* (page 47)

CHAPTER

NINE

TOPOLOGY

The afem.topology package provides tools for the creation and use of OpenCASCADE topology (i.e., shapes). While geometry defines curves and surfaces, topology describes their connectivity and boundary representation. OpenCASCADE shapes are the core building blocks for building more complex parts and assemblies. The topology entities and tools can be imported by:

```
from afem.topology import *
```

The user should review the OpenCASCADE topology documentation to become with the underlying types and data structures:

- OpenCASCADE Reference Manual-Topology[11]

The types in afem.topology.entities are essentially wrappers of their underlying TopoDS_Shape object. This is an attempt to provide a more "Pythonic" interface for the OpenCASCADE TopoDS_Shape types. Many tools are provided for the user to easily create, modify, and operate on shapes. If a tool is not available, the user can still use the pyOCCT package and access the wrapped TopoDS_Shape instance using the object property of *Shape* (page 109).

A subset of afem.topology entities and tools are featured in the following example:

```
from afem.geometry import *
from afem.graphics import *
from afem.topology import *

gui = Viewer()

# Create a box by size
builder = BoxBySize(10, 10, 10)
box = builder.solid
box.set_transparency(0.5)

# Create a cylinder partially inside the box
circle = CircleByNormal((5, 5, 5), (0, 0, 1), 2).circle
face = FaceByPlanarWire(circle).face
cyl = SolidByDrag(face, (0, 0, 15)).solid

# View the two shapes
gui.add(box, cyl)
gui.start()
gui.clear()

# Fuse the shapes
fuse = FuseShapes(box, cyl)
```

(continues on next page)

[11] https://www.opencascade.com/doc/occt-7.2.0/overview/html/occt_user_guides__modeling_data.html#occt_modat_5

(continued from previous page)

```
fused_shape = fuse.shape
fused_shape.set_transparency(0.5)

gui.add(fused_shape)
gui.start()
gui.clear()

# Cut the cylinder from the box
cut = CutShapes(box, cyl)
cut_shape = cut.shape

gui.add(cut_shape)
gui.start()
gui.clear()

# Common material between the two shapes
common = CommonShapes(box, cyl)
common_shape = common.shape

# Show original box for reference
gui.add(common_shape, box)
gui.start()
gui.clear()

# Intersect the shapes
sec = IntersectShapes(box, cyl)
sec_shape = sec.shape

# Original shapes shown for reference
gui.add(sec_shape, box, cyl)
gui.start()
gui.clear()

# Split the box with the cylinder. The resulting shape is a compound with
# two solids.
split = SplitShapes(box, cyl)
split_shape = split.shape
split_shape.set_transparency(0.5)

gui.add(split_shape)
gui.start()
gui.clear()

# Locally split one face of the box with a plane
pln = PlaneByAxes((5, 5, 5), 'xz').plane

local = LocalSplit(builder.front_face, pln, box)
local_shape = local.shape
local_shape.set_transparency(0.5)

gui.add(local_shape)
gui.start()
gui.clear()

# Offset the box
offset = OffsetShape(box, 2)
offset_shape = offset.shape
```

(continues on next page)

(continued from previous page)

```
offset_shape.set_transparency(0.5)

gui.add(box, offset_shape)
gui.start()
gui.clear()

# Rebuild the box with the results of the cut tool.
rebuild = RebuildShapeByTool(box, cut)
new_shape = rebuild.new_shape

gui.add(new_shape)
gui.start()

# Check the new shape for errors
check = CheckShape(new_shape)
print('Shape is valid:', check.is_valid)
print('Shape type:', new_shape.shape_type)

# Since a face is removed it is no longer a valid solid but a shell. Try to
# fix the shape.
fix = FixShape(new_shape)
fixed_shape = fix.shape

check = CheckShape(fixed_shape)
print('Shape is valid:', check.is_valid)
print('Shape type:', fixed_shape.shape_type)

gui.add(fixed_shape)
gui.start()

# Find free edges of a shape
tool = ExploreFreeEdges(fixed_shape)

gui.add(*tool.free_edges)
gui.start()
```

The needed entities and tools are imported by:

```
from afem.geometry import *
from afem.graphics import *
from afem.topology import *
```

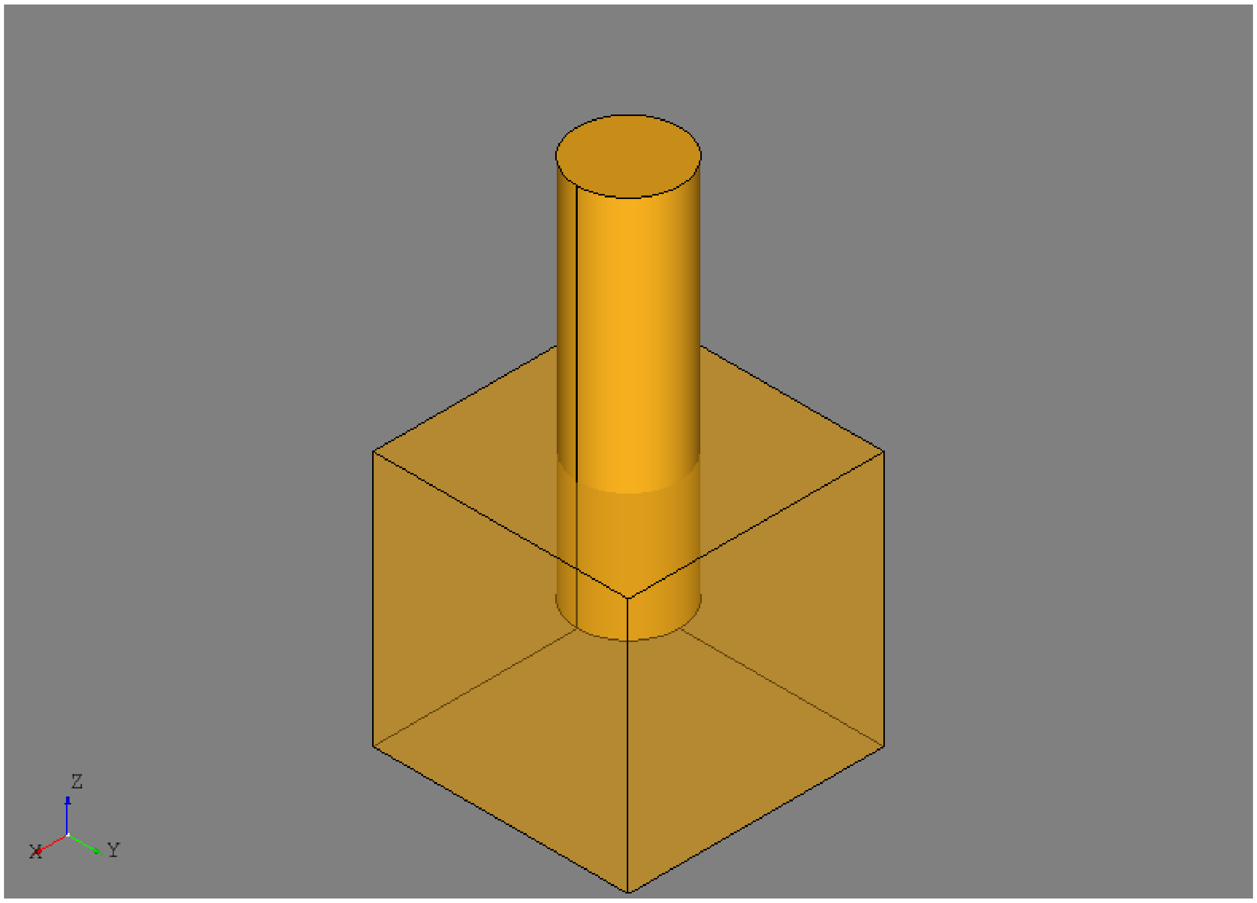
A number of tools exist to create shapes but these examples the primary shapes are a solid box and a solid cylinder. The box is created by a length, width, and height:

```
builder = BoxBySize(10, 10, 10)
box = builder.solid
box.set_transparency(0.5)
```

The cylinder is created by extruding a circular face along a vector:

```
circle = CircleByNormal((5, 5, 5), (0, 0, 1), 2).circle
face = FaceByPlanarWire(circle).face
cyl = SolidByDrag(face, (0, 0, 15)).solid
```

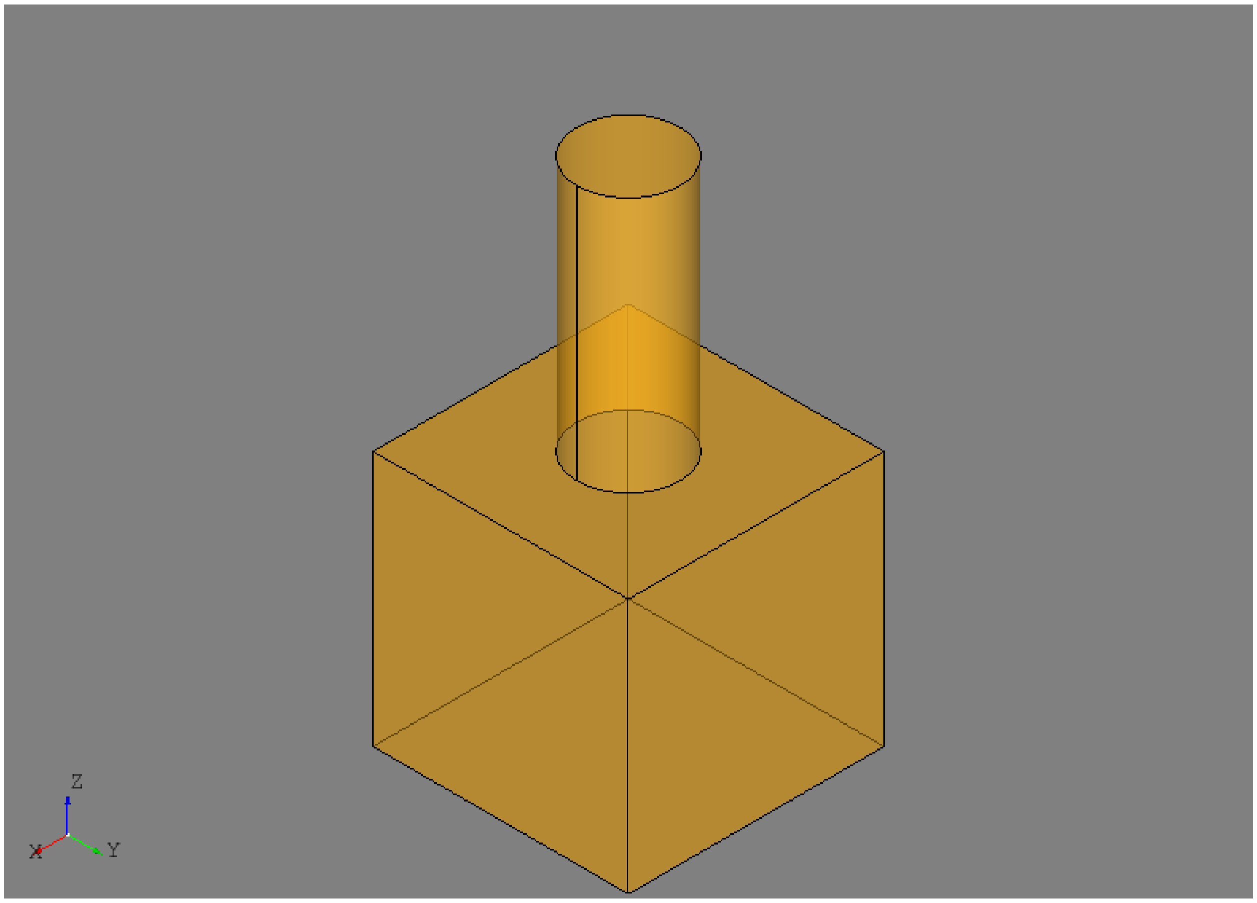
The two shapes are shown below:



Boolean operations are some of the most commonly used and most powerful modeling tools. The two shapes are fused together to form, in this case, a single solid using the *FuseShapes* (page 149) tool:

```
fuse = FuseShapes(box, cyl)
fused_shape = fuse.shape
```

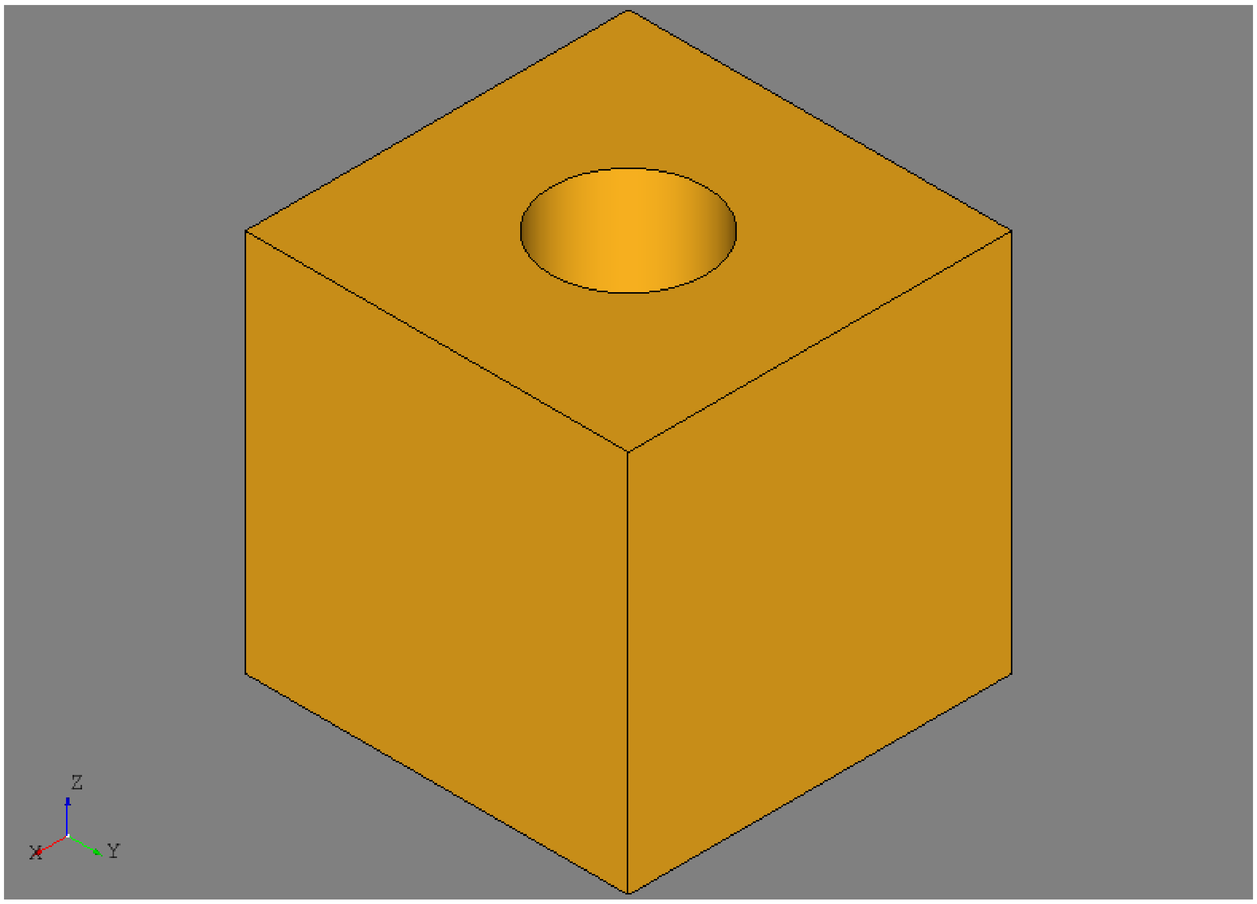
In the resulting shape, the portion of the cylinder that was inside the solid box has been removed and the union of the box and cylinder is shown below:



The cylinder is cut from the box using the *CutShapes* (page 149) tool:

```
cut = CutShapes(box, cyl)
cut_shape = cut.shape
```

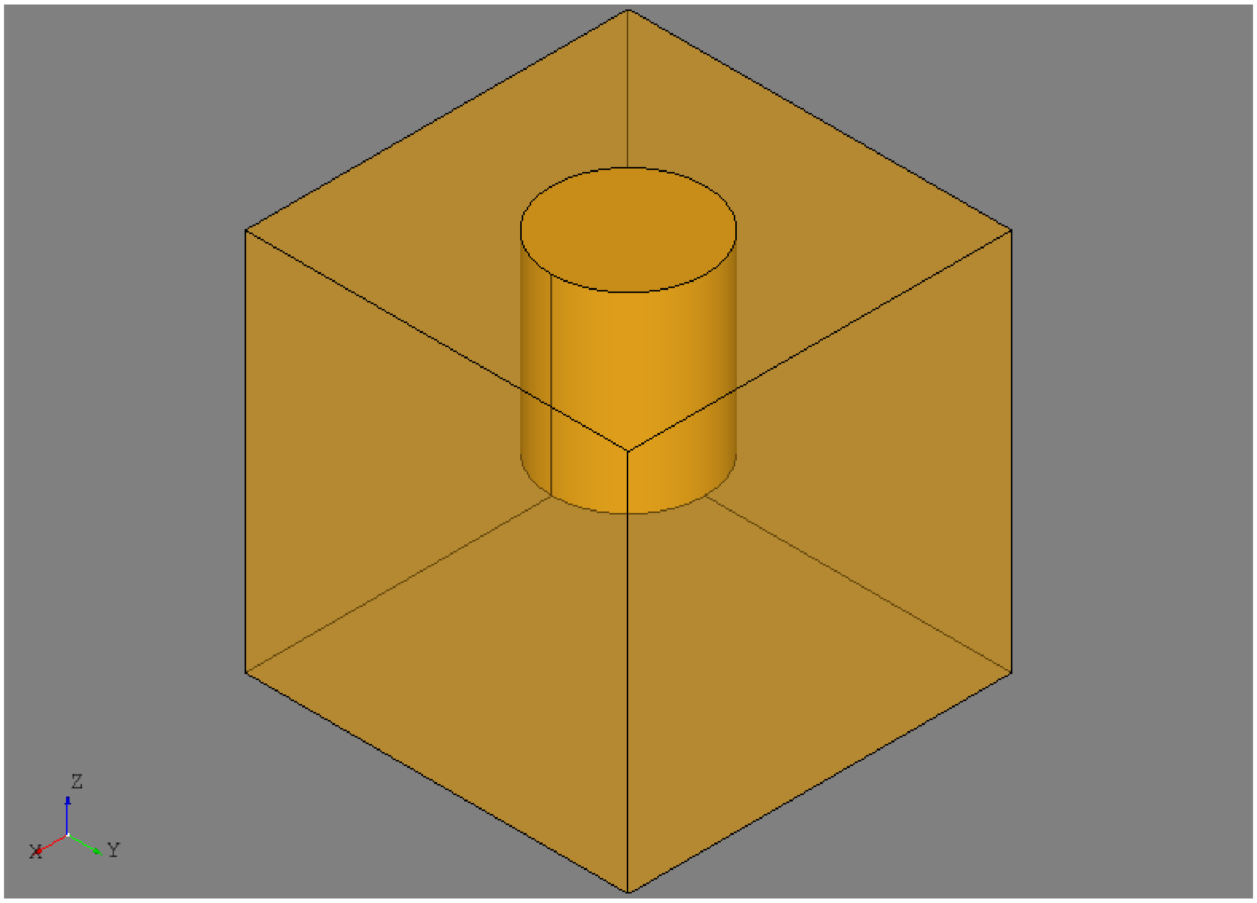
In this tool, material from `shape2` is cut away from `shape1` and the result is shown below:



Finding common material between two shapes is done by the *CommonShapes* (page 150) tool:

```
common = CommonShapes(box, cyl)
common_shape = common.shape
```

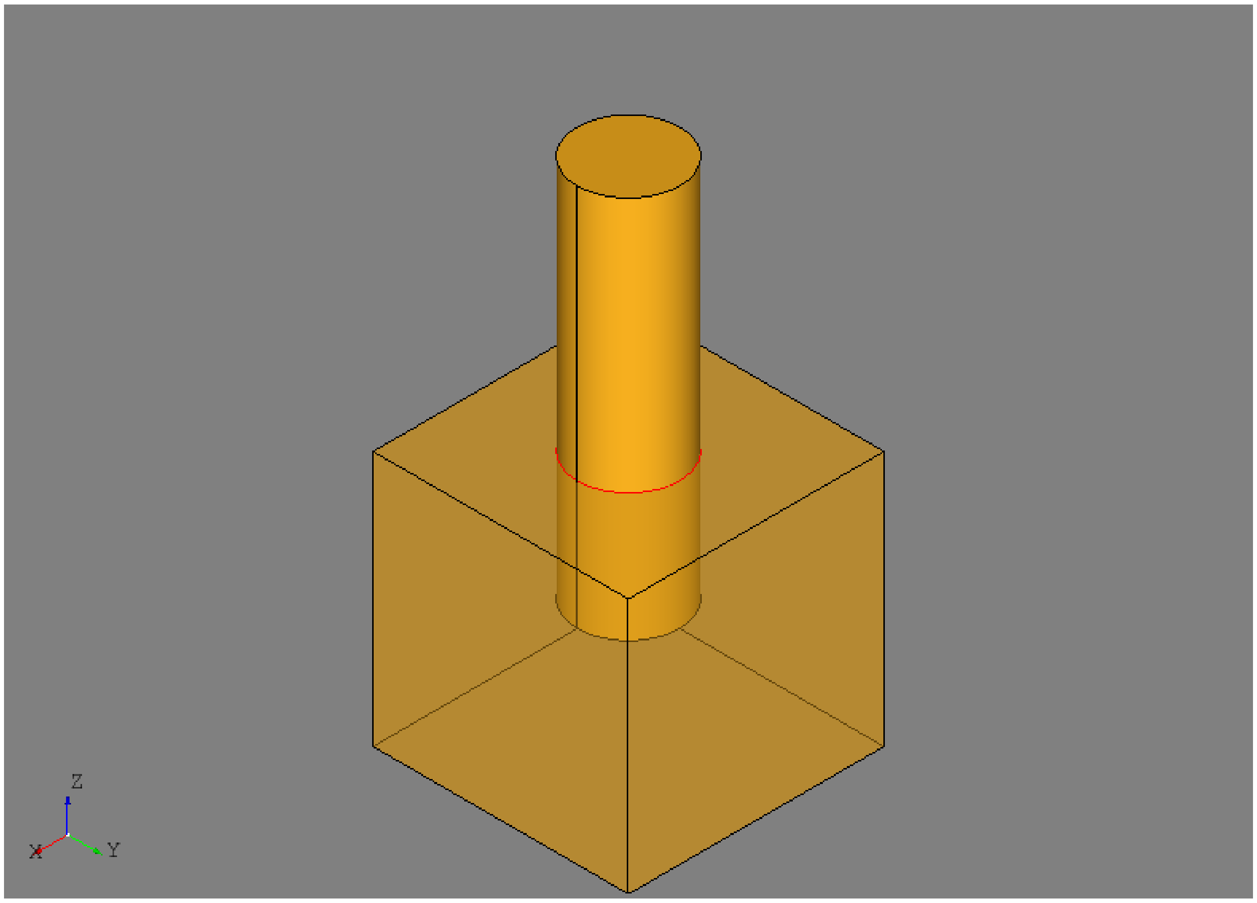
This tool will find the material that is shared by both the box and the cylinder, which in this case in the segment of the cylinder inside the box. The result is shown below where the original box is shown for reference:



The intersection of shapes is done by `IntersectShapes` (page 150) and in this case the resulting shape is a compound of edges:

```
sec = IntersectShapes(box, cyl)
sec_shape = sec.shape
```

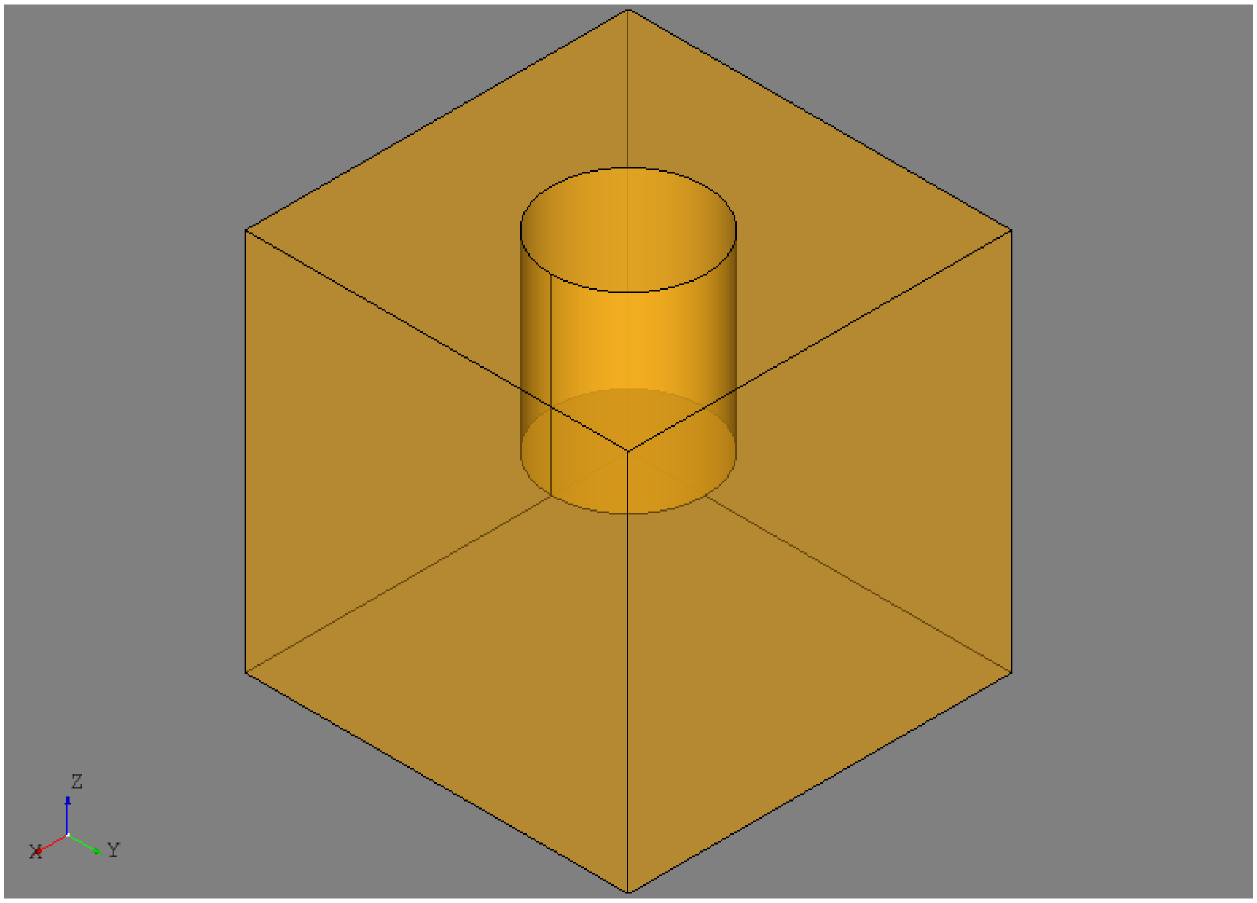
Sometimes it's possible for the result to only contain vertices if, for example, to edges are used in the intersection. The intersecting shape is shown in red in the image below:



The *SplitShapes* (page 151) tool splits arbitrary shapes with each other and in this example is used to split the box with the cylinder:

```
split = SplitShapes(box, cyl)
split_shape = split.shape
```

The resulting shape in this case in a compound consisting of two different solids as shown below:

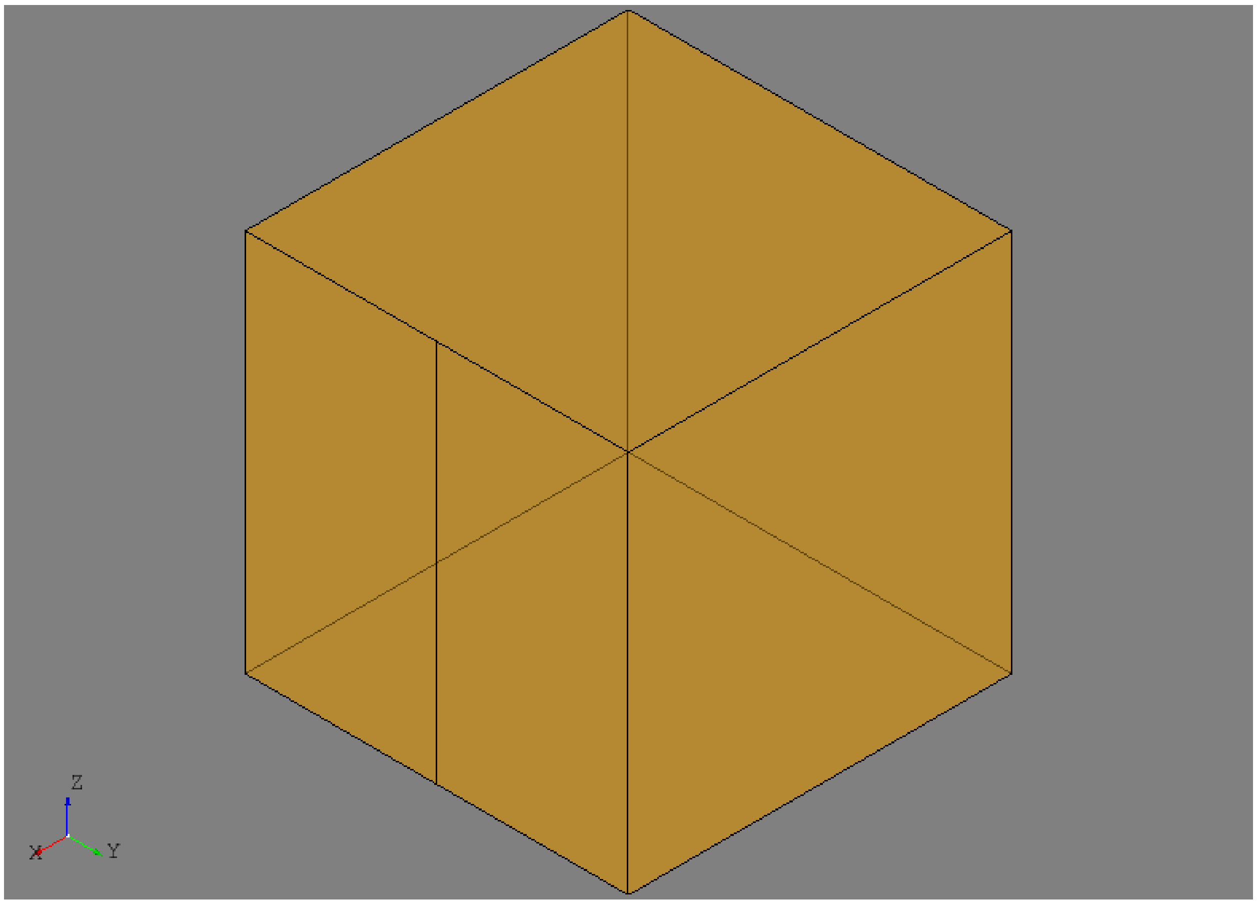


The *SplitShapes* (page 151) tool is very general and can be used for splitting faces with edges, among other applications.

In some cases it may be desirable to only split a sub-shape of a basis shape. This example uses the *LocalSplit* (page 152) tool to split one of the front face of the original solid box with a plane:

```
pln = PlaneByAxes((5, 5, 5), 'xz').plane

local = LocalSplit(builder.front_face, pln, box)
local_shape = local.shape
```

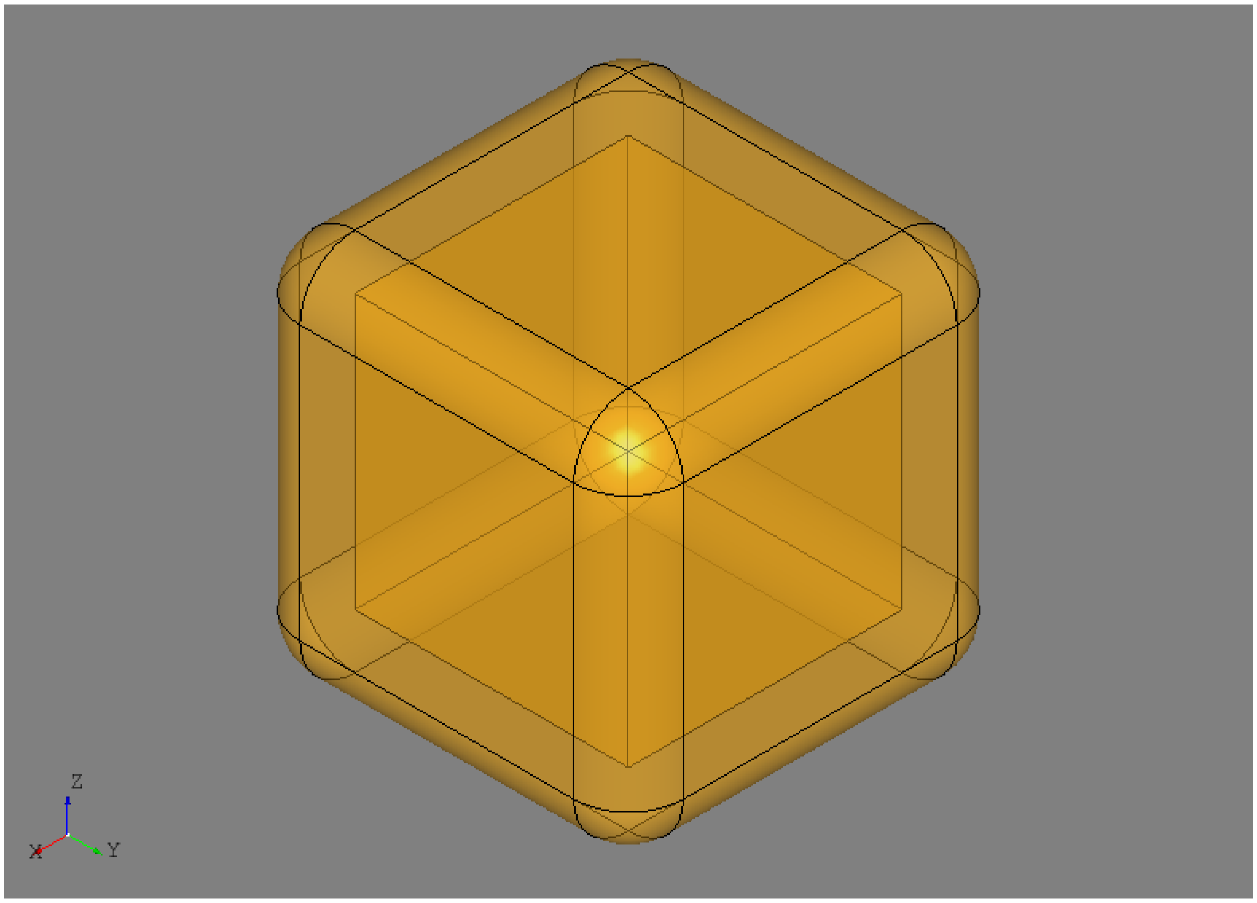
The front face of the box is provided as the shape to split and the box is provided as the "basis shape", where the shape to fit must be a sub-shape of the basis shape. The result is still a solid but with one of the faces now split:



The *OffsetShape* (page 156) tool is used to offset the solid box:

```
offset = OffsetShape(box, 2)
offset_shape = offset.shape
```

The resulting offset shape is shown below:



The rest of the example script demonstrates rebuilding (or substituting) shapes using specified substitutions or, in this example, the results of a Boolean operation. The rebuilding tools may not be used that often in practice, but set up an example of using the `CheckShape` (page 168) and `FixShape` (page 164) tools. Since a face was removed from the solid, it is not longer a valid, closed solid. This is detected by the `CheckShape` (page 168) tool and fixed with the generic `FixShape` (page 164) tool. In this example the only fix was just switching the type from a `Solid` (page 118) to a `Shell` (page 117), but they can be used to detect and fix more complicated errors like self-intersecting shapes or improperly defined geometry.

9.1 Entities

9.1.1 Shape

class `afem.topology.entities.`**`Shape`**(*shape*)

Bases: `afem.base.entities.ViewableItem` (page 11)

Shape.

Parameters **shape** (*OCCT.TopoDS.TopoDS_Shape*) – The underlying shape.

Variables

- **SHAPE** (*OCCT.TopAbs.TopAbs_ShapeEnum.TopAbs_SHAPE*) – Shape type.
- **VERTEX** (*OCCT.TopAbs.TopAbs_ShapeEnum.TopAbs_VERTEX*) – Vertex type.
- **EDGE** (*OCCT.TopAbs.TopAbs_ShapeEnum.TopAbs_EDGE*) – Edge type.

- **WIRE** (*OCCT.TopAbs.TopAbs_ShapeEnum.TopAbs_WIRE*) – Wire type.
- **FACE** (*OCCT.TopAbs.TopAbs_ShapeEnum.TopAbs_FACE*) – Face type.
- **SHELL** (*OCCT.TopAbs.TopAbs_ShapeEnum.TopAbs_SHELL*) – Shell type.
- **SOLID** (*OCCT.TopAbs.TopAbs_ShapeEnum.TopAbs_SOLID*) – Solid type.
- **COMPSOLID** (*OCCT.TopAbs.TopAbs_ShapeEnum.TopAbs_COMPSOLID*) – CompSolid type.
- **COMPOUND** (*OCCT.TopAbs.TopAbs_ShapeEnum.TopAbs_COMPOUND*) – Compound type.

Raises **TypeError** – If `shape` is not a `TopoDS_Shape`.

displayed_shape

Returns The shape to be displayed.

Return type OCCT.TopoDS.TopoDS_Shape

object

Returns The underlying shape.

Return type OCCT.TopoDS.TopoDS_Shape

hash_code

Returns The hash code of the shape computed using the TShape and Location. Orientation is not used. The upper limit is 99,999.

Return type int

is_null

Returns *True* if shape is null, *False* if not.

Return type bool

shape_type

Returns The shape type.

Return type OCCT.TopAbs.TopAbs_ShapeEnum

is_vertex

Returns *True* if a Vertex, *False* if not.

Return type bool

is_edge

Returns *True* if an Edge, *False* if not.

Return type bool

is_wire

Returns *True* if a Wire, *False* if not.

Return type bool

is_face

Returns *True* if a Face, *False* if not.

Return type bool

`is_shell`

Returns *True* if a Shell, *False* if not.

Return type bool

`is_solid`

Returns *True* if a Solid, *False* if not.

Return type bool

`is_compsolid`

Returns *True* if a CompSolid, *False* if not.

Return type bool

`is_compound`

Returns *True* if a Compound, *False* if not.

Return type bool

`closed`

Returns The closed flag of the shape.

Return type bool

`infinite`

Returns The infinite flag of the shape.

Return type bool

`vertices`

Returns The vertices of the shape.

Return type list(*afem.topology.entities.Vertex* (page 115))

`edges`

Returns The edges of the shape.

Return type list(*afem.topology.entities.Edge* (page 115))

`wires`

Returns The wires of the shape.

Return type list(*afem.topology.entities.Wire* (page 116))

`faces`

Returns The Face of the shape.

Return type list(*afem.topology.entities.Face* (page 117))

`shells`

Returns The shells of the shape.

Return type list(*afem.topology.entities.Shell* (page 117))

`solids`

Returns The solids of the shape.

Return type list(*afem.topology.entities.Solid* (page 118)) `compounds`

Returns The compounds of the shape.

Return type list(*afem.topology.entities.Compound* (page 119))

`compsolids`

Returns The compsolids of the shape.

Return type list(*afem.topology.entities.CompSolid* (page 118))

`num_vertices`

Returns The number of vertices in the shape.

Return type int

`num_edges`

Returns The number of edges in the shape.

Return type int

`num_faces`

Returns The number of faces in the shape.

Return type int

`tol_avg`

Returns The average global tolerance.

Return type float

`tol_min`

Returns The minimum global tolerance.

Return type float

`tol_max`

Returns The minimum global tolerance.

Return type float

`shape_iter`

Returns Yield the underlying sub-shape(s).

Return type collections.Iterable(*afem.topology.entities.Shape* (page 109))

`length`

Returns The length of all edges of the shape.

Return type float

`area`

Returns The area of all faces of the shape.

Return type float

`volume`

Returns The voume of all solids of the shape.

Return type float

point

Returns *None* unless overridden in derived class.

Return type *afem.geometry.entities.Point* (page 35) or None

curve

Returns *None* unless overridden in derived class.

Return type *afem.geometry.entities.Curve* (page 43) or None

surface

Returns *None* unless overridden in derived class.

Return type *afem.geometry.entities.Surface* (page 49) or None

nullify()

Destroy reference to underlying shape and make it null.

Returns None.

reverse()

Reverse the orientation of the shape.

Returns None.

reversed()

Create a new shape with reversed orientation.

Returns The reversed shape.

Return type *afem.topology.entities.Shape* (page 109)

is_partner(*other*)

Check if this shape shares the same TShape with the other. Locations and Orientations may differ.

Parameters **other** (`afem.topology.entities.Shape` (page 109)) – Other shape.

Returns *True* if partner, *False* if not.

Return type bool

is_same(*other*)

Check if this shape shares the same TShape and Location with the other. Orientations may differ.

Parameters **other** (`afem.topology.entities.Shape` (page 109)) – Other shape.

Returns *True* if same, *False* if not.

Return type bool

is_equal(*other*)

Check if this shape is equal to the other. That is, they share the same TShape, Location, and Orientation.

Parameters **other** (`afem.topology.entities.Shape` (page 109)) – Other shape.

Returns *True* if equal, *False* if not.

Return type bool

copy(*geom=True*)

Copy this shape.

Parameters **geom** (*bool*) – Option to copy geometry.

Returns The copied shape.

Return type *afem.topology.entities.Shape* (page 109)

shared_vertices(*other*, *as_compound=False*)
Get shared vertices between this shape and the other.

Parameters

- **other** (`afem.topology.entities.Shape` (page 109)) – The other shape.
- **as_compound** (*bool*) – Option to return shared shapes as a single compound.

Returns Shared vertices.

Return type list(*afem.topology.entities.Vertex* (page 115)) or *afem.topology.entities.Compound* (page 119)

shared_edges(*other*, *as_compound=False*)
Get shared edges between this shape and the other.

Parameters

- **other** (`afem.topology.entities.Shape` (page 109)) – The other shape.
- **as_compound** (*bool*) – Option to return shared shapes as a single compound.

Returns Shared edges.

Return type list(*afem.topology.entities.Edge* (page 115)) or *afem.topology.entities.Compound* (page 119)

shared_faces(*other*, *as_compound=False*)
Get shared faces between this shape and the other.

Parameters

- **other** (`afem.topology.entities.Shape` (page 109)) – The other shape.
- **as_compound** (*bool*) – Option to return shared shapes as a single compound.

Returns Shared faces.

Return type list(*afem.topology.entities.Face* (page 117)) or *afem.topology.entities.Compound* (page 119)

static wrap(*shape*)
Wrap the OpenCASCADE shape based on its type.

Parameters **shape** (*OCCT.TopoDS.TopoDS_Shape*) – The shape.

Returns The wrapped shape.

Return type *afem.topology.entities.Shape* (page 109) or *afem.topology.entities.Vertex* (page 115) or *afem.topology.entities.Edge* (page 115) or *afem.topology.entities.Wire* (page 116) or *afem.topology.entities.Face* (page 117) or *afem.topology.entities.Shell* (page 117) or *afem.topology.entities.Solid* (page 118) or *afem.topology.entities.CompSolid* (page 118) or *afem.topology.entities.Compound* (page 119)

Warning: If a null shape is provided then it is returned as a `Shape`.

static to_shape(*entity*)
Convent an entity to a shape. If already a shape the entity is returned. If the entity is geometry it is converted to its corresponding shape.

Parameters **entity** (`afem.topology.entities.Shape` (page 109) *or* `afem.geometry.entities.Curve` (page 43) *or* `afem.geometry.entities.Surface` (page 49) *or* *point_like*) – The entity.

Returns The shape.

Return type *afem.topology.entities.Shape* (page 109)

Raises **TypeError** – If entity cannot be converted to a shape.

static from_topods_list(*topods_list*)
Create a Python list of shapes from a TopoDS_ListOfShape.

Parameters **topods_list** (*OCCT.TopoDS.TopoDS_ListOfShape*) – The list.

Returns The list of shapes.

Return type list(*afem.topology.entities.Shape* (page 109))

9.1.2 Vertex

class afem.topology.entities.**Vertex**(*shape*)
Bases: *afem.topology.entities.Shape* (page 109)

Vertex.

Parameters **shape** (*OCCT.TopoDS.TopoDS_Vertex*) – The vertex.

Raises **TypeError** – If `shape` is not a `TopoDS_Vertex`.

point

Returns A point at vertex location.

Return type *afem.geometry.entities.Point* (page 35)

static by_point(*pnt*)
Create a vertex by a point.

Parameters **pnt** (*point_like*) – The point.

Returns The vertex.

Return type *afem.topology.entities.Vertex* (page 115)

9.1.3 Edge

class afem.topology.entities.**Edge**(*shape*)
Bases: *afem.topology.entities.Shape* (page 109)

Edge.

Parameters **shape** (*OCCT.TopoDS.TopoDS_Edge*) – The edge.

Raises **TypeError** – If `shape` is not a `TopoDS_Edge`.

curve

Returns The underlying curve of the edge.

Return type *afem.geometry.entities.Curve* (page 43)

first_vertex

Returns The first vertex of the edge.

Return type *afem.topology.entities.Vertex* (page 115)

`last_vertex`

Returns The last vertex of the edge.

Return type *afem.topology.entities.Vertex* (page 115)

`same_parameter`

Returns The same parameter flag for the edge.

Return type bool

`same_range`

Returns The same range flag for the edge.

Return type bool

`static by_curve`(*curve*)

Create an edge by a curve.

Parameters **`curve`** (`afem.geometry.entities.Curve` (page 43)) – The curve.

Returns The edge.

Return type *afem.topology.entities.Edge* (page 115)

9.1.4 Wire

`class` `afem.topology.entities.`**`Wire`**(*shape*)

Bases: *`afem.topology.entities.Shape`* (page 109)

Wire.

Parameters **`shape`** (*`OCCT.TopoDS.TopoDS_Wire`*) – The wire.

Raises **`TypeError`** – If `shape` is not a `TopoDS_Wire`.

`curve`

Returns The curve formed by concatenating all the underlying curves of the edges.

Return type *afem.geometry.entities.NurbsCurve* (page 47)

`static by_curve`(*curve*)

Create a wire from a curve.

Parameters **`curve`** (`afem.geometry.entities.Curve` (page 43)) – The curve.

Returns The wire.

Return type *afem.topology.entities.Wire* (page 116)

`static by_edge`(*edge*)

Create a wire from an edge.

Parameters **`edge`** (`afem.topology.entities.Edge` (page 115)) – The edge.

Returns The wire.

Return type *afem.topology.entities.Wire* (page 116)

`static by_points`(*pnts*, *close=False*)

Create polygonal wire by connecting points.

Parameters

- **pnts** (*collections.Sequence(point_like)*) – The ordered points.
- **close** (*bool*) – Option to close the wire.

Returns The new wire.

Return type *afem.topology.entities.Wire* (page 116)

9.1.5 Face

class afem.topology.entities.**Face**(*shape*)

Bases: *afem.topology.entities.Shape* (page 109)

Face.

Parameters **shape** (*OCCT.TopoDS.TopoDS_Face*) – The face.

Raises **TypeError** – If shape is not a TopoDS_Face.

surface

Returns The underlying surface of the face.

Return type *afem.geometry.entities.Surface* (page 49)

outer_wire

Returns The outer wire of the face.

Return type *afem.topology.entities.Wire* (page 116)

to_shell()

Create a shell from the face.

Returns The shell.

Return type *afem.topology.entities.Shell* (page 117)

static by_surface(*surface*)

Create a face by a surface.

Parameters **surface** (afem.geometry.entities.Surface (page 49)) – The surface.

Returns The face.

Return type *afem.topology.entities.Face* (page 117)

static by_wire(*wire*)

Create a face by a planar wire.

Parameters **wire** (afem.topology.entities.Wire (page 116)) – The wire.

Returns The new face.

Return type *afem.topology.entities.Face* (page 117)

9.1.6 Shell

class afem.topology.entities.**Shell**(*shape*)

Bases: *afem.topology.entities.Shape* (page 109)

Shell.

Parameters **shape** (*OCCT.TopoDS.TopoDS_Shell*) – The shell.

Raises **`TypeError`** – If `shape` is not a `TopoDS_Shell`.

`surface`

Returns The underlying surface of the largest face.

Return type *afem.geometry.entities.Surface* (page 49)

`static by_surface`(*surface*)
Create a shell from a surface.

Parameters **`surface`** (`afem.geometry.entities.Surface` (page 49)) – The surface.

Returns The shell.

Return type *afem.topology.entities.Shell* (page 117)

`static by_face`(*face*)
Create a shell from a face.

Parameters **`face`** (`afem.topology.entities.Face` (page 117)) – The face.

Returns The shell.

Return type *afem.topology.entities.Shell* (page 117)

9.1.7 Solid

`class` `afem.topology.entities.`**`Solid`**(*shape*)
Bases: *`afem.topology.entities.Shape`* (page 109)

Solid.

Parameters **`shape`** (*`OCCT.TopoDS.TopoDS_Solid`*) – The solid.

Raises **`TypeError`** – If `shape` is not a `TopoDS_Solid`.

`outer_shell`

Returns The outer shell of the face.

Return type *afem.topology.entities.Shell* (page 117)

`static by_shell`(*shell*)
Create a solid from the shell.

Parameters **`shell`** (`afem.topology.entities.Shell` (page 117)) – The shell.

Returns The new solid.

Return type *afem.topology.entities.Solid* (page 118)

9.1.8 CompSolid

`class` `afem.topology.entities.`**`CompSolid`**(*shape*)
Bases: *`afem.topology.entities.Shape`* (page 109)

CompSolid.

Parameters **`shape`** (*`OCCT.TopoDS.TopoDS_CompSolid`*) – The compsolid.

Raises **`TypeError`** – If `shape` is not a `TopoDS_CompSolid`.

9.1.9 Compound

class afem.topology.entities.**Compound**(*shape*)
Bases: *afem.topology.entities.Shape* (page 109)

Compound.

Parameters **shape** (*OCCT.TopoDS.TopoDS_Compound*) – The compound.

Raises **TypeError** – If `shape` is not a `TopoDS_Compound`.

surface

Returns The underlying surface of the largest face, or *None* if compound has no faces.

Return type *afem.geometry.entities.Surface* (page 49)

static by_shapes(*shapes*)
Create a new compound from a collection of shapes.

Parameters **shapes** (*collections.Sequence(*afem.topology.entities.Shape (page 109)*)*) – The shapes.

Returns The new compound.

Return type *afem.topology.entities.Compound* (page 119)

9.1.10 Bounding Box

class afem.topology.entities.**BBox**
Bases: `OCCT.Bnd.Bnd_Box`

Bounding box in 3-D space.

is_void

Returns *True* if bounding box is empty, *False* if not.

Return type bool

pmin

Returns Lower corner of bounding box. *None* if empty.

Return type *afem.geometry.entities.Point* (page 35)

pmax

Returns Upper corner of bounding box. *None* if empty.

Return type *afem.geometry.entities.Point* (page 35)

xmin

Returns Minimum x-component.

Return type float

xmax

Returns Maximum x-component.

Return type float

ymin

Returns Minimum y-component.

Return type float

ymax

Returns Maximum y-component.

Return type float

zmin

Returns Minimum z-component.

Return type float

zmax

Returns Maximum z-component.

Return type float

gap

Returns The gap of the bounding box.

Return type float

diagonal

Returns The diagonal length of the box.

Return type float

set_gap(*gap*)
Set the gap of the bounding box.

Parameters **gap** (*float*) – The gap.

Returns None.

enlarge(*tol*)
Enlarge the box with a tolerance value.

Parameters **tol** (*float*) – The tolerance.

Returns None.

add_box(*bbox*)
Add the other box to this one.

Parameters **bbox** (afem.topology.entities.BBox (page 119)) – The other box.

Returns None.

Raises **TypeError** – If *bbox* cannot be converted to a bounding box.

add_pnt(*pnt*)
Add a point to the bounding box.

Parameters **pnt** (*point_like*) – The point.

Returns None.

Raises **TypeError** – If *pnt* cannot be converted to a point.

add_shape(*shape*)
Add shape to the bounding box.

Parameters **shape** (afem.topology.entities.Shape (page 109)) – The shape.

Returns None.

is_pnt_out (*pnt*)
Check to see if the point is outside the bounding box.

Parameters **pnt** (*point_like*) – The point.

Returns *True* if outside, *False* if not.

Return type bool

Raises **TypeError** – If *pnt* cannot be converted to a point.

is_line_out (*line*)
Check to see if the line intersects the box.

Parameters **line** (afem.geometry.entities.Line (page 45)) – The line.

Returns *True* if outside, *False* if it intersects.

Return type bool

Raises **TypeError** – If *line* is not a line.

is_pln_out (*pln*)
Check to see if the plane intersects the box.

Parameters **pln** (afem.geometry.entities.Plane (page 52)) – The plane.

Returns *True* if outside, *False* if it intersects.

Return type bool

Raises **TypeError** – If *pln* is not a plane.

is_box_out (*bbox*)
Check to see if the bounding box intersects this one.

Parameters **bbox** (afem.topology.entities.BBox (page 119)) – The other box.

Returns *True* if outside, *False* if it intersects or is inside.

