omf Documentation

Release 2.0.0a0

Global Mining Guidelines Group

Mar 23, 2023

Contents

Version: 2.0.0a0

API library for Open Mining Format, a new standard for mining data backed by the Global Mining Guidelines Group.

Warning: Pre-Release Notice

Version 2 of the Open Mining Format (OMF) and the associated Python API is under active development, and subject to backwards-incompatible changes at any time. The latest stable release of Version 1 is available on PyPI.

CHAPTER 1

Why?

An open-source serialization format and API library to support data interchange across the entire mining community.

CHAPTER 2

Scope

This library provides an abstracted object-based interface to the underlying OMF serialization format, which enables rapid development of the interface while allowing for future changes under the hood.

CHAPTER 3

Goals

- The goal of Open Mining Format is to standardize data formats across the mining community and promote collaboration
- The goal of the API library is to provide a well-documented, object-based interface for serializing OMF files

CHAPTER 4

Alternatives

OMF is intended to supplement the many alternative closed-source file formats used in the mining community.

CHAPTER 5

Connections

This library makes use of the properties open-source project, which is designed and publicly supported by Seequent.

CHAPTER 6

Installation

To install the repository, ensure that you have pip installed and run:

```
pip install --pre omf
```

Or from github:

```
git clone https://github.com/gmggroup/omf.git
cd omf
pip install -e .
```

CHAPTER 7

3D Visualization

To easily visualize OMF project files and data objects in a pure Python environment, check out omfvista which provides a module for loading OMF datasets into PyVista mesh objects for 3D visualization and analysis.

Contents:

7.1 OMF API Index

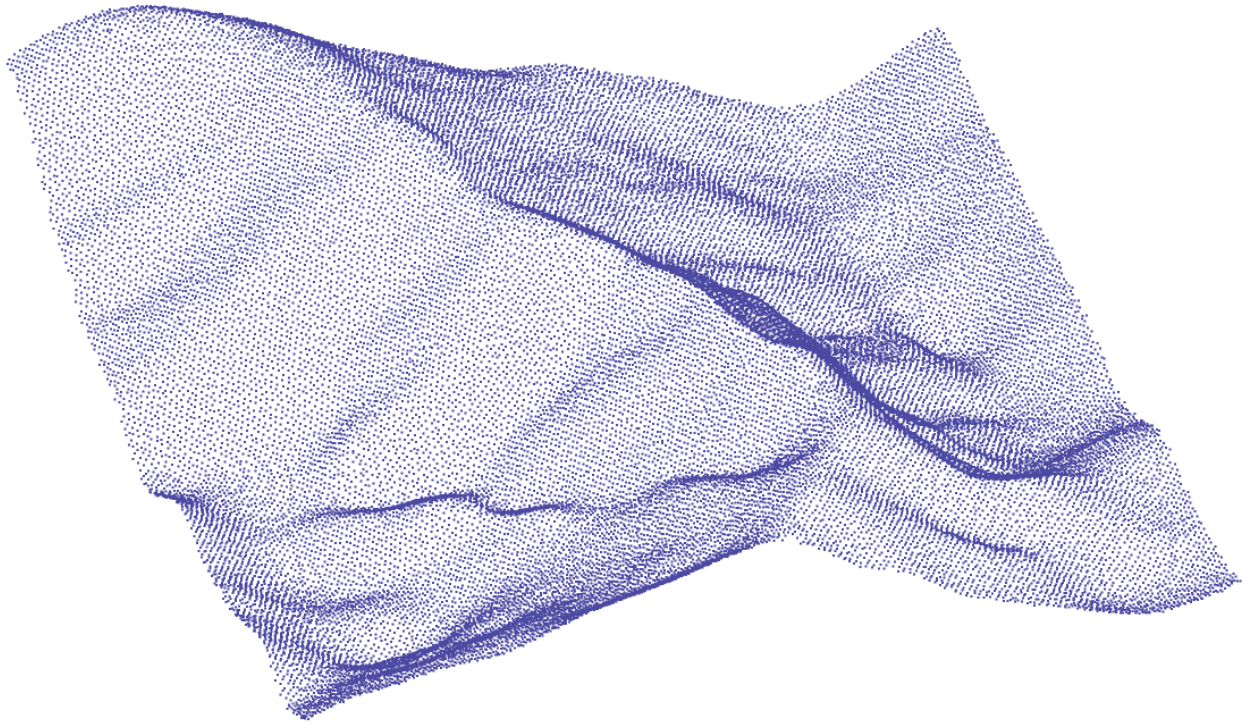
The OMF API contains tools for creating *Projects* and adding *PointSets*, *LineSets*, *Surfaces*, *Block Models*, and *Composites*. These different elements may have *Attributes* or image *Textures*.