Return type bool

Raises **TypeError** – If *bbox* cannot be converted to a bounding box.

distance (*bbox*)
Calculate distance to other box.

Parameters **bbox** (*afem.geometry.entities.BBox*) – The other box.

Returns Distance to other box.

Return type float

Raises **TypeError** – If *bbox* cannot be converted to a bounding box.

9.2 Create

9.2.1 VertexByPoint

class afem.topology.create.**VertexByPoint** (*pnt*)
Bases: object

Create a vertex using a point.

Parameters **pnt** (*point_like*) – The point.

vertex

Returns The vertex.

Return type *afem.topology.entities.Vertex* (page 115)

9.2.2 EdgeByPoints

class `afem.topology.create.`**EdgeByPoints**(*p1*, *p2*)

Bases: `object`

Create an edge between two points.

Parameters

- **p1** (*point_like*) – The first point.
- **p2** (*point_like*) – The second point.

edge

Returns The edge.

Return type *afem.topology.entities.Edge* (page 115)

vertex1

Returns The first vertex.

Return type *afem.topology.entities.Vertex* (page 115)

vertex2

Returns The second vertex.

Return type *afem.topology.entities.Vertex* (page 115)

9.2.3 EdgeByVertices

class `afem.topology.create.`**EdgeByVertices**(*v1*, *v2*)

Bases: `object`

Create an edge between two vertices.

Parameters

- **v1** (`afem.topology.entities.Vertex` (page 115)) – The first vertex.
- **v2** (`afem.topology.entities.Vertex` (page 115)) – The second vertex.

edge

Returns The edge.

Return type *afem.topology.entities.Edge* (page 115)

9.2.4 EdgeByCurve

class `afem.topology.create.`**EdgeByCurve**(*crv*)

Bases: `object`

Create an edge using a curve.

Parameters **crv** (`afem.geometry.entities.Curve` (page 43)) – The curve.

edge

Returns The edge.

Return type *afem.topology.entities.Edge* (page 115)

vertex1

Returns The first vertex.

Return type *afem.topology.entities.Vertex* (page 115)

vertex2

Returns The second vertex.

Return type *afem.topology.entities.Vertex* (page 115)

9.2.5 EdgeByDrag

class `afem.topology.create.`**EdgeByDrag**(*vertex*, *v*)

Bases: `object`

Create an edge by dragging a vertex along a vector.

Parameters

- **vertex** (`afem.topology.entities.Vertex` (page 115)) – The vertex.
- **v** (*vector_like*) – The vector to drag the shape.

edge

Returns The edge.

Return type *afem.topology.entities.Edge* (page 115)

first_vertex

Returns The vertex at the bottom of the edge.

Return type *afem.topology.entities.Edge* (page 115)

last_vertex

Returns The vertex at the top of the edge.

Return type *afem.topology.entities.Vertex* (page 115)

9.2.6 EdgeByWireConcat

class `afem.topology.create.`**EdgeByWireConcat**(*wire*)

Bases: `object`

Create an edge by concatenating all the edges of a wire. The edge may have C0 continuity.

Parameters **wire** (`afem.topology.entities.Wire` (page 116)) – The wire.

edge

Returns The edge.

Return type *afem.topology.entities.Edge* (page 115)

9.2.7 WireByEdges

class `afem.topology.create.`**`WireByEdges`**(**edges*)

Bases: `object`

Create a wire using topologically connected edges.

Parameters **`edges`** (*`collections.Sequence`*(`afem.topology.entities.Edge` (page 115))) – The edges. They must share a common vertex to be connected.

`wire`

Returns The wire.

Return type *afem.topology.entities.Wire* (page 116)

`last_edge`

Returns The last edge added to the wire.

Return type *afem.topology.entities.Edge* (page 115)

`last_vertex`

Returns The last vertex added to the wire.

Return type *afem.topology.entities.Vertex* (page 115)

9.2.8 WiresByConnectedEdges

class `afem.topology.create.`**`WiresByConnectedEdges`**(*edges*, *tol=None*, *shared=False*)

Bases: `object`

Create wires from a list of unsorted edges.

Parameters

- **`edges`** (*`collections.Sequence`*(`afem.topology.entities.Edge` (page 115))) – The edges.
- **`tol`** (*`float`*) – Connection tolerance. If *None* if provided then the maximum tolerance of all edge will be used.
- **`shared`** (*`bool`*) – Option to use only shared vertices to connect edges. If *False* then geometric coincidence will be also checked.

`nwires`

Returns Number of wires.

Return type int

`wires`

Returns The wires.

Return type list(*afem.topology.entities.Wire* (page 116))

9.2.9 WireByPlanarOffset

class `afem.topology.create.`**`WireByPlanarOffset`**(*spine*, *distance*, *altitude=0.0*, *join=GeomAbs_JoinType.GeomAbs_Arc*, *is_open=False*)

Bases: `object`

Create a wire by offsetting a planar wire or face.

Parameters

- **`spine`** (`afem.topology.entities.Wire` (page 116) *or* `afem.topology.entities.Face` (page 117)) – The wire to offset. If a face is provided the outer wire will be used.
- **`distance`** (*`float`*) – Offset distance in the plane.
- **`altitude`** (*`float`*) – Offset altitude normal to the plane.
- **`join`** (*`OCCT.GeomAbs.GeomAbs_JoinType`*) – Join type.

`wire`

Returns The wire.

Return type *afem.topology.entities.Wire* (page 116)

9.2.10 WiresByShape

class `afem.topology.create.`**`WiresByShape`**(*shape*)

Bases: *`afem.topology.create.WiresByConnectedEdges`* (page 124)

Create wires by connecting all the edges of a shape. This method gathers all the unique edges of a shape and then uses *`WiresByConnectedEdges`* (page 124).

Parameters **`shape`** (`afem.topology.entities.Shape` (page 109)) – The shape.

Raises **`ValueError`** – If no edges are found in the shape.

9.2.11 WireByPoints

class `afem.topology.create.`**`WireByPoints`**(*pnts*, *close=False*)

Bases: `object`

Create a polygonal wire by connecting points.

Parameters

- **`pnts`** (*`collections.Sequence(point_like)`*) – The ordered points.
- **`close`** (*`bool`*) – Option to close the wire.

`wire`

Returns The wire.

Return type *afem.topology.entities.Wire* (page 116)

9.2.12 WireByConcat

class afem.topology.create.**WireByConcat**(*wire*)

Bases: object

Create a wire by concatenating all the edges of the wire. The wire may have C0 continuity.

Parameters **wire** (afem.topology.entities.Wire (page 116)) – The wire.

wire

Returns The wire.

Return type *afem.topology.entities.Wire* (page 116)

9.2.13 FaceBySurface

class afem.topology.create.**FaceBySurface**(*srf*, *tol=1e-07*)

Bases: object

Create a face from a surface.

Parameters

- **srf** (afem.geometry.entities.Surface (page 49)) – The surface.
- **tol** (*float*) – Tolerance for resolution of degenerate edges.

face

Returns The face.

Return type *afem.topology.entities.Face* (page 117)

9.2.14 FaceByPlane

class afem.topology.create.**FaceByPlane**(*pln*, *umin*, *umax*, *vmin*, *vmax*)

Bases: object

Create a finite face from a plane.

Parameters

- **pln** (afem.geometry.entities.Plane (page 52)) – The plane.
- **umin** (*float*) – Minimum u-parameter.
- **umax** (*float*) – Maximum u-parameter.
- **vmin** (*float*) – Minimum v-parameter.
- **vmax** (*float*) – Maximum v-parameter.

face

Returns The face.

Return type *afem.topology.entities.Face* (page 117)

9.2.15 FaceByPlanarWire

class afem.topology.create.**FaceByPlanarWire**(*wire*)

Bases: `object`

Create a face from a planar wire.

Parameters **wire** (`afem.topology.entities.Wire` (page 116) *or* `afem.topology.entities.Edge` (page 115) *or* `afem.geometry.entities.Curve` (page 43)) – The wire.

face

Returns The face.

Return type *afem.topology.entities.Face* (page 117)

9.2.16 FaceByDrag

class afem.topology.create.**FaceByDrag**(*edge*, *v*)

Bases: `object`

Create a face by dragging an edge along a vector.

Parameters

- **edge** (`afem.topology.entities.Edge` (page 115)) – The edge.
- **v** (*vector_like*) – The vector to drag the shape.

face

Returns The face.

Return type *afem.topology.entities.Face* (page 117)

first_edge

Returns The edge at the bottom of the face.

Return type *afem.topology.entities.Edge* (page 115)

last_edge

Returns The edge at the top of the face.

Return type *afem.topology.entities.Edge* (page 115)

9.2.17 ShellBySurface

class afem.topology.create.**ShellBySurface**(*srf*)

Bases: `object`

Create a shell from a surface.

Parameters **srf** (`afem.geometry.entities.Surface` (page 49)) – The surface.

shell

Returns The shell.

Return type *afem.topology.entities.Shell* (page 117)

9.2.18 ShellByFaces

class afem.topology.create.**ShellByFaces**(*faces*)

Bases: `object`

Create a shell from connected faces. This method initializes a shell an then simply adds the faces to it. The faces should already have shared edges. This is not checked.

Parameters **faces** (*collections.Sequence(*afem.topology.entities.Face (page 117)*)*) – The faces.

shell

Returns The shell.

Return type *afem.topology.entities.Shell* (page 117)

9.2.19 ShellByDrag

class afem.topology.create.**ShellByDrag**(*wire*, *v*)

Bases: `object`

Create a shell by dragging a wire along a vector.

Parameters

- **wire** (afem.topology.entities.Wire (page 116)) – The wire.
- **v** (*vector_like*) – The vector to drag the shape.

shell

Returns The shell.

Return type *afem.topology.entities.Shell* (page 117)

first_wire

Returns The wire at the bottom of the shell.

Return type *afem.topology.entities.Wire* (page 116)

last_wire

Returns The wire at the top of the shell.

Return type *afem.topology.entities.Wire* (page 116)

9.2.20 ShellBySewing

class afem.topology.create.**ShellBySewing**(*faces*, *tol=None*, *cut_free_edges=False*, *non_manifold=False*)

Bases: `object`

Create a shell by sewing faces.

Parameters

- **faces** (*collections.Sequence(*afem.topology.entities.Face (page 117)*)*) – The faces.
- **tol** (*float*) – Sewing tolerance. If *None* the maximum tolerance of all the faces is used.
- **cut_free_edges** (*bool*) – Option for cutting of free edges.

- **non_manifold** (*bool*) – Option for non-manifold processing.

Raises RuntimeError – If an invalid shape type results.

nshells

Returns Number of shells.

Return type int

shell

Returns The sewn shell.

Return type *afem.topology.entities.Shell* (page 117)

shells

Returns The sewn shells if more than one is found.

Return type list(*afem.topology.entities.Shell* (page 117))

9.2.21 SolidByShell

class afem.topology.create.**SolidByShell**(*shell*)

Bases: object

Create a solid using a shell. The shell can either be closed (finite solid) or open (infinite solid).

Parameters shell (afem.topology.entities.Shell (page 117)) – The shell.

solid

Returns The solid.

Return type *afem.topology.entities.Solid* (page 118)

9.2.22 SolidByPlane

class afem.topology.create.**SolidByPlane**(*pln*, *width*, *height*, *depth*)

Bases: object

Create a solid box using a plane. The plane will be extruded in the direction of the plane's normal. The solid's width and height will be centered at the plane's origin.

Parameters

- **pln** (afem.geometry.entities.Plane (page 52)) – The plane.
- **width** (*float*) – Width of the box.
- **height** (*float*) – Height of the box.
- **depth** (*float*) – Depth of the box.

solid

Returns The solid.

Return type *afem.topology.entities.Solid* (page 118)

9.2.23 SolidByDrag

class `afem.topology.create.`**`SolidByDrag`**(*face*, *v*)

Bases: `object`

Create a solid by dragging a face along a vector.

Parameters

- **`face`** (`afem.topology.entities.Face` (page 117)) – The face.
- **`v`** (*`vector_like`*) – The vector to drag the shape.

`solid`

Returns The solid.

Return type *afem.topology.entities.Solid* (page 118)

`first_face`

Returns The face at the bottom of the solid.

Return type *afem.topology.entities.Face* (page 117)

`last_face`

Returns The face at the top of the solid.

Return type *afem.topology.entities.Face* (page 117)

9.2.24 CompoundByShapes

class `afem.topology.create.`**`CompoundByShapes`**(*shapes*)

Bases: `object`

Create a compound from a list of shapes.

Parameters **`shapes`** – List of shapes.

Type collections.Sequence(*afem.topology.entities.Shape* (page 109))

`compound`

Returns The compound.

Return type *afem.topology.entities.Compound* (page 119)

9.2.25 HalfspaceByShape

class `afem.topology.create.`**`HalfspaceByShape`**(*shape*, *pnt*)

Bases: `object`

Create a half-space by a face or shell and a reference point.A half-space is an infinite solid, limited by a surface. It is built from a face or a shell, which bounds it, and with a reference point, which specifies the side of the surface where the matter of the half-space is located. A half-space is a tool commonly used in topological operations to cut another shape.

Parameters

- **`shape`** (`afem.topology.entities.Face` (page 117) *or* `afem.topology.entities.Shell` (page 117)) – The face or shell.
- **`pnt`** (*`point_like`*) – The reference point where the matter is located.

Raises `RuntimeError` – If *shape* is not a face or shell.

solid

Returns The half-space as a solid.

Return type *afem.topology.entities.Solid* (page 118)

9.2.26 HalfspaceBySurface

class afem.topology.create.**HalfspaceBySurface**(*srf*, *pnt*)

Bases: *afem.topology.create.HalfspaceByShape* (page 130)

Create a half-space using a surface.

Parameters

- **afem.geometry.entities.Surface** – The surface.
- **pnt** (*point_like*) – The reference point where the matter is located.

9.2.27 ShapeByFaces

class afem.topology.create.**ShapeByFaces**(*faces*, *sew=False*, *tol=None*, *cut_free_edges=False*, *non_manifold=False*)

Bases: object

Create either a face or a shell from faces.

Parameters

- **faces** (*collections.Sequence(afem.topology.entities.Face* (page 117)*)*) – The faces.
- **sew** (*bool*) – Option to sew the faces if more than one face is provided.
- **tol** (*float*) – Sewing tolerance. If *None* the maximum tolerance of all the faces is used.
- **cut_free_edges** (*bool*) – Option for cutting of free edges.
- **non_manifold** (*bool*) – Option for non-manifold processing.

shape

Returns The shape. Will be a face if there was only one face in the list, it will be a shell if only one shell was formed, or it will be a compound of shells if more than one shell was found after sewing.

Return type *afem.topology.entities.Face* (page 117) or *afem.topology.entities.Shell* (page 117) or *afem.topology.entities.Compound* (page 119)

is_face

Returns *True* if the shape is a face, *False* if not.

Return type bool

is_shell

Returns *True* if the shape is a shell, *False* if not.

Return type bool

is_compound

Returns *True* if the shape is a compound, *False* if not.

Return type bool

9.2.28 ShapeByDrag

class afem.topology.create.**ShapeByDrag**(*shape*, *v*)

Bases: object

Create a shape by dragging another shape along a vector.

Parameters

- **shape** (afem.topology.entities.Shape (page 109)) – The shape.
- **v** (*vector_like*) – The vector to drag the shape.

shape

Returns The shape.

Return type *afem.topology.entities.Shape* (page 109)

first_shape

Returns The shape at the bottom.

Return type *afem.topology.entities.Shape* (page 109)

last_shape

Returns The shape at the top.

Return type *afem.topology.entities.Shape* (page 109)

is_edge

Returns *True* if shape is an edge, *False* if not.

Return type bool

is_face

Returns *True* if shape is a face, *False* if not.

Return type bool

is_shell

Returns *True* if shape is a shell, *False* if not.

Return type bool

is_solid

Returns *True* if shape is a solid, *False* if not.

Return type bool

is_compsolid

Returns *True* if shape is a compsolid, *False* if not.

Return type bool

is_compound

Returns *True* if shape is a compound, *False* if not.

Return type bool

9.2.29 BoxBuilder

class afem.topology.create.**BoxBuilder**(**args*)

Bases: `object`

Base class for building boxes.

shell

Returns The box as a shell.

Return type *afem.topology.entities.Shell* (page 117)

solid

Returns The box as a solid.

Return type *afem.topology.entities.Solid* (page 118)

bottom_face

Returns The bottom face.

Return type *afem.topology.entities.Face* (page 117)

back_face

Returns The back face.

Return type *afem.topology.entities.Face* (page 117)

front_face

Returns The front face.

Return type *afem.topology.entities.Face* (page 117)

left_face

Returns The left face.

Return type *afem.topology.entities.Face* (page 117)

right_face

Returns The right face.

Return type *afem.topology.entities.Face* (page 117)

top_face

Returns The top face.

Return type *afem.topology.entities.Face* (page 117)

9.2.30 BoxBySize

class afem.topology.create.**BoxBySize**(*dx=1.0*, *dy=1.0*, *dz=1.0*)

Bases: *afem.topology.create.BoxBuilder* (page 133)

Build a box with the corner at (0, 0, 0) and the other at (dx, dy, dz).

Parameters

- **dx** (*float*) – The corner x-location.
- **dy** (*float*) – The corner y-location.
- **dz** (*float*) – The corner z-location.

9.2.31 BoxBy2Points

class afem.topology.create.**BoxBy2Points**(*p1*, *p2*)

Bases: *afem.topology.create.BoxBuilder* (page 133)

Build a box between two points.

Parameters

- **p1** (*point_like*) – The first corner point.
- **p2** (*point_like*) – The other corner point.

9.2.32 CylinderByAxis

class afem.topology.create.**CylinderByAxis**(*radius*, *height*, *axis2=None*)

Bases: object

Create a cylinder.

Parameters

- **radius** (*float*) – The radius.
- **height** (*float*) – The height.
- **axis2** – Not yet implemented. Solid will be constructed in xy-plane.

Raises NotImplementedError – If an axis is provided.

face

Returns The lateral face of the cylinder

Return type *afem.topology.entities.Face* (page 117)

shell

Returns The cylinder as a shell.

Return type *afem.topology.entities.Shell* (page 117)

solid

Returns The cylinder as a solid.

Return type *afem.topology.entities.Solid* (page 118)

9.2.33 SphereByRadius

class afem.topology.create.**SphereByRadius**(*origin=(0.0, 0.0, 0.0)*, *radius=1.0*)

Bases: object

Create a sphere by a center and radius.

Parameters

- **origin** (*point_like*) – The origin.
- **radius** (*float*) – The radius.

face

Returns The sphere as a face.

Return type *afem.topology.entities.Face* (page 117)

shell

Returns The sphere as a shell.

Return type *afem.topology.entities.Shell* (page 117)

solid

Returns The sphere as a solid.

Return type *afem.topology.entities.Face* (page 117)

sphere

Returns The sphere primitive.

Return type OCCT.BRepPrim.BRepPrim_Sphere

9.2.34 SphereBy3Points

class afem.topology.create.**SphereBy3Points**(*p1*, *p2*, *p3*)

Bases: *afem.topology.create.SphereByRadius* (page 134)

Create a sphere using three points.

Parameters

- **p1** (*point_like*) – The first point.
- **p2** (*point_like*) – The second point.
- **p3** (*point_like*) – The third point.

9.2.35 PointAlongShape

class afem.topology.create.**PointAlongShape**(*shape*, *ds*, *tol=1e-07*)

Bases: *afem.geometry.create.PointFromParameter* (page 58)

Create a point along an edge or wire at a specified distance from the first parameter.

Parameters

- **shape** (afem.topology.entities.Edge (page 115) *or* afem.topology.entities.Wire (page 116)) – The shape.
- **ds** (*float*) – The distance along the curve from the given parameter.
- **tol** (*float*) – Tolerance.

Raises

- **TypeError** – If *shape* if not a curve or wire.
- **RuntimeError** – If OCC method fails.

9.2.36 PointsAlongShapeByNumber

class afem.topology.create.**PointsAlongShapeByNumber**(*shape*, *n*, *d1=None*, *d2=None*, *shape1=None*, *shape2=None*)

Bases: *afem.geometry.create.PointsAlongCurveByNumber* (page 59)

Create a specified number of points along an edge or wire.

Parameters

- **shape** (afem.topology.entities.Edge (page 115) *or* afem.topology.entities.Wire (page 116)) – The shape.
- **n** (*int*) – Number of points to create ($n > 0$).
- **d1** (*float*) – An offset distance for the first point. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last point. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **shape1** (afem.topology.entities.Shape (page 109)) – A shape to define the first point. This shape is intersected with the edge or wire.
- **shape2** (afem.topology.entities.Shape (page 109)) – A shape to define the last point. This shape is intersected with the edge or wire.

Raises

- **TypeError** – If *shape* if not an edge or wire.
- **RuntimeError** – If OCC method fails.

9.2.37 PointsAlongShapeByDistance

class afem.topology.create.**PointsAlongShapeByDistance**(*shape*, *maxd*, *d1=None*, *d2=None*, *shape1=None*, *shape2=None*, *nmin=0*)

Bases: *afem.geometry.create.PointsAlongCurveByDistance* (page 60)

Create a specified number of points along an edge or wire.

Parameters

- **shape** (afem.topology.entities.Edge (page 115) *or* afem.topology.entities.Wire (page 116)) – The shape.
- **maxd** (*float*) – The maximum allowed spacing between points. The actual spacing will be adjusted to not to exceed this value.
- **d1** (*float*) – An offset distance for the first point. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last point. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **shape1** (afem.topology.entities.Shape (page 109)) – A shape to define the first point. This shape is intersected with the edge or wire.
- **shape2** (afem.topology.entities.Shape (page 109)) – A shape to define the last point. This shape is intersected with the edge or wire.
- **nmin** (*int*) – Minimum number of points to create.

Raises

- **`TypeError`** – If *shape* if not a curve or wire.
- **`RuntimeError`** – If OCC method fails.

9.2.38 PlaneByEdges

class `afem.topology.create.`**`PlaneByEdges`**(*shape*, *tol=-1.0*)

Bases: `object`

Create a plane by fitting it to all the edges of a shape.

Parameters

- **`shape`** (`afem.topology.entities.Shape` (page 109)) – The shape containing the edges.
- **`tol`** (*`float`*) – Edges must be within this planar tolerance. The tolerance is the largest value between the value provided or the largest tolerance of any one of the edges in the shape.

`found`

Returns *True* if plane was found, *False* if not.

Return type bool

`plane`

Returns The plane. Returns *None* if no plane was found.

Return type *afem.geometry.entities.Plane* (page 52)

9.2.39 PlaneByIntersectingShapes

class `afem.topology.create.`**`PlaneByIntersectingShapes`**(*shape1*, *shape2*, *pnt=None*, *tol=-1.0*)

Bases: `object`

Create a plane by intersection two shapes. If no additional point is provided, then *`PlaneByEdges`* (page 137). is used. If a point is provided, then the edges are tessellated and the point is added to this list. Then the tool *`PlaneByApprox`* (page 68) is used.

Parameters

- **`shape1`** (`afem.topology.entities.Shape` (page 109)) – The first shape.
- **`shape2`** (`afem.topology.entities.Shape` (page 109)) – The second shape.
- **`pnt`** (`afem.geometry.entities.Point` (page 35)) – Additional point to add to the edges since they might be collinear.
- **`tol`** (*`float`*) – Edges must be within this planar tolerance. The tolerance is the largest value between the value provided or the largest tolerance of any one of the edges in the shape.

Raises **`ValueError`** – If there are less than three points after tessellating the edges.

`found`

Returns *True* if plane was found, *False* if not.

Return type bool

plane

Returns The plane. Returns *None* if no plane was found.

Return type *afem.geometry.entities.Plane* (page 52)

9.2.40 PlanesAlongShapeByNumber

class `afem.topology.create.`**`PlanesAlongShapeByNumber`**(*shape*, *n*, *ref_pln=None*, *d1=None*, *d2=None*, *shape1=None*, *shape2=None*)

Bases: `afem.geometry.create.PlanesAlongCurveByNumber` (page 70)

Create a specified number of planes along an edge or wire.

Parameters

- **shape** (`afem.topology.entities.Edge` (page 115) *or* `afem.topology.entities.Wire` (page 116)) – The shape.
- **n** (*int*) – Number of points to create ($n > 0$).
- **ref_pln** (`afem.geometry.entities.Plane` (page 52)) – The normal of this plane will be used to define the normal of all planes along the curve. If no plane is provided, then the first derivative of the curve will define the plane normal.
- **d1** (*float*) – An offset distance for the first point. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last point. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **shape1** (`afem.topology.entities.Shape` (page 109)) – A shape to define the first point. This shape is intersected with the edge or wire.
- **shape2** (`afem.topology.entities.Shape` (page 109)) – A shape to define the last point. This shape is intersected with the edge or wire.

Raises

- **TypeError** – If *shape* if not an edge or wire.
- **RuntimeError** – If OCC method fails.

9.2.41 PlanesAlongShapeByDistance

class `afem.topology.create.`**`PlanesAlongShapeByDistance`**(*shape*, *maxd*, *ref_pln=None*, *d1=None*, *d2=None*, *shape1=None*, *shape2=None*, *nmin=0*)

Bases: `afem.geometry.create.PlanesAlongCurveByDistance` (page 71)

Create planes along an edge or wire by distance between them.

Parameters

- **shape** (`afem.topology.entities.Edge` (page 115) *or* `afem.topology.entities.Wire` (page 116)) – The shape.

- **maxd** (*float*) – The maximum allowed spacing between planes. The actual spacing will be adjusted to not to exceed this value.
- **ref_pln** (afem.geometry.entities.Plane (page 52)) – The normal of this plane will be used to define the normal of all planes along the curve. If no plane is provided, then the first derivative of the curve will define the plane normal.
- **d1** (*float*) – An offset distance for the first point. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last point. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **shape1** (afem.topology.entities.Shape (page 109)) – A shape to define the first point. This shape is intersected with the edge or wire.
- **shape2** (afem.topology.entities.Shape (page 109)) – A shape to define the last point. This shape is intersected with the edge or wire.
- **nmin** (*int*) – Minimum number of planes to create.

Raises

- **TypeError** – If *shape* if not an edge or wire.
- **RuntimeError** – If OCC method fails.

9.3 Explore

9.3.1 ExploreWire

class afem.topology.explore.**ExploreWire**(*wire*, *face=None*)

Bases: object

Explore the edges of a wire.

Parameters

- **wire** (afem.topology.entities.Wire (page 116)) – The wire.
- **face** (afem.topology.entities.Face (page 117)) – The face.

nedges

Returns Number of edges.

Return type int

edges

Returns The ordered edges.

Return type list(*afem.topology.entities.Edge* (page 115))

current_vertices

Returns The result of the BRepTools_WireExplorer::CurrentVertex method. As the explorer traverses the edges, this stores the vertex connecting the current edge to the previous one. This will not be a complete list of ordered vertices.

Return type list(*afem.topology.entities.Vertex* (page 115))

ordered_vertices

Returns Attempt to provide the ordered vertices of a wire. If the wire is closed the first and last vertices will be the same.

Return type list(*afem.topology.entities.Vertex* (page 115))

9.3.2 ExploreFreeEdges

class `afem.topology.explore.`**`ExploreFreeEdges`**(*shape*)

Bases: `object`

Explore the free bounds of a shape.

Parameters **`shape`** (`afem.topology.entities.Shape` (page 109)) – The shape.

`closed_wires`

Returns Closed wires of free edges.

Return type list(*afem.topology.entities.Wire* (page 116))

`open_wires`

Returns Open wires of free edges.

Return type list(*afem.topology.entities.Wire* (page 116))

`free_edges`

Returns All free edges of the shape.

Return type list(*afem.topology.entities.Edge* (page 115))

9.4 Modify

9.4.1 DivideClosedShape

class `afem.topology.modify.`**`DivideClosedShape`**(*shape*)

Bases: `object`

Divide all closed faces in a shape.

Parameters **`shape`** (`afem.topology.entities.Shape` (page 109)) – The shape.

`shape`

Returns The divided shape.

Return type *afem.topology.entities.Shape* (page 109)

9.4.2 DivideContinuityShape

class `afem.topology.modify.`**`DivideContinuityShape`**(*shape*, *tol=0.001*, *continuity=GeomAbs_Shape.GeomAbs_C1*)

Bases: `object`

Divide a shape for a given continuity and tolerance.

Parameters

- **`shape`** (`afem.topology.entities.Shape` (page 109)) – The shape.

- **tol** (*float*) – The tolerance.
- **continuity** (*OCCT.GeomAbs.GeomAbs_Shape*) – The continuity to divide.

shape

Returns The divided shape.

Return type *afem.topology.entities.Shape* (page 109)

9.4.3 DivideC0Shape

class afem.topology.modify.**DivideC0Shape**(*shape*, *tol=0.001*)
Bases: *afem.topology.modify.DivideContinuityShape* (page 140)

Divide a shape at all C0 boundaries to form a C1 shape.

Parameters

- **shape** (afem.topology.entities.Shape (page 109)) – The shape.
- **tol** (*float*) – The tolerance.

9.4.4 UnifyShape

class afem.topology.modify.**UnifyShape**(*shape*, *edges=True*, *faces=True*, *bsplines=False*)
Bases: object

Unify edges and faces of a shape that lie on the same geometry.

Parameters

- **shape** (afem.topology.entities.Shape (page 109)) – The shape.
- **edges** (*bool*) – Option to unify all possible edges.
- **faces** (*bool*) – Option to unify all possible faces.
- **bsplines** (*bool*) – Option to concatenate the curves of edges if they are C1 continuous.

shape

Returns The unified shape.

Return type *afem.topology.entities.Shape* (page 109)

modified(*shape*)
Return a list of shapes modified from the given shape.

Parameters **shape** (afem.topology.entities.Shape (page 109)) – The shape.

Returns List of modified shapes.

Return type list(*afem.topology.entities.Shape* (page 109))

generated(*shape*)
Return a list of shapes generated from the given shape.

Parameters **shape** (afem.topology.entities.Shape (page 109)) – The shape.

Returns List of generated shapes.

Return type list(*afem.topology.entities.Shape* (page 109))

is_deleted(*shape*)
Check to see if shape is deleted.

Parameters **shape** (afem.topology.entities.Wire (page 116)) – The shape.

Returns *True* if deleted, *False* if not.

Return type bool

9.4.5 SewShape

class afem.topology.modify.**SewShape**(*shape=None*, *tol=None*, *min_tol=None*, *max_tol=None*, *cut_free_edges=False*, *non_manifold=False*)
Bases: object

Sew the shape.

Parameters

- **shape** (afem.topology.entities.Shape (page 109)) – The context shape to sew.
- **tol** (*float*) – Sewing tolerance. If *None* is provided then the average tolerance of the shape will be used. If no shape is provided, then a default value of 1.0e-7 is used.
- **min_tol** (*float*) – Minimum tolerance.
- **max_tol** (*float*) – Maximum tolerance.
- **cut_free_edges** (*bool*) – Option for cutting of free edges.
- **non_manifold** (*bool*) – Option for non-manifold processing.

Note: If *shape* is *None* then the user is expected to manually load the shape and perform the operation.

load(*shape*)
Load the context shape to sew.

Parameters **shape** (afem.topology.entities.Shape (page 109)) – The shape.

Returns None.

add(*shape*)
Add a shape to be sewed or controlled.

Parameters **shape** (afem.topology.entities.Shape (page 109)) – The shape.

Returns None.

perform()
Perform the sewing operation.

Returns None.

sewed_shape

Returns The sewed shape. May be a null shape if nothing is constructed.

Return type *afem.topology.entities.Shape* (page 109)

n_free_edges

Returns Number of free edges.

Return type int

`free_edges`

Returns Free edges.

Return type list(*afem.topology.entities.Edge* (page 115))

`n_multiple_edges`

Returns Number of edges connected to more than two faces.

Return type int

`multiple_edges`

Returns Multiple edges.

Return type list(*afem.topology.entities.Edge* (page 115))

`n_manifold_edges`

Returns Number of manifold edges.

Return type int

`manifold_edges`

Returns Manifold edges.

Return type list(*afem.topology.entities.Edge* (page 115))

`is_modified`(*shape*)

Check to see if input shape has been modified.

Parameters **`shape`** (`afem.topology.entities.Shape` (page 109)) – The shape.

Returns *True* if modified, *False* if not.

Return type bool

`modified`(*shape*)

Get a modified shape.

Parameters **`shape`** (`afem.topology.entities.Shape` (page 109)) – The shape.

Returns The modified shape.

Return type *afem.topology.entities.Shape* (page 109)

`is_modified_subshape`(*subshape*)

Check to see if input sub-shape has been modified.

Parameters **`subshape`** (`afem.topology.entities.Shape` (page 109)) – The sub-shape.

Returns *True* if modified, *False* if not.

Return type bool

`modified_subshape`(*subshape*)

Get a modified sub-shape.

Parameters **`subshape`** (`afem.topology.entities.Shape` (page 109)) – The sub-shape.

Returns The modified sub-shape.

Return type *afem.topology.entities.Shape* (page 109)

9.4.6 RebuildShapeWithShapes

class `afem.topology.modify.`**`RebuildShapeWithShapes`**(*old_shape*)

Bases: `object`

Rebuild a shape by requesting substitutions on a shape.

Parameters **`old_shape`** (`afem.topology.entities.Shape` (page 109)) – The old shape.

`remove`(*old_shape*)

Request to remove the old shape.

Parameters **`old_shape`** (`afem.topology.entities.Shape` (page 109)) – The old shape. This is usually a sub-shape of the original old shape.

Returns None.

`replace`(*old_shape*, *new_shapes*)

Request to replace the old shape with a list of new shapes.

Parameters

- **`old_shape`** (`afem.topology.entities.Shape` (page 109)) – The old shape. This is usually a sub-shape of the original old shape.
- **`new_shapes`** (*`list`*(`afem.topology.entities.Shape` (page 109))) – The new shapes.

Returns None.

`apply`()

Apply the substitutions to the original old shape and return a new shape.

Returns The new shape.

Return type *afem.topology.entities.Shape* (page 109)

9.4.7 RebuildShapeByTool

class `afem.topology.modify.`**`RebuildShapeByTool`**(*old_shape*, *tool*)

Bases: `object`

Rebuild a shape using a supported tool.

Parameters

- **`old_shape`** (`afem.topology.entities.Shape` (page 109)) – The old shape.
- **`tool`** (`afem.topology.bop.BopCore` (page 146)) – The tool.

Raises **`ValueError`** – If there are no sub-shapes to substitute.

Note: This tool will first try to make substitutions on the faces of the shape. If no faces exist, it will try the edges. If no edges exist it will try the vertices.

`new_shape`

Returns The new shape after substitutions.

Return type *afem.topology.entities.Shape* (page 109)

9.4.8 RebuildShapesByTool

class afem.topology.modify.**RebuildShapesByTool**(*old_shapes*, *tool*)

Bases: `object`

Rebuild multiple shapes using a supported tool. This method is intended to address the case where modified shapes from an old shape are not duplicated in adjacent shapes, like what would happen if rebuilding a single shape at a time without any context. If a modified shape has already been replaced in an old shape and is encountered again, it is not substituted in the later shape. This method will first try to make substitutions on the faces of the shape. If not faces exist it will try the edges. If no edges exist it will try the vertices.

Parameters

- **old_shapes** (*collections.Sequence(*afem.topology.entities.Shape (page 109)*)*) – The old shapes.
- **tool** (afem.topology.bop.BopCore (page 146)) – The tool.

new_shape(*old_shape*)

Get the new shape from the old shape.

Parameters **old_shape** (afem.topology.entities.Shape (page 109)) – The old shape provided in the initial inputs.

Returns The new shape after substitutions.

Return type *afem.topology.entities.Shape* (page 109)

Raises **RuntimeError** – If the old shape is not a key in the final results.

9.4.9 ShapeBSplineRestriction

class afem.topology.modify.**ShapeBSplineRestriction**(*shape*, *is_mutable=False*, *approx_srf=True*, *approx_crv3d=True*, *approx_crv2d=True*, *tol3d=0.01*, *tol2d=1e-06*, *dmax=9*, *nmax=10000*, *degree=True*, *rational=False*, *continuity3d=GeomAbs_Shape.GeomAbs_C1*, *continuity2d=GeomAbs_Shape.GeomAbs_C2*)

Bases: `object`

Re-approximate shape surfaces with B-splines.

Parameters

- **shape** (afem.topology.entities.Shape (page 109)) – The shape.
- **is_mutable** (*bool*) – Flag for mutable input.
- **approx_srf** (*bool*) – Flag to approximate surface.
- **approx_crv3d** (*bool*) – Flag to approximate 3-d curves.
- **approx_crv2d** (*bool*) – Flag to approximate 2-d curves.
- **tol3d** (*float*) – Tolerance for 3-d approximations.
- **tol2d** (*float*) – Tolerance for 2-d approximations.
- **dmax** (*int*) – Maximum allowed degree for approximation.

- **nmax** (*int*) – Maximum allowed number of segments for approximation.
- **degree** (*bool*) – If *True*, the approximation is made with degree limited by *dmax* but at the expense of *nmax*. If *False*, the approximation is made with number of spans limited by *nmax* but at the expense of *dmax*.
- **rational** (*bool*) – If *True*, the approximation for rational B-Spline and Bezier are converted to polynomial.
- **continuity3d** (*OCCT.GeomAbs.GeomAbs_Shape*) – Desired continuity for 3-d curve and surface approximation.
- **continuity2d** (*OCCT.GeomAbs.GeomAbs_Shape*) – Desired continuity for 2-d curve and surface approximation.

is_done

Returns *True* if modification was successful.

Return type bool

error_curve2d

Returns Error for 2-d curve approximation.

Return type float

error_curve3d

Returns Error for 3-d curve approximation.

Return type float

error_surface

Returns Error for surface approximation.

Return type float

nspan

Returns Number of spans for approximation.

Return type int

modified_shape(*shape*)

Return the modified shape corresponding to the given shape.

Parameters **shape** (afem.topology.entities.Shape (page 109)) – A shape.

Returns The modified shape.

Return type *afem.topology.entities.Shape* (page 109)

9.5 Boolean

9.5.1 BopCore

class afem.topology.bop.**BopCore**

Bases: object

Core class for Boolean operations and enabling attributes and methods for rebuilding shapes.

build()

Build the results.

Returns None.

is_done

Returns *True* if operation is done, *False* if not.

Return type bool

shape

Returns The resulting shape.

Return type *afem.topology.entities.Shape* (page 109)

modified(*shape*)

Return a list of shapes modified from the given shape.

Parameters **shape** (`afem.topology.entities.Shape` (page 109)) – The shape.

Returns List of modified shapes.

Return type list(*afem.topology.entities.Shape* (page 109))

generated(*shape*)

Return a list of shapes generated from the given shape.

Parameters **shape** (`afem.topology.entities.Shape` (page 109)) – The shape.

Returns List of generated shapes.

Return type list(*afem.topology.entities.Shape* (page 109))

is_deleted(*shape*)

Check to see if shape is deleted.

Parameters **shape** (`afem.topology.entities.Shape` (page 109)) – The shape.

Returns *True* if deleted, *False* if not.

Return type bool

9.5.2 BopAlgo

class afem.topology.bop.**BopAlgo**(*shape1*, *shape2*, *fuzzy_val*, *nondestructive*, *bop*)

Bases: `afem.topology.bop.BopCore` (page 146)

Base class for Boolean operations.

Parameters

- **shape1** (`afem.topology.entities.Shape` (page 109) *or None*) – The first shape.
- **shape2** (`afem.topology.entities.Shape` (page 109) *or None*) – The second shape.
- **fuzzy_val** (*float*) – Fuzzy tolerance value.
- **nondestructive** (*bool*) – Option to not modify the input shapes.
- **bop** – The OpenCASCADE class for the Boolean operation.

Note: If *shape1* or *shape2* is *None* then the user is expected to manually set the arguments and tools and build the result.

static set_parallel_mode(*flag*)
Global option to set the Boolean operations for parallel execution.

Parameters **flag** (*bool*) – Option for parallel execution. *True* turns parallel execution on, *False* turns it off.

Returns None.

debug(*path='.'*)
Export files for debugging Boolean operations.

Parameters **path** –

Returns

arguments

Returns The arguments.

Return type list(*afem.topology.entities.Shape* (page 109))

tools

Returns The tools.

Return type list(*afem.topology.entities.Shape* (page 109))

set_args(*shapes*)
Set the arguments.

Parameters **shapes** (*list(*afem.topology.entities.Shape (page 109)*)*) – The arguments.

Returns None.

set_tools(*shapes*)
Set the tools.

Parameters **shapes** (*list(*afem.topology.entities.Shape (page 109)*)*) – The tools.

Returns None.

vertices

Returns The vertices of the resulting shape.

Return type list(*afem.topology.entities.Vertex* (page 115))

edges

Returns The edges of the resulting shape.

Return type list(*afem.topology.entities.Edge* (page 115))

refine_edges()
Fuse C1 edges.

Returns None.

fuse_edges

Returns The result flag of edge refining.

Return type bool

section_edges

Returns A list of section edges as a result of intersection between the shapes.

Return type list(*afem.topology.entities.Edge* (page 115))

has_modified

Returns *True* if there is at least one modified shape.

Return type bool

has_generated

Returns *True* if there is at least one generated shape.

Return type bool

has_deleted

Returns *True* if there is at least one deleted shape.

Return type bool

9.5.3 FuseShapes

class afem.topology.bop.**FuseShapes**(*shape1=None*, *shape2=None*, *fuzzy_val=None*, *nondestructive=False*)

Bases: *afem.topology.bop.BopAlgo* (page 147)

Boolean fuse operation.

Parameters

- **shape1** (afem.topology.entities.Shape (page 109) *or None*) – The first shape.
- **shape2** (afem.topology.entities.Shape (page 109) *or None*) – The second shape.
- **fuzzy_val** (*float*) – Fuzzy tolerance value.
- **nondestructive** (*bool*) – Option to not modify the input shapes.

Note: If *shape1* or *shape2* is *None* then the user is expected to manually set the arguments and tools and build the result.

9.5.4 CutShapes

class afem.topology.bop.**CutShapes**(*shape1=None*, *shape2=None*, *fuzzy_val=None*, *nondestructive=False*)

Bases: *afem.topology.bop.BopAlgo* (page 147)

Boolean cut operation.

Parameters

- **shape1** (afem.topology.entities.Shape (page 109) *or None*) – The first shape.

- **shape2** (afem.topology.entities.Shape (page 109) *or None*) – The second shape.
- **fuzzy_val** (*float*) – Fuzzy tolerance value.
- **nondestructive** (*bool*) – Option to not modify the input shapes.

Note: If *shape1* or *shape2* is *None* then the user is expected to manually set the arguments and tools and build the result.

9.5.5 CommonShapes

class afem.topology.bop.**CommonShapes**(*shape1=None*, *shape2=None*, *fuzzy_val=None*, *nondestructive=False*)

Bases: *afem.topology.bop.BopAlgo* (page 147)

Boolean common operation.

Parameters

- **shape1** (afem.topology.entities.Shape (page 109) *or None*) – The first shape.
- **shape2** (afem.topology.entities.Shape (page 109) *or None*) – The second shape.
- **fuzzy_val** (*float*) – Fuzzy tolerance value.
- **nondestructive** (*bool*) – Option to not modify the input shapes.

Note: If *shape1* or *shape2* is *None* then the user is expected to manually set the arguments and tools and build the result.

9.5.6 IntersectShapes

class afem.topology.bop.**IntersectShapes**(*shape1=None*, *shape2=None*, *compute_pcurve1=False*, *compute_pcurve2=False*, *approximate=False*, *fuzzy_val=None*, *nondestructive=False*)

Bases: *afem.topology.bop.BopAlgo* (page 147)

Boolean intersect operation.

Parameters

- **shape1** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Surface (page 49)) – The first shape.
- **shape2** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Surface (page 49)) – The second shape.
- **compute_pcurve1** (*bool*) – Option to compute p-curves on shape 1.
- **compute_pcurve2** (*bool*) – Option to compute p-curves on shape 2.
- **approximate** (*bool*) – Option to approximate intersection curves.
- **fuzzy_val** (*float*) – Fuzzy tolerance value.

- **nondestructive** (*bool*) – Option to not modify the input shapes.

Note: If *shape1* or *shape2* is *None* then the user is expected to manually set the arguments and tools and build the result.

has_ancestor_face1(*edge*)
Get the ancestor face on the intersection edge on the first shape if available.

Parameters **edge** (afem.topology.entities.Edge (page 115)) – The edge.

Returns *True* and the face if available, *False* and *None* if not.

Return type tuple(bool, *afem.topology.entities.Face* (page 117) or None)

has_ancestor_face2(*edge*)
Get the ancestor face on the intersection edge on the second shape if available.

Parameters **edge** (afem.topology.entities.Edge (page 115)) – The edge.

Returns *True* and the face if available, *False* and *None* if not.

Return type tuple(bool, *afem.topology.entities.Face* (page 117) or None)

9.5.7 SplitShapes

class afem.topology.bop.**SplitShapes**(*shape1=None*, *shape2=None*, *fuzzy_val=None*, *nondestructive=False*)
Bases: *afem.topology.bop.BopAlgo* (page 147)

Split arbitrary shapes. This is a wrapper for the SALOME GEOMAlgo_Splitter tool.

Parameters

- **shape1** (afem.topology.entities.Shape (page 109) *or None*) – The first shape.
- **shape2** (afem.topology.entities.Shape (page 109) *or None*) – The second shape.
- **fuzzy_val** (*float*) – Fuzzy tolerance value.
- **nondestructive** (*bool*) – Option to not modify the input shapes.

Note: If *shape1* or *shape2* is *None* then the user is expected to manually set the arguments and tools and build the result.

9.5.8 VolumesFromShapes

class afem.topology.bop.**VolumesFromShapes**(*shapes*, *intersect=False*, *fuzzy_val=None*, *nondestructive=False*)
Bases: *afem.topology.bop.BopAlgo* (page 147)

Build solids from a list of shapes.

Parameters

- **shapes** (*list*(afem.topology.entities.Shape (page 109))) – The shapes.
- **intersect** (*bool*) – Option to intersect the shapes before building solids.

- **fuzzy_val** (*float*) – Fuzzy tolerance value.
- **nondestructive** (*bool*) – Option to not modify the input shapes.

box

Returns The bounding box of all provided shapes.

Return type *afem.topology.entities.Solid* (page 118)

nsolids

Returns The number of solids in the shape.

Return type int

solids

Returns The list of solids.

Return type list(*afem.topology.entities.Solid* (page 118))

9.5.9 CutCylindricalHole

class afem.topology.bop.**CutCylindricalHole**(*shape*, *radius*, *ax1*, *fuzzy_val=None*, *nondestructive=False*)

Bases: *afem.topology.bop.BopAlgo* (page 147)

Cut a cylindrical hole on a shape.

Parameters

- **shape** (afem.topology.entities.Shape (page 109)) – The shape.
- **radius** (*float*) – The radius of the hole.
- **afem.geometry.entities.Axis1** – The axis for the hole.
- **fuzzy_val** (*float*) – Fuzzy tolerance value.
- **nondestructive** (*bool*) – Option to not modify the input shapes.

9.5.10 LocalSplit

class afem.topology.bop.**LocalSplit**(*shape*, *tool*, *basis_shape*, *approximate=False*, *fuzzy_val=None*, *nondestructive=False*)

Bases: *afem.topology.bop.BopCore* (page 146)

Perform a local split of a shape in the context of a basis shape. This tool only splits faces.

Parameters

- **shape** (afem.topology.entities.Shape (page 109)) – The local shape.
- **tool** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Surface (page 49)) – The tool to split with.
- **basis_shape** (afem.topology.entities.Shape (page 109)) – The basis shape that the local shape is part of.
- **approximate** (*bool*) – Option to approximate intersection curves.
- **fuzzy_val** (*float*) – Fuzzy tolerance value.
- **nondestructive** (*bool*) – Option to not modify the input shapes.

9.5.11 SplitShapeByEdges

class afem.topology.bop.**SplitShapeByEdges**(*shape*, *edges=None*, *check_interior=True*)

Bases: *afem.topology.bop.BopCore* (page 146)

Split a shape using edges.

Parameters

- **shape** (afem.topology.entities.Shape (page 109)) – The basis shape.
- **edges** (*collections.Sequence(*afem.topology.entities.Edge (page 115)*) or None*) – The edges to split the shape with. If provided, then the results will be built during initialization. If none are provided then the user is expected to add edges and build manually.
- **check_interior** (*bool*) – Option to check internal intersections.

add_edges(*shapes*)

Add splittings edges or wires for the initial shape.

Parameters shapes (*collections.Sequence(*afem.topology.entities.Shape (page 109)*)*) – The splitting edges or wires.

Returns *True* if added, *False* if not.

Return type bool

add_wire_on_face(*w*, *f*)

Add the wire on the face.

Parameters

- **w** (afem.topology.entities.Wire (page 116)) – The wire.
- **f** (afem.topology.entities.Face (page 117)) – The face.

Returns None.

add_edge_on_face(*e*, *f*)

Add the edge on the face.

Parameters

- **e** (afem.topology.entities.Edge (page 115)) – The edge.
- **f** (afem.topology.entities.Face (page 117)) – The face.

Returns None

add_edges_on_face(*e*, *f*)

Parameters

- **e** (*collections.Sequence(*afem.topology.entities.Edge (page 115)*)*) – The edges.
- **f** (afem.topology.entities.Face (page 117)) – The face.

Returns None

add_edge_on_edge(*e1*, *e2*)

Add the edge on an existing edge.

Parameters

- **e1** (afem.topology.entities.Edge (page 115)) – The edge.

- **e2** (`afem.topology.entities.Edge` (page 115)) – The existing edge.

Returns None.

9.5.12 SplitWire

class `afem.topology.bop.`**`SplitWire`**(*wire*, *splitter*)

Bases: `object`

Split a wire with a shape.

Parameters

- **wire** (`afem.topology.entities.Wire` (page 116)) – The wire.
- **splitter** (`afem.topology.entities.Shape` (page 109)) – The splitter shape.

Raises **RuntimeError** – If the splitting algorithm fails.

wire

Returns The split wire.

Return type *afem.topology.entities.Wire* (page 116)

9.5.13 TrimOpenWire

class `afem.topology.bop.`**`TrimOpenWire`**(*wire*, *shape1*, *shape2=None*)

Bases: `object`

Trim an open wire between one or two shapes.

Parameters

- **wire** (`afem.topology.entities.Wire` (page 116)) – The wire.
- **shape1** (`afem.topology.entities.Shape` (page 109) *or* `afem.geometry.entities.Geometry` (page 34)) – The first shape.
- **shape2** (`afem.topology.entities.Shape` (page 109) *or* `afem.geometry.entities.Geometry` (page 34)) – The second shape.

Raises

- **TypeError** – If a wire is not provided or it is closed.
- **RuntimeError** – If zero or more than two split locations are found. The split shapes must result in one or two split locations. That is, they should intersect the wire at only one location.

split_wire

Returns The wire after splitting.

Return type *afem.topology.entities.Wire* (page 116)

first_wire

Returns The first trimmed segment.

Return type *afem.topology.entities.Wire* (page 116)

last_wire

Returns The last trimmed segment.

Return type *afem.topology.entities.Wire* (page 116)

trimmed_wire

Returns The interior trimmed segment.

Return type *afem.topology.entities.Wire* (page 116)

new_vertices

Returns New vertices after splitting.

Return type list(*afem.topology.entities.Vertex* (page 115))

all_vertices

Returns All ordered vertices after splitting the original wire but before trimming.

Return type list(*afem.topology.entities.Vertex* (page 115))

9.6 Offset

9.6.1 ProjectShape

class afem.topology.offset.**ProjectShape**(*shape*, *to_project*, *tol3d=0.0001*, *tol2d=None*, *continuity=GeomAbs_Shape.GeomAbs_C2*, *max_degree=14*, *max_seg=16*, *max_dist=None*, *limit=True*)

Bases: object

Project edges and wires onto a basis shape.

Parameters

- **shape** (afem.topology.entities.Shape (page 109)) – The shape to project to.
- **to_project** (*collections.Sequence*(afem.topology.entities.Edge (page 115) *or* afem.topology.entities.Wire (page 116))) – List of edges or wires to project.
- **tol3d** (*float*) – The 3-D tolerance.
- **tol2d** (*float*) – The 2-D tolerance. If not provided then *sqrt(tol3d)* is used.
- **continuity** (*OCCT.GeomAbs.GeomAbs_Shape*) – Desired continuity.
- **max_degree** (*int*) – Max degree.
- **max_seg** (*int*) – Max segments.
- **max_dist** (*float*) – Max distance between target shape and shapes to project. If not satisfied then results for the corresponding shape are discarded.
- **limit** (*bool*) – Option to limit projected edges to the face boundaries.

is_done

Returns *True* if projection was correctly built, *False* if not.

Return type bool

projection

Returns The projected shape. Tries to build the result as a compound of wires.

Return type *afem.topology.entities.Shape* (page 109)

nwires

Returns The number of wires in the projection.

Return type int

wires

Returns A list of wires in the projected shape.

Return type list(*afem.topology.entities.Wire* (page 116))

nedges

Returns The number of edges in the projection.

Return type int

edges

Returns A list of edges in the projected shape.

Return type list(*afem.topology.entities.Edge* (page 115))

9.6.2 OffsetShape

class afem.topology.offset.**OffsetShape**(*shape, offset, tol=None, join_mode=GeomAbs_JoinType.GeomAbs_Arc, remove_internal_edges=False, perform_simple=False*)

Bases: `object`

Offset a shape.

Parameters

- **shape** (`afem.topology.entities.Shape` (page 109)) – The shape. It may be a face, shell, a solid, or a compound of these kinds.
- **offset** (*float*) – The offset value. The offset will be outside the shape if positive and inside if negative.
- **tol** (*float*) – Tolerance for coincidence for generated shapes. If not provided the average tolerance of the shape is used.
- **join_mode** (*OCCT.GeomAbs.GeomAbs_JoinType*) – Option for how to fill holes that may appear when offsetting two adjacent faces.
- **remove_internal_edges** (*bool*) – Option to remove internal edges from the result.
- **perform_simple** (*bool*) – Option to use simple algorithm without intersection computation.

is_done

Returns *True* if done, *False* if not.

Return type bool

shape

Returns The offset shape.

Return type *afem.topology.entities.Shape* (page 109)

9.6.3 LoftShape

class afem.topology.offset.**LoftShape**(*sections*, *is_solid=False*, *make_ruled=False*, *pres3d=1e-06*, *check_compatibility=None*, *use_smoothing=None*, *par_type=None*, *continuity=None*, *max_degree=None*)

Bases: `object`

Loft a shape using a sequence of sections.

Parameters

- **sections** (*collections.Sequence(*afem.topology.entities.Vertex (page 115) *or* afem.topology.entities.Edge (page 115) *or* afem.topology.entities.Wire (page 116)*)*) – The sections of the loft. These are usually wires but the first and last section can be vertices. Edges are converted to wires before adding to the loft tool.
- **is_solid** (*bool*) – If *True* the tool will build a solid, otherwise it will build a shell.
- **make_ruled** (*bool*) – If *True* the faces between sections will be ruled surfaces, otherwise they are smoothed out by approximation.
- **pres3d** (*float*) – Defines the precision for the approximation algorithm.
- **check_compatibility** (*bool*) – Option to check the orientation of the sections to avoid twisted results and update to have the same number of edges.
- **use_smoothing** (*bool*) – Option to use approximation algorithm.
- **par_type** (*OCCT.Approx.Approx_ParametrizationType*) – Parametrization type.
- **continuity** (*OCCT.GeomAbs.GeomAbs_Shape*) – The desired continuity.
- **max_degree** (*int*) – The maximum degree for the approximation algorithm.

Raises **TypeError** – If any of the sections cannot be added to the tool because they are of the wrong type.

is_done

Returns *True* if done, *False* if not.

Return type bool

shape

Returns The lofted shape.

Return type *afem.topology.entities.Shape* (page 109)

first_shape

Returns The first/bottom shape of the loft if a solid was constructed.

Return type *afem.topology.entities.Shape* (page 109)

last_shape

Returns The last/top shape of the loft if a solid was constructed.

Return type *afem.topology.entities.Shape* (page 109)

max_degree

Returns The max degree used in the approximation algorithm

Return type int

generated_face(*edge*)

Get a face(s) generated by the edge. If the ruled option was used, then this returns each face generated by the edge. If the smoothing option was used, then this returns the face generated by the edge.

Parameters **edge** (`afem.topology.entities.Edge` (page 115)) – The edge.

Returns The face(s) generated by the edge.

Return type *afem.topology.entities.Shape* (page 109)

9.6.4 SweepShape

class `afem.topology.offset.`**SweepShape**(*spine*, *profile*)

Bases: `object`

Sweep a profile along a spine.

Parameters

- **spine** (`afem.geometry.entities.Curve` (page 43) *or* `afem.topology.entities.Edge` (page 115) *or* `afem.topology.entities.Wire` (page 116)) – The path for the sweep. This must be at least G1 continuous.
- **profile** (`afem.geometry.entities.Geometry` (page 34) *or* `afem.topology.entities.Shape` (page 109)) – The profile.

Raises **TypeError** – If `spine` is not or cannot be converted to a wire.

is_done

Returns *True* if done, *False* if not.

Return type bool

shape

Returns The swept shape.

Return type *afem.topology.entities.Shape* (page 109)

first_shape

Returns The first/bottom shape of the sweep.

Return type *afem.topology.entities.Shape* (page 109)

last_shape

Returns The last/top shape of the sweep.

Return type *afem.topology.entities.Shape* (page 109)

9.6.5 SweepShapeWithNormal

class `afem.topology.offset.`**SweepShapeWithNormal**(*spine*, *spine_support=None*, *tol3d=0.0001*, *tol_bound=0.0001*, *tol_angular=0.01*, *max_degree=None*, *max_segments=None*, *force_c1=None*, *transition_mode=BRepBuilderAPI_TransitionMode.BRepBuilderAPI*

Bases: `object`

Sweep sections along a spine using a support shape to define the local orientation.

Parameters

- **`spine`** (`afem.topology.entities.Wire` (page 116)) – The spine.
- **`spine_support`** (`afem.topology.entities.Shape` (page 109)) – The shape that will define the normal during the sweeping algorithm. To be effective, each edge of the spine must have a representation on one face of the spine support.
- **`tol3d`** (*`float`*) – The 3-D tolerance.
- **`tol_bound`** (*`float`*) – The boundary tolerance.
- **`tol_angular`** (*`float`*) – The angular tolerance.
- **`max_degree`** (*`int`*) – The maximum degree allowed in the resulting surface.
- **`max_segments`** (*`int`*) – The maximum number of segments allowed in the resulting surface.
- **`force_c1`** (*`bool`*) – If *True*, the tool will attempt to approximate a C1 surface if a swept surface proved to be C0.
- **`transition_mode`** (*`OCCT.BRepBuilderAPI.BRepBuilderAPI_TransitionMode`*) – The transition mode to manage discontinuities on the swept shape.

`add_profile`(*profile*, *with_contact=False*, *with_correction=False*)

Add the profile to the tool.

Parameters

- **`profile`** (`afem.topology.entities.Vertex` (page 115) *or* `afem.topology.entities.Edge` (page 115) *or* `afem.topology.entities.Wire` (page 116)) – The profile to add.
- **`with_contact`** (*`bool`*) – If *True*, then the profile is translated to be in contact with the spine.
- **`with_correction`** (*`bool`*) – If *True*, then the profile is rotated to be orthogonal to the spine's tangent.

Returns None.

Raises **`TypeError`** – If the profile is not a vertex, edge, or wire.

`is_ready`

Returns *True* if tool is ready to build the shape. *False* if not.

Return type bool

`build`()

Build the resulting shape.

Returns None.

`make_solid`()

Attempts to transform the sweeping shell into a solid.

Returns *True* if done, *False* if not.

Return type bool

`is_done`

Returns *True* if done, *False* if not.

Return type bool

`shape`

Returns The resulting shape.

Return type *afem.topology.entities.Shape* (page 109)

`first_shape`

Returns The first/bottom shape of the sweep.

Return type *afem.topology.entities.Shape* (page 109)

`last_shape`

Returns The last/top shape of the sweep.

Return type *afem.topology.entities.Shape* (page 109)

9.7 Distance

9.7.1 DistanceShapeToShape

class `afem.topology.distance.`**`DistanceShapeToShape`**(*shape1*, *shape2*, *deflection=1e-07*)

Bases: `object`

Calculate minimum distance between two shapes. If geometry is provided it will be converted to a shape.

Parameters

- **shape1** (`afem.topology.entities.Shape` (page 109) *or* `afem.geometry.entities.Geometry` (page 34)) – The first shape or geometry.
- **shape2** (`afem.topology.entities.Shape` (page 109) *or* `afem.geometry.entities.Geometry` (page 34)) – The second or geometry.

`is_done`

Returns *True* if algorithm is done, *False* if not.

Return type bool

`nsol`

Returns The number of solutions satisfying the minimum distance.

Return type

`dmin`

Returns The minimum distance.

Return type float

`inner_solution`

Returns *True* if one of the shapes is a solid and the other is completely or partially inside the solid.

Return type bool

`point_on_shape1`(*n=1*)

The point for the *n-th* solution on the first shape.

Parameters **n** (*int*) – The index.

Returns The point.

Return type *afem.geometry.entities.Point* (page 35)

point_on_shape2(*n=1*)
The point for the *n-th* solution on the second shape.

Parameters **n** (*int*) – The index.

Returns The point.

Return type *afem.geometry.entities.Point* (page 35)

support_type_shape1(*n=1*)
The type of support for the *n-th* solution on the first shape.

Parameters **n** (*int*) – The index.

Returns The support type.

Return type OCCT.BRepExtrema.BRepExtrema_SupportType

is_vertex_shape1(*n=1*)
Check if support type is a vertex for the first shape.

Parameters **n** (*int*) – The index.

Returns *True* if a vertex, *False* otherwise.

Return type bool

is_on_edge_shape1(*n=1*)
Check if support type is on an edge for the first shape.

Parameters **n** (*int*) – The index.

Returns *True* if on an edge, *False* otherwise.

Return type bool

is_in_face_shape1(*n=1*)
Check if support type is in a face for the first shape.

Parameters **n** (*int*) – The index.

Returns *True* if in a face, *False* otherwise.

Return type bool

support_type_shape2(*n=1*)
The type of support for the *n-th* solution on the second shape.

Parameters **n** (*int*) – The index.

Returns The support type.

Return type OCCT.BRepExtrema.BRepExtrema_SupportType

is_vertex_shape2(*n=1*)
Check if support type is a vertex for the second shape.

Parameters **n** (*int*) – The index.

Returns *True* if a vertex, *False* otherwise.

Return type bool

is_on_edge_shape2(*n=1*)
Check if support type is on an edge for the second shape.

Parameters **n** (*int*) – The index.

Returns *True* if on an edge, *False* otherwise.

Return type bool

is_in_face_shape2(*n=1*)
Check if support type is in a face for the second shape.

Parameters **n** (*int*) – The index.

Returns *True* if in a face, *False* otherwise.

Return type bool

support_on_shape1(*n=1*)
Get the shape where the *n-th* solution is on the first shape.

Parameters **n** (*int*) – The index.

Returns The support shape.

Return type *afem.topology.entities.Shape* (page 109)

support_on_shape2(*n=1*)
Get the shape where the *n-th* solution is on the second shape.

Parameters **n** (*int*) – The index.

Returns The support shape.

Return type *afem.topology.entities.Shape* (page 109)

par_on_edge_shape1(*n=1*)
Get the parameter of the *n-th* solution if it is on an edge of the first shape.

Parameters **n** (*int*) – The index.

Returns The parameter.

Return type float

par_on_edge_shape2(*n=1*)
Get the parameter of the *n-th* solution if it is on an edge of the second shape.

Parameters **n** (*int*) – The index.

Returns The parameter.

Return type float

par_on_face_shape1(*n=1*)
Get the parameters of the *n-th* solution if it is in a face of the first shape.

Parameters **n** (*int*) – The index.

Returns The parameters.

Return type tuple(float, float)

par_on_face_shape2(*n=1*)
Get the parameters of the *n-th* solution if it is in a face of the second shape.

Parameters **n** (*int*) – The index.

Returns The parameters.

Return type tuple(float, float)

normal_on_shape1(*n=1*)
Get a unit normal on the first shape where the *n-th* solution is located if it is in a face.

Parameters **n** (*int*) – The index.

Returns The unit normal.

Return type *afem.geometry.entities.Direction* (page 38)

Raises **ValueError** – If the solution is not in a face.

normal_on_shape2(*n=1*)
Get a unit normal on the second shape where the *n-th* solution is located if it is in a face.

Parameters **n** (*int*) – The index.

Returns The unit normal.

Return type *afem.geometry.entities.Direction* (page 38)

Raises **ValueError** – If the solution is not in a face.

9.7.2 DistanceShapeToShapes

class afem.topology.distance.**DistanceShapeToShapes**(*shape*, *other_shapes*)
Bases: object

Calculate the minimum distance between a shape and other shapes. Sort the results by distance.

Parameters

- **shape** (afem.topology.entities.Shape (page 109)) – The main shape.
- **other_shapes** (*list* (afem.topology.entities.Shape (page 109))) – The other shapes.

dmin

Returns The minimum distance of all shapes.

Return type float

dmax

Returns The maximum distance of all shapes.

Return type float

sorted_distances

Returns List of sorted distances.

Return type list(float)

nearest_shape

Returns The nearest shape.

Return type *afem.topology.entities.Shape* (page 109)

farthest_shape

Returns The farthest shape.

Return type *afem.topology.entities.Shape* (page 109)

sorted_shapes

Returns List of shapes sorted by distance.

Return type list(*afem.topology.entities.Shape* (page 109))

9.8 Fix

9.8.1 FixShape

class afem.topology.fix.**FixShape**(*shape*, *precision=None*, *min_tol=None*, *max_tol=None*, *context=None*)

Bases: `object`

Attempt to fix the shape by applying a number of general fixes.

Parameters

- **shape** (`afem.topology.entities.Shape` (page 109)) – The shape.
- **precision** (*float*) – Basic precision value.
- **min_tol** (*float*) – Minimum allowed tolerance.
- **max_tol** (*float*) – Maximum allowed tolerance.
- **context** (`afem.topology.entities.Shape` (page 109)) – The context shape.

Note: By default, the precision, minimum, and maximum tolerance values are Precision::Confusion() with OCCT, which is typically 1.0e-7.

shape

Returns The fixed shape.

Return type *afem.topology.entities.Shape* (page 109)

context

Returns The context.

Return type OCCT.ShapeBuild.ShapeBuild_ReShape

apply(*shape*)

Apply substitutions to the shape (or sub-shape) and get the result.

Parameters **shape** (`afem.topology.entities.Shape` (page 109)) – The shape.

Returns The new shape.

Return type *afem.topology.entities.Shape* (page 109)

static limit_tolerance(*shape*, *tol=1e-07*, *styp=TopAbs_ShapeEnum.TopAbs_SHAPE*)

Limit tolerances in a shape.

Parameters

- **shape** (`afem.topology.entities.Shape` (page 109)) – The shape.
- **tol** (*float*) – Target tolerance.
- **styp** (*OCCT.TopAbs.TopAbs_ShapeEnum*) – The level of shape to set (i.e., only vertices, only edges, only faces, or all shapes).

Returns *True* if the shape is valid after limiting tolerance, *False* if not.

Return type bool

static set_tolerance(*shape*, *tol*, *styp=TopAbs_ShapeEnum.TopAbs_SHAPE*)
Enforce tolerance on the given shape.

Parameters

- **shape** (`afem.topology.entities.Shape` (page 109)) – The shape.
- **tol** (*float*) – The tolerance.
- **styp** (*OCCT.TopAbs.TopAbs_ShapeEnum*) – The level of shape to set (i.e., only vertices, only edges, only faces, or all shapes).

Returns None.

9.9 Properties

9.9.1 ShapeProps

class afem.topology.props.**ShapeProps**
Bases: `object`

Base class for shape properties.

mass

Returns The mass of the shape. This corresponds to total length for linear properties, total area for surface properties, or total volume for volume properties.

Return type float

cg

Returns The center of gravity.

Return type *afem.geometry.entities.Point* (page 35)

static_moments

Returns The static moments of inertia Ix, Iy, and Iz.

Return type tuple(float)

matrix_of_inertia

Returns The 3 x 3 matrix of inertia.

Return type numpy.ndarray

moment_of_inertia(*axis*)
Compute the moment of inertia about the axis.

Parameters **axis** (`afem.geometry.entities.Axis1` (page 41)) – The axis.

Returns The moment of inertia.

Return type float

9.9.2 LinearProps

class afem.topology.props.**LinearProps**(*shape*, *skip_shared=True*)

Bases: *afem.topology.props.ShapeProps* (page 165)

Calculate linear properties of a shape.

Parameters

- **shape** (afem.topology.entities.Shape (page 109)) – The shape.
- **skip_shared** (*bool*) – If *True*, edges shared by two or more faces are taken into calculation only once.

length

Returns The total length of all edges of the shape.

Return type float

9.9.3 SurfaceProps

class afem.topology.props.**SurfaceProps**(*shape*, *tol=1e-07*, *skip_shared=False*)

Bases: *afem.topology.props.ShapeProps* (page 165)

Calculate surface properties of a shape.

Parameters

- **shape** (afem.topology.entities.Shape (page 109)) – The shape.
- **tol** (*float*) – Maximum relative error of computed area for each face.
- **skip_shared** (*bool*) – If *True*, faces shared by two or more shells are taken into calculation only once.

area

Returns The total area of all faces of the shape.

Return type float

9.9.4 VolumeProps

class afem.topology.props.**VolumeProps**(*shape*, *tol=1e-07*, *only_closed=False*, *skip_shared=False*)

Bases: *afem.topology.props.ShapeProps* (page 165)

Calculate volume properties of a shape.

Parameters

- **shape** (afem.topology.entities.Shape (page 109)) – The shape.
- **tol** (*float*) – Maximum relative error of computed volume for each solid.
- **only_closed** (*bool*) – If *True*, then faces must belong to closed shells.
- **skip_shared** (*bool*) – If *True*, volumes formed by equal faces (i.e., the same TShape, location, and orientation) are taken into calculation only once.

volume

Returns The total volume of all solids of the shape.

Return type float

9.9.5 LengthOfShapes

class afem.topology.props.**LengthOfShapes**(*shapes*)

Bases: object

Calculate the total length of all edges of each shape and sort the results.

Parameters shapes (*collections.Sequence(*afem.topology.entities.Shape (page 109)*)*) – The shapes.

min_length

Returns The minimum length.

Return type float

max_length

Returns The maximum length.

Return type float

sorted_lengths

Returns List of sorted lengths.

Return type list(float)

shortest_shape

Returns The shortest shape.

Return type *afem.topology.entities.Shape* (page 109)

longest_shape

Returns The longest shape.

Return type *afem.topology.entities.Shape* (page 109)

sorted_shapes

Returns List of shapes sorted by length.

Return type list(*afem.topology.entities.Shape* (page 109))

9.9.6 AreaOfShapes

class afem.topology.props.**AreaOfShapes**(*shapes*)

Bases: object

Calculate the total area of each face for each shape and sort the results.

Parameters shapes (*collections.Sequence(*afem.topology.entities.Shape (page 109)*)*) – The shapes.

min_area

Returns The minimum area.

Return type float

max_area

Returns The maximum area.

Return type float

sorted_areas

Returns List of sorted areas.

Return type list(float)

smallest_shape

Returns The smallest shape.

Return type *afem.topology.entities.Shape* (page 109)

largest_shape

Returns The largest shape.

Return type *afem.topology.entities.Shape* (page 109)

sorted_shape

Returns List of shapes sorted by area.

Return type list(*afem.topology.entities.Shape* (page 109))

9.10 Check

9.10.1 CheckShape

class afem.topology.check.**CheckShape**(*shape*, *geom=True*)

Bases: object

Check shape and its sub-shapes for errors.

Parameters

- **shape** (afem.topology.entities.Shape (page 109)) – The shape.
- **geom** (*bool*) – Option to check geometry in additional to topology.

is_valid

Returns *True* if the shape and all of its sub-shapes are valid, *False* if not.

Return type bool

invalid_shapes

Returns List of invalid shapes.

Return type list(*afem.topology.entities.Shape* (page 109))

print_errors()

Print the errors.

Returns None.

log_errors()

Log the errors at the “info” level.

Returns None.

is_subshape_valid(*shape*)
Check if a sub-shape of the original shape is valid.

Parameters **shape** (afem.topology.entities.Shape (page 109)) – The sub-shape.

Returns *True* if valid, *False* if not.

9.10.2 ClassifyPointInSolid

class afem.topology.check.**ClassifyPointInSolid**(*solid*, *pnt=None*, *tol=1e-07*)
Bases: object

Classify a point in a solid.

Parameters

- **solid** (afem.topology.entities.Solid (page 118)) – The solid.
- **pnt** (*point_like*) – The point. If not provided the *perform()* method will need to be used.
- **tol** (*float*) – The tolerance.

is_in

Returns *True* if point is in solid, *False* if not.

Return type bool

is_out

Returns *True* if point is outside the solid, *False* if not.

Return type bool

is_on

Returns *True* if point is on the solid, *False* if not.

Return type bool

is_unknown

Returns *True* if classification is unknown, *False* if not.

Return type bool

is_on_face

Returns *True* if point is on a face, *False* if not.

Return type bool

perform(*pnt*, *tol=1e-07*)
Perform the classification with the point and tolerance.

Parameters

- **pnt** (*point_like*) – The point.
- **tol** (*float*) – The tolerance.

Returns None.

face()
Get the face the point is on.

Returns The face.

Return type *afem.topology.entities.Face* (page 117)

9.11 Transform

`afem.topology.transform.`**`mirror_shape`**(*shape*, *pln*)

Mirror a shape about a plane.

Parameters

- **`shape`** (`afem.topology.entities.Shape` (page 109)) – The shape.
- **`pln`** (`afem.geometry.entities.Plane` (page 52)) – The plane.

Returns The mirrored shape.

Return type *afem.topology.entities.Shape* (page 109)

Raises **`RuntimeError`** – If the transformation fails or is not done.

CHAPTER

TEN

ADAPTOR

The afem.adaptor package provides a set of entities that enable a curve-like interface for edges and wires, and a surface-like interface for faces. This can be useful when needing to use shapes in some geometrical tools and operations. The entities and toosl can be imported by:

```
from afem.adaptor import *
```

The example below demonstrates some of the main features of the adaptor package:

```
from afem.adaptor import *
from afem.geometry import *
from afem.graphics import *
from afem.topology import *

gui = Viewer()

# Define some points to interpolate
pts = [(0, 0, 0), (5, 5, 0), (10, 0, 0)]

# Interpolate the points
curve = NurbsCurveByInterp(pts).curve

# Create an edge from the curve
edge = EdgeByCurve(curve).edge
edge.set_color(1, 0, 0)

# Create an adaptor curve from the edge
adp_crv = EdgeAdaptorCurve.by_edge(edge)

# Adaptors work in some geometry tools like projection
p1 = Point(8, 8, 0)
proj = ProjectPointToCurve(p1, adp_crv)

# Create an edge to shown projection
e1 = EdgeByPoints(p1, proj.nearest_point).edge
e1.set_color(0, 0, 1)

# View the results
gui.add(edge, p1, e1)
gui.view_top()
gui.start()

# Create a face by dragging the edge
face = FaceByDrag(edge, (0, 0, 10)).face
face.set_color(0, 1, 0)
```

(continues on next page)

(continued from previous page)

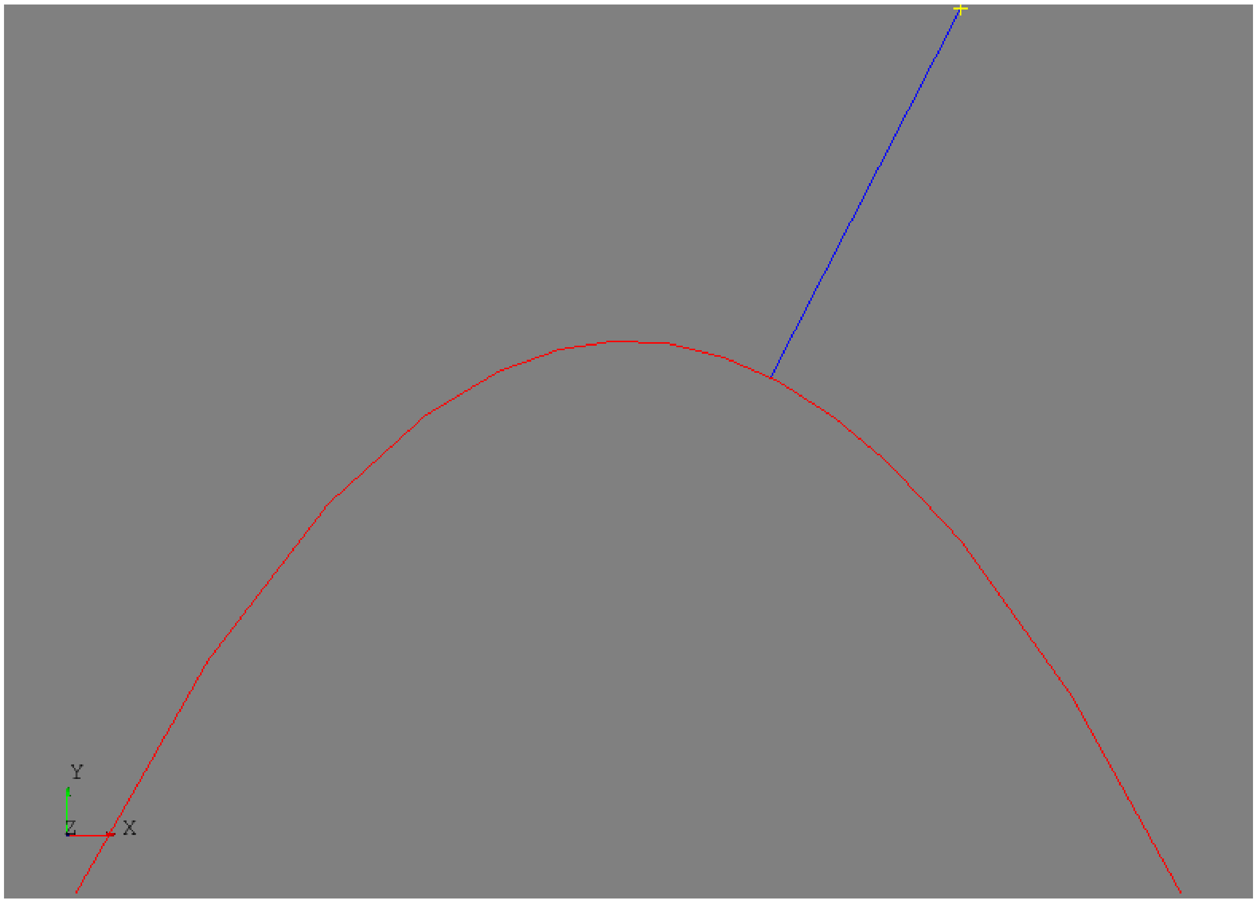
```
# Create an adaptor surface from the face
adp_srf = FaceAdaptorSurface.by_face(face)

# Project a point to the adaptor surface
p2 = Point(8, 8, 5)
proj = ProjectPointToSurface(p2, adp_srf)

# Create an edge to shown projection
e2 = EdgeByPoints(p2, proj.nearest_point).edge
e2.set_color(0, 0, 1)

# View the results
gui.add(face, p2, e2)
gui.view_iso()
gui.start()

# Adaptors have similar interfaces to curves and surfaces
p3 = adp_crv.eval(10)
p4 = adp_srf.eval(10, -5)

# View the points
gui.add(p3, p4)
gui.start()

# Create points along an adaptor curve
tool = PointsAlongCurveByNumber(adp_crv, 10)

# View the points
gui.clear()
gui.add(edge, *tool.points)
gui.view_top()
gui.start()

# Define some new points to interpolate
pts = [(10, 0, 0), (15, 5, 0), (20, 0, 0)]

# Interpolate the points
curve2 = NurbsCurveByInterp(pts).curve

# Create an edge from the curve
edge2 = EdgeByCurve(curve2).edge
edge.set_color(1, 0, 0)

# Build a wire from the edges
wire = WiresByConnectedEdges([edge, edge2]).wires[0]
wire.set_color(1, 0, 0)

# Create an adaptor curve from a wire. The static method "to_adaptor" can
# be used for convenience to convert entities to an adaptor if possible.
adp_crv2 = AdaptorCurve.to_adaptor(wire)

# Create points along the wire adaptor curve
tool = PointsAlongCurveByNumber(adp_crv2, 20)

# View the points
gui.clear()
```

(continues on next page)

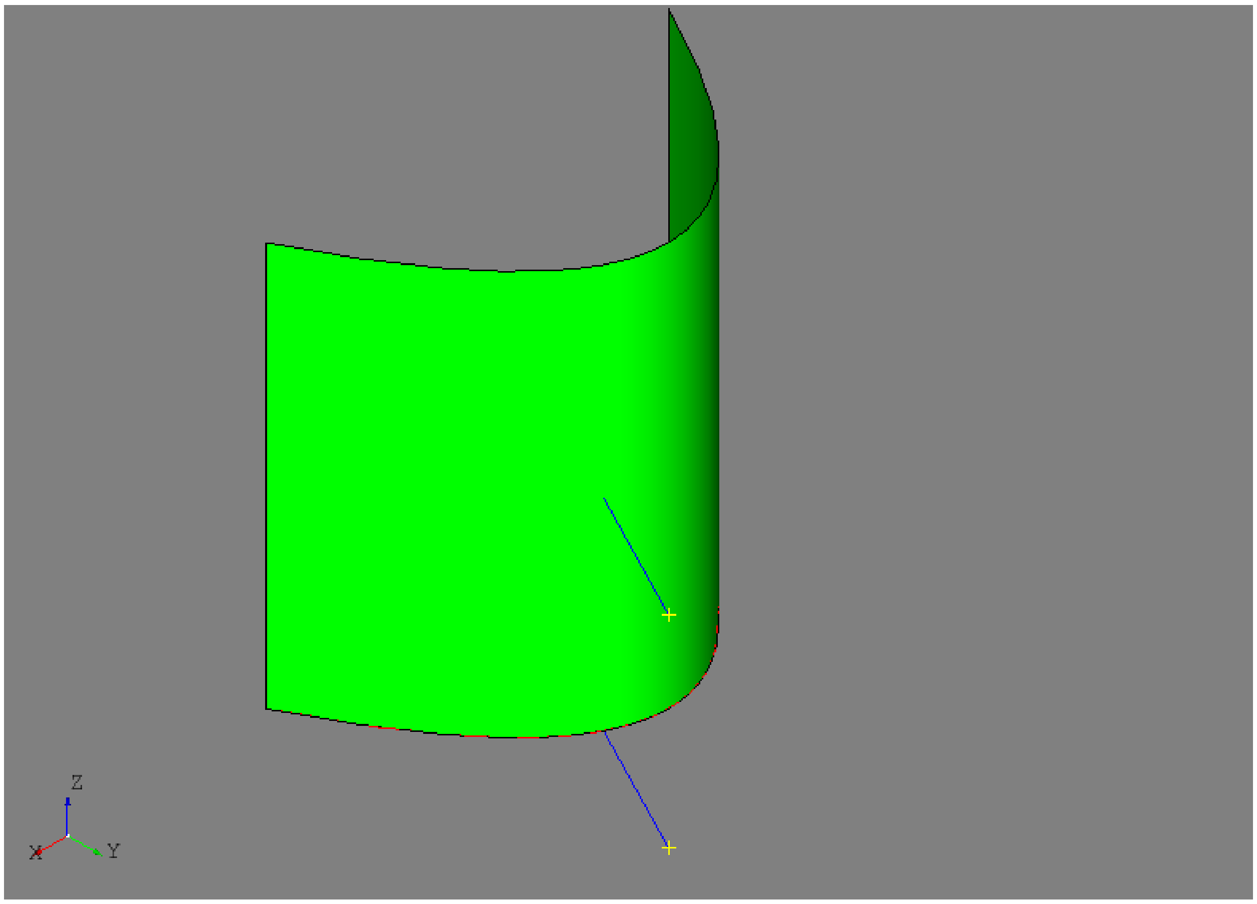
(continued from previous page)

```
gui.add(wire, *tool.points)
gui.view_top()
gui.start()
```

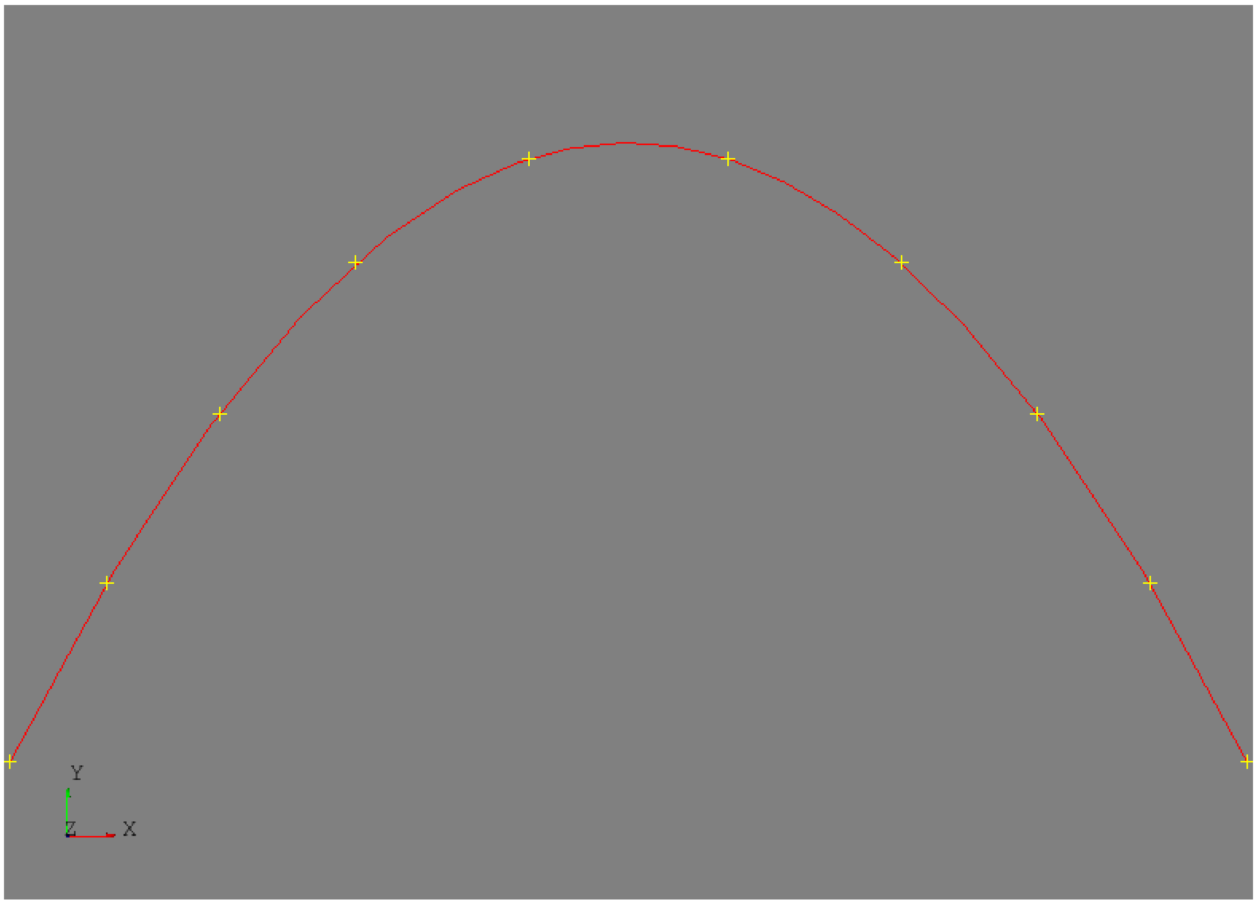
The results of projecting the point to the adaptor curve (i.e., the edge) are shown below:



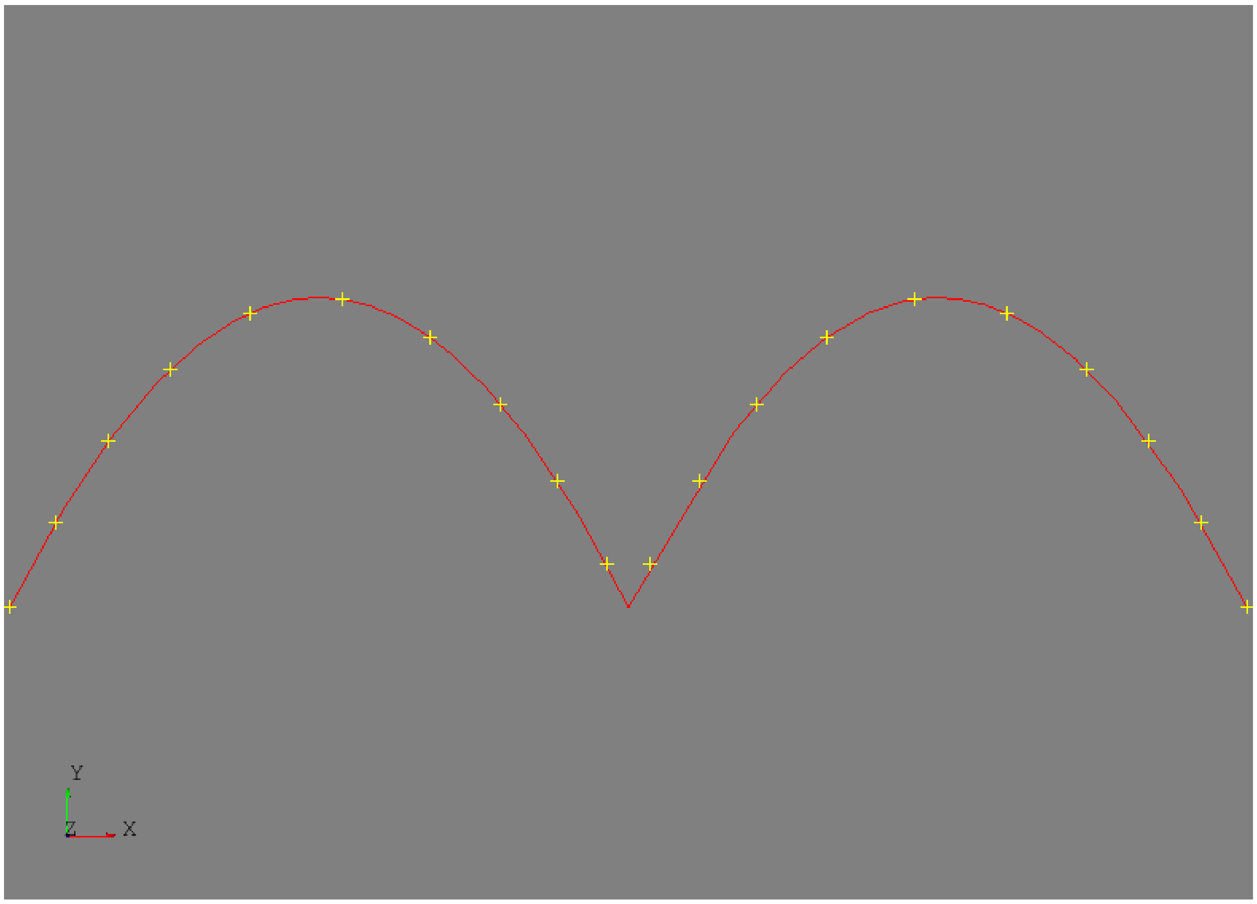
Extending the edge to form a face and then projection a point to the adaptor surface (i.e., the face) is shown below:



Creating points along the adaptor curve for the edge can be done using the *PointsAlongCurveByNumber* (page 59) tool:



Creating points along the adaptor curve for the wire can also be done using the `PointsAlongCurveByNumber` (page 59) tool:



10.1 Entities

10.1.1 AdaptorBase

`class afem.adaptor.entities.`**`AdaptorBase`**`(`*obj*`)`

Bases: `object`

Base class for adaptor types.

Parameters **`obj`** (*`OCCT.Adaptor3d.Adaptor3d_Curve or OCCT.Adaptor3d.Adaptor3d_Surface`*) – The underlying OpenCASCADE type.

10.1.2 AdaptorCurve

`class afem.adaptor.entities.`**`AdaptorCurve`**`(`*obj*`)`

Bases: *`afem.adaptor.entities.AdaptorBase`* (page 176)

Base class for adaptor curves around `Adaptor3d_Curve`.

`object`

Returns The underlying OpenCASCADE object.

Return type OCCT.Adaptor3d.Adaptor3d_Curve

`u1`

Returns The first parameter.

Return type float

`u2`

Returns The last parameter.

Return type float

`continuity`

Returns The continuity of the adaptor curve.

Return type OCCT.GeomAbs.GeomAbs_Shape

`is_closed`

Returns *True* if curve is closed, *False* if not.

Return type bool

`is_periodic`

Returns *True* if curve is periodic, *False* if not.

Return type bool

`type`

Returns The type of the adaptor curve.

Return type OCCT.GeomAbs.GeomAbs_CurveType

`length`

Returns Curve length.

Return type float

`eval`(*u*)

Evaluate a point on the curve.

Parameters **u** (*`float`*) – Curve parameter.

Returns Curve point.

Return type *afem.geometry.entities.Point* (page 35)

`deriv`(*u*, *d=1*)

Evaluate a derivative on the curve.

Parameters

- **u** (*`float`*) – Curve parameter.
- **d** (*`int`*) – Derivative to evaluate.

Returns Curve derivative.

Return type *afem.geometry.entities.Vector* (page 39)

`arc_length`(*u1*, *u2*, *tol=1e-07*)

Calculate the curve length between the parameters.

Parameters

- **u1** (*`float`*) – First parameter.

- **u2** (*float*) – Last parameter.
- **tol** (*float*) – The tolerance.

Returns Curve length.

Return type float

static to_adaptor(*entity*)
Convert the entity to an adaptor type if possible.

Parameters **entity** (afem.adaptor.entities.AdaptorCurve (page 176) *or* afem.geometry.entities.Curve (page 43) *or* afem.topology.entities.Edge (page 115) *or* afem.topology.entities.Wire (page 116)) – The entity.

Returns The adaptor curve.

Return type *afem.adaptor.entities.AdaptorCurve* (page 176)

Raises **TypeError** – If *entity* cannot be converted to an adaptor.

10.1.3 GeomAdaptorCurve

class afem.adaptor.entities.**GeomAdaptorCurve**(*obj*)
Bases: *afem.adaptor.entities.AdaptorCurve* (page 176)

Geometry adaptor curve around `GeomAdaptor_Curve`.

classmethod by_curve(*curve*, *u1=None*, *u2=None*)
Create by a curve.

Parameters

- **curve** (afem.geometry.entities.Curve (page 43)) – The curve.
- **u1** (*float*) – First parameter.
- **u2** (*float*) – Last parameter.

Returns The adaptor curve.

Return type *afem.adaptor.entities.GeomAdaptorCurve* (page 178)

Note: Both *u1* and *u2* must be provided in order to be used.

10.1.4 EdgeAdaptorCurve

class afem.adaptor.entities.**EdgeAdaptorCurve**(*obj*)
Bases: *afem.adaptor.entities.AdaptorCurve* (page 176)

Edge adaptor curve around `BRepAdaptor_Curve`.

classmethod by_edge(*edge*, *face=None*)
Create by an edge.

Parameters

- **edge** (afem.topology.entities.Edge (page 115)) – The edge.
- **face** (afem.topology.entities.Edge (page 115)) – The face the edge lies in.

Returns The adaptor curve.

Return type *afem.adaptor.entities.EdgeAdaptorCurve* (page 178)

10.1.5 WireAdaptorCurve

class afem.adaptor.entities.**WireAdaptorCurve**(*obj*)

Bases: *afem.adaptor.entities.AdaptorCurve* (page 176)

Wire adaptor curve around `BRepAdaptor_CompCurve`.

classmethod by_wire(*wire*, *curvilinear_knots=False*)

Create by a wire.

Parameters

- **wire** (afem.topology.entities.Wire (page 116)) – The wire.
- **curvilinear_knots** (*bool*) – Option to determine the knot values of the curve by their curvilinear abscissa.

Returns The adaptor curve.

Return type *afem.adaptor.entities.WireAdaptorCurve* (page 179)

10.1.6 AdaptorSurface

class afem.adaptor.entities.**AdaptorSurface**(*obj*)

Bases: *afem.adaptor.entities.AdaptorBase* (page 176)

Base class for adaptor surfaces around `Adaptor3d_Surface`.

object

Returns The underlying OpenCASCADE object.

Return type OCCT.Adaptor3d.Adaptor3d_Surface

u1

Returns The first parameter in u-direction.

Return type float

u2

Returns The last parameter in u-direction.

Return type float

v1

Returns The first parameter in v-direction.

Return type float

v2

Returns The last parameter in v-direction.

Return type float

type

Returns The type of the adaptor surface.

Return type OCCT.GeomAbs.GeomAbs_SurfaceType

eval (*u=0.0*, *v=0.0*)
Evaluate a point on the surface.

Parameters

- **u** (*float*) – Surface u-parameter.
- **v** (*float*) – Surface v-parameter.

Returns Surface point.

Return type *afem.geometry.entities.Point* (page 35)

deriv (*u*, *v*, *nu*, *nv*)
Evaluate a derivative on the surface.

Parameters

- **u** (*float*) – Surface u-parameter.
- **v** (*float*) – Surface v-parameter.
- **nu** (*int*) – Derivative in u-direction.
- **nv** (*int*) – Derivative in v-direction.

Returns Surface derivative.

Return type *afem.geometry.entities.Vector* (page 39)

norm (*u*, *v*)
Evaluate a normal on the surface.

Parameters

- **u** (*float*) – Surface u-parameter.
- **v** (*float*) – Surface v-parameter.

Returns Surface normal.

Return type *afem.geometry.entities.Vector* (page 39)

static to_adaptor (*entity*)
Convert the entity to an adaptor type if possible.

Parameters **entity** (`afem.adaptor.entities.AdaptorSurface` (page 179) *or* `afem.geometry.entities.Surface` (page 49) *or* `afem.topology.entities.Face` (page 117)) – The entity.

Returns The adaptor surface.

Return type *afem.adaptor.entities.AdaptorSurface* (page 179)

Raises **TypeError** – If *entity* cannot be converted to an adaptor.

10.1.7 GeomAdaptorSurface

class afem.adaptor.entities.**GeomAdaptorSurface** (*obj*)
Bases: *afem.adaptor.entities.AdaptorSurface* (page 179)

Geometry adaptor surface around `GeomAdaptor_Surface`.

classmethod by_surface (*surface*, *u1=None*, *u2=None*, *v1=None*, *v2=None*, *tolu=0.0*, *tolv=0.0*)
Create by a surface.

Parameters

- **surface** (afem.geometry.entities.Surface (page 49)) – The surface.
- **u1** – The lower u-parameter.
- **u2** – The upper u-parameter.
- **v1** – The lower v-parameter.
- **v2** – The upper v-parameter.
- **tolu** – Tolerance for u-direction.
- **tolv** – Tolerance for v-direction.

Returns The adaptor surface.

Return type *afem.adaptor.entities.GeomAdaptorSurface* (page 180)

Note: All *u1*, *u2*, *v1*, and *v2* must be provided to be used.

10.1.8 FaceAdaptorSurface

class afem.adaptor.entities.**FaceAdaptorSurface**(*obj*)

Bases: *afem.adaptor.entities.AdaptorSurface* (page 179)

Face adaptor surface around `BRepAdaptor_Surface`.

classmethod by_face(*face*, *restrict=True*)

Create by a face.

Parameters

- **face** (afem.topology.entities.Face (page 117)) – The face.
- **restrict** (*bool*) – Option to restruct the uv-domain of the surface to the boundary of the face.

Returns The adaptor surface.

Return type *afem.adaptor.entities.FaceAdaptorSurface* (page 181)

CHAPTER

ELEVEN

SMESH

The afem.smesh packages provides entities and tools for working with the meshing library which is a standalone version from the Salome Platform. Similar to the OpenCASCADE C++ library, the SMESH library is linked to OpenCASCADE and exposed to Python via pybind11. To provide a more intuitive and "Pythonic" user interface, the smesh package within AFEM provides a number of wrappers around the core SMESH entities. This approach is similar to the geometry package. The entities and tools can be imported by:

```
from afem.smesh import *
```

There are a few key concepts and entities that need to be understood by the user in order use the meshing package effectively. These are best discussed by working through a simple but complete example:

```
from afem.graphics import Viewer
from afem.smesh import (MeshGen, NetgenAlgo2D, NetgenSimple2D, MaxLength1D,
                        Regular1D)
from afem.topology import *

# Create a simple solid box
box = BoxBySize(10, 10, 10).solid

# Get a list of faces and edges of the shape for later use
faces = box.faces
edges = box.edges

# Initialize the mesh generator
gen = MeshGen()

# Create a new mesh using the box as the shape
mesh = gen.create_mesh(box)

# Define algorithms and hypotheses
alg2d = NetgenAlgo2D(gen)
hyp2d_1 = NetgenSimple2D(gen, 1.)
hyp2d_2 = NetgenSimple2D(gen, 1., allow_quads=False)
alg1d = Regular1D(gen)
hyp1d = MaxLength1D(gen, 0.25)

# Add them to the mesh
mesh.add_hypotheses([alg2d, hyp2d_1])
mesh.add_hypothesis(hyp2d_2, faces[-1])
mesh.add_hypotheses([alg1d, hyp1d], edges[-1])

# Compute the mesh
gen.compute(mesh)
```

(continues on next page)

(continued from previous page)

```

# View the mesh
gui = Viewer()
gui.add(mesh)
gui.start()
gui.clear()

# Get sub-meshes from sub-shapes
face_submesh = mesh.get_submesh(faces[0])
edge_submesh = mesh.get_submesh(edges[0])

# View the face sub-mesh (2-D elements)
gui.add(face_submesh)
gui.start()
gui.clear()

# View the edge sub-mesh (1-D elements)
gui.add(edge_submesh)
gui.start()
```

This example creates a simple solid box and then generates a 2-D mesh on its faces. Like any other Python script or module, the required entities must be imported:

```
from afem.graphics import Viewer
from afem.smesh import (MeshGen, NetgenAlgo2D, NetgenSimple2D, MaxLength1D,
                        Regular1D)
from afem.topology import *
```

The `BoxBySize` (page 133) tool is used to create a simple solid box:

```
bbox = BoxBySize(10, 10, 10).solid
```

Providing three numbers creates a box with one corner at (0, 0, 0) and the other at x=10, y=10, and z=10.

When it is time to begin the meshing process, it is required to first initialize the core mesh generator and data structure `MeshGen` (page 192). This entity is responsible for managing the entire meshing data structure and typically only one should be created. Generating more than one instance might be used if two entirely different meshes need to be created in a single process. Create an instance by:

```
gen = MeshGen()
```

The reference to this variable (`gen` is this example) should not be destroyed until the meshing process is complete. When the reference count drops to zero, the entire meshing data structure will be destroyed and the memory freed.

A new mesh entity is created by:

```
mesh = gen.create_mesh(box)
```

The `box` variable is provided to this method so that is sets the box as the "shape to mesh." Selecting the shape to mesh is an important step as this main shape will be used to derive and access information if sub-meshes or local meshing controls are applied later in the process. In practice, the main shape to mesh will typically be a single `Compound` (page 119) that contains all the shapes of the current model. An error will result if a sub-mesh or local mesh control is applied to a sub-shape that is not a member of the original main shape.

The next step in the meshing process is usually to define algorithms and hypotheses that will control mesh types and parameters like edge length and gradation. Usually, it is required to apply both an algorithm and hypothesis to a shape. The algorithm controls execution within the meshing process while a hypothesis controls the meshing parameters. In the example, a 2-D algorithm and hypothesis is created for the Netgen mesh generator that will generate 2-D elements on the faces of the shape:

```
alg2d = NetgenAlgo2D(gen)
hyp2d_1 = NetgenSimple2D(gen, 1.)
hyp2d_2 = NetgenSimple2D(gen, 1., allow_quads=False)
```

The `hyp2d_1` will be the default global hypothesis and `hyp2d_2` is created to generate a triangular mesh. At the same time, a 1-D algorithm and hypothesis is created that will be used to control the mesh on specific edges:

```
alg1d = Regular1D(gen)
hyp1d = MaxLength1D(gen, 0.25)
```

These algorithms and hypotheses can be added to the mesh and sub-shapes if applicable. First, the global algorithm and hypothesis is applied:

```
mesh.add_hypotheses([alg2d, hyp2d_1])
```

Here, no shape is provided so this algorithm and hypothesis is applied to the main shape and all applicable sub-shapes. If more local control is desired, algorithms and hypotheses must be added along with their associated sub-shape(s):

```
mesh.add_hypothesis(hyp2d_2, faces[-1])
```

The second hypothesis using triangles is added to one of the faces of the main shape. Only the new hypothesis was applied since the same algorithm can be used. Controlling the mesh along a specific edge can be done by:

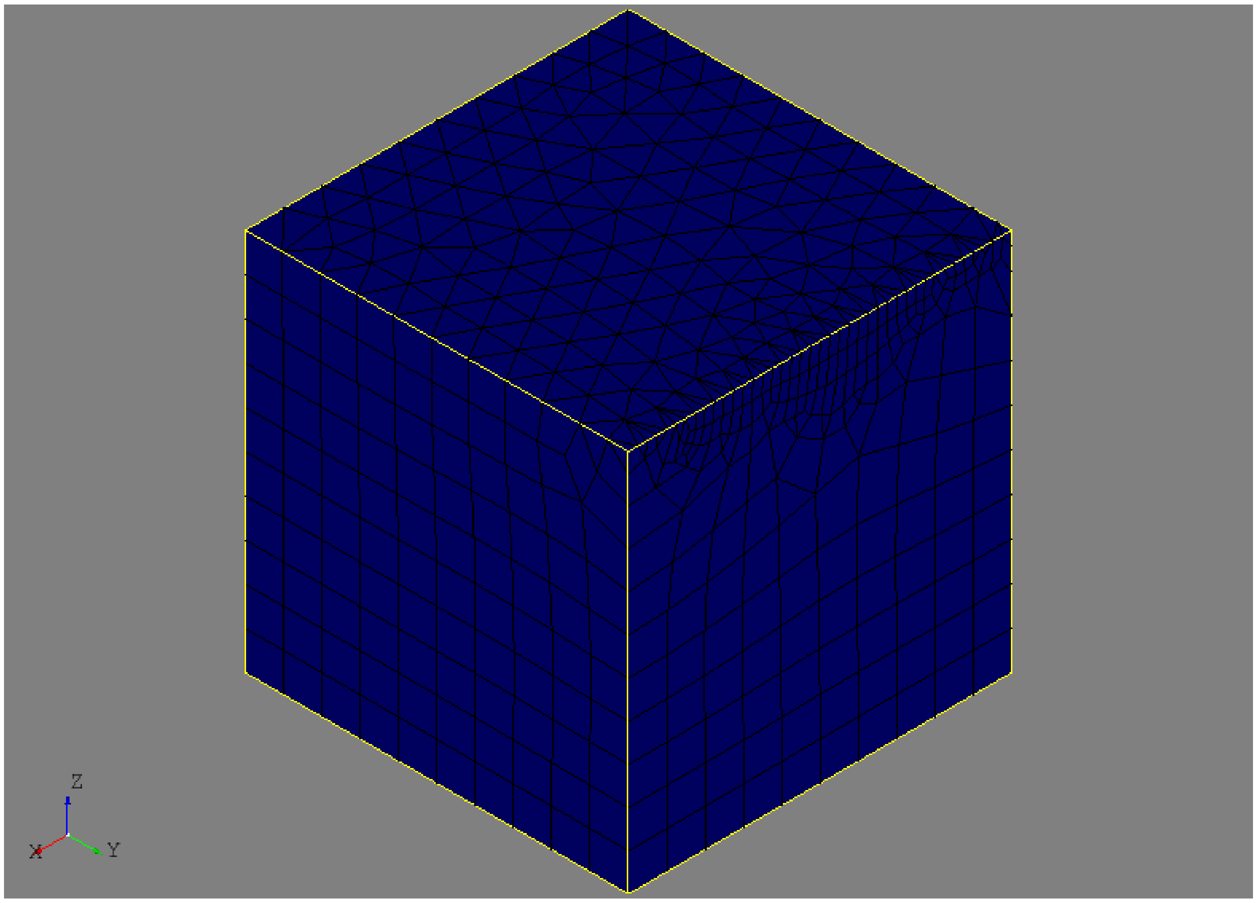
```
mesh.add_hypotheses([alg1d, hyp1d], edges[-1])
```

It should be noted that the provided shape (or sub-shape) can be a single shape like an edge, face, or solid, but it could also be a compound of multiple shapes or sub-shapes. That is, edges could be put into a *`Compound`* (page 119) and the 1-D controls would be applied to all edges in that compound.