7.1.1 Projects

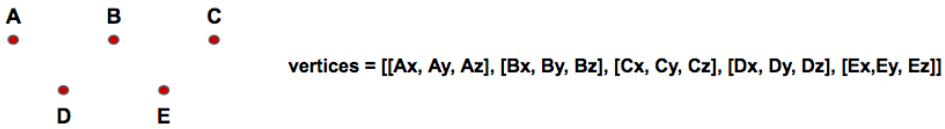
Projects contain a list of *PointSets*, *LineSets*, *Surfaces*, *Block Models*, and *Composites*. Projects can be serialized and deserialized to file using `omf.fileio.save()` and `omf.fileio.load()`.

For more details on how to build a project, see the *OMF API Example*.

7.1.2 PointSets



Element



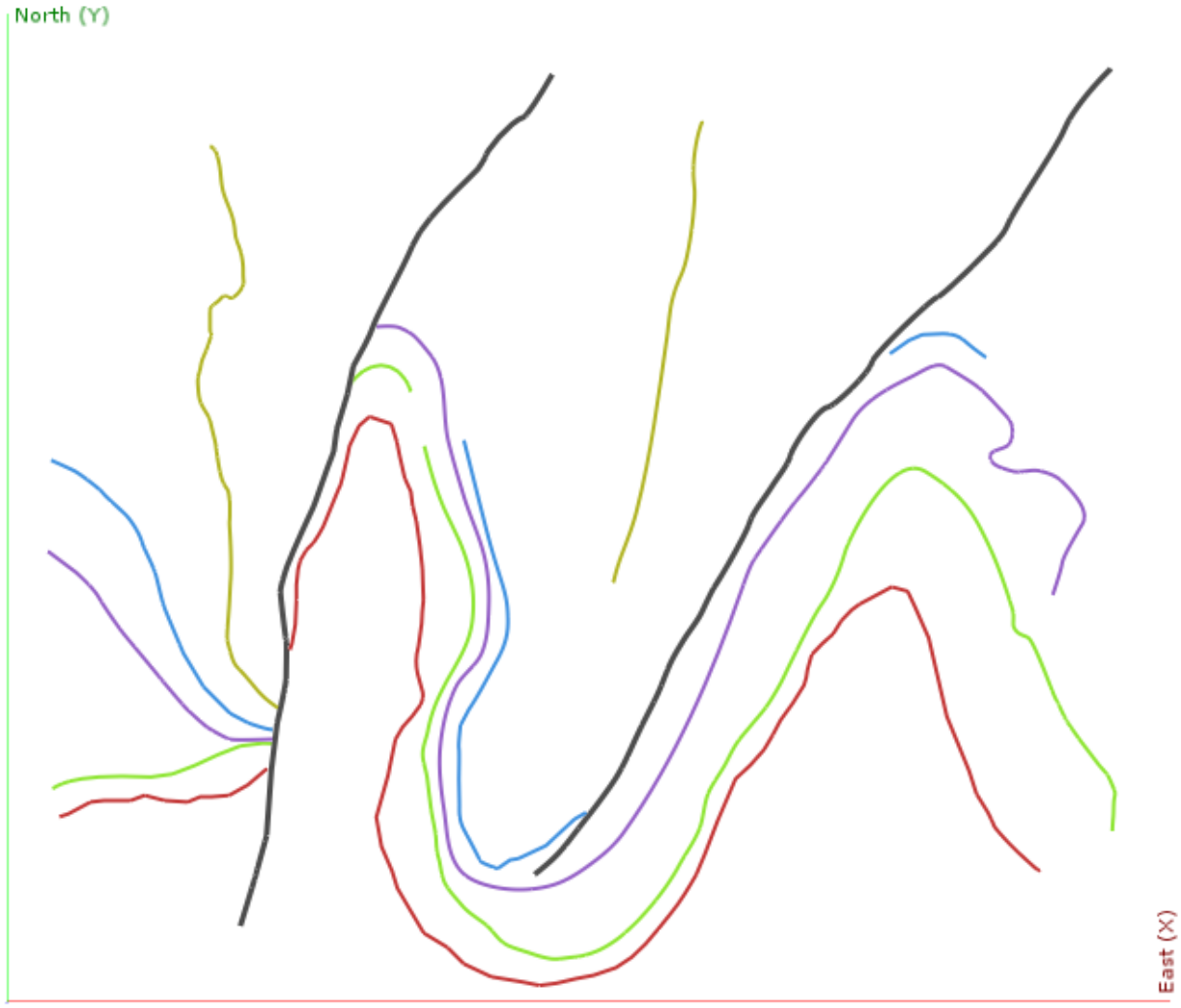
Attributes

Attributes is a list of *attributes*. For PointSets, only `location='vertices'` is valid.