With the shape set and meshing controls applied, the mesh is computed by:

```
gen.compute(mesh)
```

The next part of the script simply displays the mesh and sub-meshes derived from sub-shapes of the main shape. These sub-meshes can be used to access nodes and elements.



Towards the end of the script a small demonstration of the *MeshGroup* (page 202) class is provided:

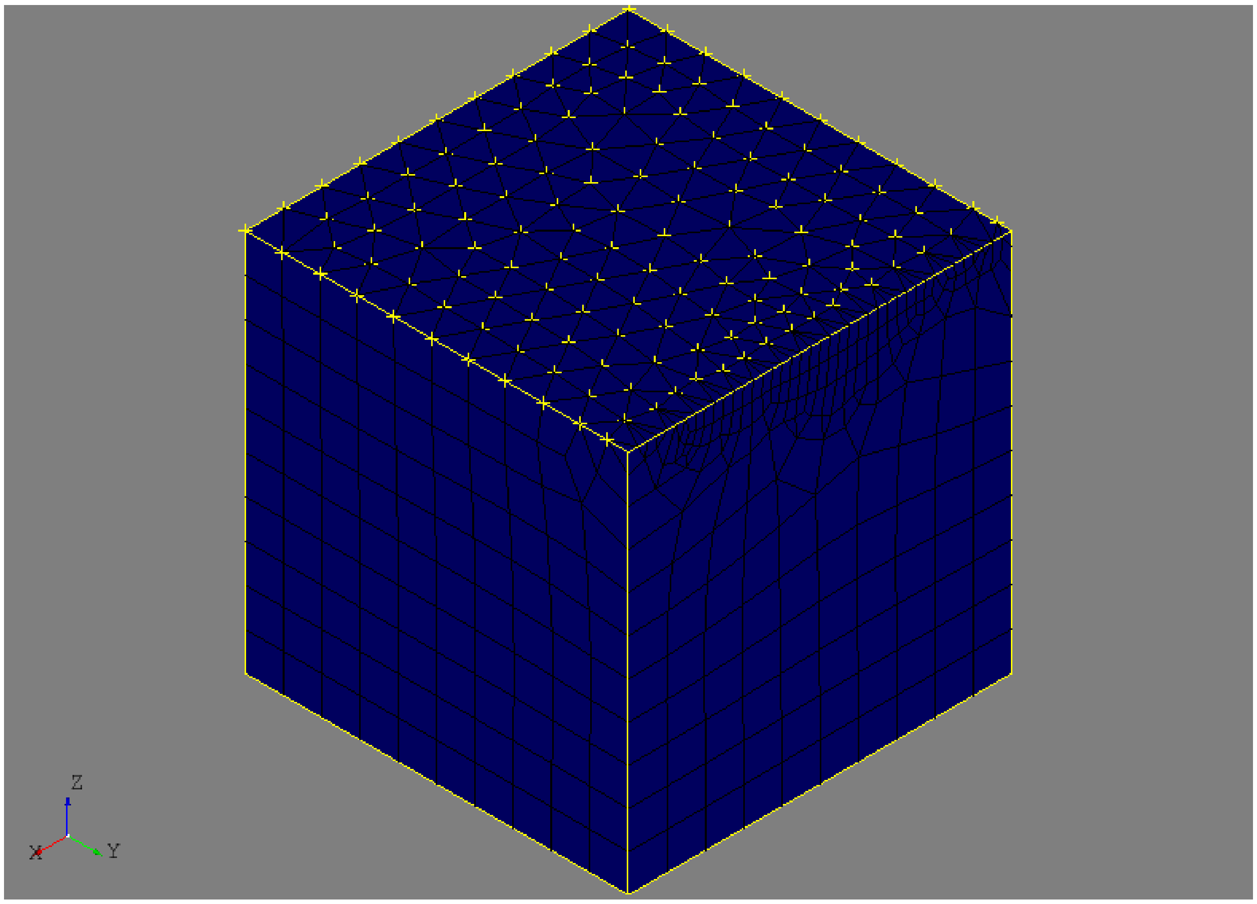
```
# Use groups to organize mesh data
edge_nodes = mesh.create_group('edge nodes', mesh.NODE, edges[-1])
face_nodes = mesh.create_group('face nodes', mesh.NODE, faces[-1])
```

Mesh groups are an excellent mechanism for organizing and accessing mesh data. The code above creates two groups of nodes on an edge and a face of the box. Other types of groups can be created for 1-D edge elements, as well as 2-D face elements including triangles and quadrangles. These groups make accessing the mesh data from the given shape much easier, flexible, and robust then using sub-meshes (this is not the intended purpose of sub-meshes anyway). Mesh groups can actually be created before the mesh is computed.

For mesh groups of similar type, additional organizational methods are provided including union, intersect, and subtracting the mesh data between two groups. In the example below, the nodes from the edge group are subtracted from the nodes in the face group:

```
group = face_nodes.subtract(edge_nodes)
```

A new group is created (without association to any shape) and is shown below. The same operations will be available for any two groups of similar type.



11.1 Entities

11.1.1 Node

class afem.smesh.entities.**Node**(*the_node*)

Bases: `object` (page 187)

Mesh node.

Parameters **the_node** (*OCCT.SMDS.SMDS_MeshNode*) – The SMDS_MeshNode object.

object

Returns The underlying node.

Return type OCCT.SMDS.SMDS_MeshNode

id

Returns The node ID.

Return type int

x

Returns The node x-location.

Return type float

y

Returns The node y-location.

Return type float

z

Returns The node z-location.

Return type float

xyz

Returns The node xyz-location.

Return type numpy.ndarray

11.1.2 Element

class afem.smesh.entities.**Element**(*the_element*)

Bases: *object* (page 188)

Generic base class for elements.

Parameters **the_element** (*OCCT.SMDS.SMDS_MeshElement*) – The element.

object

Returns The underlying element.

Return type OCCT.SMDS.SMDS_MeshElement

id

Returns The element ID.

Return type int

is_quadratic

Returns *True* if element is quadratic, *False* if not.

Return type bool

num_nodes

Returns Number of nodes.

Return type int

num_edges

Returns Number of edges.

Return type int

num_faces

Returns Number of faces.

Return type int

num_corner_nodes

Returns Number of corner nodes.

Return type int

node_iter

Returns Yield nodes of the element. The corner nodes will be first followed by medium nodes if quadratic.

Return type collections.Iterable(*afem.smesh.entities.Node* (page 187))

point_iter

Returns Yield nodes of the element as points. The corner points will be first followed by medium points if quadratic.

Return type collections.Iterable(*afem.geometry.entities.Point* (page 35))

edge_iter

Returns Yield edges of the element.

Return type collections.Iterable(*afem.smesh.entities.Element* (page 188))

nids

Returns The node ID's of the element.

Return type list(int)

is_0d

Returns *True* if the element has one node, *False* if not.

Return type bool

is_1d

Returns *True* if the element has two nodes, *False* if not.

Return type bool

is_2d

Returns *True* if the element has more than two nodes, *False* if not.

Return type bool

is_tri

Returns *True* if the element has three nodes, *False* if not.

Return type bool

is_quad

Returns *True* if the element has four nodes, *False* if not.

Return type bool

length

Returns Element length.

Return type float

area

Returns Element area.

Return type float

min_angle

Returns The minimum element angle in degrees.

Return type float

max_angle

Returns The maximum element angle in degrees.

Return type float

aspect_ratio

Returns The element aspect ratio.

Return type float

warp_angle

Returns The element warping in degrees.

Return type float

taper_ratio

Returns The element taper ratio.

Return type float

skew_angle

Returns The element skew angle.

Return type float

jacobian

Returns The element Jacobian.

Return type float

is_medium_node(*node*)

Check to see if the node is a medium node.

Parameters **node** (`afem.smesh.entities.Node` (page 187)) – A node.

Returns *True* if medium node, *False* if not.

Return type bool

node_index(*node*)

Get the node index if it belongs to the element.

Parameters **node** (`afem.smesh.entities.Node` (page 187)) – A node.

Returns Node index.

Return type int

wrapped_index(*indx*)

Get a valid node index if given one needs fixed.

Parameters **indx** (*`int`*) – A node index.

Returns The wrapped node index.

Return type int

get_node(*indx*)

Get the node at a given index.

Parameters **indx** (*`int`*) – The node index.

Returns The node.

Return type *afem.smesh.entities.Node* (page 187)

get_node_wrap(*indx*)
Get the node at a given index, fixing it if necessary.

Parameters **indx** (*int*) – The node index.

Returns The node.

Return type *afem.smesh.entities.Node* (page 187)

11.1.3 FaceSide

class afem.smesh.entities.**FaceSide**(*the_side*)
Bases: *object* (page 191)

Entity the represents the side of a quasi-quadrilateral face. It can be composed of several edges and gives access to geometry and 1-D mesh.

Parameters **the_side** (*OCCT.StdMeshers.StdMeshers_FaceSide*) – The StdMeshers_FaceSide instance.

object

Returns The underlying object.

Return type OCCT.StdMeshers.StdMeshers_FaceSide

num_edges

Returns The number of edges.

Return type int

num_nodes

Returns The number of nodes.

Return type int

num_segments

Returns The number of mesh segments.

Return type int

missed_vertices

Returns *True* if there are vertices without nodes.

Return type bool

ordered_nodes

Returns List of ordered nodes along the side.

Return type list(*afem.smesh.entities.Node* (page 187))

is_closed

Returns *True* if chain of edges is closed.

Return type bool

length

Returns Length of side.

Return type float

edges

Returns List of side edges.

Return type list(*afem.topology.entities.Edge* (page 115))

first_vertex

Returns First vertex of side.

Return type *afem.topology.entities.Vertex* (page 115)

last_vertex

Returns Last vertex of side.

Return type *afem.topology.entities.Vertex* (page 115)

vertex_node(*indx*)

Get the node from a vertex.

Parameters **indx** (*int*) – The vertex index (starts with 0).

Returns The vertex node.

Return type *afem.smesh.entities.Node* (page 187)

11.1.4 MeshGen

class afem.smesh.entities.**MeshGen**

Bases: *object* (page 192)

This class is the primary meshing database for a given instance.

object

Returns The underlying mesh object.

Return type OCCT.SMESH.SMESH_Gen

classmethod wrap(*gen*)

Create a new instance using an existing SMESH_Gen instance.

Parameters **gen** (*OCCT.SMESH.SMESH_Gen*) – A SMESH_Gen instance.

Returns The new instance.

Return type *afem.smesh.entities.MeshGen* (page 192)

new_id()

Generate a new unique ID within this generator.

Returns A new unique ID.

Return type int

create_mesh(*shape=None*, *is_embedded=False*)

Create a mesh.

Parameters

- **shape** (afem.topology.entities.Shape (page 109)) – The shape to mesh.
- **is_embedded** (*bool*) – Option for embedding mesh.

Returns The mesh.

Return type *afem.smesh.entities.Mesh* (page 193)

check_algo_state (*mesh*, *shape*)
Check if computation would fail because of some bad algorithm state.

Parameters

- **mesh** (`afem.smesh.entities.Mesh` (page 193)) – A mesh.
- **shape** (`afem.topology.entities.Shape` (page 109)) – A shape.

Returns *True* if ok, *False* if not.

Return type bool

compute (*mesh*, *shape=None*)
Compute a mesh on a shape.

Parameters

- **mesh** (`afem.smesh.entities.Mesh` (page 193)) – A mesh.
- **shape** (`afem.topology.entities.Shape` (page 109)) – The shape to compute mesh on. If not provided then the shape associated to the mesh is used.

Returns *True* if computed, *False* if not.

Return type bool

Raises **ValueError** – If no shape is available to apply the hypothesis to.

11.1.5 Mesh

class afem.smesh.entities.**Mesh** (*gen*, *is_embedded=False*)
Bases: *object* (page 193)

Mesh.

Parameters

- **gen** (`afem.smesh.entities.MeshGen` (page 192)) – The MeshGen instance.
- **is_embedded** (*bool*) – Option for embedding mesh.

Variables

- **ALL** (*OCCT.SMDSAbs.SMDSAbs_ElementType*) – SMESH all elements type.
- **NODE** (*OCCT.SMDSAbs.SMDSAbs_ElementType*) – SMESH node type.
- **EDGE** (*OCCT.SMDSAbs.SMDSAbs_ElementType*) – SMESH edge type.
- **VOLUME** (*OCCT.SMDSAbs.SMDSAbs_ElementType*) – SMESH volume type.

object

Returns The underlying mesh object.

Return type OCCT.SMESH.SMESH_Mesh

id

Returns The mesh ID.

Return type int

shape

Returns The shape to mesh.

Return type *afem.topology.entities.Shape* (page 109)

has_shape

Returns `True` if the mesh has a shape, `False` if not.

Return type bool

num_nodes

Returns Number of nodes.

Return type int

num_edges

Returns Number of edges.

Return type int

num_faces

Returns Number of faces.

Return type int

num_tris

Returns Number of triangles.

Return type int

num_quads

Returns Number of quadrangles.

Return type int

num_volumes

Returns Number of volume elements.

Return type int

num_submesh

Returns Number of sub-meshes.

Return type int

num_group

Returns Number of groups.

Return type int

ds

Returns The mesh data structure.

Return type *afem.smesh.entities.MeshDS* (page 196)

shape_to_mesh(*shape*)

Set the shape to mesh.

Parameters **shape** (`afem.topology.entities.Shape` (page 109)) – The shape.

Returns None

add_hypothesis(*hypothesis*, *shape=None*)
Add a hypothesis to the shape.

Parameters

- **hypothesis** (afem.smesh.hypotheses.Hypothesis (page 204)) – The hypothesis.
- **shape** (afem.topology.entities.Shape (page 109)) – The shape the hypothesis applies to. This can be a sub-shape of the master shape. If not provided then the master shape is used.

Returns Status of adding hypothesis.

Return type OCCT.SMESH.SMESH_Hypothesis.Hypothesis_Status

Raises **ValueError** – If no shape is available to apply the hypothesis to.

add_hypotheses(*hypotheses*, *shape=None*)
Add a hypotheses to the shape.

Parameters

- **hypotheses** (*collections.Sequence*(afem.smesh.hypotheses.Hypothesis (page 204)*)*) – The hypotheses to add.
- **shape** (afem.topology.entities.Shape (page 109)) – The shape the hypothesis applies to. This can be a sub-shape of the master shape. If not provided then the master shape is used.

Returns Dictionary where the key is the hypothesis and the value is the hypothesis status.

Return type dict

Raises **ValueError** – If no shape is available to apply the hypothesis to.

clear()
Clear all nodes and elements.

Returns None.

clear_submesh(*shape*)
Clear the nodes and elements from the shape.

Parameters **shape** (afem.topology.entities.Shape (page 109)) – The shape.

Returns None.

get_submesh(*sub_shape*)
Get a sub-mesh for the sub-shape.

Parameters **sub_shape** (afem.topology.entities.Shape (page 109)) – A sub-shape.

Returns A sub-mesh.

Return type *afem.smesh.entities.SubMesh* (page 199)

get_submesh_containing(*sub_shape*)
Get a sub-mesh for containing the sub-shape.

Parameters **sub_shape** (afem.topology.entities.Shape (page 109)) – A sub-shape.

Returns A sub-mesh.

Return type *afem.smesh.entities.SubMesh* (page 199)

create_group(*name*, *type_*, *shape=None*)
Create a group belonging to the mesh.

Parameters

- **name** (*str*) – The name of the group.
- **type** (*OCCT.SMDSAbs.SMDSAbs_ElementType*) – The element type for the group.
- **shape** (`afem.topology.entities.Shape` (page 109)) – The shape to create the group on. If *None* provided, then the group is not on geometry.

Returns The group.

Return type *afem.smesh.entities.MeshGroup* (page 202)

export_dat(*fn*)
Export the mesh to a DAT file.

Parameters **fn** (*str*) – The output file.

Returns None

export_stl(*fn*, *is_ascii=True*)
Export the mesh to an STL file.

Parameters

- **fn** (*str*) – The output file.
- **is_ascii** (*bool*) – ASCII text output or binary.

Returns None.

export_unv(*fn*)
Export the mesh to a UNV file.

Parameters **fn** (*str*) – The output file.

Returns None

import_unv(*fn*)
Import a mesh from a UNV file.

Parameters **fn** (*str*) – The input file.

Returns None.

classmethod wrap(*mesh*)
Create a new instance using an existing SMESH_Mesh instance.

Parameters **mesh** (*OCCT.SMESH.SMESH_Mesh*) – A SMESH_Mesh instance.

Returns The new instance.

Return type *afem.smesh.entities.Mesh* (page 193)

11.1.6 MeshDS

class afem.smesh.entities.**MeshDS**(*mesh*)
Bases: *object* (page 197)

Mesh data structure.

Parameters **mesh** (`afem.smesh.entities.Mesh` (page 193)) – A mesh.

object

Returns The underlying mesh object.

Return type OCCT.SMESH.SMESHDS_Mesh

is_embedded_mode

Returns *True* if embedded, *False* if not.

Return type bool

id

Returns The mesh ID.

Return type int

min_node_id

Returns The minimum node id.

Return type int

max_node_id

Returns The maximum node id.

Return type int

min_elm_id

Returns The minimum element id.

Return type int

max_elm_id

Returns The maximum element id.

Return type int

num_nodes

Returns Number of nodes.

Return type int

num_elms

Returns Number of elements.

Return type int

num_edges

Returns Number of edges.

Return type int

num_faces

Returns Number of faces.

Return type int

node_iter

Returns Yield nodes of the mesh.

Return type collections.Iterable(*afem.smesh.entities.Node* (page 187))

edge_iter

Returns Yield edges of the mesh.

Return type collections.Iterable(*afem.smesh.entities.Element* (page 188))

faces_iter

Returns Yield faces of the mesh.

Return type collections.Iterable(*afem.smesh.entities.Element* (page 188))

move_node (*node*, *x*, *y*, *z*)
Move node to given location.

Parameters

- **node** (*afem.smesh.nodes.Node*) – The node.
- **x** (*float*) – The x-location.
- **y** (*float*) – The y-location.
- **z** (*float*) – The z-location.

Returns None.

renumber_nodes (*start=1*, *step=1*)
Renumber nodes.

Parameters

- **start** (*int*) – Starting node id.
- **step** (*int*) – Step between id's.

Returns None.

renumber_elements (*start=1*, *step=1*)
Renumber elements.

Parameters

- **start** (*int*) – Starting element id.
- **step** (*int*) – Step between id's.

Returns None.

has_elements (*shape*)
Check to see if the shape has mesh elements.

Parameters **shape** (afem.topology.entities.Shape (page 109)) – The shape.

Returns *True* if the shape has elements, *False* if not.

Return type bool

mesh_elements (*shape*)
Get elements of shape.

Parameters **shape** (afem.topology.entities.Shape (page 109)) – The shape.

Returns Sub-mesh data structure of elements.

Return type *afem.smesh.entities.SubMeshDS* (page 200)

shape_to_index (*shape*)
Get the shape index.

Parameters **shape** (`afem.topology.entities.Shape` (page 109)) – The shape.

Returns The shape index.

Return type int

index_to_shape (*indx*)

Get the shape from an index.

Parameters **indx** (*int*) – The index.

Returns The shape.

Return type *afem.topology.entities.Shape* (page 109)

classmethod wrap (*mesh*)

Create a new instance using an existing SMESHDS_Mesh instance.

Parameters **mesh** (*OCCT.SMESHDS.SMESHDS_Mesh*) – A SMESHDS_Mesh instance.

Returns The new instance.

Return type *afem.smesh.entities.MeshDS* (page 196)

11.1.7 SubMesh

class afem.smesh.entities.**SubMesh** (*gen*, *mesh*, *mesh_ds*, *sub_shape*)

Bases: *object* (page 199)

Sub-mesh.

Parameters

- **gen** (`afem.smesh.entities.MeshGen` (page 192)) – A generator.
- **mesh** (`afem.smesh.entities.Mesh` (page 193)) – A mesh.
- **mesh_ds** (`afem.smesh.entities.MeshDS` (page 196)) – A mesh data structure.
- **sub_shape** (`afem.topology.entities.Shape` (page 109)) – The sub-shape.

object

Returns The underlying sub-mesh object.

Return type OCCT.SMESH.SMESH_subMesh

id

Returns The mesh ID.

Return type int

shape

Returns The sub-shape.

Return type *afem.topology.entities.Shape* (page 109)

is_empty

Returns *True* if the mesh is empty, *False* if not.

Return type bool

is_computed

Returns *True* if the mesh is computed, *False* if not.

Return type bool

ds

Returns The sub-mesh data structure.

Return type *afem.smesh.entities.SubMeshDS* (page 200)

classmethod wrap(*sub_mesh*)
Create a new instance using an existing SMESH_subMesh instance.

Parameters **sub_mesh** (*OCCT.SMESH.SMESH_subMesh*) – A SMESH_subMesh instance.

Returns The new instance.

Return type *afem.smesh.entities.SubMesh* (page 199)

can_add_hypothesis(*hyp*)
Check to see if the hypothesis can be attached.

Parameters **hyp** (`afem.smesh.hypotheses.Hypothesis` (page 204)) – The hypothesis.

Returns *True* if can be attached, *False* if not.

Return type bool

is_applicable_hypothesis(*hyp*)
Check to see if the hypothesis can be used to mesh.

Parameters **hyp** (`afem.smesh.hypotheses.Hypothesis` (page 204)) – The hypothesis.

Returns *True* if applicable, *False* if not.

Return type bool

11.1.8 SubMeshDS

class afem.smesh.entities.**SubMeshDS**(*sub_mesh*)
Bases: *object* (page 200)

Sub-mesh data structure.

Parameters **sub_mesh** (`afem.smesh.entities.SubMesh` (page 199)) – A sub-mesh.

object

Returns The underlying mesh object.

Return type OCCT.SMESH.SMESHDS_subMesh

id

Returns The mesh ID.

Return type int

is_complex

Returns *True* if sub-mesh is complex, *False* if not.

Return type bool

is_quadratic

Returns *True* if sub-mesh is quadratic, *False* if not.

Return type bool

num_nodes

Returns Number of nodes.

Return type int

num_elms

Returns Number of elements.

Return type int

node_iter

Returns Yield nodes of the sub-mesh.

Return type collections.Iterable(*afem.smesh.entities.Node* (page 187))

elm_iter

Returns Yield elements of the sub-mesh.

Return type collections.Iterable(*afem.smesh.entities.Element* (page 188))

classmethod wrap(*sub_meshds*)

Create a new instance using an existing SMESHDS_SubMesh instance.

Parameters **sub_meshds** (*OCCT.SMESHDS.SMESHDS_SubMesh*) – A SMESHDS_SubMesh instance.

Returns The new instance.

Return type *afem.smesh.entities.SubMeshDS* (page 200)

get_node(*id_*)

Get a node.

Parameters **id** (*int*) – The id of the node in the shape.

Returns The node.

Return type *afem.smesh.entities.Node* (page 187)

get_element(*id_*)

Get an element.

Parameters **id** (*int*) – The id of the element in the shape.

Returns The element.

Return type *afem.smesh.entities.Element* (page 188)

contains(*elm*)

Check to see if node or element is in the sub-mesh.

Parameters **elm** (`afem.smesh.entities.Node` (page 187) *or* `afem.smesh.entities.Element` (page 188)) – The node or element.

Returns *True* if present, *False* if not.

Return type bool

clear()

Clear all nodes and elements.

Returns None.

11.1.9 MeshGroup

class `afem.smesh.entities.`**`MeshGroup`**(*mesh*, *name*, *type_*, *shape=None*)

Bases: *`object`* (page 202)

Mesh group for nodes and elements.

Parameters

- **`mesh`** (`afem.smesh.entities.Mesh` (page 193)) – The mesh the group belongs to.
- **`name`** (*`str`*) – The name of the group.
- **`type`** (*`OCCT.SMDSAbs.SMDSAbs_ElementType`*) – The element type for the group.
- **`shape`** (`afem.topology.entities.Shape` (page 109)) – The shape to create the group on. If *None* provided, then the group is not on geometry.

`object`

Returns The underlying SMESH_Group object.

Return type OCCT.SMESH.SMESH_Group

`id`

Returns The group ID.

Return type int

`type`

Returns The group type.

Return type OCCT.SMDSAbs.SMDSAbs_ElementType

`mesh`

Returns The mesh the group belongs to.

Return type *afem.smesh.entities.Mesh* (page 193)

`name`

Getter The group name.

Setter Set the group name.

Type str

`shape`

Getter The shape of the group.

Setter Set the group shape.

Type *afem.topology.entities.Shape* (page 109)

Raises **`AttributeError`** – If the group is not associated with a shape.

`is_empty`

Returns Check if group is empty.

Return type bool

`size`

Returns The number of entities in the group.

Return type int

node_iter

Returns Yield the nodes in the group.

Return type collections.Iterable(*afem.smesh.entities.Node* (page 187))

Raises **TypeError** – If group is not a node group.

edge_iter

Returns Yield the edges in the group.

Return type collections.Iterable(*afem.smesh.entities.Element* (page 188))

Raises **TypeError** – If group is not an edge group.

face_iter

Returns Yield the faces in the group.

Return type collections.Iterable(*afem.smesh.entities.Element* (page 188))

Raises **TypeError** – If group is not a face group.

set_name(*name*)

Set the group name.

Parameters **name** (*str*) – The group name.

Returns None.

set_shape(*shape*)

Set the shape of the group.

Parameters **shape** (afem.topology.entities.Shape (page 109)) – The shape.

Returns None.

contains_id(*eid*)

Check if group contains the element by using its ID.

Parameters **eid** (*int*) – The element ID.

Returns *True* if in the group, *False* otherwise.

Return type bool

contains_node(*node*)

Check if group contains the node.

Parameters **node** (afem.smesh.entities.Node (page 187)) – The node.

Returns *True* if in the group, *False* otherwise.

Return type bool

Raises **TypeError** – If the group is not a node group.

contains_elm(*elm*)

Check if group contains the element.

Parameters **elm** (afem.smesh.entities.Element (page 188)) – The element.

Returns *True* if in the group, *False* otherwise.

Return type bool

Raises **TypeError** – If the group is not an element group.

union (*other*, *name='union group'*)
Union the entities of this group with another.

Parameters

- **other** (`afem.smesh.entities.MeshGroup` (page 202)) – The other group.
- **name** (*str*) – The name of the new group.

Returns New group.

Return type *afem.smesh.entities.MeshGroup* (page 202)

Raises **TypeError** – If the two groups are of different type or they do not share the same mesh.

intersect (*other*, *name='intersect group'*)
Intersect the entities between this group and another.

Parameters

- **other** (`afem.smesh.entities.MeshGroup` (page 202)) – The other group.
- **name** (*str*) – The name of the new group.

Returns New group.

Return type *afem.smesh.entities.MeshGroup* (page 202)

Raises **TypeError** – If the two groups are of different type or they do not share the same mesh.

subtract (*other*, *name='Subtract group'*)
Subtract the entities from this group by another.

Parameters

- **other** (`afem.smesh.entities.MeshGroup` (page 202)) – The other group.
- **name** (*str*) – The name of the new group.

Returns New group.

Return type *afem.smesh.entities.MeshGroup* (page 202)

Raises **TypeError** – If the two groups are of different type or they do not share the same mesh.

11.2 Hypotheses

11.2.1 Hypothesis

class afem.smesh.hypotheses.**Hypothesis** (*hyp*)
Bases: *object* (page 204)

Base class for all hypotheses.

Parameters **hyp** (*OCCT.SMESH.SMESH_Hypothesis*) – The SMESH hypothesis.

object

Returns The underlying hypothesis.

Return type OCCT.SMESH.SMESH_Hypothesis

id

Returns The hypothesis ID.

Return type int

name

Returns The hypothesis name.

Return type str

dim

Returns Hypothesis dimension.

Return type int

11.2.2 Algorithm

class `afem.smesh.hypotheses.`**`Algorithm`**(*hyp*)

Bases: *`afem.smesh.hypotheses.Hypothesis`* (page 204)

Base class for algorithms.

static edge_length(*e*)

Compute length of an edge.

Parameters **e** (`afem.topology.entities.Edge` (page 115)) – The edge.

Returns Edge length.

Return type float

static continuity(*e1*, *e2*)

Calculate the continuity of the two edges.

Parameters

- **e1** (`afem.topology.entities.Vertex` (page 115)) – The first edge.
- **e2** (`afem.topology.entities.Vertex` (page 115)) – The second edge.

Returns The continuity.

Return type OCCT.GeomAbs.GeomAbs_Shape

static is_continuous(*e1*, *e2*)

Check if the two edges can be considered continuous.

Parameters

- **e1** (`afem.topology.entities.Edge` (page 115)) – The first edge.
- **e2** (`afem.topology.entities.Edge` (page 115)) – The second edge.

Returns *True* if continuous, *False* otherwise.

Return type bool

static is_straight(*e*)

Check if the edge can be considered straight.

Parameters **e** (`afem.topology.entities.Edge` (page 115)) – The edge.

Returns *True* if straight, *False* otherwise.

Return type bool

static is_degenerated(*e*)

Check if the edge has no 3-D curve.

Parameters **e** (`afem.topology.entities.Edge` (page 115)) – The edge.

Returns *True* if no 3-D curve, *False* otherwise.

Return type bool

static vertex_node (*v*, *mesh*)
Get a node built on a vertex.

Parameters

- **v** (`afem.topology.entities.Vertex` (page 115)) – The vertex.
- **mesh** (`afem.smesh.entities.Mesh` (page 193) *or* `afem.smesh.entities.MeshDS` (page 196)) – The mesh.

Returns The node.

Return type *afem.smesh.entities.Node* (page 187)

compatible_hypotheses

Returns A list of OCCT hypotheses that are compatible with this algorithm.

Return type list(str)

compute_error

Returns The compute error.

Return type OCCT.SMESH.SMESH_ComputeError

compute_error_name

Returns The name of the compute error.

Return type str

check_hypothesis (*mesh*, *shape*, *status=Hypothesis_Status.HYP_OK*)
Check the hypothesis in the given mesh and shape.

Parameters

- **mesh** (`afem.smesh.entities.Mesh` (page 193)) – The mesh.
- **shape** (`afem.topology.entities.Shape` (page 109)) – The shape.
- **status** (*OCCT.SMESH.SMESH_Hypothesis.Hypothesis_Status*) – The status to check.

Returns *True* if check matches status, *False* otherwise.

Return type bool

compute (*mesh*, *shape*)
Compute the mesh on a shape.

Parameters

- **mesh** (`afem.smesh.entities.Mesh` (page 193)) – The mesh.
- **shape** (`afem.topology.entities.Shape` (page 109)) – The shape.

Returns *True* if completed, *False* if not.

Return type bool

11.2.3 Regular1D

class afem.smesh.hypotheses.**Regular1D**(*gen*)

Bases: *afem.smesh.hypotheses.Algorithm* (page 205)

Regular 1-D algorithm. Use this with a hypothesis to mesh edges.

Parameters **gen** (afem.smesh.entities.MeshGen (page 192)) – A mesh generator.

11.2.4 CompositeSide1D

class afem.smesh.hypotheses.**CompositeSide1D**(*gen*)

Bases: *afem.smesh.hypotheses.Algorithm* (page 205)

Composite side 1-D discretization. This allows a 1-D hypothesis to be applied to a whole side of a geometrical face even if it is composed of several edges provided that they form C1 curve in all faces of the main shape.

Parameters **gen** (afem.smesh.entities.MeshGen (page 192)) – A mesh generator.

static get_face_side(*mesh, e, f, ignore_meshed=False*)

Return a face side the edge belongs to.

Parameters

- **mesh** (afem.smesh.entities.Mesh (page 193)) – The mesh.
- **e** (afem.topology.entities.Edge (page 115)) – The edge.
- **f** (afem.topology.entities.Face (page 117)) – The face.
- **ignore_meshed** (*bool*) – Unclear what option does.

Returns The face side.

Return type *afem.smesh.entities.FaceSide* (page 191)

11.2.5 MaxLength1D

class afem.smesh.hypotheses.**MaxLength1D**(*gen, max_length*)

Bases: *afem.smesh.hypotheses.Hypothesis* (page 204)

Maximum length 1-D hypothesis.

Parameters

- **gen** (afem.smesh.entities.MeshGen (page 192)) – A mesh generator.
- **max_length** (*float*) – Maximum edge length.

max_length

Returns The maximum edge length.

Return type float

11.2.6 LocalLength1D

class afem.smesh.hypotheses.**LocalLength1D**(*gen, local_length, precision=1e-07*)

Bases: *afem.smesh.hypotheses.Hypothesis* (page 204)

Local length 1-D hypothesis.

Parameters

- **gen** (`afem.smesh.entities.MeshGen` (page 192)) – A mesh generator.
- **local_length** (*float*) – The desired edge length.
- **precision** (*float*) – The value used to control the rounding of the number of segments. Use 0.5 to provide rounding to nearest integer, 1.0 for lower integer, 0.0 for higher integer.

local_length

Returns The local edge length.

Return type float

precision

Returns The precision.

Return type float

11.2.7 NumberOfSegments1D

class afem.smesh.hypotheses.**NumberOfSegments1D**(*gen*, *nseg*)

Bases: *afem.smesh.hypotheses.Hypothesis* (page 204)

Number of segments 1-D hypothesis.

Parameters

- **gen** (`afem.smesh.entities.MeshGen` (page 192)) – A mesh generator.
- **nseg** (*int*) – The number of segments.

nseg

Returns The number of segments.

Return type int

11.2.8 Adaptive1D

class afem.smesh.hypotheses.**Adaptive1D**(*gen*, *min_size*, *max_size*, *deflection*)

Bases: *afem.smesh.hypotheses.Hypothesis* (page 204)

Adaptive length 1-D hypothesis.

Parameters

- **gen** (`afem.smesh.entities.MeshGen` (page 192)) – A mesh generator.
- **min_size** (*float*) – Minimum edge size.
- **max_size** (*float*) – Maximum edge size.
- **deflection** (*float*) – Maximum distance from a segment to a curved edge.

11.2.9 Deflection1D

class afem.smesh.hypotheses.**Deflection1D**(*gen*, *deflection*)

Bases: *afem.smesh.hypotheses.Hypothesis* (page 204)

Deflection length 1-D hypothesis.

Parameters

- **gen** (afem.smesh.entities.MeshGen (page 192)) – A mesh generator.
- **deflection** (*float*) – Maximum distance from a segment to a curved edge.

11.2.10 QuadrangleAlgo2D

class afem.smesh.hypotheses.**QuadrangleAlgo2D**(*gen*)

Bases: *afem.smesh.hypotheses.Algorithm* (page 205)

Quadrangle 2-D algorithm.

Parameters **gen** (afem.smesh.entities.MeshGen (page 192)) – A mesh generator.

static is_applicable(*shape*, *check_all=True*)

Check if this algorithm can mesh the shape.

Parameters

- **shape** (afem.topology.entities.Shape (page 109)) – The shape.
- **check_all** (*bool*) – If *True*, this check returns *True* if all shapes are applicable. If *False* this check returns *True* if at least one shape is ok.

Returns Check whether algorithm is applicable.

Return type bool

11.2.11 QuadrangleHypo2D

class afem.smesh.hypotheses.**QuadrangleHypo2D**(*gen*, *quad_type=StdMeshers_QuadType.QUAD_STANDARD*)

Bases: *afem.smesh.hypotheses.Hypothesis* (page 204)

Quadrangle 2-D parameters.

Parameters

- **gen** (afem.smesh.entities.MeshGen (page 192)) – A mesh generator.
- **quad_type** (*OCCT.StdMeshers.StdMeshers_QuadType*) – The quadrangle preference for transitions.

set_enforced_nodes(*shapes*, *pnts*)

Set enforced nodes on shapes.

Parameters

- **shapes** (*list(*afem.topology.entities.Shape (page 109)*)*) – List of shapes.
- **pnts** (*list(point_like)*) – List of points for enforced nodes.

Returns None.

11.2.12 NetgenHypothesis

class afem.smesh.hypotheses.**NetgenHypothesis**(*gen*, *max_size=1000.0*, *min_size=0.0*, *allow_quads=False*, *second_order=False*, *optimize=True*, *fineness=Fineness.Moderate*, *growth_rate=0.3*, *nseg_per_edge=1*, *nseg_per_radius=2*, *surface_curvature=False*, *fuse_edges=False*)

Bases: *afem.smesh.hypotheses.Hypothesis* (page 204)

NETGEN hypothesis.

Parameters

- **gen** (afem.smesh.entities.MeshGen (page 192)) – A mesh generator.
- **max_size** (*float*) – Maximum edge length.
- **min_size** (*float*) – Minimum edge length.
- **allow_quads** (*bool*) – Enable quad-dominated mesh.
- **second_order** (*bool*) – Enable second-order mesh.
- **optimize** (*bool*) – Enable mesh optimization.
- **fineness** (*OCCT.NETGENPlugin.NETGENPlugin_Hypothesis.Fineness*) – Mesh fineness.
- **growth_rate** (*float*) – Growth rate between 0 to 1.
- **nseg_per_edge** (*int*) – Number of segments per edge.
- **nseg_per_radius** (*int*) – Number of segments per radius.
- **surface_curvature** (*bool*) – Enable surface curvature to define edge size.
- **fuse_edges** (*bool*) – Option to fuse edges.

Variables

- **VERYCOARSE** (*OCCT.NETGENPlugin.NETGENPlugin_Hypothesis.Fineness*) –
- **COARSE** (*OCCT.NETGENPlugin.NETGENPlugin_Hypothesis.Fineness*) –
- **MODERATE** (*OCCT.NETGENPlugin.NETGENPlugin_Hypothesis.Fineness*) –
- **FINE** (*OCCT.NETGENPlugin.NETGENPlugin_Hypothesis.Fineness*) –
- **VERYFINE** (*OCCT.NETGENPlugin.NETGENPlugin_Hypothesis.Fineness*) –

11.2.13 NetgenAlgo2D

class afem.smesh.hypotheses.**NetgenAlgo2D**(*gen*)

Bases: *afem.smesh.hypotheses.Algorithm* (page 205)

NETGEN 2-D algorithm.

Parameters **gen** (afem.smesh.entities.MeshGen (page 192)) – A mesh generator.

11.2.14 NetgenAlgoOnly2D

class afem.smesh.hypotheses.**NetgenAlgoOnly2D**(*gen*)

Bases: *afem.smesh.hypotheses.Algorithm* (page 205)

NETGEN 2-D only algorithm. Takes into account pre-existing nodes on face boundaries.

Parameters **gen** (afem.smesh.entities.MeshGen (page 192)) – A mesh generator.

11.2.15 NetgenHypo2D

class afem.smesh.hypotheses.**NetgenHypo2D**(*gen*, *max_size=1000.0*, *min_size=0.0*, *allow_quads=False*, *second_order=False*, *optimize=True*, *fineness=Fineness.Moderate*, *growth_rate=0.3*, *nseg_per_edge=1*, *nseg_per_radius=2*, *surface_curvature=False*, *fuse_edges=False*)

Bases: *afem.smesh.hypotheses.Hypothesis* (page 204)

NETGEN 2-D hypothesis.

Parameters

- **gen** (afem.smesh.entities.MeshGen (page 192)) – A mesh generator.
- **max_size** (*float*) – Maximum edge length.
- **min_size** (*float*) – Minimum edge length.
- **allow_quads** (*bool*) – Enable quad-dominated mesh.
- **second_order** (*bool*) – Enable second-order mesh.
- **optimize** (*bool*) – Enable mesh optimization.
- **fineness** (*OCCT.NETGENPlugin.NETGENPlugin_Hypothesis.Fineness*) – Mesh fineness.
- **growth_rate** (*float*) – Growth rate between 0 to 1.
- **nseg_per_edge** (*int*) – Number of segments per edge.
- **nseg_per_radius** (*int*) – Number of segments per radius.
- **surface_curvature** (*bool*) – Enable surface curvature to define edge size.
- **fuse_edges** (*bool*) – Option to fuse edges.

11.2.16 NetgenSimple2D

class afem.smesh.hypotheses.**NetgenSimple2D**(*gen*, *local_length=None*, *nseg=None*, *allow_quads=True*, *length_from_edges=False*, *max_area=0.0*)

Bases: *afem.smesh.hypotheses.Hypothesis* (page 204)

NETGEN 2-D simple hypothesis.

Parameters

- **gen** (afem.smesh.entities.MeshGen (page 192)) – A mesh generator.
- **local_length** (*float*) – Local edge segment length.

- **nseg** (*int*) – Number of edge segments.
- **allow_quads** (*bool*) – Enable quad-dominated mesh.
- **length_from_edges** (*bool*) – Set maximum element area to be dependent on 1-D discretization.
- **max_area** (*float*) – Set maximum element area.

11.2.17 NetgenAlgo3D

class afem.smesh.hypotheses.**NetgenAlgo3D**(*gen*)

Bases: *afem.smesh.hypotheses.Algorithm* (page 205)

NETGEN 3-D algorithm.

Parameters **gen** (afem.smesh.entities.MeshGen (page 192)) – A mesh generator.

11.2.18 NetgenAlgo2D3D

class afem.smesh.hypotheses.**NetgenAlgo2D3D**(*gen*)

Bases: *afem.smesh.hypotheses.Algorithm* (page 205)

NETGEN 2-D/3-D algorithm.

Parameters **gen** (afem.smesh.entities.MeshGen (page 192)) – A mesh generator.

11.2.19 MeshGemsAlgo2D

class afem.smesh.hypotheses.**MeshGemsAlgo2D**(*gen*)

Bases: *afem.smesh.hypotheses.Algorithm* (page 205)

MeshGems MGCAD-Surf algorithm.

Parameters **gen** (afem.smesh.entities.MeshGen (page 192)) – A mesh generator.

Raises **NotImplementedError** – If MeshGems is not available.

11.2.20 MeshGemsHypo2D

class afem.smesh.hypotheses.**MeshGemsHypo2D**(*gen*, *size=None*, *allow_quads=True*)

Bases: *afem.smesh.hypotheses.Hypothesis* (page 204)

MeshGems MGCAD-Surf hypothesis.

Parameters

- **gen** (afem.smesh.entities.MeshGen (page 192)) – A mesh generator.
- **size** (*float*) – Desired global edge size.
- **allow_quads** (*bool*) – Enable quad-dominant mesh.

Raises **NotImplementedError** – If MeshGems is not available.

set_physical_size(*size*, *is_rel=False*)

Set physical size.

Parameters

- **size** (*float*) –
- **is_rel** (*bool*) –

Returns

set_min_size(*size*, *is_rel=False*)
Set minimum size.

Parameters

- **size** (*float*) –
- **is_rel** (*bool*) –

Returns

set_max_size(*size*, *is_rel=False*)
Set maximum size.

Parameters

- **size** (*float*) –
- **is_rel** (*bool*) –

Returns

set_use_gradation(*val=True*)

Parameters **val** (*bool*) –

Returns

set_gradation(*val*)

Parameters **val** (*float*) –

Returns

set_quads_allowed(*val*)

Parameters **val** (*bool*) –

Returns

set_angle_mesh(*val*)

Parameters **val** (*float*) –

Returns

set_chordal_error(*val*)

Parameters **val** (*float*) –

Returns

set_anisotropic(*val*)

Parameters **val** (*bool*) –

Returns

set_anisotropic_ratio(*val*)

Parameters **val** (*float*) –

Returns

set_remove_tiny_edges(*val*)

Parameters **val** (*bool*) –

Returns

set_tiny_edge_length(*val*)

Parameters **val** (*float*) –

Returns

set_optimize_tiny_edges(*val*)

Parameters **val** (*bool*) –

Returns

set_tiny_edge_optimization_length(*val*)

Parameters **val** (*float*) –

Returns

set_correct_surface_intersection(*val*)

Parameters **val** (*bool*) –

Returns

set_correct_surface_intersection_max_cost(*val*)

Parameters **val** (*float*) –

Returns

set_bad_element_removal(*val*)

Parameters **val** (*bool*) –

Returns

set_bad_element_aspect_ratio(*val*)

Parameters **val** (*float*) –

Returns

set_optimize_mesh(*val*)

Parameters **val** (*bool*) –

Returns

set_respect_geometry(*val*)

Parameters **val** (*bool*) –

Returns

set_max_number_of_threads(*val*)

Parameters **val** (*int*) –

Returns

set_debug(*val*)

Parameters **val** (*bool*) –

Returns

set_required_entities(*val*)

Parameters **`val`** (*`str`*) – "respect", "ignore", "clear"

Returns

`set_sewing_tolerance`(*val*)

Parameters **`val`** (*`float`*) –

Returns

`set_tags`(*val*)

Parameters **`val`** (*`str`*) – "respect", "ignore", "clear"

Returns

`add_option`(*name*, *val*)

Parameters

- **`name`** (*`str`*) –
- **`val`** (*`str`*) –

Returns

`add_hyperpatch`(*patch_ids*)

Parameters **`patch_ids`** (*`list(set(int))`*) –

Returns

11.3 Utilities

11.3.1 MeshEditor

class `afem.smesh.utils.`**`MeshEditor`**(*mesh*)

Bases: `object` (page 215)

Mesh editor.

Parameters **`mesh`** (`afem.smesh.entities.Mesh` (page 193)) – A mesh.

`object`

Returns The underlying mesh editor.

Return type OCCT.SMESH.SMESH_MeshEditor

`reorient`(*elm*)

Reverse the element orientation.

Parameters **`elm`** (`afem.smesh.entities.Element` (page 188)) – The element.

Returns *True* if reoriented, *False* if not.

Return type bool

`smooth`(*elms=()*, *fixed_nodes=()*, *method='laplacian'*, *iters=5*, *target_ar=1.0*, *in_2d=True*)

Smooth the mesh.

Parameters

- **`elms`** (*`collections.Sequence(`*`afem.smesh.entities.Element` (page 188)*`)`*) – The elements to smooth. If empty then the whole mesh is smoothed.

- **fixed_nodes** (*collections.Sequence(afem.smesh.entities.Node (page 187))*) – The fixed nodes if any. Nodes on edges and boundaries are always fixed.
- **method** (*str*) – Either 'laplacian', 'l', 'centroidal', or 'c'.
- **iters** (*int*) – Number of iterations.
- **target_ar** (*float*) – Target aspect ratio.
- **in_2d** (*bool*) – Perform smoothing in 2-d parameters of nodes on face.

Returns None.

find_coincident_nodes(*nodes=()*, *tol=1e-07*)
Find coincident nodes.

Parameters

- **nodes** (*collections.Sequence(afem.smesh.entities.Node (page 187))*) – The nodes to search. If not provided then the whole mesh will be searched.
- **tol** (*float*) – Search tolerance.

Returns Coincident nodes as a list of list of nodes. The length of the returned list will be the number of coincident nodes. Each row is a list of nodes that are coincident.

Return type list(list(*afem.smesh.entities.Node* (page 187)))

merge_nodes(*nodes=None*, *avoid_making_holes=False*, *tol=1e-07*)
Merge nodes.

Parameters

- **nodes** (*list(list(afem.smesh.entities.Node (page 187)))*) – The nodes to merge. Each row is a list of nodes where the first node is kept and the others are replaced with the first. If *None* is provided then the entire mesh is first searched.
- **avoid_making_holes** (*bool*) – Avoid modifications that may spoil mesh topology.
- **tol** (*float*) – Search tolerance.

Returns None.

find_equal_elements(*elements=()*)
Find equal elements.

Parameters elements (*collections.Sequence(afem.smesh.entities.Element (page 188))*) – The elements to search. If not provided then the whole mesh will be searched.

Returns Equal elements as a list of list of integers. The length of the returned list will be the number of equal elements. Each row is a list of element ID's that are equal.

Return type list(list(int))

merge_elements(*elements=None*)
Merge elements.

Parameters elements (*list(list(int))*) – The elements to merge. Each row is a list of element ID's where the first ID is kept and the others are replaced with the first. If *None* is provided then the entire mesh is first searched.

Returns None.

merge_equal_elements()
Merge elements built on same nodes.

Returns None.

find_shape(*elm*)
Find the shape index that the element or node is on.

Parameters elm (`afem.smesh.entities.Node` (page 187) *or* `afem.smesh.entities.Element` (page 188)) – The node or element.

Returns The shape index.

Return type int

is_medium(*node*)
Check if node is a medium.

Parameters node (`afem.smesh.entities.Node` (page 187)) – The node.

Returns *True* if medium, *False* if not.

Return type bool

double_elements(*elms*)
Double elements.

Parameters elms (*collections.Sequence(*`afem.smesh.entities.Element` (page 188)*)*) – The elements to double.

Returns None.

double_nodes(*nids*)
Double nodes.

Parameters nids (*collections.Sequence(int)*) – Sequence of node ID's.

Returns *True* if doubled, *False* if not.

Return type bool

transform(*trsf*, *elements=()*, *copy=False*, *make_groups=False*, *target_mesh=None*)
Transform the elements.

Parameters

- **trsf** (*OCCT.gp.gp_Trsf*) – The transformation.
- **elements** (*collection.Sequence(*`afem.smesh.entities.Element` (page 188)*)*) – The elements to transform. If none are provided then the whole mesh is used.
- **copy** (*bool*) – Option to copy elements.
- **make_groups** (*bool*) – Option to make groups.
- **target_mesh** (`afem.smesh.entities.Mesh` (page 193)) – The target mesh to place elements.

Returns List of element ID's.

Return type list(int)

translate(*vector*, *elements=()*, *copy=False*, *make_groups=False*, *target_mesh=None*)
Translate the elements.

Parameters

- **`vector`** (*`vector_like`*) – The translation vector.
- **`elements`** (*`collection.Sequence(`*`afem.smesh.entities.Element` (page 188)*`)`*) – The elements to transform. If none are provided then the whole mesh is used.
- **`copy`** (*`bool`*) – Option to copy elements.
- **`make_groups`** (*`bool`*) – Option to make groups.
- **`target_mesh`** (`afem.smesh.entities.Mesh` (page 193)) – The target mesh to place elements.

Returns List of element ID's.

Return type list(int)

`check_free_border`(*n1*, *n2*, *n3=None*)

Check to see if the nodes are on a free border.

Parameters

- **`n1`** (`afem.smesh.entities.Node` (page 187)) – The first node.
- **`n2`** (`afem.smesh.entities.Node` (page 187)) – The second node.
- **`n3`** (`afem.smesh.entities.Node` (page 187)) – The third node.

Returns *True* if on a free border, *False* if not.

Return type bool

`find_free_border`(*n1*, *n2*, *n3*)

Return nodes and faces of a free border if found.

Parameters

- **`n1`** (`afem.smesh.entities.Node` (page 187)) – The first node.
- **`n2`** (`afem.smesh.entities.Node` (page 187)) – The second node.
- **`n3`** (`afem.smesh.entities.Node` (page 187)) – The third node.

Returns List of nodes and elements of free borders.

Return type tuple(list(*afem.smesh.entities.Node* (page 187)), list(*afem.smesh.entities.Element* (page 188)))

`tri_to_quad`(*elms=()*, *method=0*, *max_bending_angle=30.0*)

Try to merge triangular elements into quadrangles.

Parameters

- **`elms`** (*`collections.Sequence(`*`afem.smesh.entities.Element` (page 188)*`)`*) – The elements to combine. If empty then the whole mesh is used.
- **`method`** (*`int`*) – The criteria used for determining which elements to combine (0=Aspect ratio, 1=Minimum angle, 2=Skew, 3=Area, 4=Warping, 5=Taper).
- **`max_bending_angle`** (*`float`*) – The maximum bending angle.

Returns *True* if some elements were merged, *False* if not.

Return type bool

`convert_to_quadratic`()

Convert to quadratic mesh by inserting mid-sided nodes along the edge between existing nodes.

Returns *True* if successful, *False* if not.

Return type bool

11.3.2 MeshHelper

class afem.smesh.utils.**MeshHelper**(*mesh*)

Bases: `object` (page 219)

Mesh helper.

Parameters **mesh** (`afem.smesh.entities.Mesh` (page 193)) – A mesh.

object

Returns The underlying object.

Return type OCCT.SMESH.SMESH_MesherHelper

static is_structured(*submesh*)

Check if a 2-D mesh on a face is structured.

Parameters **submesh** (`afem.smesh.entities.SubMesh` (page 199)) – The submesh.

Returns *True* if structured, *False* if not.

Return type bool

static is_distorted2d(*submesh*, *check_uv=False*)

Check if a 2-D mesh on a face is distorted.

Parameters

- **submesh** (`afem.smesh.entities.SubMesh` (page 199)) – The submesh.
- **check_uv** (*bool*) – Option to check using *uv* parameters.

Returns *True* if distored, *False* if not.

Return type bool

static subshape_by_node(*node*, *mesh_ds*)

Get the support shape of a node.

Parameters

- **node** (`afem.smesh.entities.Node` (page 187)) – The node.
- **mesh_ds** (`afem.smesh.entities.MeshDS` (page 196)) – The mesh data structure.

Returns The support shape.

Return type *afem.topology.entities.Shape* (page 109)

static common_ancestor(*shape1*, *shape2*, *mesh*, *ancestor_type=TopAbs_ShapeEnum.TopAbs_EDGE*)

Get a common ancestor between the two shapes.

Parameters

- **shape1** (`afem.topology.entities.Shape` (page 109)) – The first shape.
- **shape2** (`afem.topology.entities.Shape` (page 109)) – The second shape.
- **mesh** (`afem.smesh.entities.Mesh` (page 193)) – The mesh.
- **ancestor_type** (*OCCT.TopAbs.TopAbs_ShapeEnum*) – The shape type.

Returns The common ancestor.

Return type *afem.topology.entities.Edge* (page 115)

static is_subshape_by_shape(*shape*, *main_shape*)
Check to see if the shape is a sub-shape in a shape.

Parameters

- **shape** (`afem.topology.entities.Shape` (page 109)) – The shape.
- **main_shape** (`afem.topology.entities.Shape` (page 109)) – The main shape.

Returns *True* if a sub-shape, *False* if not.

Return type bool

static is_subshape_by_mesh(*shape*, *mesh*)
Check to see if the shape is a sub-shape in a mesh.

Parameters

- **shape** (`afem.topology.entities.Shape` (page 109)) – The shape.
- **mesh** (`afem.smesh.entities.Mesh` (page 193)) – The mesh.

Returns *True* if a sub-shape, *False* if not.

Return type bool

static get_angle(*e1*, *e2*, *f*, *v*)
Determine the angle between the two edges on a face.

Parameters

- **e1** (`afem.topology.entities.Edge` (page 115)) – The first edge.
- **e2** (`afem.topology.entities.Edge` (page 115)) – The second edge.
- **f** (`afem.topology.entities.Face` (page 117)) – The face the edges belong to.
- **v** (`afem.topology.entities.Vertex` (page 115)) – The vertex connecting the two edges.

Returns The angle between the edges.

Return type float

static is_closed_edge(*edge*)
Check if edge is closed.

Parameters **edge** (`afem.topology.entities.Edge` (page 115)) – The edge.

Returns *True* if closed, *False* if not.

Return type bool

static shape_by_hypothesis(*hyp*, *shape*, *mesh*)
Get a shape the hypothesis is applied to.

Parameters

- **hyp** (`afem.smesh.hypotheses.Hypothesis` (page 204)) – The hypothesis.
- **shape** (`afem.topology.entities.Shape` (page 109)) – The shape.
- **mesh** (`afem.smesh.entities.Mesh` (page 193)) – The mesh.

Returns The shape that the hypothesis is applied to.

Return type *afem.topology.entities.Shape* (page 109)

is_reversed_submesh(*face*)
Check to see if the elements have opposite orientation on the face.

Parameters **face** (`afem.topology.entities.Face` (page 117)) – The face.

Returns *True* if reversed, *False* if not.

Return type bool

set_subshape(*shape*)
Set the shape to make elements on.

Parameters **shape** (`afem.topology.entities.Shape` (page 109) *or* *int*) – The shape or the shape ID.

Returns None.

shape_to_index(*shape*)
Get the shape index.

Parameters **shape** (`afem.topology.entities.Shape` (page 109)) – The shape.

Returns The shape index.

Return type int

add_node(*x*, *y*, *z*, *id_=0*, *u=0.0*, *v=0.0*)
Create a node.

Parameters

- **x** (*float*) – The x-location.
- **y** (*float*) – The y-location.
- **z** (*float*) – The z-location.
- **id** (*int*) – The node ID. If zero then the ID will be assigned.
- **u** (*float*) – The node u-parameter.
- **v** (*float*) – The node v-parameter.

Returns The created node.

Return type *afem.smesh.entities.Node* (page 187)

add_edge(*n1*, *n2*, *id_=0*, *force3d=True*)
Add an edge.

Parameters

- **n1** (`afem.smesh.entities.Node` (page 187)) – The first node.
- **n2** (`afem.smesh.entities.Node` (page 187)) – The second node.
- **id** (*int*) – The edge ID. If zero then the ID will be assigned.
- **force3d** – Unknown option.

Returns The created edge.

Return type *afem.smesh.entities.Element* (page 188)

add_face(*n1*, *n2*, *n3*, *n4=None*, *id_=0*, *force3d=False*)
Add a face.

Parameters

- **n1** (`afem.smesh.entities.Node` (page 187)) – The first node.
- **n2** (`afem.smesh.entities.Node` (page 187)) – The second node.
- **n3** (`afem.smesh.entities.Node` (page 187)) – The third node.
- **n4** (`afem.smesh.entities.Node` (page 187)) – The fourth node. If provided then the face will be a quadrangle, if not provided then the face is a triangle.
- **id** (*int*) – The face ID. If zero then the ID will be assigned.
- **force3d** – Unknown option.

Returns The created face.

Return type *afem.smesh.entities.Element* (page 188)

CHAPTER

TWELVE

OML

The afem.oml package provides a minimal set of entities and tools intended to make defining structural components easier and more efficient. Many of the tools used to create parts require a solid defining the common material of the part given its basis shape. Some tools are more automated and parametric and therefore need more information, like a reference surface or curve. The oml package, and more specifically the *Body* (page 228) class, are intended to provide a single type that contains all necessary information. The entities and tools can be imported by:

```
from afem.oml import *
```

The intent of the oml package and the *Body* (page 228) class is best understood by manually building a solid shape that represents a basic wing as shown in the following example:

```
from math import tan, radians

from afem.geometry import *
from afem.graphics import Viewer
from afem.oml import *
from afem.sketch import *
from afem.topology import *

# Parameters
semispan = 107.  # wing semi-span
sweep = 34.  # Leading edge sweep
uk = 0.28  # Percent semi-span to locate section cross section
c1 = 51.5  # Root chord
c2 = 31.  # Chord of second section
c3 = 7.  # Tip chord
t3 = 3.  # Tip washout in degrees

# Define leading edge points of cross sections
p1x, p1y = 0., 0.
p2x = semispan * uk * tan(radians(sweep))
p2y = semispan * uk
p3x = semispan * tan(radians(sweep))
p3y = semispan

# Create a cross section using an UIUC airfoil file
cs = Airfoil()
cs.read_uiuc('../models/clarky.dat')

# Define cross section planes
pln1 = PlaneByAxes((p1x, p1y, 0), axes='xz').plane
pln2 = PlaneByAxes((p2x, p2y, 0), axes='xz').plane
pln3 = PlaneByAxes((p3x, p3y, 0), axes='xz').plane
```

(continues on next page)

(continued from previous page)

```
# Build cross sections
cs.build(pln1, scale=c1)
wire1 = cs.wires[0]
cs.build(pln2, scale=c2)
wire2 = cs.wires[0]
cs.build(pln3, scale=c3, rotate=t3)
wire3 = cs.wires[0]

# Loft a solid
shape = LoftShape([wire1, wire2, wire3], True, make_ruled=True).shape

# Make a body
wing = Body(shape, 'Wing')
wing.set_transparency(0.5)
wing.set_color(1, 0, 0)

# Build a reference surface using chord lines of the cross sections. Make sure
# to use the same scaling and rotation parameters. Set the parametric domains
# to be between 0 and 1 for convenience.
chord1 = cs.build_chord(pln1, scale=c1)
chord2 = cs.build_chord(pln2, scale=c2)
chord3 = cs.build_chord(pln3, scale=c3, rotate=t3)
sref = NurbsSurfaceByInterp([chord1, chord2, chord3], 1).surface
sref.set_udomain(0., 1.)
sref.set_vdomain(0., 1.)

# Set the wing reference surface
wing.set_sref(sref)

# Show wing and reference surface
gui = Viewer()
gui.add(wing, wing.sref)
gui.start()

# Show the underlying shape of the reference surface
gui.clear()
gui.add(wing.sref_shape)
gui.start()

# Evaluate point
p = wing.eval(0.5, 0.5)

# Extract a plane
pln = wing.extract_plane(0.5, 0., 0.5, 0.5)
face = FaceByPlane(pln, -10, 10, -10, 10).face

# Extract a trimmed curve
crv = wing.extract_curve(0.15, 0.05, 0.15, 0.95)

gui.clear()
gui.add(wing.sref, p, face, crv)
gui.start()
```

After the necessary modules are imported, a few parameters are specified that will be used to define the shape of the wing:

```
semispan = 107.  # wing semi-span
sweep = 34.  # Leading edge sweep
uk = 0.28  # Percent semi-span to locate section cross section
c1 = 51.5  # Root chord
c2 = 31.  # Chord of second section
c3 = 7.  # Tip chord
t3 = 3.  # Tip washout in degrees
```

The wing will be defined by three cross sections using the same airfoil but at different locations and scales. A constant leading edge sweep is assumed so the leading edge locations of each cross section are:

```
p1x, p1y = 0., 0.
p2x = semispan * uk * tan(radians(sweep))
p2y = semispan * uk
p3x = semispan * tan(radians(sweep))
p3y = semispan
```

An airfoil from the UIUC airfoil database is used to create the cross section:

```
cs = Airfoil()
cs.read_uiuc('../models/clarky.dat')
```

One advantage of using an *Airfoil* (page 97) is that the leading and trailing edge points are stored and from that a chord line can be built which later will be used to build additional reference geometry stored with the wing.

Three planes are defined using the points derived from the wing parameters:

```
pln1 = PlaneByAxes((p1x, p1y, 0), axes='xz').plane
pln2 = PlaneByAxes((p2x, p2y, 0), axes='xz').plane
pln3 = PlaneByAxes((p3x, p3y, 0), axes='xz').plane
```

These planes represent the eventual location of the 3-D airfoil cross sections where the local origin of the plane is also the leading edge in this case. At each section, a 3-D wire is built by providing the 3-D plane and a scaling and rotation parameter:

```
cs.build(pln1, scale=c1)
wire1 = cs.wires[0]
cs.build(pln2, scale=c2)
wire2 = cs.wires[0]
cs.build(pln3, scale=c3, rotate=t3)
wire3 = cs.wires[0]
```

These wires are then used to loft a solid shape:

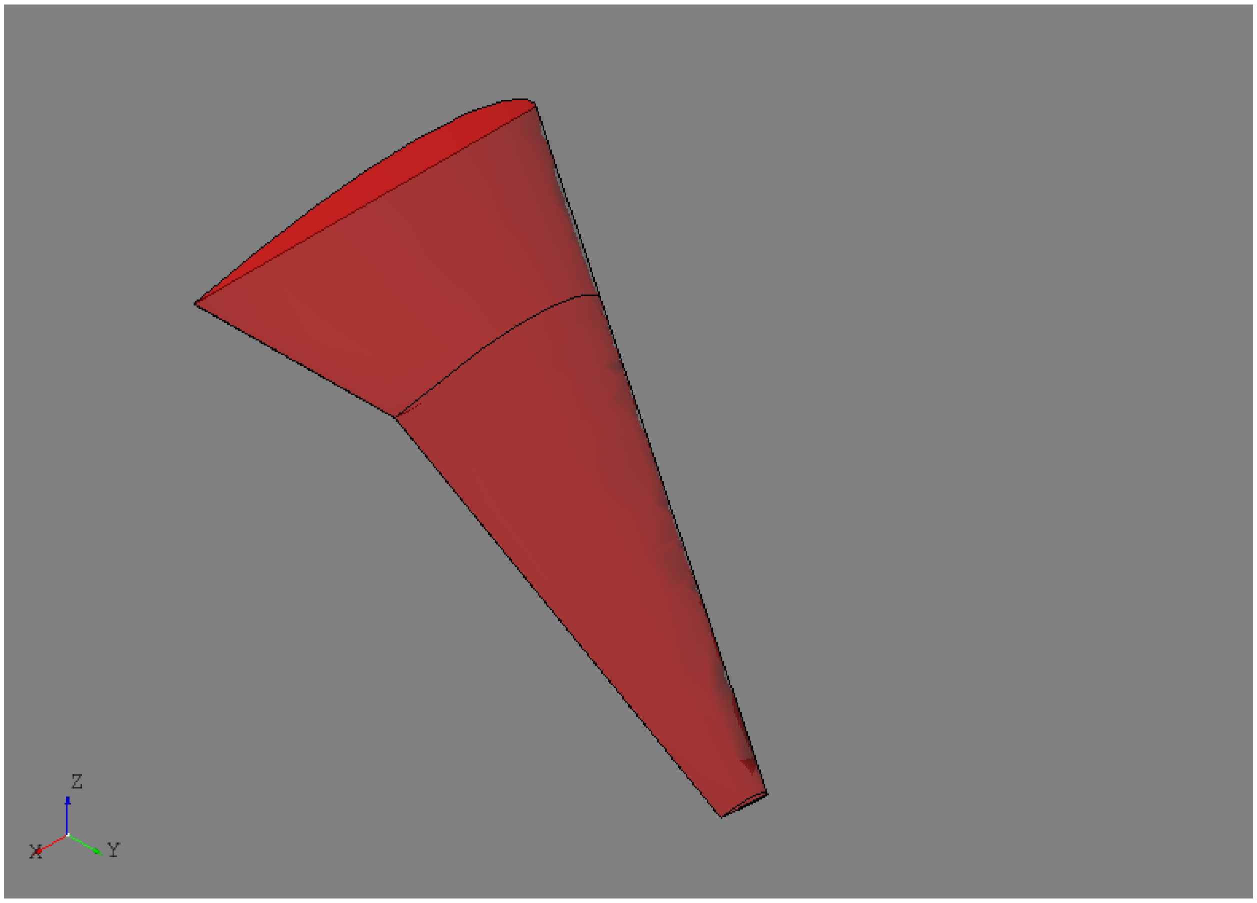
```
shape = LoftShape([wire1, wire2, wire3], True, make_ruled=True).shape
```

The `make_ruled=True` option was used to produce linear segments between the airfoil cross sections, rather than blending them together with approximation.

At this point, a solid body (i.e., a *Solid* (page 118)) has been defined and a *Body* (page 228) can be initialized:

```
wing = Body(shape, 'Wing')
```

The `name` parameter, given as "Wing" in this example, is optional but is useful and can be treated as a default type of metadata. Defining a valid *Solid* (page 118) is the only hard requirement when defining a *Body* (page 228) for use during structural modeling. The user can use all the tools available to create them manually as done above, or they can be derived by importing data from other sources such as STEP files. The wing derived above is shown below:



It is at times convenient to define structural components using normalized or relative parameters rather than in absolute terms. For example, in a wing it is common to define a spar's location and orientation by percent chord and percent semispan. This leads to the need for some kind of "reference geometry" on which to evaluate these types of parameters. For a wing, a natural solution is to define a continuous surface that generally represents its planform shape. Since many of the structural modeling tools use this type of surface, storing a "reference surface" and other reference geometry is provided by the *Body* (page 228) class.

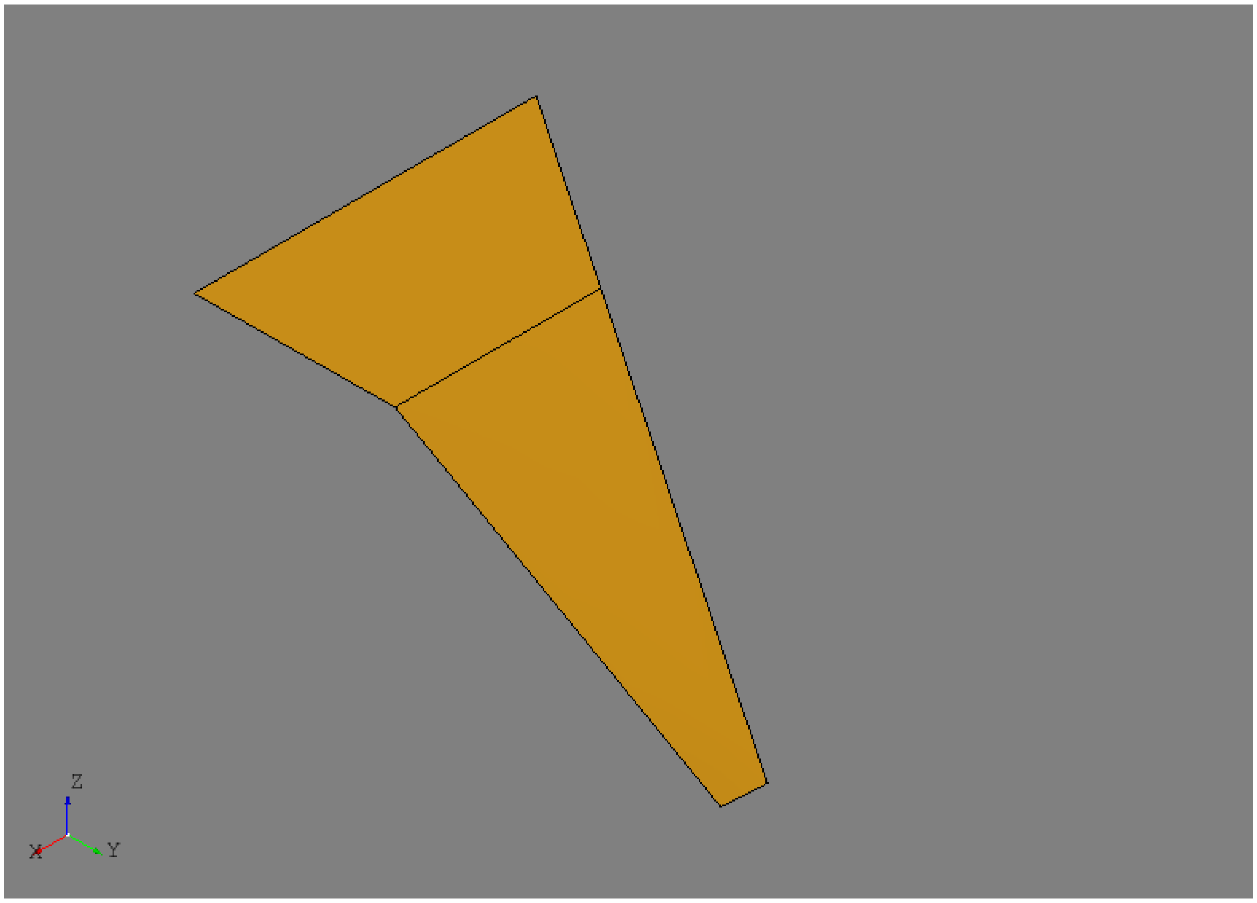
For the wing shown above, a reference surface will be built by lofting the chord lines of the three airfoil cross sections:

```
chord1 = cs.build_chord(pln1, scale=c1)
chord2 = cs.build_chord(pln2, scale=c2)
chord3 = cs.build_chord(pln3, scale=c3, rotate=t3)
sref = NurbsSurfaceByInterp([chord1, chord2, chord3], 1).surface
sref.set_udomain(0., 1.)
sref.set_vdomain(0., 1.)
```

The chord lines are built using the same scaling and rotation parameters and a linear surface is constructed. The parametric domain of the surface is set to be between 0 and 1 for convenience. This essentially corresponds to the u-direction being percent chord and the v-direction percent semispan (in this case along the sweep direction). The surface is set as the reference surface by:

```
wing.set_sref(sref)
```

When a reference surface is set, the method will automatically construct an underlying shape by splitting the surface at any C1 discontinuities. This shape is primarily used for intersection methods when extracting curves since C1 discontinuous surfaces may be numerically unreliable, but it is provided to the user if needed. This `wing.sref_shape` is shown below:



A number of methods are provided for convenience if the Body contains reference geometry such as as reference surface. A point on the surface can be evaluated by:

```
p = wing.eval(0.5, 0.5)
```

A plane can be defined between two points using the reference surface:

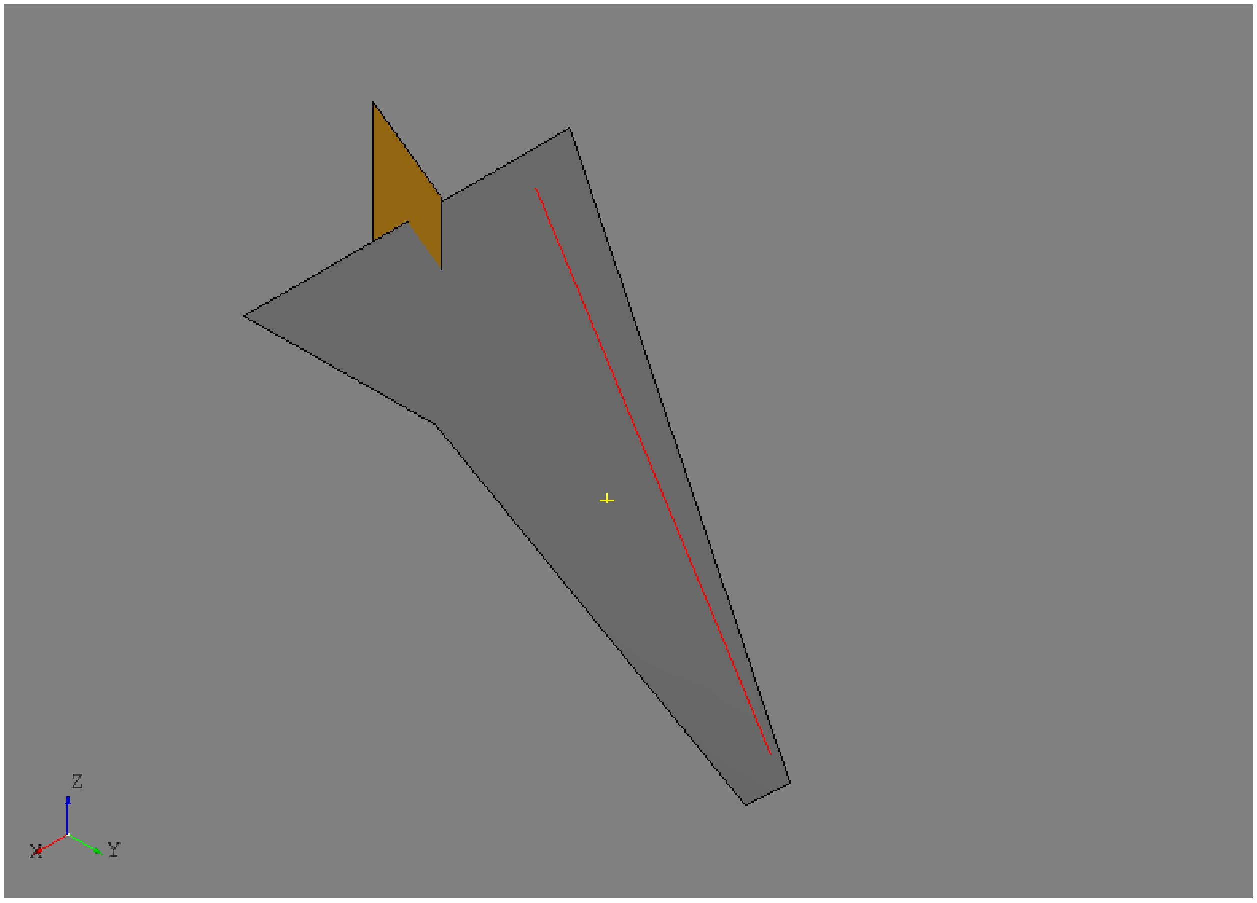
```
pln = wing.extract_plane(0.5, 0., 0.5, 0.5)
```

Here, two points are defined and then a third is defined by translating the first point in the direction of the reference surface normal. These three points are used to define a plane which can be useful for structural modeling.

Sometimes it may be useful to build a curve within the reference surface between two points as shown below:

```
crv = wing.extract_curve(0.15, 0.05, 0.15, 0.95)
```

This method will generated a *`TrimmedCurve`* (page 48) between the parameters by intersecting a `basis_shape` with the reference surface. If a basis shape is not provided, a plane will be constructed as described above. The result of these methods are shown below:



The main takeaway for the `oml` package and the *Body* (page 228) class is that it is meant to be a container for minimum data needed for structural modeling with minimal restrictions and assumptions. Creating the solid contained by the *Body* (page 228), as well as any associated reference geometry, can be as complex or as simple as the user needs and/or desires.

12.1 Entities

12.1.1 Body

class `afem.oml.entities.`**`Body`**(*shape*, *name='Body'*)

Bases: *afem.core.entities.ShapeHolder* (page 13)

Generic class for solid bodies and encapsulating necessary information when creating structural components.

Parameters

- **shape** (`afem.topology.entities.Solid` (page 118) *or* `afem.topology.entities.Shape` (page 109)) – The shape which should be a solid.
- **name** (*str*) – The name.

`mirrored`(*pln*, *name=None*)

Mirror this Body using the plane.

Parameters

- **pln** (`afem.geometry.entities.Plane` (page 52)) – The plane.

- **name** (*str*) – The name of the new Body.

Returns Mirrored Body.

Return type *afem.oml.entities.Body* (page 228)

static save_bodies(*fn*, *bodies*, *binary=True*)

Save Body instances.

Parameters

- **fn** (*str*) – The filename.
- **bodies** (*collections.Sequence(afem.oml.entities.Body (page 228))*) – The Body instances
- **binary** (*bool*) – If *True*, the document will be saved in a binary format. If *False*, the document will be saved in an XML format.

Returns *True* if saved, *False* otherwise.

Return type bool

Note: This method only saves the name, shape, reference surface, and color of the Body. Other user-defined metadata is currently not saved.

static load_bodies(*fn*)

Load saved Body instances.

Parameters **fn** (*str*) – The filename. The extension should be either ".xbf" for a binary file or ".xml" for an XML file.

Returns A dictionary where the key is the name and the value is the Body instance.

Return type dict(str, *afem.oml.entities.Body* (page 228))

Raises **TypeError** – If the file extension type is not supported.

12.2 Check

12.2.1 CheckOML

class afem.oml.check.**CheckOML**

Bases: `object`

Check OML.

static is_body(*entity*)

Check to see if the entity is a Body.

Parameters **entity** – Entity to check.

Returns *True* if Body, *False* if not.

Return type bool

CHAPTER

THIRTEEN

STRUCTURE

The `afem.structure` package provides the low-level types and tools that the user can leverage to build more complete and useful aircraft structural models. This package is the main focus of the entire AFEM project and is built on top of almost all the other packages. The entities and tools can be imported by:

```
from afem.structure import *
```

The sub-sections and example below will describe the package in more detail, but some discussion about the general modeling approach and "best-practices" is helpful. The AFEM library does not necessarily provide new methods for building structural geometry, but rather focuses on providing a more specialized API and workflow that streamlines and automates many of the repetitive and mundane tasks typically encountered in the traditional modeling process. The end result is still B-Rep topology and a finite element model and therefore is still subject to the limitations and/or weaknesses of these modeling paradigms. This may include robustness of Boolean operations within the geometric modeling kernel or the quality of the finite element mesh. Through experience, a procedure has been found to work well for a wide range of applications:

1. **Build initial parts** The first step of the process is simply defining the initial reference geometry and shape for the part without any real topological connection to any others. At this point, the parts and their shapes can be used to build others but there is less concern or emphasis for details like establishing topologically congruent edges. The goal for this initial step is to define the initial part shape as quickly and easily as possible while keeping its representation flexible.

2. **Join parts** The initial part shapes may be connected spatially but disconnected topologically. It's during this step that modeling tools are used to join the various shapes together to form shared and congruent edges between the connecting parts. Modeling flexibility is reduced after this step and joining operations are best done to large groups of connected parts so that joints with multiple and complex intersections can be handled robustly. If new parts are created after this step and joined locally with ones that already exist and have been joined, it is not guaranteed that the overall model will maintain the proper topology. In this case, it would be better to define the part in the first step and include in the overall joining operation. The structure usually involved in this step is the internal structure such as ribs, spars, bulkheads, frames, etc. The external skin is usually joined in a later step.

3. **Trim parts** After the parts have been joined, it is usually the case that certain pieces or regions of the parts should be discarded. For example, if modeling only the primary load-carrying structure of a wing box, the regions of the ribs aft of the rear spar may need to be removed from the model. In an interactive GUI, this would be a "select and delete" action. Typically, no new topology is introduced in this step but rather certain regions of the existing model are removed without modifying others. For the wing box example mentioned earlier, this may be first identifying the faces aft of the rear spar using various tools and then removing those faces from the shape. As mentioned earlier, Step 2 usually involves joining internal structure and not necessarily external structure like the wing skin. Joining the external skins with internal structure is is best done after the internal structure has been joined and trimmed. This avoids creating unnecessary edges in the shape of the external structure since internal regions will be discarded.

4. **Applying meshing controls** At this point, it is assumed the geometrical modeling operations are complete and

now a finite element mesh can be created. Various tools are provided for applying both global and local meshing controls on any part, shape, or sub-shape. The number of elements along an edge or max element size on a face can be applied on a per edge or face level.

5. **Generate FEM** The last step is simply computing the FEM considering the global shape to mesh and all the local meshing controls (if any). The result is a mesh data structure that can be supplied to downstream processes. Local mesh data for a part, shape, or sub-shape can be extracted from this global mesh to further support downstream tools and processes.

The basic wing box example below goes through each of these steps in more detail and highlights the key features, types, and tools in the AFEM toolkit:

```
from afem.config import Settings
from afem.exchange import ImportVSP
from afem.geometry import *
from afem.graphics import Viewer
from afem.structure import *
from afem.topology import *

Settings.log_to_console()

# Set units to inch.
Settings.set_units('in')

# Initialize a viewer
gui = Viewer()

# Import an OpenVSP model that includes OML reference geometry
fn = r'../models/simple_wing.stp'
vsp_import = ImportVSP(fn)
wing = vsp_import['WingGeom']
wing.set_color(1., 0., 0.)
wing.set_transparency(0.75)

gui.add(wing.sref, wing)
gui.start()
gui.clear()

# Define a group to put the structure in
wingbox = GroupAPI.create_group('wing box')
spars_assy = wingbox.create_subgroup('spar assy', active=True)
ribs_assy = wingbox.create_subgroup('rib assy', active=False)

# Define a front spar between parameters on the wing reference surface
fspar = SparByParameters('fspar', 0.15, 0.1, 0.15, 0.98, wing).part

gui.add(wing.sref, fspar, fspar.cref, fspar.cref.p1, fspar.cref.p2)
gui.start()

# Define a rear spar between parameters on the wing reference surface
rspar = SparByParameters('rspar', 0.65, 0.1, 0.65, 0.98, wing).part

gui.add(rspar, rspar.cref, rspar.cref.p1, rspar.cref.p2)
gui.start()

# Activate rib assembly
ribs_assy.activate()
```

(continues on next page)

(continued from previous page)

```
# Define root rib between front and rear spar
root = RibByPoints('root', fspar.cref.p1, rspar.cref.p1, wing).part

gui.add(root, root.cref, root.cref.p1, root.cref.p2)
gui.start()

# Define tip rib between front and rear spar
tip = RibByPoints('tip', fspar.cref.p2, rspar.cref.p2, wing).part

gui.add(tip, tip.cref, tip.cref.p1, tip.cref.p2)
gui.start()

# Add ribs between root and tip perpendicular to rear spar reference curve
ribs = RibsAlongCurveByDistance('rib', rspar.cref, 30., fspar, rspar, wing,
                                d1=30., d2=-30.).parts

for rib in ribs:
    gui.add(rib, rib.cref, rib.cref.p1, rib.cref.p2)
gui.start()

# Activate spar assembly
spars_assy.activate()

# Add a front center spar considering the intersection between the front
# spar and the root rib. If this is not considered, the front center spar
# may be oriented in such a way that causes it to have a gap with the front
# spar and root rib.
p1 = wing.sref.eval(0.25, .0)
pln = PlaneByIntersectingShapes(fspar.shape, root.shape, p1).plane
fcspar = SparByPoints('fcspar', p1, root.cref.p1, wing, pln).part

gui.add(fcspar, fcspar.cref, fcspar.cref.p1, fcspar.cref.p2)
gui.start()

# Add rear center spar
p1 = wing.sref.eval(0.75, .0)
pln = PlaneByIntersectingShapes(rspar.shape, root.shape, p1).plane
rcspar = SparByPoints('rcspar', p1, root.cref.p2, wing, pln).part

gui.add(rcspar, rcspar.cref, rcspar.cref.p1, rcspar.cref.p2)
gui.start()

# Activate rib assembly
ribs_assy.activate()

# Add center ribs using a reference plane alonge the rear center spar
ref_pln = PlaneByAxes(origin=(0., 0., 0.), axes='xz').plane
ribs = RibsAlongCurveByNumber('center rib', rcspar.cref, 3, fcspar, rcspar,
                              wing, ref_pln, d1=6, d2=-18).parts

for rib in ribs:
    gui.add(rib, rib.cref, rib.cref.p1, rib.cref.p2)
gui.start()

# Draw the part reference curves to see what the layout will eventually
# look like
gui.clear()
```

(continues on next page)

(continued from previous page)

```
gui.add(wing.sref)
for part in wingbox.get_parts():
    gui.add(part.cref)
gui.start()

# Join the internal structure using their reference curves to check for
# actual intersection
internal_parts = wingbox.get_parts()
FuseSurfacePartsByCref(internal_parts)

gui.add(wingbox)
gui.start()

# Discard faces of parts using the reference curve
DiscardByCref(internal_parts)

gui.clear()
gui.add(wingbox)
gui.start()

# Activate wingbox assembly
wingbox.activate()

# Extract the shell of wing to define the wing skin
skin = SkinByBody('skin', wing).part
skin.set_transparency(0.5)

gui.add(skin)
gui.start()

# Join the wing skin with the internal structure
skin.fuse(*internal_parts)

# Discard wing skin that is touching the wing reference surface. This
# should leave only the upper and lower skins.
print(skin.shape.shape_type)
skin.discard_by_dmin(wing.sref, 1.)

# After removing the faces, the skin is now a compound of two shells, one
# upper shell and one lower. Use the Part.fix() to alter the shape from an
# invalid shell to a compound of two shells.
print('Skin shape type before fix:', skin.shape.shape_type)
skin.fix()
print('Skin shape type after fix:', skin.shape.shape_type)

gui.clear()
gui.add(wingbox)
gui.start()

# Check for free edges
shape = GroupAPI.get_shape()
tool = ExploreFreeEdges(shape)

gui.clear()
gui.add(shape, *tool.free_edges)
gui.start()
```

(continues on next page)

(continued from previous page)

```
# Begin meshing
print('Creating mesh...')
mesh = MeshVehicle(4.)

# Set number of elements between spars and skin edges
spars = wingbox.get_parts(rtype=Spar)
skins = wingbox.get_parts(rtype=Skin)
shape = ExploreParts.shared_edges(spars, skins, True)
mesh.set_max_length_1d(4., shape)

# Set number of elements between spar and rib edges
ribs = wingbox.get_parts(rtype=Rib)
shape = ExploreParts.shared_edges(spars, ribs, True)
mesh.set_number_segments_1d(4, shape)

# Apply structured quadrilateral mesh
shape = ExploreParts.shared_edges(skins, ribs, True)
mesh.set_number_segments_1d(15, shape)
for part in wingbox.get_parts():
    mesh.set_quadrangle_2d(part.shape)

# Compute the mesh
mesh.compute()

# View
gui.clear()
gui.add(mesh)
gui.start()

# Create node groups of the spars and ribs
spar_nodes = mesh.create_node_group(spars_assy.get_shape())
rib_nodes = mesh.create_node_group(ribs_assy.get_shape())

gui.clear()
gui.add(spar_nodes, rib_nodes)
gui.start()

# Find common nodes between spars and ribs
shared_nodes = spar_nodes.intersect(rib_nodes, 'spar and rib nodes')

gui.clear()
gui.add(mesh, shared_nodes)
gui.start()

# Create edge groups of front spar and skin and find shared edges
fspar_edges = mesh.create_edge_group(fspar.shape)
skin_edges = mesh.create_edge_group(skin.shape)
shared_edges = skin_edges.intersect(fspar_edges)

gui.clear()
gui.add(shared_edges)
gui.start()

# Export mesh to Nastran
mesh.export_nastran('structure_basic.bdf')
```

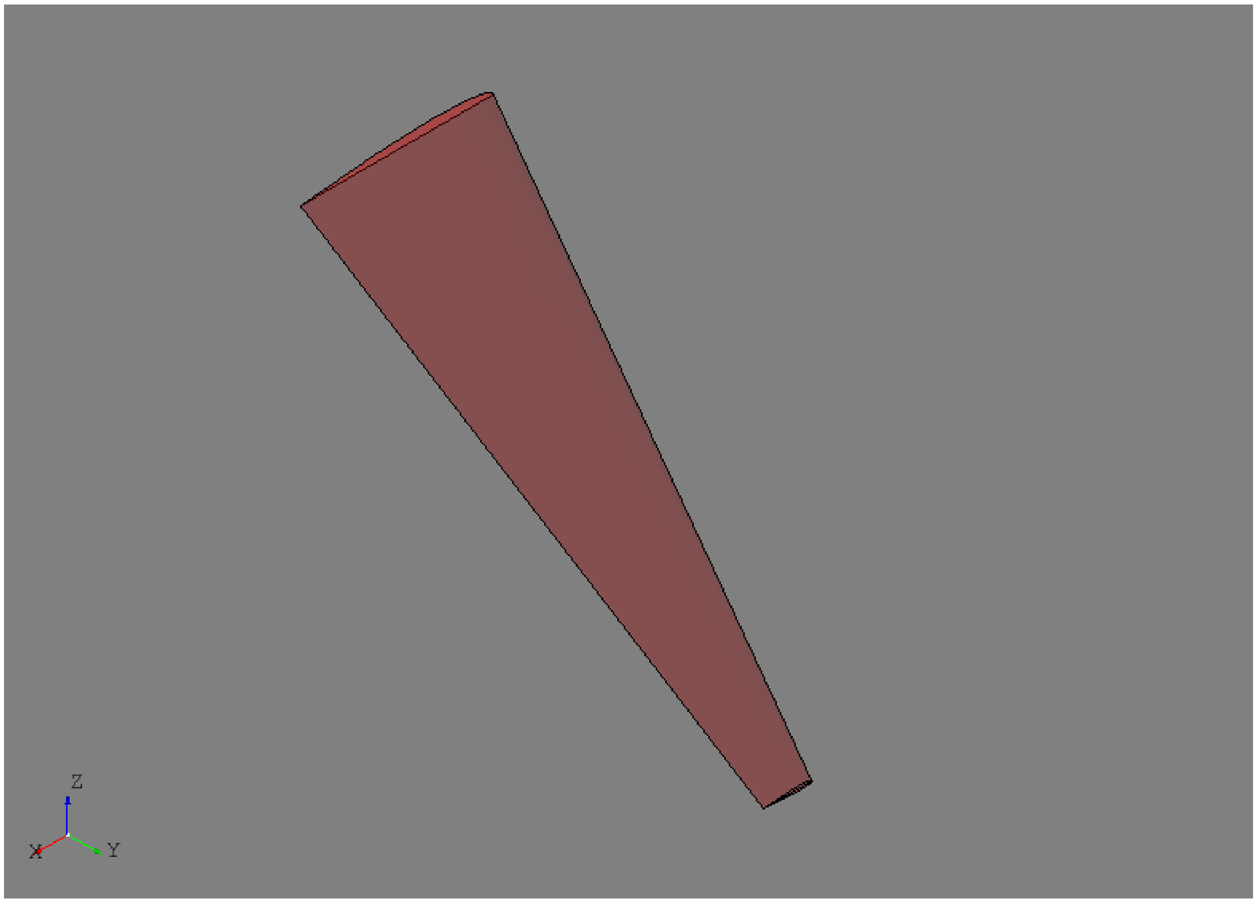
The procedure starts by settings the default length units to inches:

```
Settings.set_units('in')
```

This is done before importing the STEP file so that it can be properly translated upon import. In this example a simple wing geometry generated by OpenVSP will be used:

```
fn = r'../models/simple_wing.stp'
vsp_import = ImportVSP(fn)
wing = vsp_import['WingGeom']
wing.set_color(1., 0., 0.)
wing.set_transparency(0.75)
```

Note that this model was generated with a specialized build of OpenVSP that includes additional reference geometry in the STEP export tool. For more details see the section on *OpenVSP import* (page 310). Upon import, a `Body` (page 228) type will be created that contains the solid that represents the wing as well as the wing reference surface as shown below.



The reference surface of the wing in this case is lofted through the chord lines of each wing section and provides the basis for the parametric (i.e, uv) definition of the wing. For this particular model, the u-direction will correspond to chord and v-direction will correspond to span. The leading edge will be at u=0 and the trailing edge at u=1, while the root is at v=0 and the tip at v=1. This is not an absolute rule and is completely dependent on the underlying surface set as the wing's reference surface. This flexibility was left open in case different users found other parametrization techniques more intuitive or suitable for a given application.

To provide model organization, three different `Group` (page 261) entities are created using the `GroupAPI` (page 262) tool:

```
wingbox = GroupAPI.create_group('wing box')
spars_assy = wingbox.create_subgroup('spar assy', active=True)
ribs_assy = wingbox.create_subgroup('rib assy', active=False)
```

The `wingbox` will be the top-level group and sub-groups for the spars and ribs are created as shown. Note that the spar assembly was set `active=True` while the rib assembly was `False`. When a *Part* (page 253) is created it is by default placed in the active *Group* (page 261) if one was not provided. Since spars are the first items created in this example, the spar assembly was made active. The *Group* (page 261) can also be supplied to the *Part* (page 253) builder tool directly.

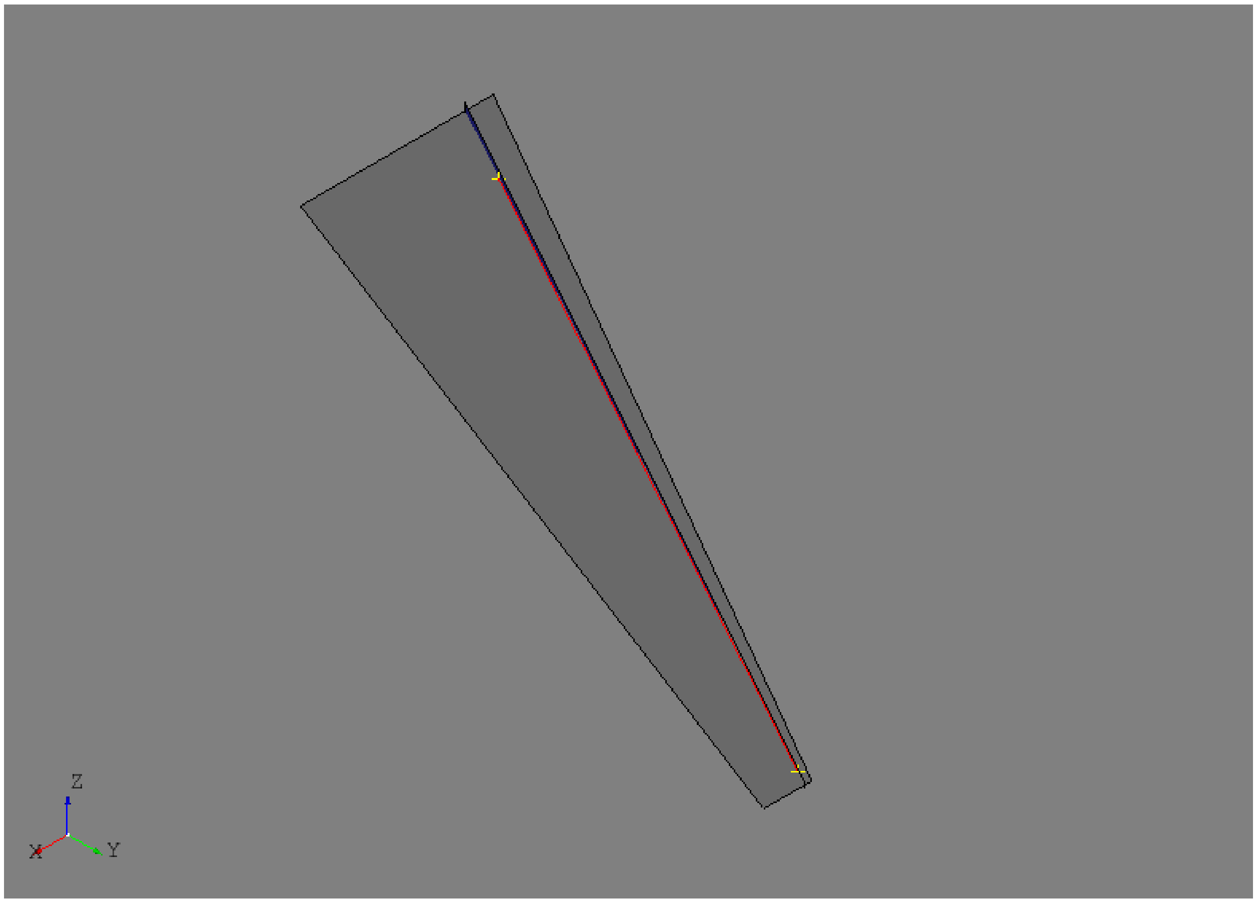
A front spar is defined in the parametric domain of the wing using its reference surface and the *SparByParameters* (page 275) tool:

```
fspar = SparByParameters('fspar', 0.15, 0.1, 0.15, 0.98, wing).part
```

Note that the *uv* parameters are relative to the reference surface of the *Body* (page 228) supplied to this tool. This tool, and others like it, are designed to operate as described in Step 1 above and generate the initial shape of the part. The initial shape is formed by simply finding the intersecting material between the *Body* (page 228) solid and a basis shape. The *SparByParameters* (page 275) tool, and others like it, can usually be initialized with an optional `basis_shape` input. If provided, this basis shape will be used to find the initial part shape and has few limitations (i.e., could be curvilinear). If a basis shape is not provided, for this tool a plane is automatically defined between two points on the wing reference surface defined by the *uv* parameters and a third which is offset in the normal direction at the (*u1*, *v1*) location. This approach usually results in a reasonable orientation of a planar part in case the wing is twists or changes dihedral. This is another reason why is it usually good practice, if not required, to provide define and attach a reference surface to the input *Body* (page 228).

At this point the initial shape may extend beyond the anticipated start and end locations since the process has simply found an intersection between the wing solid and a plane. Not trimming the actual shape to its start and end locations is done because: 1) the interfacing parts and shapes are not yet defined and 2) because no assumption is made about the shape of the interfacing structure. Point 2 is particularly important because although assumptions could be made and the shape trimmed with a plane at the start and end points based on the *uv* parameters for example, it is important to keep it general in case the interfacing structure is actually some curvilinear shape that is not yet defined (e.g., a bulkhead in a fuselage).

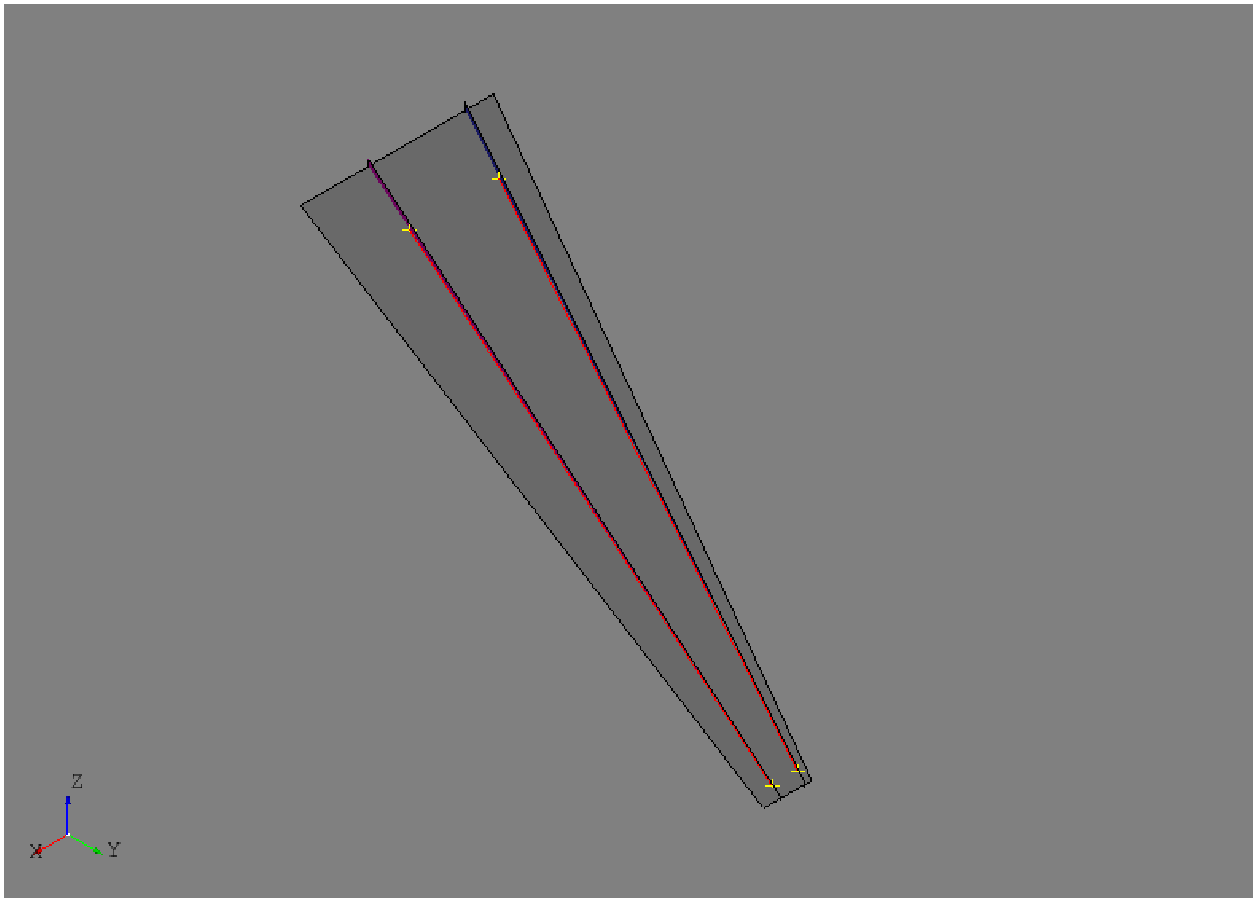
Although the part shape is not initially trimmed, having some way of tracking where it will generally start and stop is a valuable reference. In the case of the front spar, this is accomplished by building and associating a *Part* (page 253) *reference curve* within the *SparByParameters* (page 275) process. This reference curve is constructed by intersecting the wing's reference surface with the part's basis shape (a default plane in this case). The resulting intersection is converted into a *TrimmedCurve* (page 48) entity. Initially this curve also extends the full length of the reference surface, but the points defined by the *uv* parameters are used to bound the curve. These points and the trimmed curve represent a planform view of where the part will generally start and end on the wing, even though the spar's shape at this point still extends the full volume. The initial part shape, its reference curve (shown in red), and its start and end points (shown in yellow) are shown below.



The rear spar is defined using the same tool but at 65% chord:

```
rspar = SparByParameters('rspar', 0.65, 0.1, 0.65, 0.98, wing).part
```

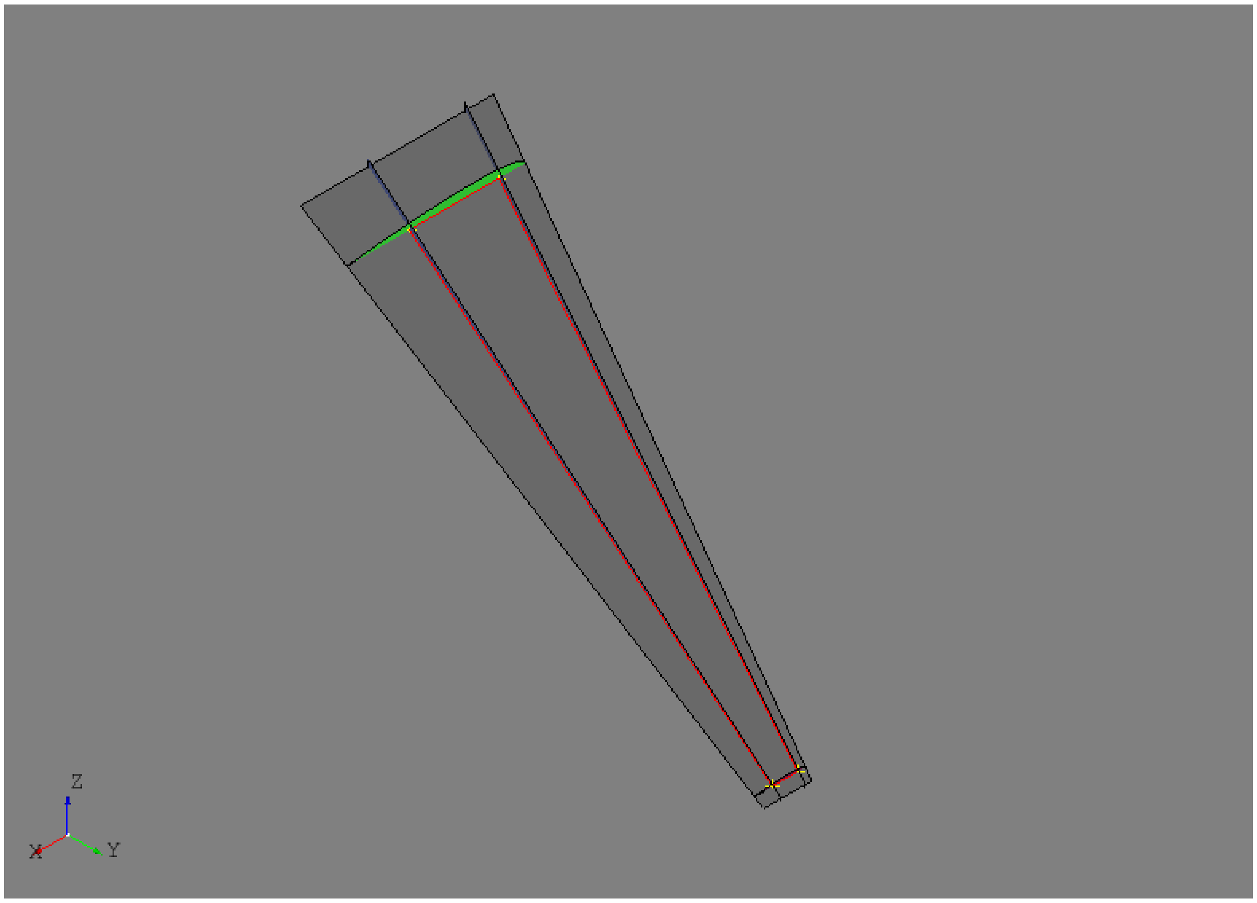
The initial shapes of the front and rear spar as well as their reference geometry are shown below.



After activating the ribs assembly, ribs will be defined at the root and tip of the front and rear spars using the `RibByPoints` (page 281) tool. In this tool, start and end points are given instead of parameters on the wing reference surface. Although, these points should generally lie close to or on the wing's reference surface because the tool simply inverts them to find *uv* parameters and then uses the `RibByParameters` (page 280) tool. The root and tip ribs are defined as:

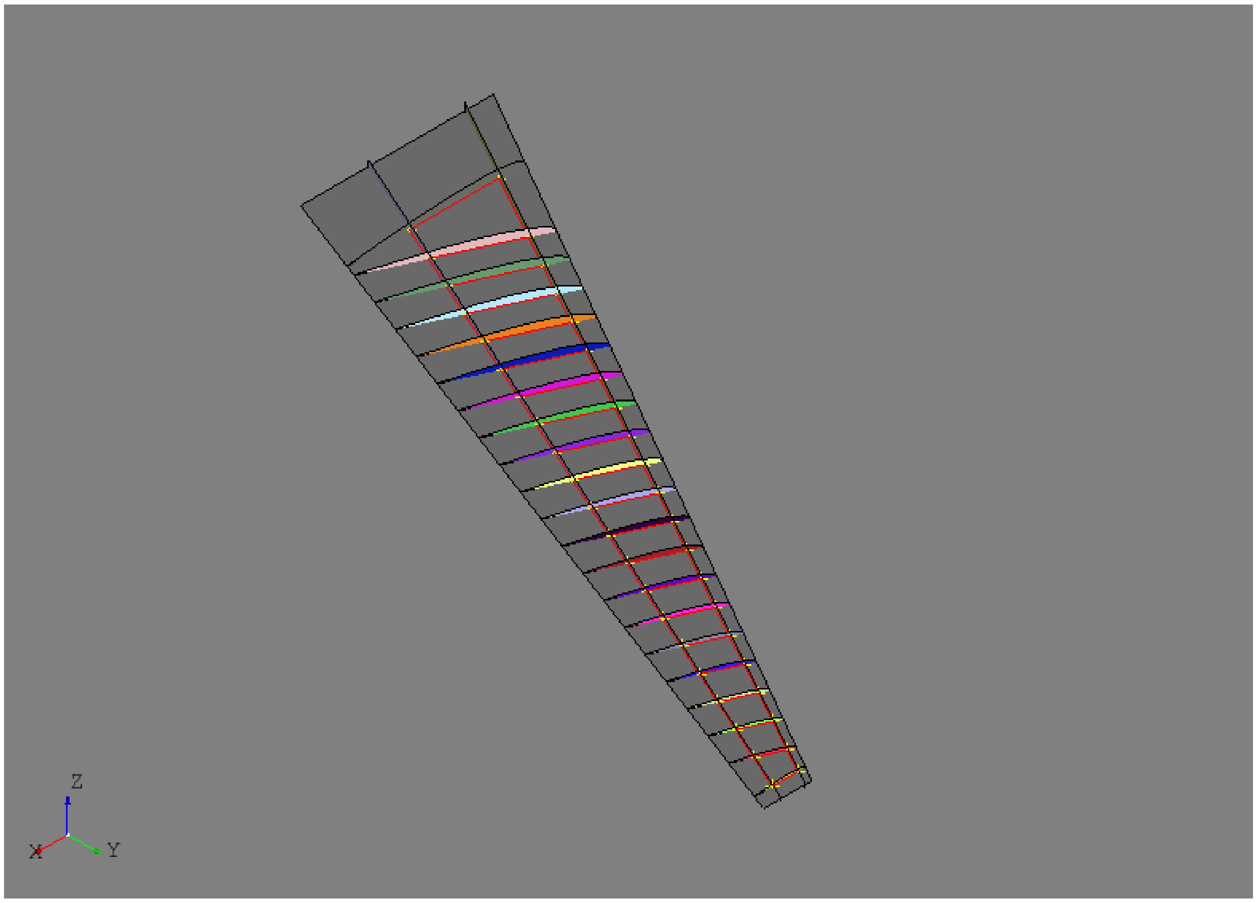
```
root = RibByPoints('root', fspar.cref.p1, rspar.cref.p1, wing).part
tip = RibByPoints('tip', fspar.cref.p2, rspar.cref.p2, wing).part
```

The root and tip ribs are shown below along with their reference geometry.



Generating multiple intermediate ribs between the root and tip will be accomplished using the *RibsAlongCurveByDistance* (page 285) tool. This tool uses a curve and a spacing to distribute planes along the curve that become the basis shapes for the ribs. The parameters `shape1` and `shape2` define the start and end points for the rib reference curves and `d1` and `d2` control initial offset spacing at the first and last points of the curve. Additionally, a *reference plane* can be supplied using the `ref_pln` parameter which will define the plane orientation at each point along the curve. If no reference plane is provided, then the first derivative of the curve will be used to define the normal orientation of the plane.

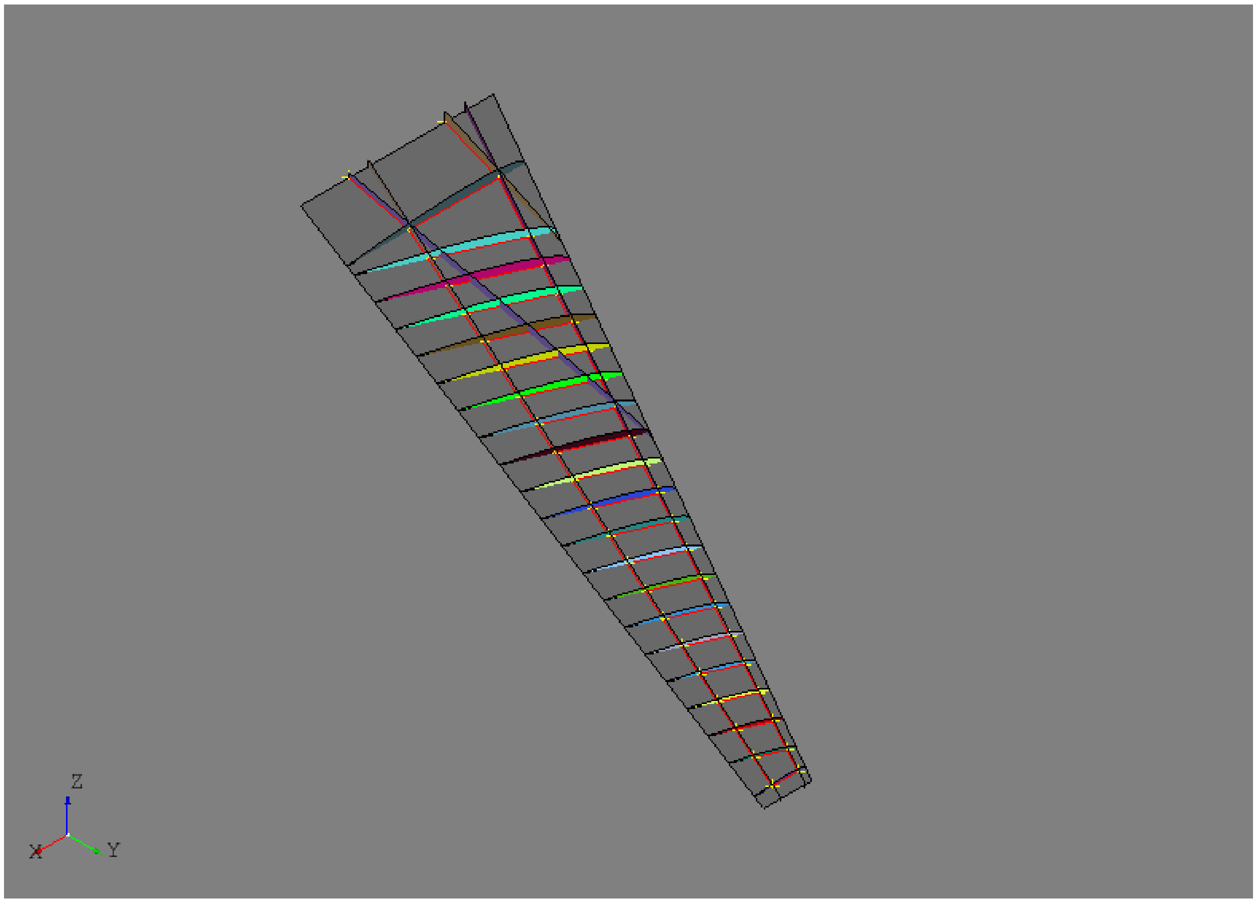
```
ribs = RibsAlongCurveByDistance('rib', rspar.cref, 30., fspar, rspar, wing,
                                d1=30., d2=-30.).parts
```



Since the rear spar reference curve was used without providing a reference plane, this tool makes the ribs perpendicular to the rear spar. Remember that all the images to this point show the initial part shapes without any joining or trimming. The images may begin to look cluttered and strange, but it will become clear in later steps how to achieve the desired result.

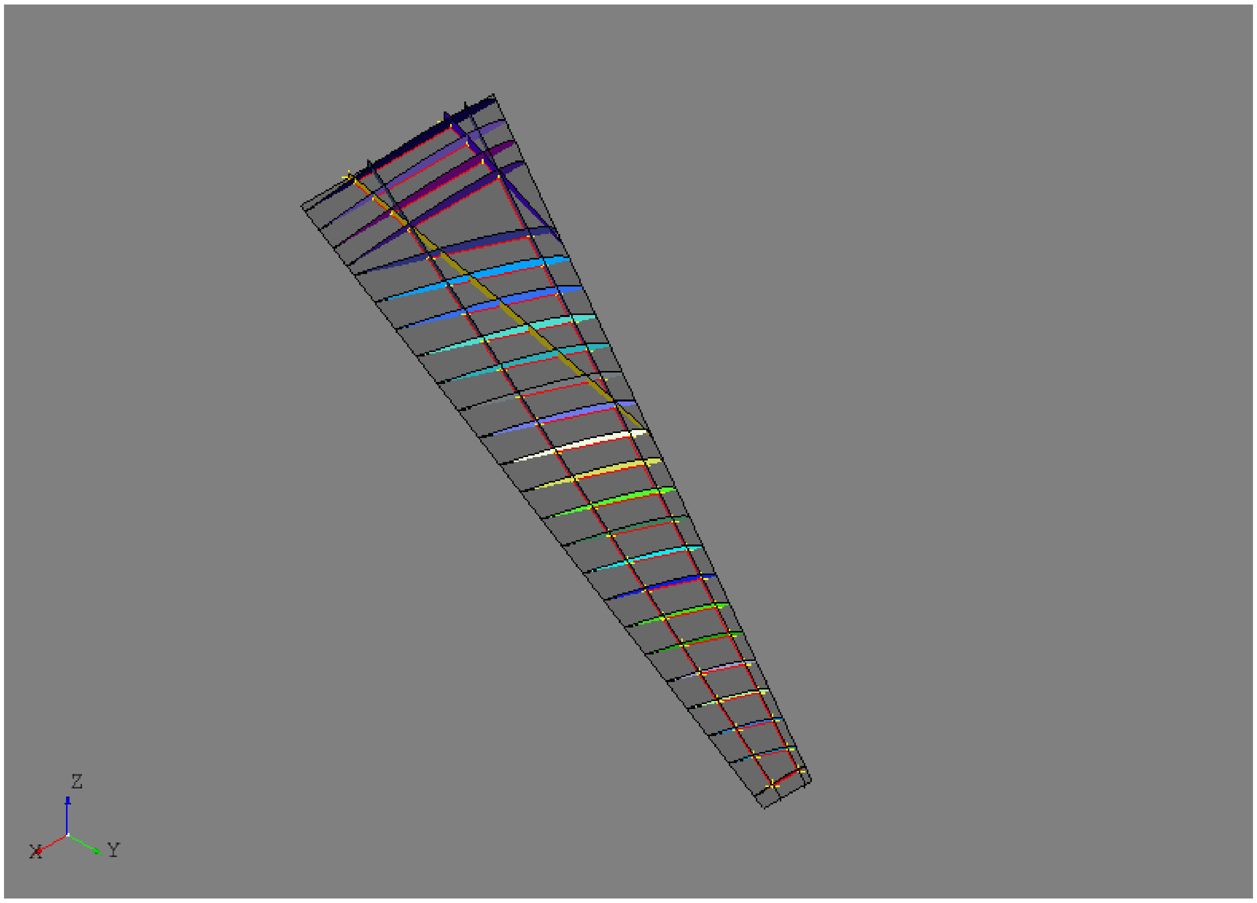
Simple center wing box structure will be modeled that includes a front and rear center spar as well as some center ribs oriented in the global xz-plane. This highlights an example where some additional thought and modeling is required before the automated tools can be used. The end point of the front center spar will interface with both the front spar and the root rib and therefore should be oriented such that is intersects cleanly. If the automated tools are used without providing a basis shape, it is likely the the automated orientation will not line up properly with the front spar and root rib intersection. To resolve this, the intersection between the front spar and root rib will be used to define the basis shape of the front center spar. The tool *PlaneByIntersectingShapes* (page 137) will return a plane based on the intersection of the two shapes if the edges of the resulting intersection are planar. To fully define the plane, a third reference point is needed that is not collinear with the intersection and is usually the other point of the structure being defined. The same process is done for the rear center spar and the results are shown below.

```
# Front center spar
p1 = wing.sref.eval(0.25, .0)
pln = PlaneByIntersectingShapes(fspar.shape, root.shape, p1).plane
fcspar = SparByPoints('fcspar', p1, root.cref.p1, wing, pln).part

# Rear center spar
p1 = wing.sref.eval(0.75, .0)
pln = PlaneByIntersectingShapes(rspar.shape, root.shape, p1).plane
rcspar = SparByPoints('rcspar', p1, root.cref.p2, wing, pln).part
```

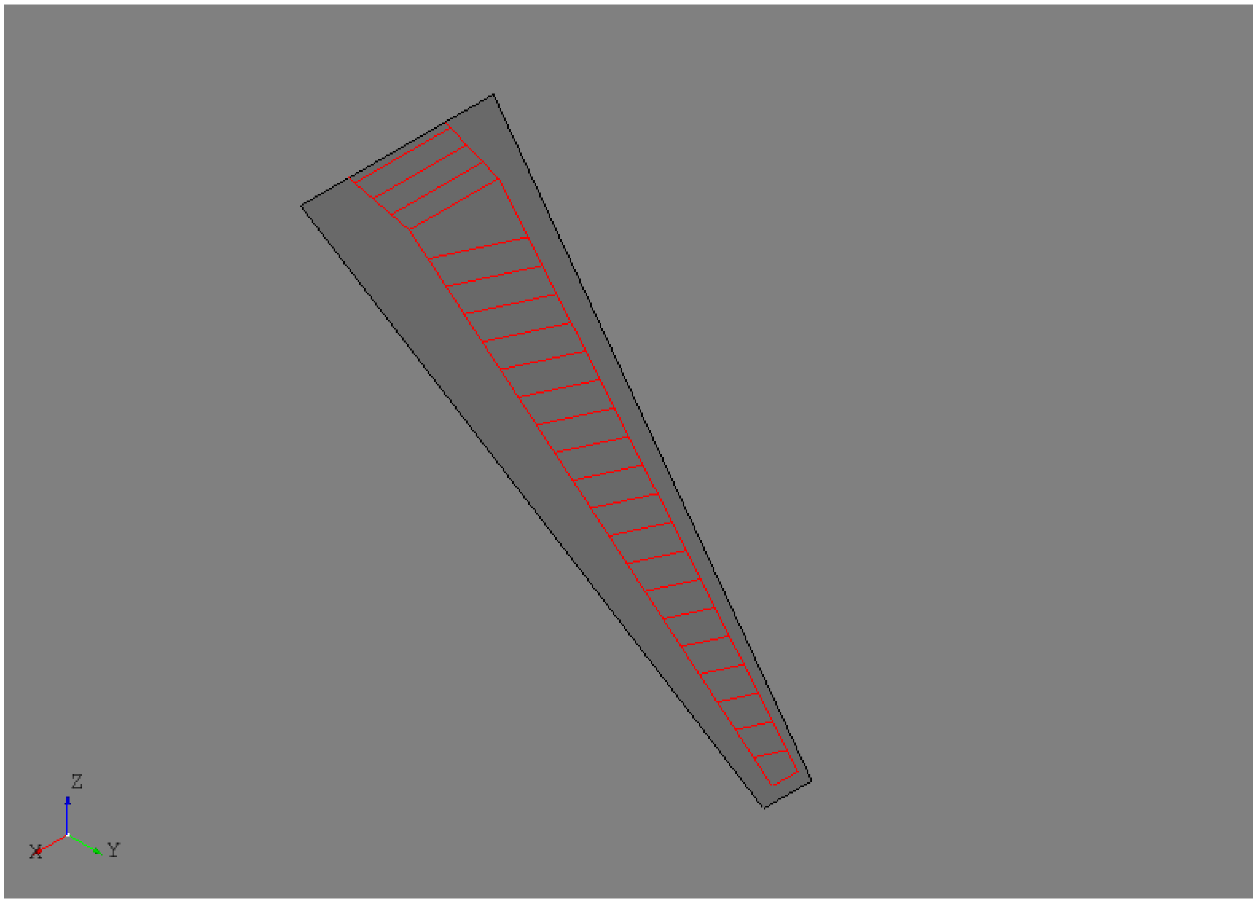


Center wing box ribs are added using the :class‘.RibsAlongCurveByNumber‘ tool and this time a basis shape (the xz-plane) will be supplied.

```
ref_pln = PlaneByAxes(origin=(0., 0., 0.), axes='xz').plane
ribs = RibsAlongCurveByNumber('center rib', rcspar.cref, 3, fcspar, rcspar,
                              wing, ref_pln, d1=6, d2=-18).parts
```



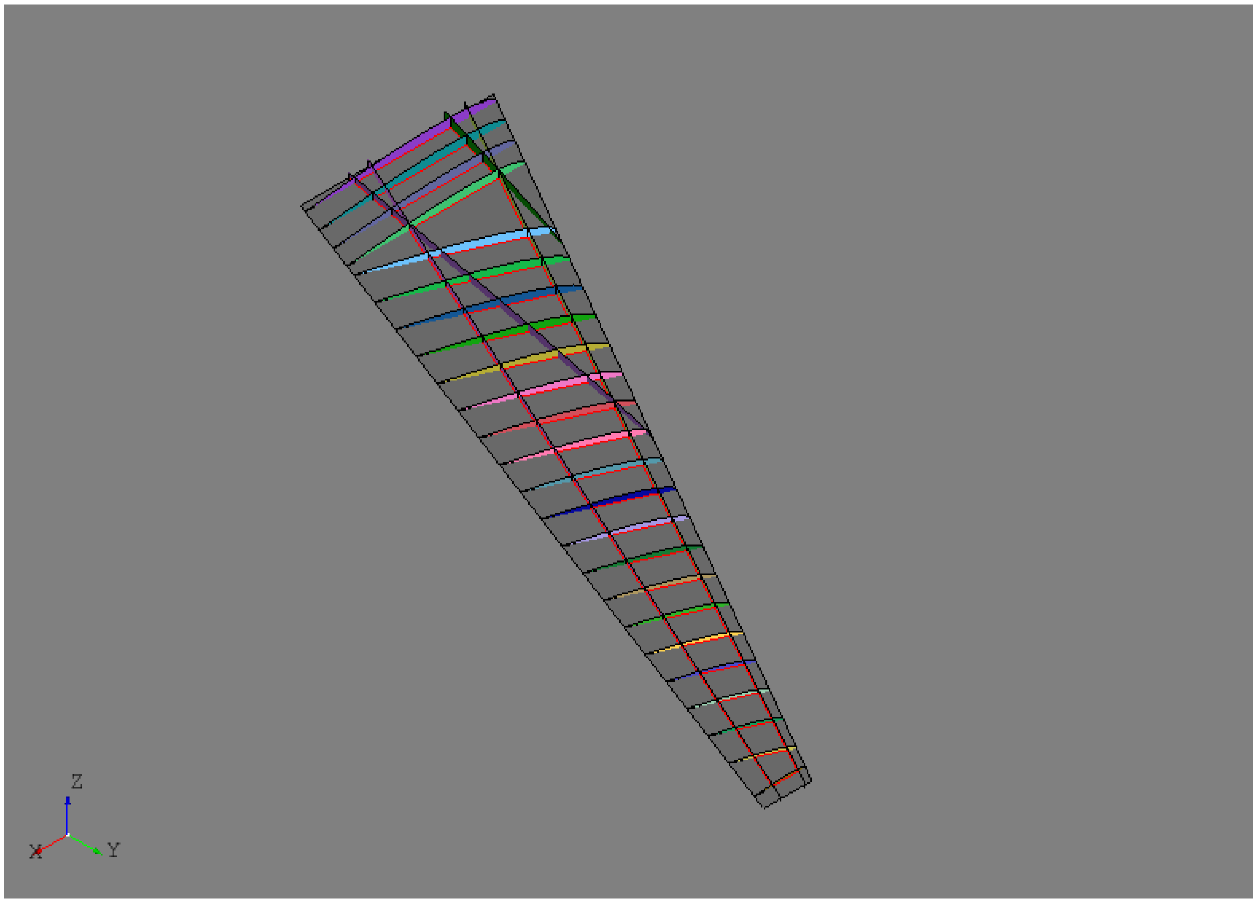
At this point it is helpful to plot only the wing reference surface and the reference curves of each of the parts defined thus far. This will be a helpful representation of how the structure will generally look at the end of the process.



Joining the internal structure can now be done using a specialized `FuseSurfacePartsByCref` (page 291) tool. This tool will take a sequence of `SurfacePart` (page 257) entities and join them only if their reference curves intersect. With this tool, even though the initial shapes may visually intersect in undesired regions, the reference curves are used as a way to intelligently select which parts should actually be topologically joined with one another.

```
internal_parts = wingbox.get_parts()
FuseSurfacePartsByCref(internal_parts)
```

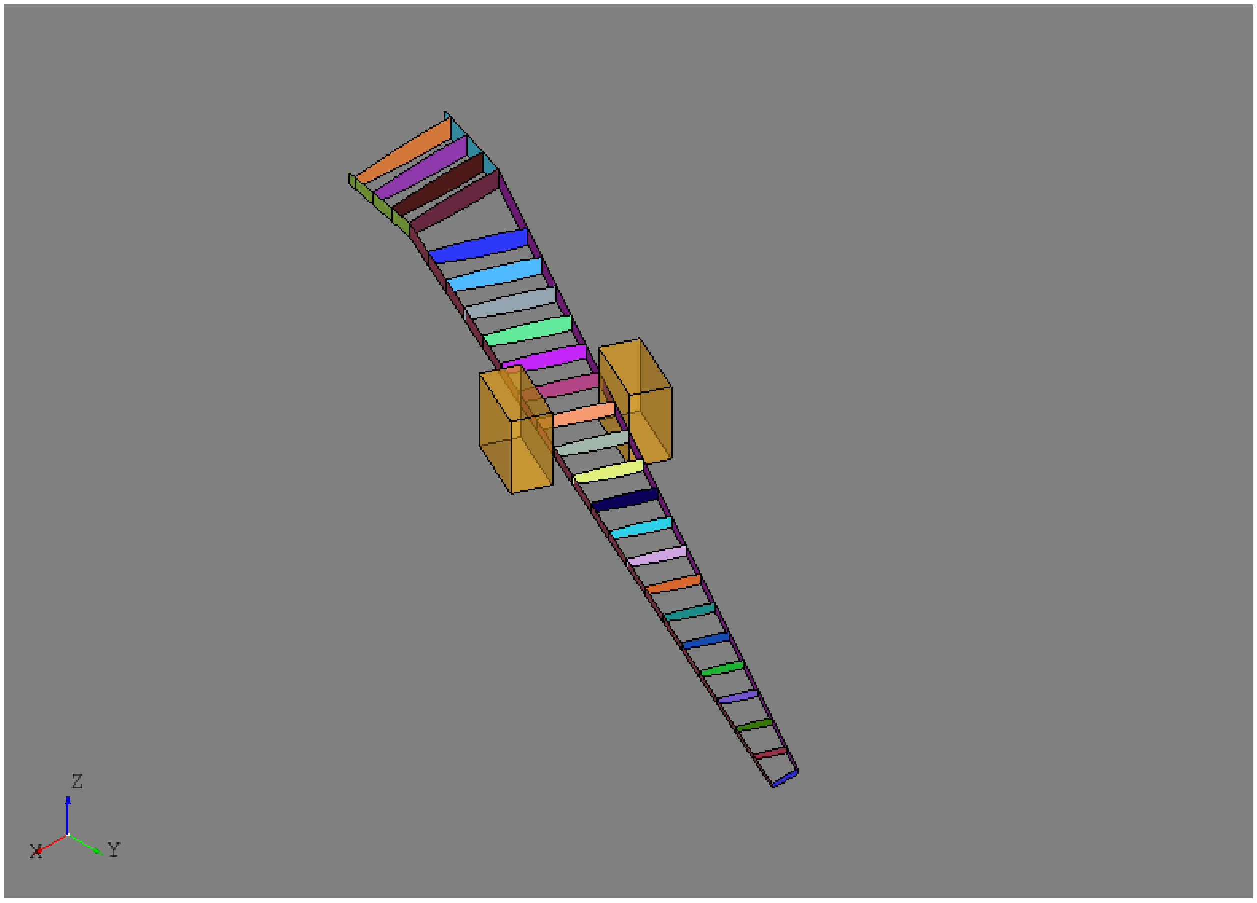
It may not be obvious in the image below, but after the joining operation new edges only exist between the parts that had intersecting reference curves.



After the joining operation the initial shapes have been updated with new shapes that are now trimmed with the other parts and should share congruent edges. Although, there are still some regions of some of the parts that must be discarded in this example to get the desired result. In an interactive GUI, this would be accomplished by a simple point-and-click delete operation. Since AFEM is most likely operating in an automated and scripted environment, other means of replicating the point-and-click action must be developed. In this example the `DiscardByCref` (page 294) tool is used:

```
DiscardByCref(internal_parts)
```

For each part in the input, this will try and determine which sub-shapes should be discarded (i.e., deleted) based on their relationship to the reference curve of the part. At each end of the part reference curve, the first derivatives of the curve are used to define normal directions for two planes located at the start and end points. These planes and their normal direction attempt to define the "outside" region of the part. For each relevant sub-shape, which will be the edges of a `CurvePart` (page 257) or faces of a `SurfacePart` (page 257), the centroid of the shape will be classified as either "inside" or "outside" the part based on its location relative to the two planes. If the centroid is classified as outside then the sub-shape is removed and the part shape is updated. This method and tool are based on some assumptions about the general part shape and it having a reference curve. There are other discarding tools available for general or complex cases. The result after discarding faces of the internal structure based on their reference curves is shown below along with one of the "outside" regions for one of the ribs. Note that the regions are drawn as finite boxes, but they are actually infinite boxes in the direction away from the part reference curve.



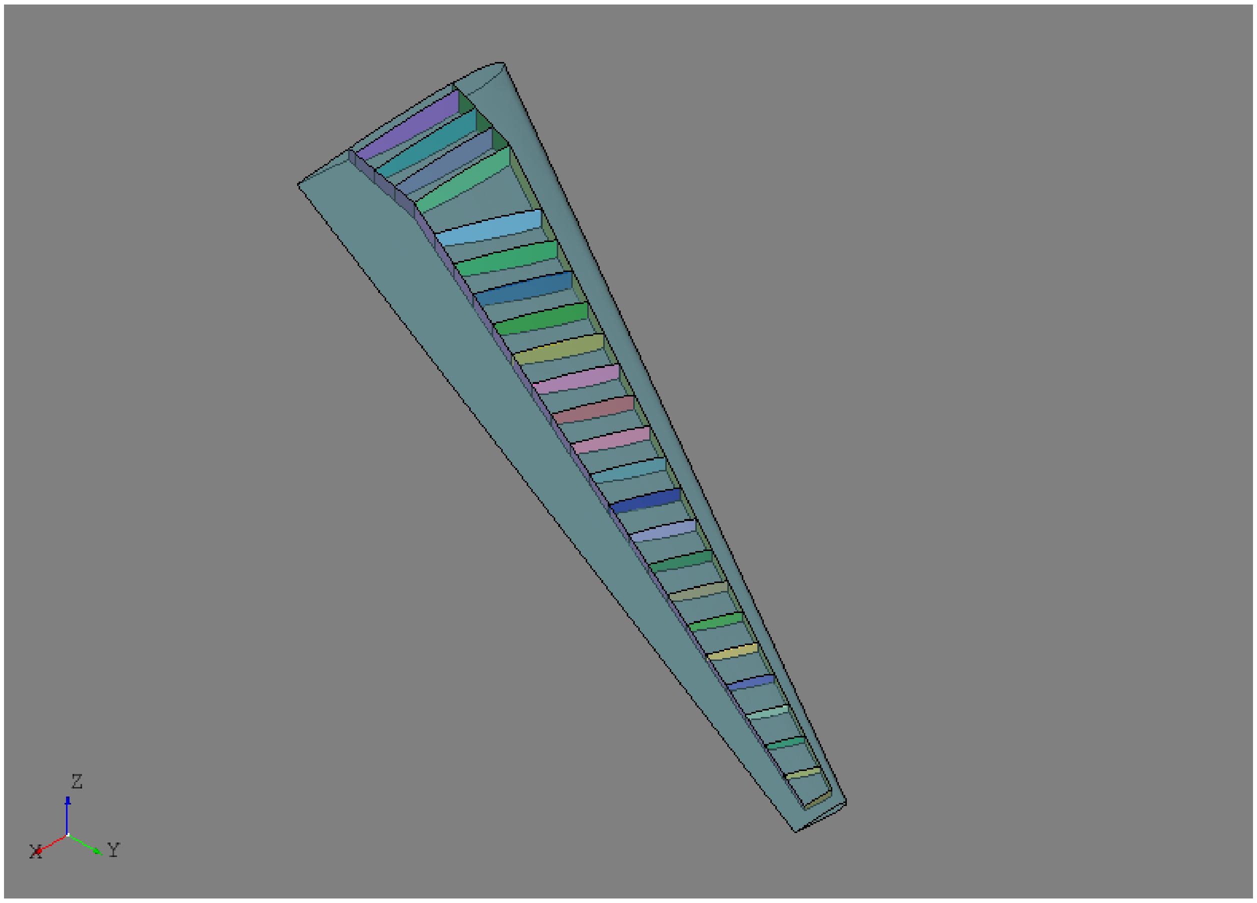
After the internal structure is defined and joined, the next step is usually initializing any external skins and joining them with the internal structure. The `SkinByBody` (page 290) extract the outer shell from a solid and uses that as the shape for a `Skin` (page 260) entity:

```
skin = SkinByBody('skin', wing).part
```

The *skin.fuse(other_parts)* will join the input parts with the skin but it does not check or join the input parts with one another. This is the desired behavior since they have already been joined:

```
skin.fuse(*internal_parts)
```

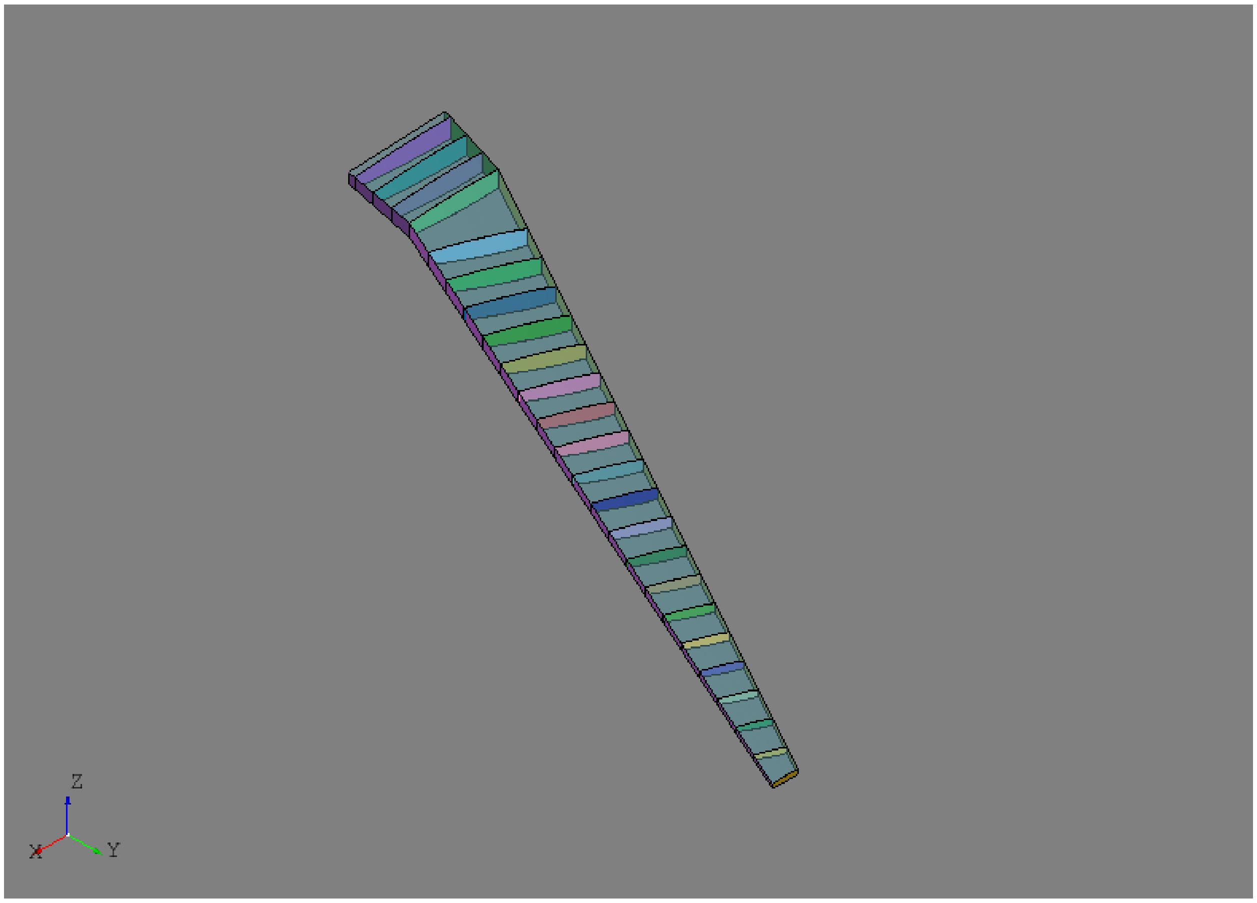
Joining the external skin with all the internal structure at once improves the chances of obtaining congruent edges between all the interfacing structure. It is generally recommended that joining of external skins to internal structure is saved for later in the geometric modeling process. The skin and internal structure are shown below.



For this example, the wing skin regions outside the primary load-carrying area of the wing box are to be excluded:

```
skin.discard_by_dmin(wing.sref, 1.)
```

This method will check each sub-shape of the part for its distance to some reference entity, in this case the wing reference surface. If the distance is below some threshold then the sub-shape is discarded from the part. For this model, any skin region that touches the wing reference surface is either along the leading or trailing edge and outside the primary load-carrying area. The resulting wing box is shown below.

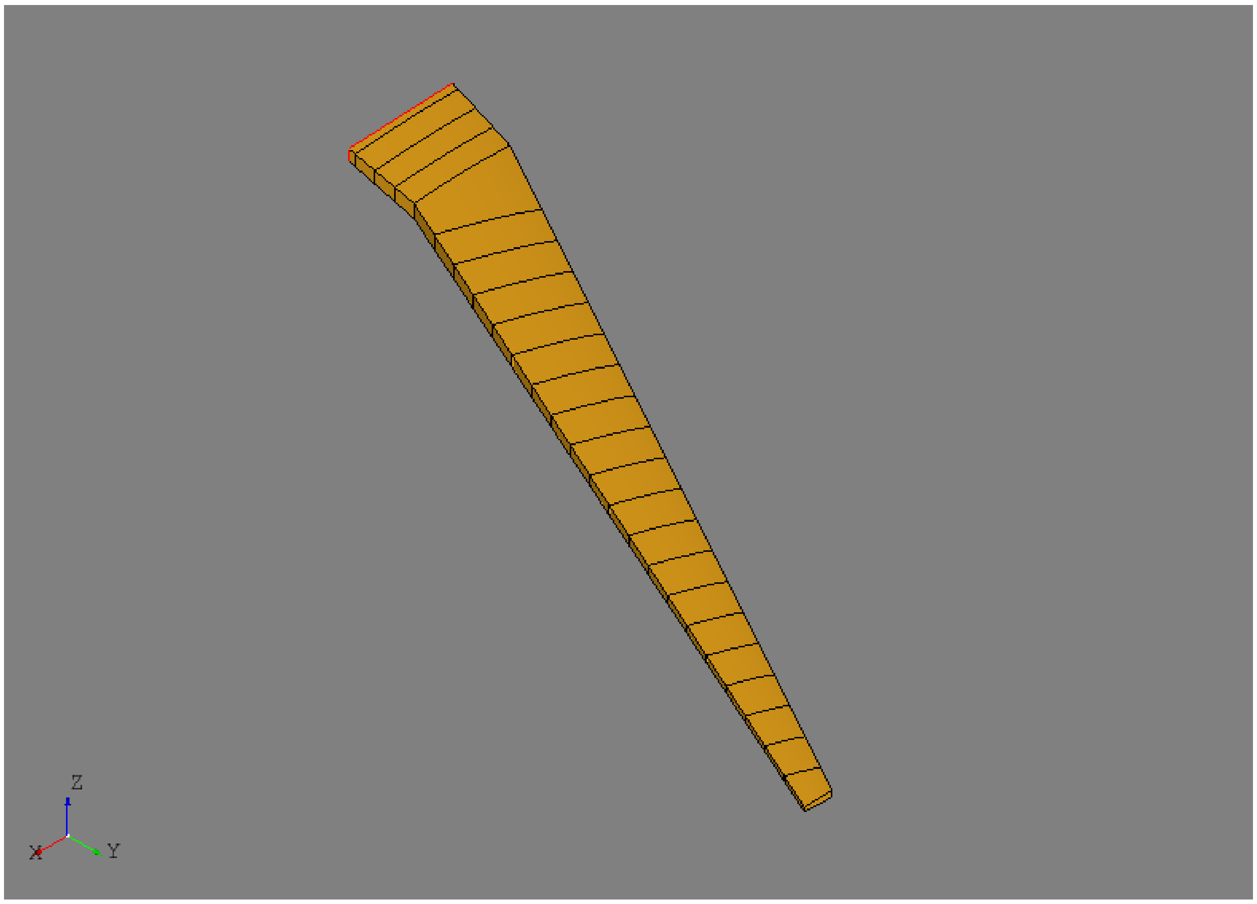


In the example, the `skin.fix()` method is used to check the shape of the skin. This method is a general shape checking and fixing tool that can catch and repair errors that may arise during the modeling process. In this case, the skin was originally a single *Shell* (page 117), but after joining and discarding become two disconnected shapes, one upper and one lower. To remedy this, the `skin.fix()` repaired the shape by changing its type to a :class‘.Compound‘ made up of two *Shell* (page 117) entities.

At this point the geometric modeling process is complete and the following code will check the shape for any free edges which may indicate an error. The `shape` variable below is a single top-level shape put together from all of the parts.

```
shape = GroupAPI.get_shape()
tool = ExploreFreeEdges(shape)
```

The free edges of `tool` are shown below with the checked geometry. Note that the only free edges (shown in red) found were along the root of the model which is expected since now structure was modeled there.



The next step is the process is usually the generation of Finite Element Model (FEM) data. For this task, AFEM focuses on providing the needed level of access to the various shapes of the model to conveniently apply meshing controls. Rules and best practices for airframe meshing are typically dependent on the application (e.g., loads vs. deflection model) and/or the user's own experience and heuristics. Developing fully-automated, widely applicable, and robust rules for meshing was deemed impractical and outside the scope of this project. Instead, effort was directed towards developing streamlined tools to enable the user to efficiently define their own controls. The low-level meshing controls are best described in the `afem.smesh` documentation.

For this example, the *`MeshVehicle`* (page 296) tool provides a convenient API to define and apply both global and local meshing controls to the desired regions. The *`MeshVehicle`* (page 296) tool is not a one-size-fits all tool, but is more an example of a meshing utility that could be developed for more advanced use cases. Although, it does provide a wide range of functionality all in one place:

```
mesh = MeshVehicle(4.)
```

Upon initialization, this tool will define a top-level global shape derived from the *master* group from the *`GroupAPI`* (page 262) tool. The `target_size` parameter is used to define a global quad-dominated meshing algorithm that will be applied to the entire shape absent of any other local mesh controls.

This simple example is actually suited for more of a structured meshing approach with the right local edge meshing controls. The code below gathers parts of specific types together and then the *`ExploreParts`* (page 294) tool is used to retrieve a *`Compound`* (page 119) that contains only the shared edges. Here, the edges shared by both the spars and skin will have a maximum edge length of four.

```
spars = wingbox.get_parts(rtype=Spar)
skins = wingbox.get_parts(rtype=Skin)
```

(continues on next page)

(continued from previous page)

```
shape = ExploreParts.shared_edges(spars, skins, True)
mesh.set_max_length_1d(4., shape)
```

In similar fashion, the shared edges between the ribs and spars will have four elements along their length.

```
ribs = wingbox.get_parts(rtype=Rib)
shape = ExploreParts.shared_edges(spars, ribs, True)
mesh.set_number_segments_1d(4, shape)
```

Next, the edges shared between the ribs and the skin will have fifteen elements along their length.

```
shape = ExploreParts.shared_edges(skins, ribs, True)
mesh.set_number_segments_1d(15, shape)
```

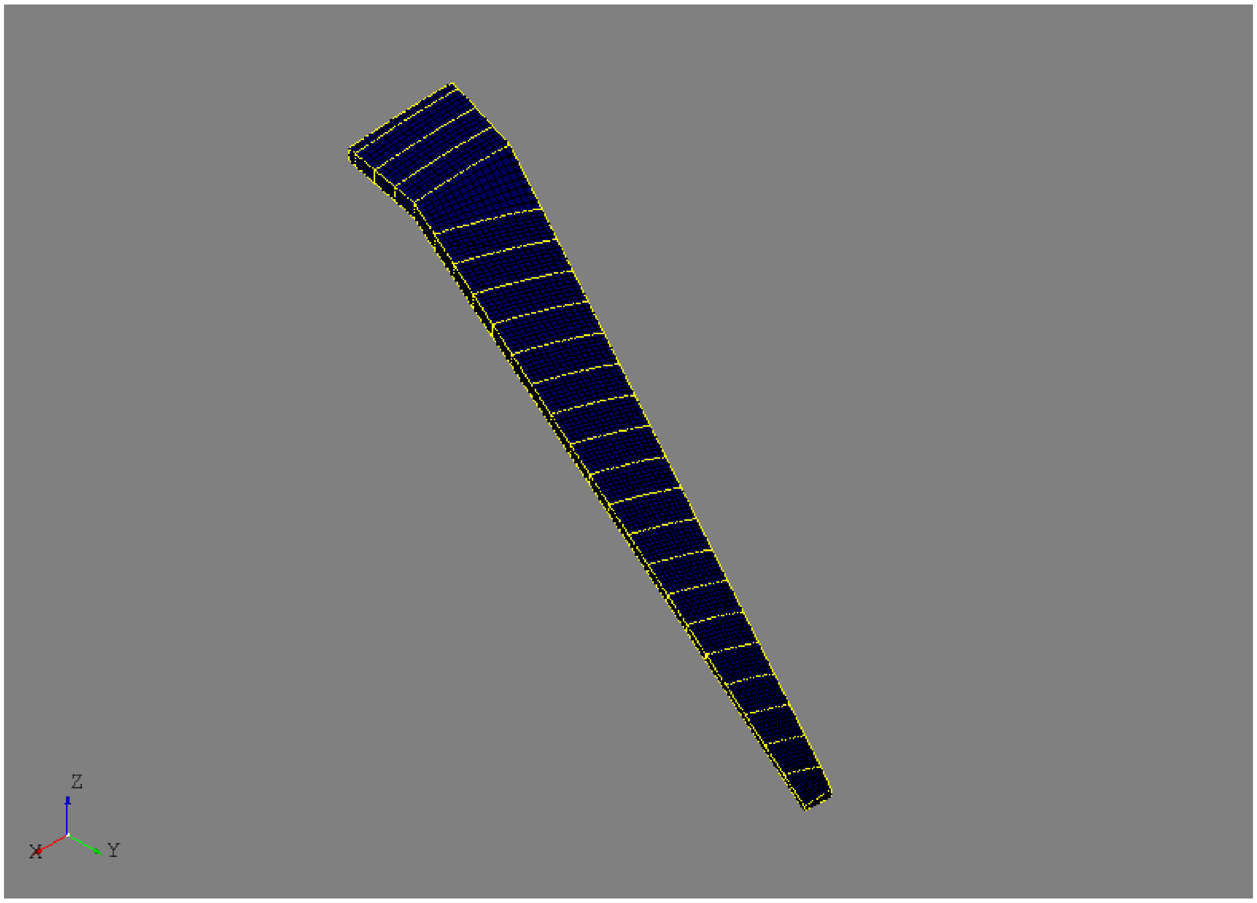
Since these controls provide a fairly structured 1-D meshing pattern, a structured quadrilateral algorithm will be applied to all the applicable faces of the model.

```
for part in wingbox.get_parts():
    mesh.set_quadrangle_2d(part.shape)
```

Finally, the mesh can be computed:

```
mesh.compute()
```

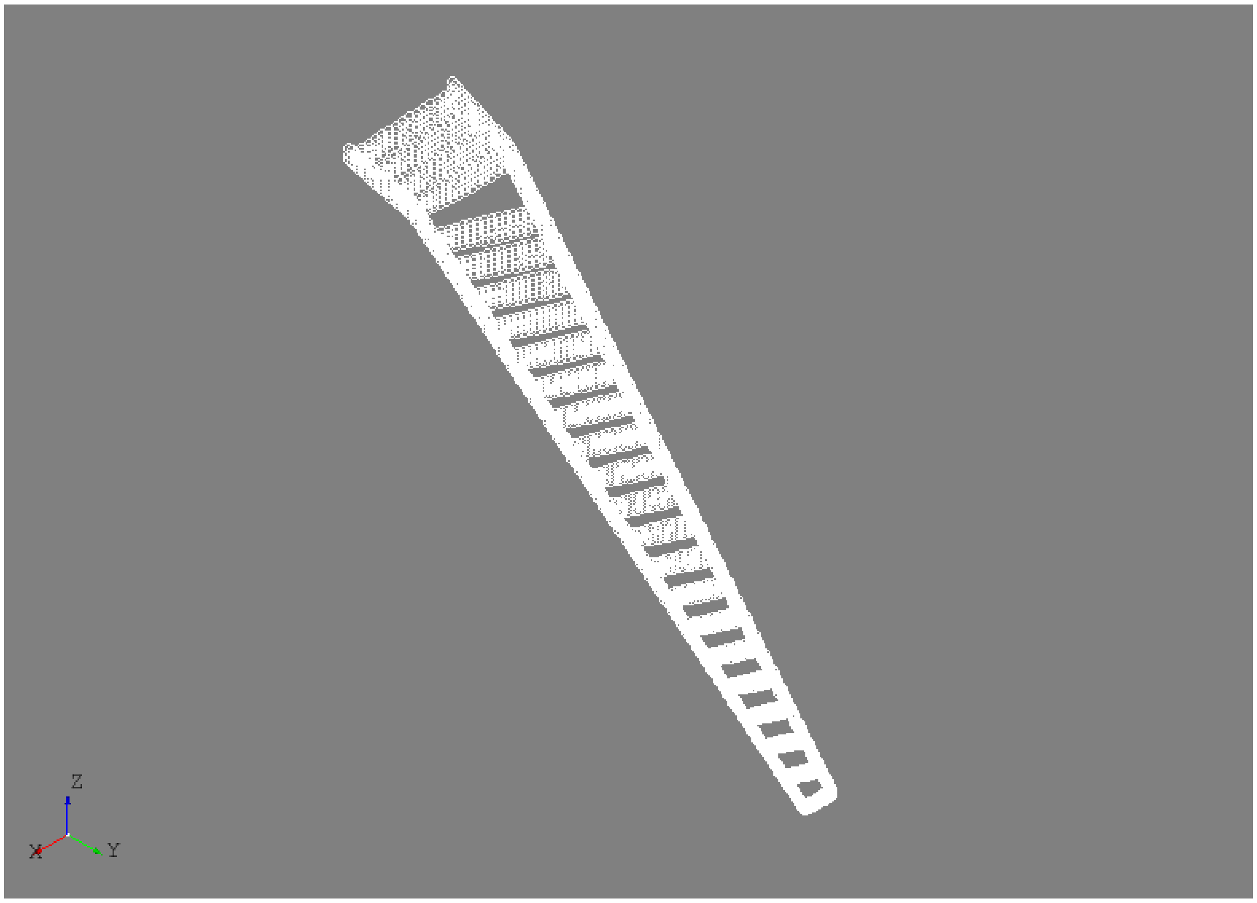
The nearly all structured quadrilateral mesh is shown below.



A convenient way to work with the mesh data is to create mesh groups for different mesh types (e.g., node, edge, or face). Mesh groups can be created before the mesh is computed and are derived and associated to a shape. Node groups for the spars and ribs can be defined by:

```
spar_nodes = mesh.create_node_group(spars_assy.get_shape())
rib_nodes = mesh.create_node_group(ribs_assy.get_shape())
```

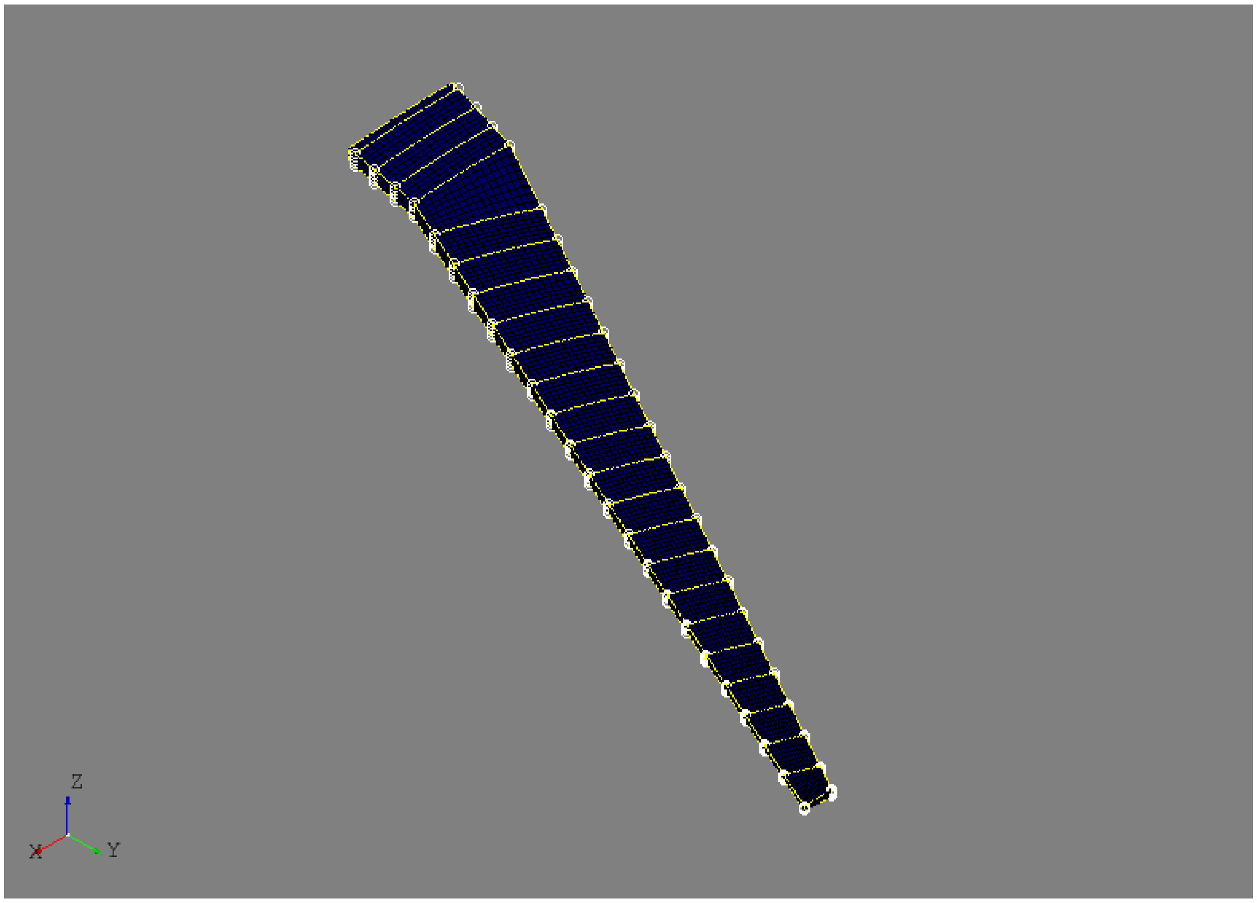
The nodes in these groups are shown below.



Useful information can be derived from these mesh groups like the shared nodes between two different groups. In this example, the shared nodes between the spars and ribs can be obtained by an intersection algorithm:

```
shared_nodes = spar_nodes.intersect(rib_nodes, 'spar and rib nodes')
```

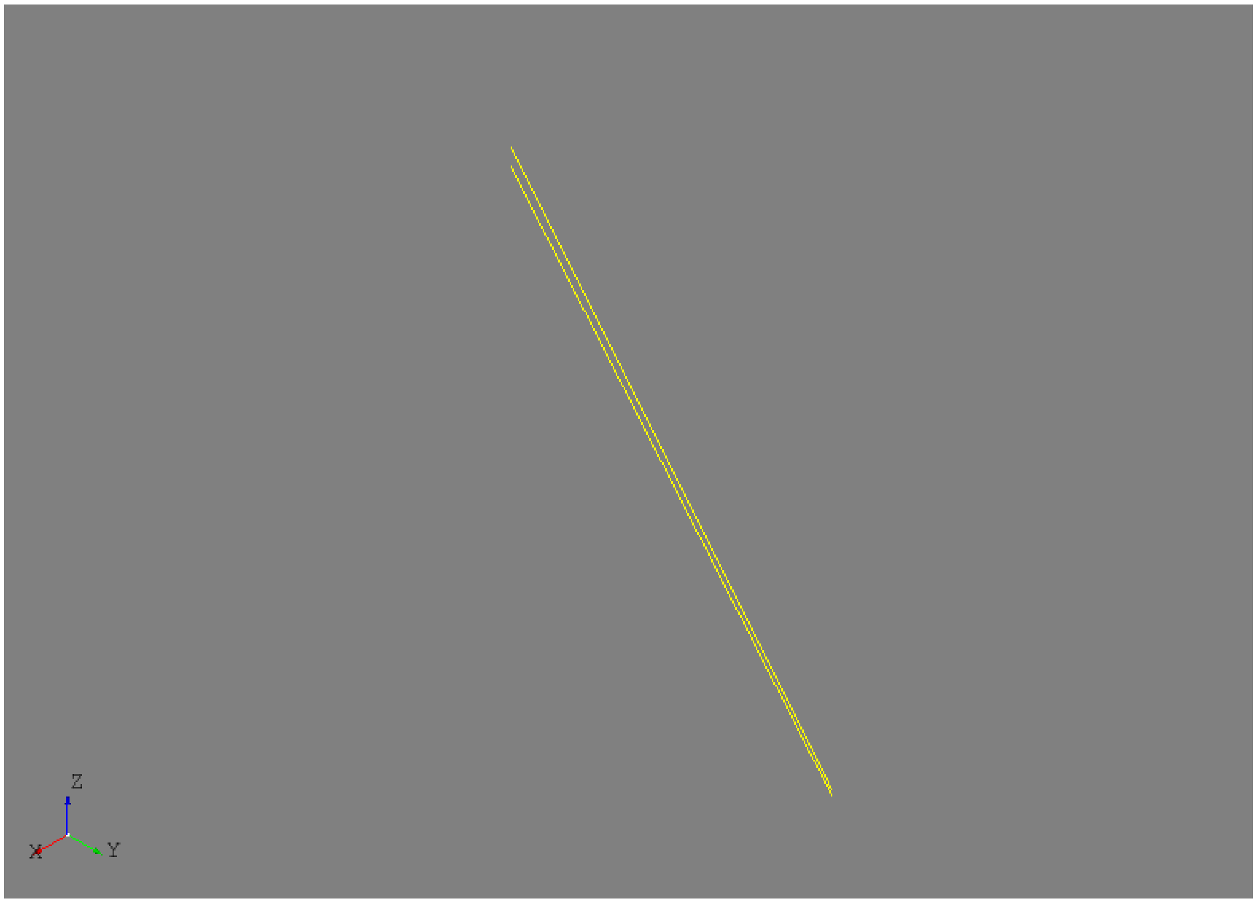
The shared nodes are shown below. The same could be done for mesh edge groups created from the same parts.



Mesh edge groups can created from the front spar and the wing skin and shared edges are calculated by:

```
fspar_edges = mesh.create_edge_group(fspar.shape)
skin_edges = mesh.create_edge_group(skin.shape)
shared_edges = skin_edges.intersect(fspar_edges)
```

This could be one way of extract spar cap elements from the mesh and assigning FEM properties in downstream properties. The shared mesh edges are shown below.



As a last example, the experimental Nastran export tool can be accessed from the *MeshVehicle* (page 296) tool to output the entire mesh to a Nastran bulk data file:

```
mesh.export_nastran('structure_basic.bdf')
```

To date, no materials or element properties are include in AFEM so this file only contains the nodes and elements. Although, this could be part of future development.

13.1 Entities

13.1.1 Part

class afem.structure.entities.**Part**(*name*, *shape*, *cref=None*, *sref=None*, *group=None*)

Bases: *afem.core.entities.ShapeHolder* (page 13)

Base class for all parts.

Parameters

- **name** (*str*) – The name.
- **shape** (afem.topology.entities.Shape (page 109)) – The shape.
- **cref** (afem.geometry.entities.Curve (page 43) *or None*) – The reference curve. If it is not a *TrimmedCurve* (page 48), then it will be converted to one.

- **sref** (`afem.geometry.entities.Surface` (page 49) *or None*) – The reference surface.
- **group** (*str or* `afem.structure.group.Group` (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

id

Returns The unique part ID.

Return type int

node_group

Returns The mesh node group.

Return type *afem.smesh.entities.MeshGroup* (page 202)

Raises AttributeError – If group doesn't exist.

edge_group

Returns The mesh edge group.

Return type *afem.smesh.entities.MeshGroup* (page 202)

Raises AttributeError – If group doesn't exist.

face_group

Returns The mesh face group.

Return type *afem.smesh.entities.MeshGroup* (page 202)

Raises AttributeError – If group doesn't exist.

distance(*other*)

Find the minimum distance between the part and other shape.

Parameters other (`afem.topology.entities.Shape` (page 109) *or* `afem.structure.entities.Part` (page 253)) – Other part or shape.

Returns The minimum distance.

Return type float

check(*raise_error=True*)

Check the shape of the part.

Parameters raise_error (*bool*) – Option to raise an error if the shape is not valid.

Returns *True* if shape is valid, *False* if not.

Return type bool

Raises RuntimeError – If the check fails and *raise_error* is `True`.

fix(*precision=None*, *min_tol=None*, *max_tol=None*, *context=None*, *include_subgroup=True*)

Attempt to fix the shape of the part using *FixShape* (page 164).

Parameters

- **precision** (*float*) – Basic precision value.
- **min_tol** (*float*) – Minimum tolerance.
- **max_tol** (*float*) – Maximum tolerance.

- **`context`** (`afem.topology.entities.Shape` (page 109) *or* `afem.structure.entities.Group` *or* `str`) – The context shape or group.
- **`include_subgroup`** (`bool`) – Option to recursively include parts from any subgroups.

Returns None.

`cut`(*cutter*)

Cut the part shape and rebuild this part.

Parameters **`cutter`** (`afem.topology.entities.Shape` (page 109) *or* `afem.structure.entities.Part` (page 253) *or* `afem.geometry.entities.Geometry` (page 34)) – The cutter. If geometry is provided it will be converted to a shape before the Boolean operation.

Returns *True* if shape was cut, *False* if not.

Return type bool

`split`(*splitter*, *rebuild_both=True*)

Split the part shape and rebuild this part. Optionally rebuild the splitter if it is a part. This method should handle splitting with parts and shapes or different types (i.e., splitting a face with an edge).

Parameters

- **`splitter`** (`afem.topology.entities.Shape` (page 109) *or* `afem.structure.entities.Part` (page 253)) – The splitter.
- **`rebuild_both`** (`bool`) – Option to rebuild both if *splitter* is a part.

Returns *True* if split, *False* if not.

Return type bool

Raises **`TypeError`** – If this part is or the splitter not a curve or surface part.

`rebuild`(*tool*)

Rebuild the part shape with a supported tool.

Parameters **`tool`** (`afem.topology.bop.BopCore` (page 146)) – The tool.

Returns *True* if modified, *False* if not.

Return type bool

Raises **`TypeError`** – If this part is not a curve or surface part.

`discard_by_solid`(*solid*, *tol=None*)

Discard shapes of the part using a solid. Any shapes of the part that have centroids inside the solid will be removed. Edges are checked for curve parts and faces are checked for surface parts.

Parameters

- **`solid`** (`afem.topology.entities.Solid` (page 118)) – The solid.
- **`tol`** (`float`) – The tolerance. If not provided then the part tolerance will be used.

Returns *True* if shapes were discarded, *False* if not.

Return type bool

Raises **`TypeError`** – If this part is not a curve or surface part.

`discard_by_dmax`(*entity*, *dmax*)

Discard shapes of the part using a shape and a distance. If the distance between a shape of the part and the given shape is greater than *dmax*, then the shape is removed. Edges are checked for curve parts and faces are checked for surface parts.

Parameters

- **entity** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Geometry (page 34)) – The shape.
- **dmax** (*float*) – The maximum distance.

Returns *True* if shapes were discarded, *False* if not.

Return type bool

Raises **TypeError** – If this part is not a curve or surface part.

discard_by_dmin(*entity*, *dmin*)

Discard shapes of the part using a shape and a distance. If the distance between a shape of the part and the given shape is less than *dmin*, then the shape is removed. Edges are checked for curve parts and faces are checked for surface parts.

Parameters

- **entity** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Geometry (page 34)) – The shape.
- **dmin** (*float*) – The minimum distance.

Returns *True* if shapes were discarded, *False* if not.

Return type bool

Raises **TypeError** – If this part is not a curve or surface part.

discard_by_cref(*size=None*)

Discard shapes of the part by using the reference curve. An infinite solid is created at each end of the reference curve using the curve tangent. Any shape that has a centroid in these solids is removed. For a curve part edges are discarded, for a SurfacePart faces are discarded.

Parameters **size** (*float*) – Option to define a finite solid box which might be more robust than an infinite solid.

Returns *True* if shapes were discard, *False* if not.

Return type bool

shared_vertices(*other*, *as_compound=False*)

Get vertices shared between the two parts.

Parameters

- **other** (afem.structure.entities.Part (page 253) *or* afem.topology.entities.Shape (page 109)) – The other part or shape.
- **as_compound** (*bool*) – Option to return the shared vertices in a compound.

Returns Shared vertices.

Return type list(*afem.topology.entities.Vertex* (page 115)) or *afem.topology.entities.Compound* (page 119)

shared_edges(*other*, *as_compound=False*)

Get edges shared between the two parts.

Parameters

- **other** (afem.structure.entities.Part (page 253) *or* afem.topology.entities.Shape (page 109)) – The other part or shape.
- **as_compound** (*bool*) – Option to return the shared edges in a compound.

Returns Shared edges.

Return type list(*afem.topology.entities.Edge* (page 115)) or *afem.topology.entities.Compound* (page 119)

init_meshing(*mesh*)
Initialize the part for meshing. This includes creating node, edge, and face groups.

Parameters **mesh** (`afem.smesh.entities.Mesh` (page 193)) – The top-level mesh that will contain the groups.

Returns None.

classmethod reset()
Reset the part index counter back to 1 and the mesh to None.

Returns None.

13.1.2 CurvePart

class afem.structure.entities.**CurvePart**(*name*, *shape*, *cref=None*, *sref=None*, *group=None*)
Bases: *afem.structure.entities.Part* (page 253)

Base class for curve parts.

length

Returns The length of all the edges of the part.

Return type float

13.1.3 Beam1D

class afem.structure.entities.**Beam1D**(*name*, *shape*, *cref=None*, *sref=None*, *group=None*)
Bases: *afem.structure.entities.CurvePart* (page 257)

Beam 1-D.

13.1.4 SurfacePart

class afem.structure.entities.**SurfacePart**(*name*, *shape*, *cref=None*, *sref=None*, *group=None*)
Bases: *afem.structure.entities.Part* (page 253)

Base class for surface parts.

length

Returns The length of the reference curve if available. Otherwise it returns the length of all edges of the part.

Return type float

area

Returns The area of all faces of the part.

Return type float

fuse (**other_parts*)
Fuse with other surface parts and rebuild both.

Parameters **other_parts** (afem.structure.entities.SurfacePart (page 257)) – The other part(s).

Returns *True* if fused, *False* if not.

Return type bool

sew (**other_parts*)
Sew with other parts and rebuild all parts.

Parameters **other_parts** (afem.structure.entities.SurfacePart (page 257)) – The other part(s).

Returns *True* if sewed, *False* if not.

Return type bool

merge (*other*, *unify=False*)
Merge other surface part or shape with this one.

Parameters

- **other** (afem.structure.entities.SurfacePart (page 257) *or* afem.topology.entities.Shape (page 109)) – The other part or shape.
- **unify** (*bool*) – Option to attempt to unify same domains.

Returns *True* if merged, *False* if not.

Return type bool

unify (*edges=True*, *faces=True*, *bsplines=False*)
Attempt to unify the same domains of the part shape.

Parameters

- **edges** (*bool*) – Option to unify all possible edges.
- **faces** (*bool*) – Option to unify all possible faces.
- **bsplines** (*bool*) – Option to concatenate the curves of edges if they are C1 continuous.

Returns *True* if unified, *False* if not.

Return type bool

split_local (*subshape*, *tool*)
Locally split the faces of he sub-shape in the context of the part shape.

Parameters

- **subshape** (afem.topology.entities.Shape (page 109)) – The sub-shape.
- **tool** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Surface (page 49)) – The tool to split the sub-shape with.

Returns *True* if split, *False* if not.

Return type bool

cut_hole (*d*, *ds*, *u0=None*, *is_rel=False*)
Cut a hole in the part and update its shape (experimental).

Parameters

- **d** (*float*) – Diameter of hole.
- **ds** (*float*) – The distance.
- **u0** (*float*) – The parameter. If not provided the first parameter of the reference curve will be used.
- **is_rel** (*bool*) – Option specifying if the distance is absolute or a relative to the length of the reference curve. If relative, then *ds* is multiplied by the curve length to get the absolute value.

Returns The status of the Boolean operation that will cut the hole. *True* if the operation was performed, *False* if not.

Return type bool

cut_holes (*n*, *d*)
Cut holes along the reference curve at evenly spaced intervals (experimental).

Parameters

- **n** (*int*) – The number of holes.
- **d** (*float*) – The diameter.

Returns None.

13.1.5 WingPart

class afem.structure.entities.**WingPart** (*name*, *shape*, *cref=None*, *sref=None*, *group=None*)
Bases: *afem.structure.entities.SurfacePart* (page 257)

Base class for wing parts.

13.1.6 Spar

class afem.structure.entities.**Spar** (*name*, *shape*, *cref=None*, *sref=None*, *group=None*)
Bases: *afem.structure.entities.WingPart* (page 259)

Wing spar.

13.1.7 Rib

class afem.structure.entities.**Rib** (*name*, *shape*, *cref=None*, *sref=None*, *group=None*)
Bases: *afem.structure.entities.WingPart* (page 259)

Wing rib.

13.1.8 FuselagePart

class afem.structure.entities.**FuselagePart** (*name*, *shape*, *cref=None*, *sref=None*, *group=None*)
Bases: *afem.structure.entities.SurfacePart* (page 257)

Base class for fuselage parts.

13.1.9 Bulkhead

class afem.structure.entities.**Bulkhead**(*name*, *shape*, *cref=None*, *sref=None*, *group=None*)

Bases: *afem.structure.entities.FuselagePart* (page 259)

Bulkhead.

13.1.10 Floor

class afem.structure.entities.**Floor**(*name*, *shape*, *cref=None*, *sref=None*, *group=None*)

Bases: *afem.structure.entities.FuselagePart* (page 259)

Floor.

13.1.11 Frame

class afem.structure.entities.**Frame**(*name*, *shape*, *cref=None*, *sref=None*, *group=None*)

Bases: *afem.structure.entities.FuselagePart* (page 259)

Frame.

13.1.12 Skin

class afem.structure.entities.**Skin**(*name*, *shape*, *cref=None*, *sref=None*, *group=None*)

Bases: *afem.structure.entities.SurfacePart* (page 257)

Skin.

13.1.13 Stiffener1D

class afem.structure.entities.**Stiffener1D**(*name*, *shape*, *cref=None*, *sref=None*, *group=None*)

Bases: *afem.structure.entities.CurvePart* (page 257)

1-D stiffener for surface parts.

13.1.14 Stiffener2D

class afem.structure.entities.**Stiffener2D**(*name*, *shape*, *cref=None*, *sref=None*, *group=None*)

Bases: *afem.structure.entities.SurfacePart* (page 257)

2-D stiffener for surface parts.

13.1.15 Stringer

class afem.structure.entities.**Stringer**(*name*, *shape*, *cref=None*, *sref=None*, *group=None*)

Bases: *afem.structure.entities.SurfacePart* (page 257)

Stringer.

13.1.16 Beam2D

class afem.structure.entities.**Beam2D**(*name*, *shape*, *cref=None*, *sref=None*, *group=None*)
Bases: *afem.structure.entities.SurfacePart* (page 257)

Beam 2-D.

13.1.17 Group

class afem.structure.group.**Group**(*name*, *parent=None*)
Bases: *afem.base.entities.NamedItem* (page 11)

Group of parts.

Parameters

- **name** (*str*) – The name.
- **parent** (afem.structure.group.Group (page 261) *or None*) – The parent group, if any.

parent

Returns The parent group, if any.

Return type *afem.structure.group.Group* (page 261) or None

parts

Returns List of all parts.

Return type list(*afem.structure.entities.Part* (page 253))

activate()
Activate this group.

Returns None

add_parts(**parts*)
Add parts to the group.

Parameters **parts** (afem.structure.entities.Part (page 253)) – Part(s) to add.

Returns None.

get_part(*name*)
Get a part in the group by name.

Parameters **name** (*str*) – Part name.

Returns The part.

Return type *afem.structure.entities.Part* (page 253)

Raises **KeyError** – If the part is not found.

get_parts(*include_subgroup=True*, *rtype=None*, *order=False*)
Get all the parts from the group and its subgroups.

Parameters

- **include_subgroup** (*bool*) – Option to recursively include parts from any subgroups.
- **rtype** – Option to return only parts of a certain type. Provide a class to check if the part is of the given type using *isinstance()*.

- **order** (*bool*) – Option to order parts by their ID.

Returns List of parts.

Return type list(*afem.structure.entities.Part* (page 253))

remove_part (*name*)

Remove a part from the group if present.

Parameters **name** (*str*) – Part name.

Returns None.

get_shape (*include_subgroup=True*)

Get a shape derived from all parts in the group. This puts all the parts into a single compound which could be used as the master shape for the meshing process.

Parameters **include_subgroup** (*bool*) – Option to recursively include parts from any sub-groups.

Returns The part shapes as a compound.

Return type *afem.topology.entities.Compound* (page 119)

create_subgroup (*name*, *active=True*)

Create a new sub-group of this one.

Parameters

- **name** (*str*) – The group name.
- **active** (*bool*) – Option to make this new group the active one.

Returns The new sub-group.

Return type *afem.structure.group.Group* (page 261)

static parts_to_compound (*parts*)

Convert the list of parts into a single compound using each of their shapes.

Parameters **parts** (*collections.Sequence(afem.structure.entities.Part* (page 253)*)*) – The parts.

Returns The compound.

Return type *afem.topology.entities.Compound* (page 119)

13.1.18 GroupAPI

class afem.structure.group.**GroupAPI**

Bases: object

Group API. This stores all created groups so data can be accessed from one place. There is always a master model named '_master' that is created at initialization. This will be the parent of all groups that are not provided a parent when created. No groups should be named '_master'.

classmethod reset ()

Reset master group and data structure and reset Part index back to 1. This should delete all groups unless they are referenced somewhere else.

Returns None.

classmethod get_master ()

Get the master group.

Returns The master group.

Return type *afem.structure.group.Group* (page 261)

classmethod get_active()

Get the active group.

Returns Active group.

Return type *afem.structure.group.Group* (page 261)

classmethod get_group(*group=None*)

Get an group. If a string is provided then the group name will be used to get the group. If a group instance is provided then it is simply returned. If `None` is provided then the active group is returned.

Parameters **group** (*str or* `afem.structure.group.Group` (page 261) *or None*) – Group to get.

Returns The group.

Return type *afem.structure.group.Group* (page 261)

classmethod make_active(*group*)

Activate the group. This method uses the *GroupAPI.get_group()* method to get the group to activate.

Parameters **group** (*str or* `afem.structure.group.Group` (page 261) *or None*) – Group to activate.

Returns None.

classmethod create_group(*name*, *parent=None*, *active=True*, **parts*)

Create an group.

Parameters

- **name** (*str*) – The name.
- **parent** (*str or* `afem.structure.group.Group` (page 261) *or None*) – The parent group. If `None` then the master group is used.
- **active** (*bool*) – Option to make the new group the active one.
- **parts** (`afem.structure.entities.Part` (page 253)) – The part(s) to add to the group, if any.

Returns New group.

Return type *afem.structure.group.Group* (page 261)

classmethod add_parts(*group*, **parts*)

Add parts to the group.

Parameters

- **group** (*str or* `afem.structure.group.Group` (page 261) *or None*) – The group. If `None` then the active group is used.
- **parts** (`afem.structure.entities.Part` (page 253)) – The part(s) to add.

Returns None.

classmethod get_part(*name*, *group=None*)

Get a part from the group using its name.

Parameters

- **name** (*str*) – The part name.

- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group. If None then the active group is used.

Returns The part.

Return type *afem.structure.entities.Part* (page 253)

Raises **KeyError** – If the part is not found.

classmethod get_parts (*group=None*, *include_subgroup=True*, *rtype=None*, *order=False*)
Get parts from group.

Parameters

- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group. If None then the active group is used.
- **include_subgroup** (*bool*) – Option to recursively include parts from any subgroups.
- **rtype** – Option to return only parts of a certain type. Provide a class to check if the part is of the given type using *isinstance()*.
- **order** (*bool*) – Option to order parts by their ID.

Returns The parts.

Return type list(*afem.structure.entities.Part* (page 253))

classmethod remove_part (*name*, *group=None*)
Remove a part from the group if present.

Parameters

- **name** (*str*) – Part name.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group. If None then the active group is used.

Returns None.

classmethod get_shape (*group='_master'*, *include_subgroup=True*)
Get a shape derived from all parts in the group. This puts all the parts into a single compound which could be used as the master shape for the meshing process.

Parameters

- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group. If None then the active group is used. By default the master model is used.
- **include_subgroup** (*bool*) – Option to recursively include parts from any subgroups.

Returns The parts as a compound.

Return type *afem.topology.entities.Compound* (page 119)

classmethod save_model (*fn*, *binary=True*)
Save the model.

Parameters

- **fn** (*str*) – The filename.
- **binary** (*bool*) – If *True*, the document will be saved in a binary format. If *False*, the document will be saved in an XML format.

Returns *True* if saved, *False* otherwise.

Return type bool

classmethod load_model(*fn*, *group=None*)

Load a model.

Parameters

- **fn** (*str*) – The filename. The extension should be either ".xbf" for a binary file or ".xml" for an XML file.
- **group** (afem.structure.group.Group (page 261)) – The group to load the parts into. If *None* then the active group is used.

Returns *True* if loaded, *False* otherwise.

Return type bool

Raises **TypeError** – If the file extension type is not supported.

13.2 Create

13.2.1 PartBuilder

class afem.structure.create.**PartBuilder**(*name*, *shape*, *cref=None*, *sref=None*, *group=None*, *type_=<class 'afem.structure.entities.Part'>*)

Bases: `object`

Base class for creating a part.

Parameters

- **name** (*str*) – Part name.
- **cref** (afem.geometry.entities.Curve (page 43) *or None*) – The reference curve.
- **sref** (afem.geometry.entities.Surface (page 49) *or None*) – The reference surface.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.
- **type** (*Type[*afem.structure.entities.Part (page 253)*]*) – The type of part to create.

part

Returns The part. The return type will be defined by `type_`.

Return type *afem.structure.entities.SurfacePart* (page 257)

13.2.2 PartsBuilder

class afem.structure.create.**PartsBuilder**

Bases: `object`

Base class for creating multiple parts.

nparts

Returns The number of parts.

Return type int

parts

Returns The parts.

Return type list(*afem.structure.entities.Part* (page 253))

spacing

Returns The spacing between first and second parts.

Return type float

next_index

Returns The next index.

Return type int

13.2.3 CurvePartByShape

class afem.structure.create.**CurvePartByShape**(*name*, *shape*, *cref=None*, *group=None*, *type_=<class 'afem.structure.entities.CurvePart'>*)

Bases: *afem.structure.create.PartBuilder* (page 265)

Create a curve part using a shape.

Parameters

- **name** (*str*) – Part name.
- **shape** (afem.geometry.entities.Curve (page 43) *or* afem.topology.entities.Shape (page 109)) – The shape.
- **cref** (afem.geometry.entities.Curve (page 43)) – The reference curve. If not provided then a curve will be extracted from the shape.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.
- **type** (*Type[*afem.structure.entities.Part (page 253)*]*) – The type of part to create.

13.2.4 Beam1DByShape

class afem.structure.create.**Beam1DByShape**(*name*, *shape*, *cref=None*, *group=None*)

Bases: *afem.structure.create.CurvePartByShape* (page 266)

Create a beam using a shape.

Parameters

- **name** (*str*) – Part name.
- **shape** (afem.geometry.entities.Curve (page 43) *or* afem.topology.entities.Shape (page 109)) – The shape.
- **cref** (afem.geometry.entities.Curve (page 43)) – The reference curve. If not provided then a curve will be extracted from the shape.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.2.5 Beam1DByCurve

class afem.structure.create.**Beam1DByCurve**(*name*, *crv*, *group=None*)
Bases: *afem.structure.create.Beam1DByShape* (page 266)

Create a beam using a curve.

Parameters

- **name** (*str*) – Part name.
- **crv** (afem.geometry.entities.Curve (page 43)) – The curve.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.2.6 Beam1DByPoints

class afem.structure.create.**Beam1DByPoints**(*name*, *p1*, *p2*, *group=None*)
Bases: *afem.structure.create.Beam1DByCurve* (page 267)

Create a beam between two points.

Parameters

- **name** (*str*) – Part name.
- **p1** (*point_like*) – First point.
- **p2** (*point_like*) – Second point.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.2.7 SurfacePartByShape

class afem.structure.create.**SurfacePartByShape**(*name*, *basis_shape*, *body*, *group=None*, *type_=<class 'afem.structure.entities.SurfacePart'>*)
Bases: *afem.structure.create.PartBuilder* (page 265)

Create a surface part using a basis shape.

Parameters

- **name** (*str*) – Part name.
- **basis_shape** (afem.topology.entities.Shape (page 109)) – The basis shape.
- **body** (afem.oml.entities.Body (page 228)) – The body.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.
- **type** (*Type[*afem.structure.entities.Part (page 253)*]*) – The type of part to create.

Note:

- The reference curve of the part will be untrimmed in this method. It will be determined by the intersection between *shape* and the body reference shape.

- The reference surface of the part will be taken as the underlying surface of the largest face in the basis shape.

13.2.8 SurfacePartByParameters

class afem.structure.create.**SurfacePartByParameters**(*name*, *u1*, *v1*, *u2*, *v2*, *body*, *basis_shape=None*, *group=None*, *type_=<class 'afem.structure.entities.SurfacePart'>*)

Bases: `afem.structure.create.PartBuilder` (page 265)

Create a surface part between wing parameters.

Parameters

- **name** (*str*) – Part name.
- **u1** (*float*) – Starting point u-parameter.
- **v1** (*float*) – Starting point v-parameter.
- **u2** (*float*) – Ending point u-parameter.
- **v2** (*float*) – Ending point v-parameter.
- **body** (`afem.oml.entities.Body` (page 228)) – The body.
- **basis_shape** (`afem.geometry.entities.Surface` (page 49) *or* `afem.topology.entities.Shape` (page 109)) – The basis shape to define the shape of the part. If not provided, then a plane will be defined between (u1, v1), (u2, v2), and a point translated from the reference surface normal at (u1, v1).
- **group** (*str or* `afem.structure.group.Group` (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.
- **type** (*Type[*`afem.structure.entities.Part` (page 253)*]*) – The type of part to create.

Raises RuntimeError – If Boolean operation fails.

Note:

- If a basis shape is provided, then the starting and ending parameters should be near the intersection between this shape and the body reference shape.

13.2.9 SurfacePartByPoints

class afem.structure.create.**SurfacePartByPoints**(*name*, *p1*, *p2*, *body*, *basis_shape=None*, *group=None*, *type_=<class 'afem.structure.entities.SurfacePart'>*)

Bases: `afem.structure.create.SurfacePartByParameters` (page 268)

Create a rib between two points.

Parameters

- **name** (*str*) – Part name.

- **p1** (*point_like*) – Starting point.
- **p2** (*point_like*) – End point.
- **body** (afem.oml.entities.Body (page 228)) – The body.
- **basis_shape** (afem.geometry.entities.Surface (page 49) *or* afem.topology.entities.Shape (page 109)) – The basis shape to define the shape of the part. If not provided, then a plane will be defined between (u1, v1), (u2, v2), and a point translated from the reference surface normal at (u1, v1).
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.
- **type** (*Type[*afem.structure.entities.Part (page 253)*]*) – The type of part to create.

Note:

- The starting and ending points should be on or near the body reference surface since they are projected to find parameters.
- If a basis shape is provided, then the starting and ending points should be near the intersection between this shape and the body reference shape.

13.2.10 SurfacePartByEnds

class afem.structure.create.**SurfacePartByEnds**(*name*, *e1*, *e2*, *body*, *basis_shape=None*, *group=None*, *type_=<class 'afem.structure.entities.SurfacePart'>*)

Bases: *afem.structure.create.SurfacePartByParameters* (page 268)

Create a surface part by defining its endpoints which can be either points or parameters on a reference surface.

Parameters

- **name** (*str*) – Part name.
- **e1** (*point_like or collections.Sequence[float]*) – Start point as a point or surface parameters (u1, v1).
- **e2** (*point_like or collections.Sequence[float]*) – End point as a point or surface parameters (u2, v2).
- **body** (afem.oml.entities.Body (page 228)) – The body.
- **basis_shape** (afem.geometry.entities.Surface (page 49) *or* afem.topology.entities.Shape (page 109)) – The basis shape to define the shape of the part. If not provided, then a plane will be defined between (u1, v1), (u2, v2), and a point translated from the reference surface normal at (u1, v1).
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.
- **type** (*Type[*afem.structure.entities.Part (page 253)*]*) – The type of part to create.

Raises TypeError – If *e1* or *e2* are not *point_like* or a sequence of two surface parameters.

13.2.11 SurfacePartBetweenShapes

class afem.structure.create.**SurfacePartBetweenShapes**(*name*, *shape1*, *shape2*, *body*, *basis_shape*, *group=None*, *type_=<class 'afem.structure.entities.SurfacePart'>*)

Bases: *afem.structure.create.SurfacePartByPoints* (page 268)

Create a surface part between shapes.

Parameters

- **name** (*str*) – Part name.
- **shape1** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Curve (page 43) *or* afem.geometry.entities.Surface (page 49) *or* afem.structure.entities.Part (page 253)) – Starting shape.
- **shape2** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Curve (page 43) *or* afem.geometry.entities.Surface (page 49) *or* afem.structure.entities.Part (page 253)) – Ending shape.
- **body** (*afem.oml.entities.body*) – The body.
- **basis_shape** (afem.geometry.entities.Surface (page 49) *or* afem.topology.entities.Shape (page 109)) – The basis shape.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.
- **type** (*Type[*afem.structure.entities.Part (page 253)*]*) – The type of part to create.

13.2.12 SurfacePartsBetweenPlanesByNumber

class afem.structure.create.**SurfacePartsBetweenPlanesByNumber**(*name*, *pln1*, *pln2*, *n*, *shape1*, *shape2*, *body*, *d1=None*, *d2=None*, *first_index=1*, *delimiter=' '*, *group=None*, *type_=<class 'afem.structure.entities.SurfacePart'>*)

Bases: *afem.structure.create.PartsBuilder* (page 265)

Create a specified number of surface parts between two planes.

Parameters

- **name** (*str*) – Part name.
- **pln1** (afem.geometry.entities.Plane (page 52)) – The first plane.
- **pln2** (afem.geometry.entities.Plane (page 52)) – The second plane.
- **n** (*int*) – The number of parts.

- **shape1** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Curve (page 43) *or* afem.geometry.entities.Surface (page 49) *or* afem.structure.entities.Part (page 253)) – Starting shape.
- **shape2** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Curve (page 43) *or* afem.geometry.entities.Surface (page 49) *or* afem.structure.entities.Part (page 253)) – Ending shape.
- **body** (afem.oml.entities.Body (page 228)) – The body.
- **d1** (*float*) – An offset distance for the first plane. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last plane. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **first_index** (*int*) – The first index appended to the part name as parts are created successively.
- **delimiter** (*str*) – The delimiter to use when joining the part name with the index. The final part name will be 'name' + 'delimiter' + 'index'.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.
- **type** (*Type[*afem.structure.entities.Part (page 253)*]*) – The type of part to create.

13.2.13 SurfacePartsBetweenPlanesByDistance

class afem.structure.create.**SurfacePartsBetweenPlanesByDistance**(*name, pln1, pln2, maxd, shape1, shape2, body, d1=None, d2=None, nmin=0, first_index=1, delimiter=' ', group=None, type_=<class 'afem.structure.entities.SurfacePart'>*)

Bases: *afem.structure.create.PartsBuilder* (page 265)

Create surface parts between two planes using a maximum spacing.

Parameters

- **name** (*str*) – Part name.
- **pln1** (afem.geometry.entities.Plane (page 52)) – The first plane.
- **pln2** (afem.geometry.entities.Plane (page 52)) – The second plane.
- **maxd** (*float*) – The maximum allowed spacing. The actual spacing will be adjusted to not to exceed this value.
- **shape1** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Curve (page 43) *or* afem.geometry.entities.Surface (page 49) *or* afem.structure.entities.Part (page 253)) – Starting shape.

- **shape2** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Curve (page 43) *or* afem.geometry.entities.Surface (page 49) *or* afem.structure.entities.Part (page 253)) – Ending shape.
- **body** (afem.oml.entities.Body (page 228)) – The body.
- **d1** (*float*) – An offset distance for the first plane. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last plane. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **nmin** (*int*) – Minimum number of parts to create.
- **first_index** (*int*) – The first index appended to the part name as parts are created successively.
- **delimiter** (*str*) – The delimiter to use when joining the part name with the index. The final part name will be 'name' + 'delimiter' + 'index'.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.
- **type** (*Type[*afem.structure.entities.Part (page 253)*]*) – The type of part to create.

13.2.14 SurfacePartsAlongCurveByNumber

class afem.structure.create.**SurfacePartsAlongCurveByNumber**(*name*, *crv*, *n*, *shape1*, *shape2*, *body*, *ref_pln=None*, *u1=None*, *u2=None*, *d1=None*, *d2=None*, *first_index=1*, *delimiter=' '*, *tol=1e-07*, *group=None*, *type_=<class 'afem.structure.entities.SurfacePart'>*)

Bases: *afem.structure.create.PartsBuilder* (page 265)

Create a specified number of surface parts along a curve.

Parameters

- **name** (*str*) – Part name.
- **crv** (afem.geometry.entities.Curve (page 43)) – The curve.
- **n** (*int*) – The number of parts.
- **shape1** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Curve (page 43) *or* afem.geometry.entities.Surface (page 49) *or* afem.structure.entities.Part (page 253)) – Starting shape.
- **shape2** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Curve (page 43) *or* afem.geometry.entities.Surface (page 49) *or* afem.structure.entities.Part (page 253)) – Ending shape.
- **body** (afem.oml.entities.Body (page 228)) – The body.

- **ref_pln** (afem.geometry.entities.Plane (page 52)) – The normal of this plane will be used to define the normal of all planes along the curve. If no plane is provided, then the first derivative of the curve will define the plane normal.
- **u1** (*float*) – The parameter of the first plane (default=crv.u1).
- **u2** (*float*) – The parameter of the last plane (default=crv.u2).
- **d1** (*float*) – An offset distance for the first plane. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last plane. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **first_index** (*int*) – The first index appended to the part name as parts are created successively.
- **delimiter** (*str*) – The delimiter to use when joining the part name with the index. The final part name will be 'name' + 'delimiter' + 'index'.
- **tol** (*float*) – Tolerance.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.
- **type** (*Type[*afem.structure.entities.Part (page 253)*]*) – The type of part to create.

13.2.15 SurfacePartsAlongCurveByDistance

class afem.structure.create.**SurfacePartsAlongCurveByDistance**(*name, crv, maxd, shape1, shape2, body, ref_pln=None, u1=None, u2=None, d1=None, d2=None, nmin=0, first_index=1, delimiter=' ', tol=1e-07, group=None, type_=<class 'afem.structure.entities.SurfacePart'>*)

Bases: *afem.structure.create.PartsBuilder* (page 265)

Create surface parts along a curve using a maximum spacing.

Parameters

- **name** (*str*) – Part name.
- **crv** (afem.geometry.entities.Curve (page 43)) – The curve.
- **maxd** (*float*) – The maximum allowed spacing between planes. The actual spacing will be adjusted to not to exceed this value.
- **shape1** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Curve (page 43) *or* afem.geometry.entities.Surface (page 49) *or* afem.structure.entities.Part (page 253)) – Starting shape.

- **shape2** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Curve (page 43) *or* afem.geometry.entities.Surface (page 49) *or* afem.structure.entities.Part (page 253)) – Ending shape.
- **body** (afem.oml.entities.Body (page 228)) – The body.
- **ref_pln** (afem.geometry.entities.Plane (page 52)) – The normal of this plane will be used to define the normal of all planes along the curve. If no plane is provided, then the first derivative of the curve will define the plane normal.
- **u1** (*float*) – The parameter of the first plane (default=crv.u1).
- **u2** (*float*) – The parameter of the last plane (default=crv.u2).
- **d1** (*float*) – An offset distance for the first plane. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last plane. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **nmin** (*int*) – Minimum number of planes to create.
- **first_index** (*int*) – The first index appended to the part name as parts are created successively.
- **delimiter** (*str*) – The delimiter to use when joining the part name with the index. The final part name will be 'name' + 'delimiter' + 'index'.
- **tol** (*float*) – Tolerance.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.
- **type** (*Type[*afem.structure.entities.Part (page 253)*]*) – The type of part to create.

13.2.16 SparByShape

class afem.structure.create.**SparByShape**(*name*, *basis_shape*, *body*, *group=None*)

Bases: *afem.structure.create.SurfacePartByShape* (page 267)

Create a spar using a basis shape.

Parameters

- **name** (*str*) – Part name.
- **basis_shape** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Surface (page 49)) – The basis shape or surface.
- **body** (afem.oml.entities.Body (page 228)) – The body.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

Note:

- The reference curve of the part will be untrimmed in this method. It will be determined by the intersection between *shape* and the body reference shape.
- The reference surface of the part will be taken as the underlying surface of the largest face in the basis shape.

13.2.17 SparByParameters

class afem.structure.create.**SparByParameters**(*name*, *u1*, *v1*, *u2*, *v2*, *body*, *basis_shape=None*, *group=None*)

Bases: *afem.structure.create.SurfacePartByParameters* (page 268)

Create a spar between wing parameters.

Parameters

- **name** (*str*) – Part name.
- **u1** (*float*) – Starting point u-parameter.
- **v1** (*float*) – Starting point v-parameter.
- **u2** (*float*) – Ending point u-parameter.
- **v2** (*float*) – Ending point v-parameter.
- **body** (afem.oml.entities.Body (page 228)) – The body.
- **basis_shape** (afem.geometry.entities.Surface (page 49) *or* afem.topology.entities.Shape (page 109)) – The basis shape to define the shape of the part. If not provided, then a plane will be defined between (u1, v1), (u2, v2), and a point translated from the reference surface normal at (u1, v1).
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.2.18 SparByPoints

class afem.structure.create.**SparByPoints**(*name*, *p1*, *p2*, *body*, *basis_shape=None*, *group=None*)

Bases: *afem.structure.create.SurfacePartByPoints* (page 268)

Create a spar between two points. This method inverts the starting and ending points and then uses *SparByParameters* (page 275).

Parameters

- **name** (*str*) – Part name.
- **p1** (*point_like*) – Starting point.
- **p2** (*point_like*) – End point.
- **body** (afem.oml.entities.Body (page 228)) – The body.
- **basis_shape** (afem.geometry.entities.Surface (page 49) *or* afem.topology.entities.Shape (page 109)) – The basis shape to define the shape of the part. If not provided, then a plane will be defined between (u1, v1), (u2, v2), and a point translated from the reference surface normal at (u1, v1).
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

Note:

- The starting and ending points should be on or near the body reference surface since they are projected to find parameters.

- If a basis shape is provided, then the starting and ending points should be near the intersection between this shape and the body reference shape.

13.2.19 SparByEnds

class afem.structure.create.**SparByEnds**(*name*, *e1*, *e2*, *body*, *basis_shape=None*, *group=None*)

Bases: *afem.structure.create.SurfacePartByEnds* (page 269)

Create a spar by defining its endpoints which can be either points or parameters on a reference surface.

Parameters

- **name** (*str*) – Part name.
- **e1** (*point_like or collections.Sequence[float]*) – Start point as a point or surface parameters (u1, v1).
- **e2** (*point_like or collections.Sequence[float]*) – End point as a point or surface parameters (u2, v2).
- **body** (afem.oml.entities.Body (page 228)) – The body.
- **basis_shape** (afem.geometry.entities.Surface (page 49) *or* afem.topology.entities.Shape (page 109)) – The basis shape to define the shape of the part. If not provided, then a plane will be defined between (u1, v1), (u2, v2), and a point translated from the reference surface normal at (u1, v1).
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.2.20 SparBetweenShapes

class afem.structure.create.**SparBetweenShapes**(*name*, *shape1*, *shape2*, *body*, *basis_shape*, *group=None*)

Bases: *afem.structure.create.SurfacePartBetweenShapes* (page 270)

Create a spar between shapes.

Parameters

- **name** (*str*) – Part name.
- **shape1** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Curve (page 43) *or* afem.geometry.entities.Surface (page 49) *or* afem.structure.entities.Part (page 253)) – Starting shape.
- **shape2** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Curve (page 43) *or* afem.geometry.entities.Surface (page 49) *or* afem.structure.entities.Part (page 253)) – Ending shape.
- **body** (*afem.oml.entities.body*) – The body.
- **basis_shape** (afem.geometry.entities.Surface (page 49) *or* afem.topology.entities.Shape (page 109)) – The basis shape.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.2.21 SparsBetweenPlanesByNumber

class afem.structure.create.**SparsBetweenPlanesByNumber**(*name*, *pln1*, *pln2*, *n*, *shape1*, *shape2*, *body*, *d1=None*, *d2=None*, *first_index=1*, *delimiter=' '*, *group=None*)

Bases: *afem.structure.create.SurfacePartsBetweenPlanesByNumber* (page 270)

Create a specified number of planar spars between two planes.

Parameters

- **name** (*str*) – Part name.
- **pln1** (afem.geometry.entities.Plane (page 52)) – The first plane.
- **pln2** (afem.geometry.entities.Plane (page 52)) – The second plane.
- **n** (*int*) – The number of parts.
- **shape1** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Curve (page 43) *or* afem.geometry.entities.Surface (page 49) *or* afem.structure.entities.Part (page 253)) – Starting shape.
- **shape2** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Curve (page 43) *or* afem.geometry.entities.Surface (page 49) *or* afem.structure.entities.Part (page 253)) – Ending shape.
- **body** (afem.oml.entities.Body (page 228)) – The body.
- **d1** (*float*) – An offset distance for the first plane. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last plane. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **first_index** (*int*) – The first index appended to the part name as parts are created successively.
- **delimiter** (*str*) – The delimiter to use when joining the part name with the index. The final part name will be 'name' + 'delimiter' + 'index'.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.2.22 SparsBetweenPlanesByDistance

class afem.structure.create.**SparsBetweenPlanesByDistance**(*name*, *pln1*, *pln2*, *maxd*, *shape1*, *shape2*, *body*, *d1=None*, *d2=None*, *nmin=0*, *first_index=1*, *delimiter=' '*, *group=None*)

Bases: *afem.structure.create.SurfacePartsBetweenPlanesByDistance* (page 271)

Create planar spars between two planes using a maximum spacing.

Parameters

- **name** (*str*) – Part name.
- **pln1** (afem.geometry.entities.Plane (page 52)) – The first plane.

- **pln2** (`afem.geometry.entities.Plane` (page 52)) – The second plane.
- **maxd** (*float*) – The maximum allowed spacing. The actual spacing will be adjusted to not to exceed this value.
- **shape1** (`afem.topology.entities.Shape` (page 109) *or* `afem.geometry.entities.Curve` (page 43) *or* `afem.geometry.entities.Surface` (page 49) *or* `afem.structure.entities.Part` (page 253)) – Starting shape.
- **shape2** (`afem.topology.entities.Shape` (page 109) *or* `afem.geometry.entities.Curve` (page 43) *or* `afem.geometry.entities.Surface` (page 49) *or* `afem.structure.entities.Part` (page 253)) – Ending shape.
- **body** (`afem.oml.entities.Body` (page 228)) – The body.
- **d1** (*float*) – An offset distance for the first plane. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last plane. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **nmin** (*int*) – Minimum number of parts to create.
- **first_index** (*int*) – The first index appended to the part name as parts are created successively.
- **delimiter** (*str*) – The delimiter to use when joining the part name with the index. The final part name will be 'name' + 'delimiter' + 'index'.
- **group** (*str or* `afem.structure.group.Group` (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.2.23 SparsAlongCurveByNumber

class afem.structure.create.**SparsAlongCurveByNumber**(*name, crv, n, shape1, shape2, body, ref_pln=None, u1=None, u2=None, d1=None, d2=None, first_index=1, delimiter=' ', tol=1e-07, group=None*)

Bases: *`afem.structure.create.SurfacePartsAlongCurveByNumber`* (page 272)

Create a specified number of planar spars along a curve.

Parameters

- **name** (*str*) – Part name.
- **crv** (`afem.geometry.entities.Curve` (page 43)) – The curve.
- **n** (*int*) – The number of parts.
- **shape1** (`afem.topology.entities.Shape` (page 109) *or* `afem.geometry.entities.Curve` (page 43) *or* `afem.geometry.entities.Surface` (page 49) *or* `afem.structure.entities.Part` (page 253)) – Starting shape.
- **shape2** (`afem.topology.entities.Shape` (page 109) *or* `afem.geometry.entities.Curve` (page 43) *or* `afem.geometry.entities.Surface` (page 49) *or* `afem.structure.entities.Part` (page 253)) – Ending shape.
- **body** (`afem.oml.entities.Body` (page 228)) – The body.

- **ref_pln** (`afem.geometry.entities.Plane` (page 52)) – The normal of this plane will be used to define the normal of all planes along the curve. If no plane is provided, then the first derivative of the curve will define the plane normal.
- **u1** (*float*) – The parameter of the first plane (default=crv.u1).
- **u2** (*float*) – The parameter of the last plane (default=crv.u2).
- **d1** (*float*) – An offset distance for the first plane. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last plane. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **first_index** (*int*) – The first index appended to the part name as parts are created successively.
- **delimiter** (*str*) – The delimiter to use when joining the part name with the index. The final part name will be 'name' + 'delimiter' + 'index'.
- **tol** (*float*) – Tolerance.
- **group** (*str or* `afem.structure.group.Group` (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.2.24 SparsAlongCurveByDistance

class afem.structure.create.**SparsAlongCurveByDistance**(*name, crv, maxd, shape1, shape2, body, ref_pln=None, u1=None, u2=None, d1=None, d2=None, nmin=0, first_index=1, delimiter=' ', tol=1e-07, group=None*)

Bases: `afem.structure.create.SurfacePartsAlongCurveByDistance` (page 273)

Create planar spars along a curve using a maximum spacing.

Parameters

- **name** (*str*) – Part name.
- **crv** (`afem.geometry.entities.Curve` (page 43)) – The curve.
- **maxd** (*float*) – The maximum allowed spacing between planes. The actual spacing will be adjusted to not to exceed this value.
- **shape1** (`afem.topology.entities.Shape` (page 109) *or* `afem.geometry.entities.Curve` (page 43) *or* `afem.geometry.entities.Surface` (page 49) *or* `afem.structure.entities.Part` (page 253)) – Starting shape.
- **shape2** (`afem.topology.entities.Shape` (page 109) *or* `afem.geometry.entities.Curve` (page 43) *or* `afem.geometry.entities.Surface` (page 49) *or* `afem.structure.entities.Part` (page 253)) – Ending shape.
- **body** (`afem.oml.entities.Body` (page 228)) – The body.
- **ref_pln** (`afem.geometry.entities.Plane` (page 52)) – The normal of this plane will be used to define the normal of all planes along the curve. If no plane is provided, then the first derivative of the curve will define the plane normal.
- **u1** (*float*) – The parameter of the first plane (default=crv.u1).
- **u2** (*float*) – The parameter of the last plane (default=crv.u2).

- **d1** (*float*) – An offset distance for the first plane. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last plane. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **nmin** (*int*) – Minimum number of planes to create.
- **first_index** (*int*) – The first index appended to the part name as parts are created successively.
- **delimiter** (*str*) – The delimiter to use when joining the part name with the index. The final part name will be 'name' + 'delimiter' + 'index'.
- **tol** (*float*) – Tolerance.
- **group** (*str or* `afem.structure.group.Group` (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.2.25 RibByShape

class afem.structure.create.**RibByShape**(*name*, *basis_shape*, *body*, *group=None*)

Bases: *afem.structure.create.SurfacePartByShape* (page 267)

Create a rib using a basis shape.

Parameters

- **name** (*str*) – Part name.
- **basis_shape** (`afem.topology.entities.Shape` (page 109) *or* `afem.geometry.entities.Surface` (page 49)) – The basis shape or surface.
- **body** (`afem.oml.entities.Body` (page 228)) – The body.
- **group** (*str or* `afem.structure.group.Group` (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

Note:

- The reference curve of the part will be untrimmed in this method. It will be determined by the intersection between *shape* and the body reference shape.
- The reference surface of the part will be taken as the underlying surface of the largest face in the basis shape.

13.2.26 RibByParameters

class afem.structure.create.**RibByParameters**(*name*, *u1*, *v1*, *u2*, *v2*, *body*, *basis_shape=None*, *group=None*)

Bases: *afem.structure.create.SurfacePartByParameters* (page 268)

Create a rib between wing parameters.

Parameters

- **name** (*str*) – Part name.
- **u1** (*float*) – Starting point u-parameter.

- **v1** (*float*) – Starting point v-parameter.
- **u2** (*float*) – Ending point u-parameter.
- **v2** (*float*) – Ending point v-parameter.
- **body** (afem.oml.entities.Body (page 228)) – The body.
- **basis_shape** (afem.geometry.entities.Surface (page 49) *or* afem.topology.entities.Shape (page 109)) – The basis shape to define the shape of the part. If not provided, then a plane will be defined between (u1, v1), (u2, v2), and a point translated from the reference surface normal at (u1, v1).
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.2.27 RibByPoints

class afem.structure.create.**RibByPoints**(*name, p1, p2, body, basis_shape=None, group=None*)

Bases: *afem.structure.create.SurfacePartByPoints* (page 268)

Create a rib between two points.

Parameters

- **name** (*str*) – Part name.
- **p1** (*point_like*) – Starting point.
- **p2** (*point_like*) – End point.
- **body** (afem.oml.entities.Body (page 228)) – The body.
- **basis_shape** (afem.geometry.entities.Surface (page 49) *or* afem.topology.entities.Shape (page 109)) – The basis shape to define the shape of the part. If not provided, then a plane will be defined between (u1, v1), (u2, v2), and a point translated from the reference surface normal at (u1, v1).
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

Note:

- The starting and ending points should be on or near the body reference surface since they are projected to find parameters.
- If a basis shape is provided, then the starting and ending points should be near the intersection between this shape and the body reference shape.

13.2.28 RibBetweenShapes

class afem.structure.create.**RibBetweenShapes**(*name, shape1, shape2, body, basis_shape, group=None*)

Bases: *afem.structure.create.SurfacePartBetweenShapes* (page 270)

Create a rib between shapes.

Parameters

- **name** (*str*) – Part name.
- **shape1** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Curve (page 43) *or* afem.geometry.entities.Surface (page 49) *or* afem.structure.entities.Part (page 253)) – Starting shape.
- **shape2** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Curve (page 43) *or* afem.geometry.entities.Surface (page 49) *or* afem.structure.entities.Part (page 253)) – Ending shape.
- **body** (afem.oml.entities.Body (page 228)) – The body.
- **basis_shape** (afem.geometry.entities.Surface (page 49) *or* afem.topology.entities.Shape (page 109)) – The basis shape.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.2.29 RibByOrientation

class afem.structure.create.**RibByOrientation**(*name*, *origin*, *body*, *alpha=0.0*, *beta=0.0*, *gamma=0.0*, *axes='xz'*, *group=None*)

Bases: *afem.structure.create.RibByShape* (page 280)

Create a planar rib using rotation angles.

Parameters

- **name** (*str*) – Part name.
- **origin** (*point_like*) – The origin of the plane after rotating and translating from the global origin.
- **body** (afem.oml.entities.Body (page 228)) – The body.
- **alpha** (*float*) – Rotation in degrees about global x-axis.
- **beta** (*float*) – Rotation in degrees about global y-axis.
- **gamma** (*float*) – Rotation in degrees about global z-axis.
- **axes** (*str*) – The axes for the original plane before rotation and translation ('xy', 'xz', 'yz').
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.2.30 RibsBetweenPlanesByNumber

class afem.structure.create.**RibsBetweenPlanesByNumber**(*name*, *pln1*, *pln2*, *n*, *shape1*, *shape2*, *body*, *d1=None*, *d2=None*, *first_index=1*, *delimiter=' '*, *group=None*)

Bases: *afem.structure.create.SurfacePartsBetweenPlanesByNumber* (page 270)

Create a specified number of planar ribs between two planes.

Parameters

- **name** (*str*) – Part name.
- **pln1** (afem.geometry.entities.Plane (page 52)) – The first plane.

- **pln2** (`afem.geometry.entities.Plane` (page 52)) – The second plane.
- **n** (*int*) – The number of parts.
- **shape1** (`afem.topology.entities.Shape` (page 109) *or* `afem.geometry.entities.Curve` (page 43) *or* `afem.geometry.entities.Surface` (page 49) *or* `afem.structure.entities.Part` (page 253)) – Starting shape.
- **shape2** (`afem.topology.entities.Shape` (page 109) *or* `afem.geometry.entities.Curve` (page 43) *or* `afem.geometry.entities.Surface` (page 49) *or* `afem.structure.entities.Part` (page 253)) – Ending shape.
- **body** (`afem.oml.entities.Body` (page 228)) – The body.
- **d1** (*float*) – An offset distance for the first plane. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last plane. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **first_index** (*int*) – The first index appended to the part name as parts are created successively.
- **delimiter** (*str*) – The delimiter to use when joining the part name with the index. The final part name will be 'name' + 'delimiter' + 'index'.
- **group** (*str or* `afem.structure.group.Group` (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.2.31 RibsBetweenPlanesByDistance

class afem.structure.create.**RibsBetweenPlanesByDistance**(*name*, *pln1*, *pln2*, *maxd*, *shape1*, *shape2*, *body*, *d1=None*, *d2=None*, *nmin=0*, *first_index=1*, *delimiter=' '*, *group=None*)

Bases: *afem.structure.create.SurfacePartsBetweenPlanesByDistance* (page 271)

Create planar ribs between two planes using a maximum spacing.

Parameters

- **name** (*str*) – Part name.
- **pln1** (`afem.geometry.entities.Plane` (page 52)) – The first plane.
- **pln2** (`afem.geometry.entities.Plane` (page 52)) – The second plane.
- **maxd** (*float*) – The maximum allowed spacing. The actual spacing will be adjusted to not to exceed this value.
- **shape1** (`afem.topology.entities.Shape` (page 109) *or* `afem.geometry.entities.Curve` (page 43) *or* `afem.geometry.entities.Surface` (page 49) *or* `afem.structure.entities.Part` (page 253)) – Starting shape.
- **shape2** (`afem.topology.entities.Shape` (page 109) *or* `afem.geometry.entities.Curve` (page 43) *or* `afem.geometry.entities.Surface` (page 49) *or* `afem.structure.entities.Part` (page 253)) – Ending shape.
- **body** (`afem.oml.entities.Body` (page 228)) – The body.
- **d1** (*float*) – An offset distance for the first plane. This is typically a positive number indicating a distance from *u1* towards *u2*.

- **d2** (*float*) – An offset distance for the last plane. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **nmin** (*int*) – Minimum number of parts to create.
- **first_index** (*int*) – The first index appended to the part name as parts are created successively.
- **delimiter** (*str*) – The delimiter to use when joining the part name with the index. The final part name will be 'name' + 'delimiter' + 'index'.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.2.32 RibsAlongCurveByNumber

class afem.structure.create.**RibsAlongCurveByNumber**(*name*, *crv*, *n*, *shape1*, *shape2*, *body*, *ref_pln=None*, *u1=None*, *u2=None*, *d1=None*, *d2=None*, *first_index=1*, *delimiter=' '*, *tol=1e-07*, *group=None*)

Bases: *afem.structure.create.SurfacePartsAlongCurveByNumber* (page 272)

Create a specified number of planar ribs along a curve.

Parameters

- **name** (*str*) – Part name.
- **crv** (afem.geometry.entities.Curve (page 43)) – The curve.
- **n** (*int*) – The number of parts.
- **shape1** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Curve (page 43) *or* afem.geometry.entities.Surface (page 49) *or* afem.structure.entities.Part (page 253)) – Starting shape.
- **shape2** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Curve (page 43) *or* afem.geometry.entities.Surface (page 49) *or* afem.structure.entities.Part (page 253)) – Ending shape.
- **body** (afem.oml.entities.Body (page 228)) – The body.
- **ref_pln** (afem.geometry.entities.Plane (page 52)) – The normal of this plane will be used to define the normal of all planes along the curve. If no plane is provided, then the first derivative of the curve will define the plane normal.
- **u1** (*float*) – The parameter of the first plane (default=crv.u1).
- **u2** (*float*) – The parameter of the last plane (default=crv.u2).
- **d1** (*float*) – An offset distance for the first plane. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last plane. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **first_index** (*int*) – The first index appended to the part name as parts are created successively.
- **delimiter** (*str*) – The delimiter to use when joining the part name with the index. The final part name will be 'name' + 'delimiter' + 'index'.

- **tol** (*float*) – Tolerance.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.2.33 RibsAlongCurveByDistance

class afem.structure.create.**RibsAlongCurveByDistance**(*name, crv, maxd, shape1, shape2, body, ref_pln=None, u1=None, u2=None, d1=None, d2=None, nmin=0, first_index=1, delimiter=' ', tol=1e-07, group=None*)

Bases: *afem.structure.create.SurfacePartsAlongCurveByDistance* (page 273)

Create planar ribs along a curve using a maximum spacing.

Parameters

- **name** (*str*) – Part name.
- **crv** (afem.geometry.entities.Curve (page 43)) – The curve.
- **maxd** (*float*) – The maximum allowed spacing between planes. The actual spacing will be adjusted to not to exceed this value.
- **shape1** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Curve (page 43) *or* afem.geometry.entities.Surface (page 49) *or* afem.structure.entities.Part (page 253)) – Starting shape.
- **shape2** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Curve (page 43) *or* afem.geometry.entities.Surface (page 49) *or* afem.structure.entities.Part (page 253)) – Ending shape.
- **body** (afem.oml.entities.Body (page 228)) – The body.
- **ref_pln** (afem.geometry.entities.Plane (page 52)) – The normal of this plane will be used to define the normal of all planes along the curve. If no plane is provided, then the first derivative of the curve will define the plane normal.
- **u1** (*float*) – The parameter of the first plane (default=crv.u1).
- **u2** (*float*) – The parameter of the last plane (default=crv.u2).
- **d1** (*float*) – An offset distance for the first plane. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last plane. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **nmin** (*int*) – Minimum number of planes to create.
- **first_index** (*int*) – The first index appended to the part name as parts are created successively.
- **delimiter** (*str*) – The delimiter to use when joining the part name with the index. The final part name will be 'name' + 'delimiter' + 'index'.
- **tol** (*float*) – Tolerance.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.2.34 RibsAlongCurveAndSurfaceByDistance

class afem.structure.create.**RibsAlongCurveAndSurfaceByDistance**(*name*, *crv*, *srf*, *maxd*, *shape1*, *shape2*, *body*, *u1=None*, *u2=None*, *d1=None*, *d2=None*, *rot_x=None*, *rot_y=None*, *nmin=0*, *first_index=1*, *delimiter=' '*, *tol=1e-07*, *group=None*)

Bases: *afem.structure.create.PartsBuilder* (page 265)

Create planar ribs along a curve and surface using a maximum spacing.

Parameters

- **name** (*str*) – Part name.
- **crv** (afem.geometry.entities.Curve (page 43)) – The curve.
- **srf** (afem.geometry.entities.Surface (page 49)) – The surface.
- **maxd** (*float*) – The maximum allowed spacing between planes. The actual spacing will be adjusted to not to exceed this value.
- **shape1** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Curve (page 43) *or* afem.geometry.entities.Surface (page 49) *or* afem.structure.entities.Part (page 253)) – Starting shape.
- **shape2** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Curve (page 43) *or* afem.geometry.entities.Surface (page 49) *or* afem.structure.entities.Part (page 253)) – Ending shape.
- **body** (afem.oml.entities.Body (page 228)) – The body.
- **u1** (*float*) – The parameter of the first plane (default=crv.u1).
- **u2** (*float*) – The parameter of the last plane (default=crv.u2).
- **d1** (*float*) – An offset distance for the first plane. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last plane. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **rot_x** (*float*) – The rotation angles of each plane along their local x-axis in degrees.
- **rot_y** (*float*) – The rotation angles of each plane along their local y-axis in degrees.
- **nmin** (*int*) – Minimum number of planes to create.
- **first_index** (*int*) – The first index appended to the part name as parts are created successively.
- **delimiter** (*str*) – The delimiter to use when joining the part name with the index. The final part name will be 'name' + 'delimiter' + 'index'.
- **tol** (*float*) – Tolerance.

- **group** (*str or* `afem.structure.group.Group` (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.2.35 BulkheadByShape

class `afem.structure.create.`**BulkheadByShape**(*name*, *basis_shape*, *body*, *group=None*)

Bases: *`afem.structure.create.PartBuilder`* (page 265)

Create a bulkhead using a shape.

Parameters

- **name** (*str*) – Part name.
- **basis_shape** (`afem.topology.entities.Shape` (page 109) *or* `afem.geometry.entities.Surface` (page 49)) – The basis shape or surface.
- **body** (`afem.oml.entities.Body` (page 228)) – The body.
- **group** (*str or* `afem.structure.group.Group` (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

Raises **RuntimeError** – If Boolean operation failed.

13.2.36 FloorByShape

class `afem.structure.create.`**FloorByShape**(*name*, *basis_shape*, *body*, *group=None*)

Bases: *`afem.structure.create.PartBuilder`* (page 265)

Create a floor using a shape.

Parameters

- **name** (*str*) – Part name.
- **basis_shape** (`afem.topology.entities.Shape` (page 109) *or* `afem.geometry.entities.Surface` (page 49)) – The basis shape or surface.
- **body** (`afem.oml.entities.Body` (page 228)) – The body.
- **group** (*str or* `afem.structure.group.Group` (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

Raises **RuntimeError** – If Boolean operation failed.

13.2.37 FrameByPlane

class `afem.structure.create.`**FrameByPlane**(*name*, *pln*, *body*, *height*, *group=None*)

Bases: *`afem.structure.create.PartBuilder`* (page 265)

Create a frame using a plane. A plane is required since the shape of frame is formed using an offset algorithm that requires planar shapes.

Parameters

- **name** (*str*) – Part name.
- **pln** (`afem.geometry.entities.Plane` (page 52)) – The plane.
- **body** (`afem.oml.entities.Body` (page 228)) – The body.

- **height** (*float*) – The height. The absolute value is used.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

Raises RuntimeError – If Boolean operation failed.

13.2.38 FramesByPlanes

class afem.structure.create.**FramesByPlanes**(*name, plns, body, height, first_index=1, delimiter=' ', group=None*)

Bases: *afem.structure.create.PartsBuilder* (page 265)

Create frames using a list of planes. This method uses *FrameByPlane* (page 287).

Parameters

- **name** (*str*) – Part name.
- **plns** (*list(*afem.geometry.entities.Plane (page 52)*)*) – The planes.
- **body** (afem.oml.entities.Body (page 228)) – The body.
- **height** (*float*) – The height.
- **first_index** (*int*) – The first index appended to the part name as parts are created successively.
- **delimiter** (*str*) – The delimiter to use when joining the part name with the index. The final part name will be 'name' + 'delimiter' + 'index'.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.2.39 FramesBetweenPlanesByNumber

class afem.structure.create.**FramesBetweenPlanesByNumber**(*name, pln1, pln2, n, body, height, d1=None, d2=None, first_index=1, delimiter=' ', group=None*)

Bases: *afem.structure.create.PartsBuilder* (page 265)

Create a specified number of frames between two planes.

Parameters

- **name** (*str*) – Part name.
- **pln1** (afem.geometry.entities.Plane (page 52)) – The first plane.
- **pln2** (afem.geometry.entities.Plane (page 52)) – The second plane.
- **n** (*int*) – The number of parts.
- **body** (afem.oml.entities.Body (page 228)) – The body.
- **height** (*float*) – The height.
- **d1** (*float*) – An offset distance for the first plane. This is typically a positive number indicating a distance from *u1* towards *u2*.

- **d2** (*float*) – An offset distance for the last plane. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **first_index** (*int*) – The first index appended to the part name as parts are created successively.
- **delimiter** (*str*) – The delimiter to use when joining the part name with the index. The final part name will be 'name' + 'delimiter' + 'index'.
- **group** (*str or* `afem.structure.group.Group` (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.2.40 FramesBetweenPlanesByDistance

class afem.structure.create.**FramesBetweenPlanesByDistance**(*name, pln1, pln2, maxd, body, height, d1=None, d2=None, nmin=0, first_index=1, delimiter=' ', group=None*)

Bases: *afem.structure.create.PartsBuilder* (page 265)

Create frames between two planes using a maximum spacing.

Parameters

- **name** (*str*) – Part name.
- **pln1** (`afem.geometry.entities.Plane` (page 52)) – The first plane.
- **pln2** (`afem.geometry.entities.Plane` (page 52)) – The second plane.
- **maxd** (*float*) – The maximum allowed spacing. The actual spacing will be adjusted to not to exceed this value.
- **body** (`afem.oml.entities.Body` (page 228)) – The body.
- **height** (*float*) – The height.
- **d1** (*float*) – An offset distance for the first plane. This is typically a positive number indicating a distance from *u1* towards *u2*.
- **d2** (*float*) – An offset distance for the last plane. This is typically a negative number indicating a distance from *u2* towards *u1*.
- **nmin** (*int*) – Minimum number of parts to create.
- **first_index** (*int*) – The first index appended to the part name as parts are created successively.
- **delimiter** (*str*) – The delimiter to use when joining the part name with the index. The final part name will be 'name' + 'delimiter' + 'index'.
- **group** (*str or* `afem.structure.group.Group` (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.2.41 SkinBySolid

class afem.structure.create.**SkinBySolid**(*name, solid, copy=False, group=None*)

Bases: *afem.structure.create.PartBuilder* (page 265)

Create a skin part from the outer shell of the solid.

Parameters

- **name** (*str*) – Part name.
- **solid** (afem.topology.entities.Solid (page 118)) – The solid.
- **copy** (*bool*) – Option to copy the outer shell.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.2.42 SkinByBody

class afem.structure.create.**SkinByBody**(*name*, *body*, *copy=False*, *group=None*)

Bases: *afem.structure.create.SkinBySolid* (page 289)

Create a skin part from the outer shell of a body.

Parameters

- **name** (*str*) – Part name.
- **body** (afem.oml.entities.Body (page 228)) – The body.
- **copy** (*bool*) – Option to copy the outer shell.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.2.43 StringerByShape

class afem.structure.create.**StringerByShape**(*name*, *basis_shape*, *support_shape*, *height*, *angle=30.0*, *shape1=None*, *shape2=None*, *group=None*)

Bases: *afem.structure.create.PartBuilder* (page 265)

Create a Stringer using a basis shape.

Parameters

- **name** (*str*) – Part name.
- **basis_shape** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Surface (page 49)) – The basis shape that will define the path of the stringer.
- **support_shape** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Surface (page 49)) – The shape that will defines the normal direction along the path.
- **height** (*float*) – The height.
- **angle** (*float*) – The runout angle at each end.
- **shape1** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Surface (page 49)) – The starting shape.
- **shape2** (afem.topology.entities.Shape (page 109) *or* afem.geometry.entities.Surface (page 49)) – The ending shape.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.2.44 Beam2DBySweep

class afem.structure.create.**Beam2DBySweep**(*name*, *spine*, *profile*, *group=None*)
Bases: *afem.structure.create.PartBuilder* (page 265)

Create a Beam2D by sweeping a profile along a path.

Parameters

- **name** (*str*) – Part name.
- **spine** (afem.geometry.entities.Curve (page 43) *or* afem.topology.entities.Edge (page 115) *or* afem.topology.entities.Wire (page 116)) – The path for the sweep.
- **profile** (afem.geometry.entities.Curve (page 43) *or* afem.topology.entities.Shape (page 109)) – The profile to sweep.
- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group to add the part to. If not provided the part will be added to the active group.

13.3 Join

13.3.1 FuseSurfaceParts

class afem.structure.join.**FuseSurfaceParts**(*parts*, *tools*, *fuzzy_val=None*)
Bases: object

Fuse together multiple surface parts and rebuild their shapes.

Parameters

- **parts** (*collections.Sequence(*afem.structure.entities.SurfacePart (page 257)*)*) – The surface parts.
- **parts** – The other surface parts.
- **fuzzy_val** (*float*) – Fuzzy tolerance value.

is_done

Returns *True* if operation is done, *False* if not.

Return type bool

shape

Returns The fused shape.

Return type *afem.topology.entities.Shape* (page 109)

13.3.2 FuseSurfacePartsByCref

class afem.structure.join.**FuseSurfacePartsByCref**(*parts*, *tol=None*)
Bases: object

Attempt to automatically fuse together adjacent parts based on the possible intersection of their reference curve. The part shapes are rebuilt in place.

Parameters

- **parts** (*collections.Sequence(*`afem.structure.entities.SurfacePart` (page 257)*)*) – The surface parts.
- **tol** (*float*) – The tolerance to use for checking possible intersections of the reference curves. Default is the maximum tolerance of the part shape.

Raises **TypeError** – If a given part is not a surface part.

is_done

Returns *True* if at least one fuse was performed, *False* if not.

Return type bool

13.3.3 CutParts

class `afem.structure.join.`**CutParts**(*parts*, *shape*)

Bases: `object`

Cut each part with a shape and rebuild the part shape.

Parameters

- **parts** (*collections.Sequence(*`afem.structure.entities.Part` (page 253)*)*) – The parts to cut.
- **shape** (`afem.topology.entities.Shape` (page 109) *or* `afem.geometry.entities.Surface` (page 49)) – The shape to cut with.

was_cut(*part*)

Check the status of the cut operation.

Parameters **part** (`afem.structure.entities.Part` (page 253)) – The part to check.

Returns *True* if part was cut, *False* if not.

Return type

13.3.4 SewSurfaceParts

class `afem.structure.join.`**SewSurfaceParts**(*parts*, *tol=None*, *max_tol=None*)

Bases: `object`

Sew edges of the surface parts and rebuild their shapes.

Parameters

- **parts** (*collections.Sequence(*`afem.structure.entities.SurfacePart` (page 257)*)*) – The parts to sew.
- **tol** (*float*) – The tolerance. If not provided then the average tolerance from all part shapes will be used.
- **max_tol** (*float*) – Maximum tolerance. If not provided then the maximum tolerance from all part shapes will be used.

13.3.5 SplitParts

class afem.structure.join.**SplitParts**(*parts*, *tools=None*, *fuzzy_val=None*)
Bases: object

Split part shapes and rebuild in place.

Parameters

- **parts** (*collections.Sequence(afem.structure.entities.Part* (page 253)*)*) – The parts that will be split and rebuilt.
- **tools** (*collection.Sequence(afem.structure.entities.Part* (page 253)*)*) – The parts or shapes used to split the parts but are not modified.
- **fuzzy_val** (*float*) – Fuzzy tolerance value.

is_done

Returns *True* if operation is done, *False* if not.

Return type bool

shape

Returns The split shape.

Return type *afem.topology.entities.Shape* (page 109)

13.3.6 FuseGroups

class afem.structure.join.**FuseGroups**(*groups*, *fuzzy_val=None*, *include_subgroup=True*)
Bases: object

Fuse groups and rebuild the part shapes. This tool puts all the part shapes into compounds before the Boolean operation.

Parameters

- **groups** (*collections.Sequence(afem.structure.group.Group* (page 261)*)*) – The groups.
- **fuzzy_val** (*float*) – Fuzzy tolerance value.
- **include_subgroup** (*bool*) – Option to recursively include parts from all subgroups.

Raises **ValueError** – If less than two groups are provided.

is_done

Returns *True* if operation is done, *False* if not.

Return type bool

shape

Returns The fused shape.

Return type *afem.topology.entities.Shape* (page 109)

13.4 Modify

13.4.1 DiscardByCref

class `afem.structure.modify.`**`DiscardByCref`**(*parts*)

Bases: `object`

Discard shapes of the parts by using their reference curves. An infinite solid is created at each end of the reference curve using the curve tangent. Any shape that has a centroid in these solids is removed. For a curve part edges are discarded, for a SurfacePart faces are discarded.

Parameters **`parts`** (*list (`afem.structure.entities.Part` (page 253))*) – The parts.

`was_modified`(*part*)

Check to see if part was modified.

Parameters **`part`** (`afem.structure.entities.Part` (page 253)) – The part.

Returns *True* if entities were discarded from the part, *False* otherwise.

Return type bool

13.5 Explore

13.5.1 ExploreParts

class `afem.structure.explore.`**`ExploreParts`**

Bases: `object`

Tool to explore parts.

static **`shared_edges`**(*parts*, *others*, *as_compound=False*)

Collect the shared edges between two sets of parts.

Parameters

- **`parts`** (*collections.Sequence(`afem.structure.entities.Part` (page 253))*) – The first set of parts.
- **`others`** (*collections.Sequence(`afem.structure.entities.Part` (page 253))*) – The other set of parts.
- **`as_compound`** (*bool*) – Option to return the shared edges as a compound.

Returns List of shared edges.

Return type list(*afem.topology.entities.Edge* (page 115)) or *afem.topology.entities.Compound* (page 119)

13.6 Fix

13.6.1 FixGroup

class `afem.structure.fix.`**`FixGroup`**(*group=None*, *precision=None*, *min_tol=None*, *max_tol=None*)

Bases: `object`

Attempt to fix the shapes of each part in an group using *FixShape* (page 164). Subgroups are included by default.

Parameters

- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group. If None then the active group is used.
- **precision** (*float*) – Basic precision value.
- **min_tol** (*float*) – Minimum allowed tolerance.
- **max_tol** (*float*) – Maximum allowed tolerance.

Raises **TypeError** – If an *Group* (page 261) instance is not found.

static limit_tolerance(*group=None*, *tol=1e-07*)

Limit tolerances for the group shapes.

Parameters

- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group. If None then the active group is used.
- **tol** (*float*) – Target tolerance.

Returns *True* if at least one tolerance of a sub-shape was modified.

Return type bool

Raises **TypeError** – If an *Group* (page 261) instance is not found.

static set_tolerance(*group=None*, *tol=10000000.0*)

Enforce tolerance on the given group.

Parameters

- **group** (*str or* afem.structure.group.Group (page 261) *or None*) – The group. If None then the active group is used.
- **tol** (*float*) – The tolerance.

Returns None.

Raises **TypeError** – If an *Group* (page 261) instance is not found.

13.7 Check

13.7.1 CheckPart

class afem.structure.check.**CheckPart**

Bases: object

Check structural components.

static is_part(*part*)

Check to see if the part is a structural component.

Parameters **part** – Part to check.

Returns *True* if part is a structural component, *False* if not.

Return type bool

static are_parts(**parts*)

Check to see if all entities are parts.

Parameters **parts** – Parts to check.

Returns *True* if all entities are parts, *False* if not.

Return type bool

static is_surface_part(*part*)

Check to see if part is surface-based part.

Parameters **part** –

Returns

static is_wing_part(*part*)

Check to see if the part is a wing structural component.

Parameters **part** – Part to check.

Returns *True* if part is a wing structural component, *False* if not.

Return type bool

static is_spar(*part*)

Check to see if the part is a spar.

Parameters **part** – Part to check.

Returns *True* if the part is a spar, *False* if not.

Return type bool

static is_rib(*part*)

Check to see if the part is a rib.

Parameters **part** – Part to check.

Returns *True* if the part is a rib, *False* if not.

Return type bool

13.8 Mesh

13.8.1 MeshVehicle

class `afem.structure.mesh.`**MeshVehicle**(*target_size=1.0*, *allow_quads=True*)

Bases: `object`

Tool for assisting in vehicle-level mesh generation. The “master” group will be used to define the top-level shape for meshing, which should include all the parts and therefore shapes to be meshed. Default mesh controls are applied to the top-level shape based on the target element size.

Parameters

- **target_size** (*float*) – Default global element size.
- **allow_quads** (*bool*) – Option to generate quad-dominated mesh.

shape

Returns The top-level shape.

Return type *afem.topology.entities.Shape* (page 109)

gen

Returns The mesh generator.

Return type *afem.smesh.entities.MeshGen* (page 192)

mesh

Returns The top-level mesh.

Return type *afem.smesh.entities.Mesh* (page 193)

add_control(*control*, *shape=None*)
Add a mesh control.

Parameters

- **control** (`afem.smesh.hypotheses.Hypothesis` (page 204)) – The control.
- **shape** (`afem.topology.entities.Shape` (page 109)) – The shape the control applies to. This can be a sub-shape of the master shape. If not provided then the master shape is used.

Returns Status of adding hypothesis.

Return type OCCT.SMESH.SMESH_Hypothesis.Hypothesis_Status

add_controls(*controls*, *shape=None*)
Add mesh controls to the shape.

Parameters

- **controls** (*collections.Sequence(*`afem.smesh.hypotheses.Hypothesis` (page 204)*)*) – The controls.
- **shape** (`afem.topology.entities.Shape` (page 109)) – The shape the control applies to. This can be a sub-shape of the master shape. If not provided then the master shape is used.

Returns Dictionary where the key is the control and the value is the control status.

Return type dict

Raises **ValueError** – If no shape is available to apply the control to.

set_number_segments_1d(*nseg*, *shape*)
Set the number of edge segments for the shape.

Parameters

- **nseg** (*int*) – The number of segments.
- **shape** (`afem.topology.entities.Shape` (page 109)) – The shape.

Returns None.

set_local_length_1d(*local_length*, *shape*)
Set the local length of edge segments for the shape.

Parameters

- **local_length** (*float*) – The local length.
- **shape** (`afem.topology.entities.Shape` (page 109)) – The shape.

Returns None.

set_max_length_1d(*max_length*, *shape*)

Set the max length of edge segments for the shape.

Parameters

- **max_length** (*float*) – The max length.
- **shape** (afem.topology.entities.Shape (page 109)) – The shape.

Returns None.

set_quadrangle_2d(*shape*)

Set the mesh control to use structured quadrangle mesh for the shape.

Parameters **shape** (afem.topology.entities.Shape (page 109)) – The shape. The algorithm is applied to each face of the shape only if it is applicable.

Returns None.

compute()

Compute the mesh.

Returns *True* if successful, *False* if not.

Return type bool

create_node_group(*shape*, *name='node_group'*)

Create a mesh node group from the shape.

Parameters

- **shape** (afem.topology.entities.Shape (page 109)) – The shape.
- **name** (*str*) – The group name.

Returns The node group.

Return type *afem.smesh.entities.MeshGroup* (page 202)

create_edge_group(*shape*, *name='edge_group'*)

Create a mesh edge group from the shape.

Parameters

- **shape** (afem.topology.entities.Shape (page 109)) – The shape.
- **name** (*str*) – The group name.

Returns The edge group.

Return type *afem.smesh.entities.MeshGroup* (page 202)

create_face_group(*shape*, *name='node_group'*)

Create a mesh face group from the shape.

Parameters

- **shape** (afem.topology.entities.Shape (page 109)) – The shape.
- **name** (*str*) – The group name.

Returns The face group.

Return type *afem.smesh.entities.MeshGroup* (page 202)

export_nastran(*fn*)

Export the mesh to a Nastran bulk data file.

Parameters **fn** (*str*) – The filename.

Returns None.

13.9 Utilities

`afem.structure.utils.`**`order_parts_by_id`**(*parts*)

Order the list of parts by id.

Parameters **`parts`** (*`list`* *(*`afem.structure.entities.Part` (page 253)*)*) – The parts.

Returns List of parts sorted by their ID.

Return type list(*afem.structure.entities.Part* (page 253))

`afem.structure.utils.`**`shape_of_entity`**(*entity*)

Get the shape of the entity. This method is useful if method inputs can either be a part or a shape. If the entity is already a shape it will be returned. If the entity is part the shape of the part will be returned. If the entity is a curve or surface then it will be converted to a shape.

Parameters **`entity`** (`afem.geometry.entities.Geometry` (page 34) *or* `afem.topology.entities.Shape` (page 109) *or* *`afem.base.entities.ShapeHolder`*) – The entity.

Returns The shape.

Return type *afem.topology.entities.Shape* (page 109)

CHAPTER

FOURTEEN

EXCHANGE

The afem.exchange package provides a number of common tools for data exchange including BREP (OpenCASCADE native format), STEP, IGES, and STL. The tools can be imported by:

```
from afem.exchange import *
```

Some specific use-case tools are provided including streamlined processing of STEP files from NASA's OpenVSP program. These data exchange tools can be used to import or export data to other CAD or CAE tools.

14.1 BREP

The afem.exchange.brep module contains two simple methods for reading and writing OpenCASCADE BREP files.

afem.exchange.brep.**write_brep**(*shape*, *fn*)
Write a BREP file using the shape.

Parameters

- **shape** (afem.topology.entities.Shape (page 109)) – The shape.
- **fn** (str) – The filename.

Returns None.

afem.exchange.brep.**read_brep**(*fn*)
Read a BREP file and return the shape.

Parameters **fn** (str) – The filename.

Returns The shape.

Return type *afem.topology.entities.Shape* (page 109)

14.2 STEP

class afem.exchange.step.**StepWrite**(*schema='AP203'*, *units=None*, *product_name=None*, *assembly_mode=None*)
Bases: object (page 302)

Write shape to a STEP file.

Parameters

- **schema** (str) – Schema for STEP file ('AP203', or 'AP214').

- **units** (*str or None*) – Units to convert STEP file to.
- **product_name** (*str or None*) – The name of the STEP product entry. If more than one product is generated during translation, then OpenCASCADE will automatically append a unique integer.
- **assembly_mode** (*int or None*) – Mode for writing assemblies (0, 1, or 2).

Note: The assembly modes are as follows:

- 0 (off, default): Writes STEP files without assemblies.
- 1(on): Writes all shapes in the form of STEP assemblies.
- 2(auto): Writes shapes having a structure of (possibly nested) compounds in the form of STEP assemblies, single shapes are written without assembly structures.

---

object

Returns The STEP writer object.

Return type OCCT.STEPControl.STEPControl_Writer

transfer (**shapes*)

Transfer and add the shapes to the exported entities.

Parameters **shapes** (afem.topology.entities.Shape (page 109)) – The shape(s).

Returns *True* if shape was transferred, *False* if not.

Return type bool

set_name (*shape, name*)

Set the name of the STEP entity for the given shape. The shape(s) should be transferred before naming them.

Parameters

- **shape** (afem.topology.entities.Shape (page 109)) – The shape (or sub-shape).
- **name** (*str*) – The name.

Returns *True* if name is set, *False* otherwise.

Return type bool

write (*fn='afem.stp'*)

Write the STEP file.

Parameters **fn** (*str*) – The filename.

Returns *True* if written, *False* if not.

Return type bool

class afem.exchange.step.**StepRead** (*fn*)

Bases: *object* (page 302)

Read a STEP file.

Parameters **fn** (*str*) – The file to read.

object

Returns The STEP reader object.

Return type OCCT.STEPControl.STEPControl_Reader

shape

Returns The main shape.

Return type *afem.topology.entities.Shape* (page 109)

name_from_shape(*shape*)

Attempt to extract the name for the STEP entity that corresponds to the shape.

Parameters **shape** (`afem.topology.entities.Shape` (page 109)) – The shape.

Returns The name or None if not found.

Return type str or None

14.3 IGES

class afem.exchange.iges.**IgesWrite**(*units='inch'*, *modecr=0*)

Bases: *object* (page 303)

Write shape to an IGES file.

Parameters

- **units** (*str*) – Units to convert IGES file to.
- **modecr** (*int*) – Option for writing faces. If 0, faces will be translated to IGES 144 (Trimmed Surface) entities. If 1, faces will be translated to IGES 510 (Face) entities and the IGES face will contain BRep entities.

object

Returns The IGES writer object.

Return type OCCT.IGESControl.IGESControl_Writer

add_shape(*shape*)

Add the shape to the exported entities.

Parameters **shape** (`afem.topology.entities.Shape` (page 109)) – The shape.

Returns *True* if shape was transferred, *False* if not.

Return type bool

add_geom(*geom*)

Add the geometry to the exported entities.

Parameters **geom** (*OCCT.Geom.Geom_Geometry*) – The geometry.

Returns *True* if shape was transferred, *False* if not.

Return type bool

Note: The input type in `Geom_Geometry` and not the wrapper.

write(*fn='afem.igs'*)

Write the IGES file.

Parameters **fn** (*str*) – The filename.

Returns *True* if written, *False* if not.

Return type bool

class `afem.exchange.iges.`**`IgesRead`**(*fn*)

Bases: `object` (page 304)

Read an IGES file.

Parameters **`fn`** (`str`) – The file to read.

`object`

Returns The IGES reader object.

Return type OCCT.IGESControl.IGESControl_Reader

`shape`

Returns The main shape.

Return type *afem.topology.entities.Shape* (page 109)

14.4 STL

class `afem.exchange.stl.`**`StlWrite`**

Bases: `OCCT.StlAPI.StlAPI_Writer`

Export shape to STL file.

`write`(*shape*, *fn*)

Converts shape to STL format and writes to a file.

Parameters

- **`shape`** (`afem.topology.entities.Shape` (page 109)) – The shape.
- **`fn`** (`str`) – The filename.

Returns None.

14.5 XDE

class `afem.exchange.xde.`**`XdeDocument`**(*binary=True*)

Bases: `object`

Wrapper class to work with Extended Data Exchange.

Parameters **`binary`** (`bool`) – If *True*, the document will be saved in a binary format. If *False*, the document will be saved in an XML format.

`main_label`

Returns The main label of the document.

Return type *afem.exchange.xde.XdeLabel* (page 308)

`shapes_label`

Returns The shapes label of the document.

Return type *afem.exchange.xde.XdeLabel* (page 308)

open (*fn*)
Open a document.

Parameters **fn** (*str*) – The filename.

Returns *True* if opened, *False* if not.

Return type bool

save_as (*fn*)
Save the document.

Parameters **fn** (*str*) – The filename.

Returns *True* if sucessfully saved, *False* if not.

Return type bool

close ()
Close the document.

Returns None.

read_step (*fn*)
Read and translate a STEP file.

Parameters **fn** (*str*) – The filename.

Returns The shapes label.

Return type afem.exchange.xde.Label.

Raises **RuntimeError** – If the file cannot be read.

transfer_step (*schema='AP203'*, *units=None*)
Transfer the document in preparation for STEP export.

Parameters

- **schema** (*str*) – Schema for STEP file ('AP203', or 'AP214').
- **units** (*str or None*) – Units to convert STEP file to.

Returns *True* if transferred, *False* otherwise.

Return type bool

set_shape_name (*shape*, *name*)
Set the name of the STEP entity for the given shape. The shape(s) should be transferred before naming them.

Parameters

- **shape** (afem.topology.entities.Shape (page 109)) – The shape (or sub-shape).
- **name** (*str*) – The name.

Returns *True* if name is set, *False* otherwise.

Return type bool

write_step (*fn*, *schema='AP203'*, *units=None*)
Write the document to a STEP file.

Parameters

- **fn** (*str*) – The filename.

- **schema** (*str*) – Schema for STEP file ('AP203', or 'AP214').
- **units** (*str or None*) – Units to convert STEP file to.

Returns *True* if written successfully, *False* otherwise.

is_top_level (*label*)
Check if label is top-level as opposed to a component of assembly or a sub-shape.

Parameters **label** (afem.exchange.xde.XdeLabel (page 308)) – The label

Returns *True* if top-level, *False* otherwise.

Return type bool

is_sub_shape (*label*, *shape*)
Check if the shape is a sub-shape of the shape stored on the label.

Parameters

- **label** (afem.exchange.xde.XdeLabel (page 308)) – The label
- **shape** (afem.topology.entities.Shape (page 109)) – The shape.

Returns *True* if a sub-shape, *False* otherwise.

Return type bool

find_shape (*shape*, *find_instance=False*)
Find the label corresponding to the shape. This method searches only top-level shapes.

Parameters

- **shape** (afem.topology.entities.Shape (page 109)) – The shape.
- **find_instance** (*bool*) – If *False*, search for the non-located shape in an assembly. If *True*, search for the shape with the same location.

Returns The shape label if found, *None* otherwise.

Return type *afem.exchange.xde.XdeLabel* (page 308)

new_shape ()
Create a new top-level label.

Returns The label.

Return type *afem.exchange.xde.XdeLabel* (page 308)

set_shape (*label*, *shape*)
Set the shape of the top-level label.

Parameters

- **label** (afem.exchange.xde.XdeLabel (page 308)) – The label.
- **shape** (afem.topology.entities.Shape (page 109)) – The shape.

Returns None.

add_shape (*shape*, *name=None*, *make_assy=True*)
Add a new top-level shape.

Parameters

- **shape** (afem.topology.entities.Shape (page 109)) – The shape.
- **name** (*str*) – The label name.

- **make_assy** (*bool*) – If *True*, then treat compounds as assemblies.

Returns The shape label.

Return type *afem.exchange.xde.XdeLabel* (page 308)

remove_shape(*label*, *remove_completely=True*)
Remove a shape.

Parameters

- **label** (`afem.exchange.xde.XdeLabel` (page 308)) – The label.
- **remove_completely** (*bool*) – If *True*, removes all shapes. If *False*, only remove the shape with the same location.

Returns *True* if removed, or *False* if not removed because it is not a free or top-level shape.

Return type bool

get_shapes()
Get a list containing all top-level shapes.

Returns List of top-level shapes.

Return type list(*afem.exchange.xde.XdeLabel* (page 308))

get_shape_by_name(*name*)
Get a shape label by its name. This only applies to top-level shapes and will return the first match.

Parameters **name** (*str*) – The name.

Returns The label or *None* if not found.

Return type *afem.exchange.xde.XdeLabel* (page 308) or None

find_subshape(*label*, *shape*)
Find a label for the sub-shape stored on the given label.

Parameters

- **label** (`afem.exchange.xde.XdeLabel` (page 308)) – The label.
- **shape** (`afem.topology.entities.Shape` (page 109)) – The sub-shape.

Returns The sub-shape label if found, *None* otherwise.

Return type *afem.exchange.xde.XdeLabel* (page 308) or None

add_subshape(*label*, *shape*, *name=None*)
Add a label for a sub-shape stored on the shape of the label.

Parameters

- **label** (`afem.exchange.xde.XdeLabel` (page 308)) – The label.
- **shape** (`afem.topology.entities.Shape` (page 109)) – The sub-shape.
- **name** (*str*) – The name of the sub-shape label.

Returns The sub-shape label.

Return type *afem.exchange.xde.XdeLabel* (page 308)

set_auto_naming(*mode*)
Set the option to auto-name shape labels. This only applies to top-level shapes.

Parameters **mode** (*bool*) – The mode. If *True*, added shapes are automatically named based on their type (e.g., "SOLID", "SHELL", etc.).

Returns None.

class `afem.exchange.xde.`**`XdeLabel`**(*label*)

Bases: *`object`* (page 308)

Wrapper class for OpenCASCADE TDF_Label.

Parameters **`OCCT.TDF.TDF_Label`** – The label.

`object`

Returns The underlying object.

Return type OCCT.TDF.TDF_Label

`tag`

Returns The label tag.

Return type int

`depth`

Returns The depth of this label.

Return type int

`father`

Returns The father label.

Return type *afem.exchange.xde.XdeLabel* (page 308)

`has_child`

Returns *True* if label has children, *False* if not.

Return type bool

`nb_children`

Returns The number of children.

Return type int

`is_null`

Returns Check if the label is null.

Return type bool

`is_root`

Returns Check if the label is a root.

Return type bool

`root`

Returns The root label of this label.

Return type *afem.exchange.xde.XdeLabel* (page 308)

`name`

Returns The label name.

Return type str or None

`shape`

Returns The label shape.

Return type *afem.topology.entities.Shape* (page 109) or None

string

Returns The label string.

Return type str or None

color

Returns The label color.

Return type OCCT.Quantity.Quantity.Color or None

children_iter

Returns Yield the children labels.

Return type collections.Iterable(*afem.exchange.xde.XdeLabel* (page 308))

is_equal(*other*)

Check if this label equals the other.

Parameters **other** (`afem.exchange.xde.XdeLabel` (page 308)) – The other label.

Returns *True* if equal, *False* if not.

Return type bool

is_descendant(*other*)

Check if this label is a descendant of the other.

Parameters **other** (`afem.exchange.xde.XdeLabel` (page 308)) – The other label.

Returns *True* if a descendant, *False* if not.

Return type bool

new_child()

Create a new child label of this one.

Returns Child label.

Return type *afem.exchange.xde.XdeLabel* (page 308)

find_child(*tag*)

Find a child label with `tag`.

Parameters **tag** (`int`) – The child label tag. If the tag doesn't exist then a new child will be created using that tag.

Returns Child label.

Return type *afem.exchange.xde.XdeLabel* (page 308)

set_name(*name*)

Set label name.

Parameters **name** (`str`) – The name.

Returns None.

set_string(*string*)

Set label name.

Parameters **string** (`str`) – The string.

Returns None.

set_color(*color*)
Set label color.

Parameters **color** (*OCCT.Quantity.Quantity_Color*) – The color.

Returns None.

14.6 OpenVSP

The `ImportVSP` tool is developed specifically for translating STEP files exported from the OpenVSP program. The primary motivation for this tool can be summarized by needs:

1. Automatically translate the OpenVSP standard components (e.g., MS_Wing, Fuselage, etc.) into watertight solid shapes.
2. Communicate additional metadata about the OpenVSP model and components (e.g., name, type, etc.) through the STEP file.

The first need is accomplished by essentially pre-processing the OpenVSP geometry from the STEP file based on a set of assumptions and rules:

- Surfaces of unique OpenVSP components are grouped into geometric sets in the STEP file. These are translated to compounds and it is assumed that they define a single OpenVSP component (or one body of a component if symmetric).
- Each surface of the referenced by the geometric set is first checked to see if it is planar. If so, the underlying surface is replaced with a plane instead of the B-Spline surface. This may be applicable to surfaces representing wing caps or thick trailing edges.
- The surfaces are sewn together and at this point should form a closed solid shape.

In a script-based environment, the user cannot "point and click" to identify which solid body represents a particular component. For that reason, a forked version of OpenVSP was made and modified to include useful metadata in the STEP file. The metadata includes the component name, type, and additional degenerate geometry like a wing reference surface. All of this can be optionally exported on the STEP export screen (default is off). Without this metadata, solid bodies can still be built, but are given generic names like "Body.1" and it will be left to the user to decipher which represents the original OpenVSP component.

For now, the forked OpenVSP version with metadata can be found at the Laughlin Research OpenVSP Github[12] under the *step_metadata_support* and *step_metadata_support_v3.5.0* branches. Support for version 3.5.0 was provided since the underlying surface parametrization seemed to have changed afterwards and OpenCASCADE performance suffered.

Windows builds for these modified versions of OpenVSP can be found here:

- OpenVSP 3.5.0 w/ metadata[13]
- OpenVSP 3.15.0 w/ metadata[14]

Both of these builds include the Python API.

class afem.exchange.vsp.**ImportVSP**(*fn=None*, *divide_closed=True*, *bspline_restrict=False*, *reloft=False*, *tol=0.01*)
Bases: `object`

Tool for importing, pre-processing, and translating OpenVSP models.

[12] https://github.com/LaughlinResearch/OpenVSP
[13] https://ci.appveyor.com/project/LaughlinResearch/openvsp/build/3.5.0.9
[14] https://ci.appveyor.com/project/LaughlinResearch/openvsp/build/3.15.0.8

Parameters

- **fn** (*str*) – The full path to a file to import. If provided, the file extension will be used to guess which import method to call. For example, if the file ended with 'stp' or 'step', then the *import_step()* method is called during initialization.
- **divide_closed** (*bool*) – Option to divide closed faces.
- **bspline_restrict** (*bool*) – Option to attempt to refit the underlying OpenVSP surfaces with surfaces of higher continuity. This may take a long time to run and no guarantee it will be successful. At the moment this is only used on known wing components. This method is experimental.
- **reloft** (*bool*) – For wings that are not split, this option will extract isocurves at each spanwise cross section, tessellate the curve using *tol*, and then approximate the section using a C1 continuous curve. These curves are then used to generate a new wing surface. This method is experimental.
- **tol** (*float*) – Tolerance for approximation if *bspline_restrict* or *reloft* is *True*.

Raises **TypeError** – If a file is provided but the extension is not recognized or supported.

bodies

Returns The dictionary of Body instances where the key is the component name and the value is the Body.

Return type dict

all_bodies

Returns List of Body instances.

Return type list(*afem.oml.entities.Body* (page 228))

has_invalid

Returns Check if any invalid shapes were found during import.

Return type bool

invalid_shapes

Returns List of invalid shapes found during import.

Return type list(*afem.topology.entities.Shape* (page 109))

clear()

Clear any translated data.

Returns None.

get_body(*name*)

Return Body by name.

Parameters **name** (*str*) – Body name.

Returns OML body.

Return type *afem.oml.entities.Body* (page 228)

Raises **KeyError** – If *name* is not present.

get_bodies()

Get a copy of the underlying dictionary storing the Body instances.

Returns Dictionary where the key is the component name and the value is the Body.

Return type dict

import_step(*fn*)
Import a STEP file generated by the OpenVSP version that has been modified to include metadata.

Parameters **fn** (*str*) – The full path to the file.

Returns None.

export_step(*fn*, *label_solids=True*, *label_faces=False*, *names=None*)
Export the OpenVSP model as a STEP file using Extended Data Exchange. Each OpenVSP component will be a named product in the STEP file structure.

Parameters

- **fn** (*str*) – The filename.
- **label_solids** (*bool*) – Option to label the solid bodies in the STEP entity. The name will be the same as the OpenVSP component.
- **label_faces** (*bool*) – Option to label the faces in each of the solids. Each face of the solid body will be labeled "Face 1", "Face 2", etc.
- **names** (*collections.Sequence(str) or None*) – List of Body names that will be included in export. If *None* then all are exported.

Returns None.

save_bodies(*fn*)
Save the Body instances.

Parameters **fn** (*str*) – The filename. The extension will be ".xbf" and appended if not provided.

Returns *True* if saved, *False* otherwise.

Return type bool

Note: This method only saves the label, shape, and color of the Body. User-defined metadata is currently not saved.

static rebuild_wing_solid(*srfs*, *divide_closed=True*, *reloft=False*, *tol=0.01*)
Rebuild a solid shape from the OpenVSP wing surface(s). If only one surface is provided then it is assumed that a single surface models the OML and it will be split and modified at the root, tip, and trailing edge. This single surface should have similar form and parametrization as the original OpenVSP surface. If more than once surface is provided then it is assumed that the surfaces were split during OpenVSP export and are simply sewn together to form the solid.

Parameters

- **srfs** (*collections.Sequence(*afem.geometry.entities.Surface (page 49)*)*) – The wing surface(s) used to rebuild the solid.
- **divide_closed** (*bool*) – Option to divide closed faces.
- **reloft** (*bool*) – For wings that are not split, this option will extract isocurves at each spanwise cross section, tessellate the curve using *tol*, and then approximate the section using a C1 continuous curve. These curves are then used to generate a new wing surface. This method is experimental.
- **tol** (*float*) – Tolerance for approximation if *bspline_restrict* or *reloft* is *True*.

Returns The new solid.

Return type *afem.topology.entities.Solid* (page 118)

Raises **`ValueError`** – If no surfaces are provided.

`static process_sref`(*compound*)

Process a wing reference surface. The compound should contain a single face. The underlying surface of this faces will be used to generate a new NurbsSurface. Since the OpenVSP surface uses uniform parametrization, a chord line is extracted at each unique knot in the v-direction. Then a linear surface is lofted through these lines to form a bilinear surface where the u-direction is chordwise and the v-direction is spanwise.

Parameters **`compound`** (`afem.topology.entities.Compound` (page 119)) – The compound.

Returns The reference surface.

Return type *afem.geometry.entities.NurbsSurface* (page 53)

`static rebuild_wing_sref`(*srf*)

Attempt to rebuild a wing reference surface using the given OML surface. This method is intended to operate on OpenVSP surfaces that define the OML of a wing component and/or have similar parametrization.

Parameters **`srf`** (`afem.geometry.entities.NurbsSurface` (page 53)) – The surface.

Returns The reference surface.

Return type *afem.geometry.entities.NurbsSurface* (page 53)

14.7 NASTRAN

`afem.exchange.nastran.`**`export_bdf`**(*the_mesh*, *fn*)

Export groups of parts to Nastran bulk data format (only nodes and elements). Dummy materials and properties are applied to enable import into some pre-processors. Intended for development and debugging. Only supports tri and quad elements for now.

Parameters

- **`the_mesh`** (`afem.smesh.entities.Mesh` (page 193)) – The mesh.
- **`fn`** (*`str`*) – The filename.

Returns *True* if done, *False* if not.

Return type bool

Warning: This method is experimental and provides minimal Nastran export capability.

CHAPTER

FIFTEEN

GRAPHICS

The afem.graphics package provides a base class for viewable objects as well as a minimal tool to visualize AFEM shapes, geometry, and meshes. The most used tool will be the *Viewer* (page 315) class which can be imported by:

```
from afem.graphics import Viewer
```

An instance can be created, entities added, and then viewed similar to the following process:

```
gui = Viewer()
gui.add(*args)
gui.start()
```

The add() method will try and process the argument given its type and is the most generic method to call. More specific methods giving the user more control are described in the class documentation. When the start() method is called an application is launched to view the contents and program execution will stop.

The viewer instance will be destroyed if exited using the "exit" button in the top right. If the user wants to continue processing with the same instance, simply press the c key on the keyboard to continue processing. Hotkeys are available as shown:

Table 1: Hotkeys for Viewer

Key	Description
c	Continue processing.
f	Fit the contents.
s	Shaded view.
w	Wireframe view.
i	Isometric view.
t	Top view.

class afem.graphics.display.**Viewer**(*width=800*, *height=600*)

Bases: OCCT.Visualization.Basic.BasicViewer

Simple tool for viewing entities.

display_item(*item*)

Display a type derived from ViewableItem.

Parameters **item** (afem.base.entities.ViewableItem (page 11)) – The item.

Returns The AIS_Shape created for the item.

Return type OCCT.AIS.AIS_Shape

display_group(*group*, *include_subgroup=True*)
Display all parts of a group.

Parameters

- **group** (`afem.structure.group.Group` (page 261)) – The group.
- **include_subgroup** (*bool*) – Option to recursively include parts from any subgroups.

Returns None.

add(**items*)
Add items to be displayed.

Parameters **items** (`afem.base.entities.ViewableItem` (page 11) *or OCCT.TopoDS.TopoDS_Shape or* `afem.structure.group.Group` (page 261) *or OCCT.SMESH.SMESH_Mesh or OCCT.SMESH.SMESH_subMesh or* `afem.smesh.entities.Mesh` (page 193) *or* `afem.smesh.entities.SubMesh` (page 199) *or* `afem.smesh.entities.MeshGroup` (page 202) *or* `afem.structure.mesh.MeshVehicle` (page 296)) – The items.

Returns None.

CHAPTER SIXTEEN

ACKNOWLEDGMENTS

AFEM development was partially funded by NASA through award No. NNX15AV85A under the Transformational Tools and Technology (TTT) project.

CHAPTER

SEVENTEEN

HOW TO CITE AFEM

The following Bibtex template can be used to cite AFEM in scientific and engineering publications:

```
@misc{AFEM,
   author = {Laughlin Research, LLC},
   year = {2018},
   note = {https://github.com/LaughlinResearch/AFEM},
   title = {AFEM -- Airframe Finite Element Modeler}
}
```

PYTHON MODULE INDEX

INDEX