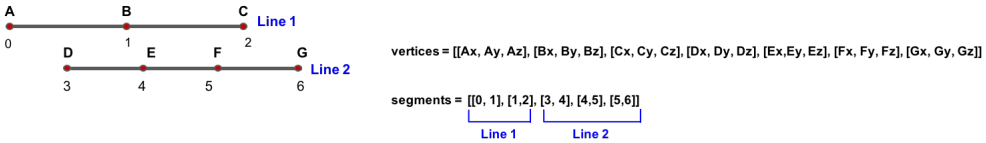
Textures

Textures is a list of *Textures* mapped to the PointSet.

7.1.3 LineSets



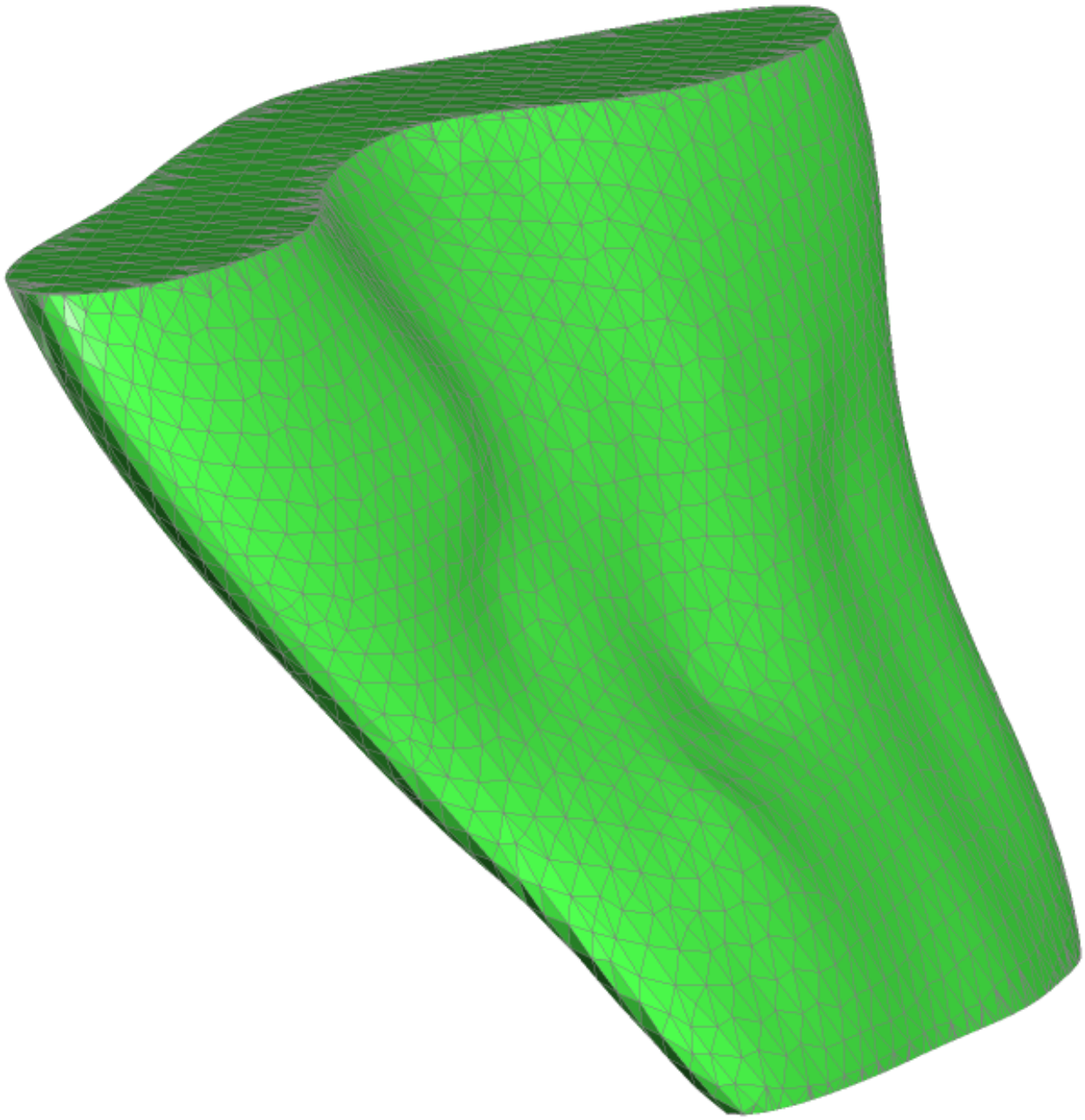
Element



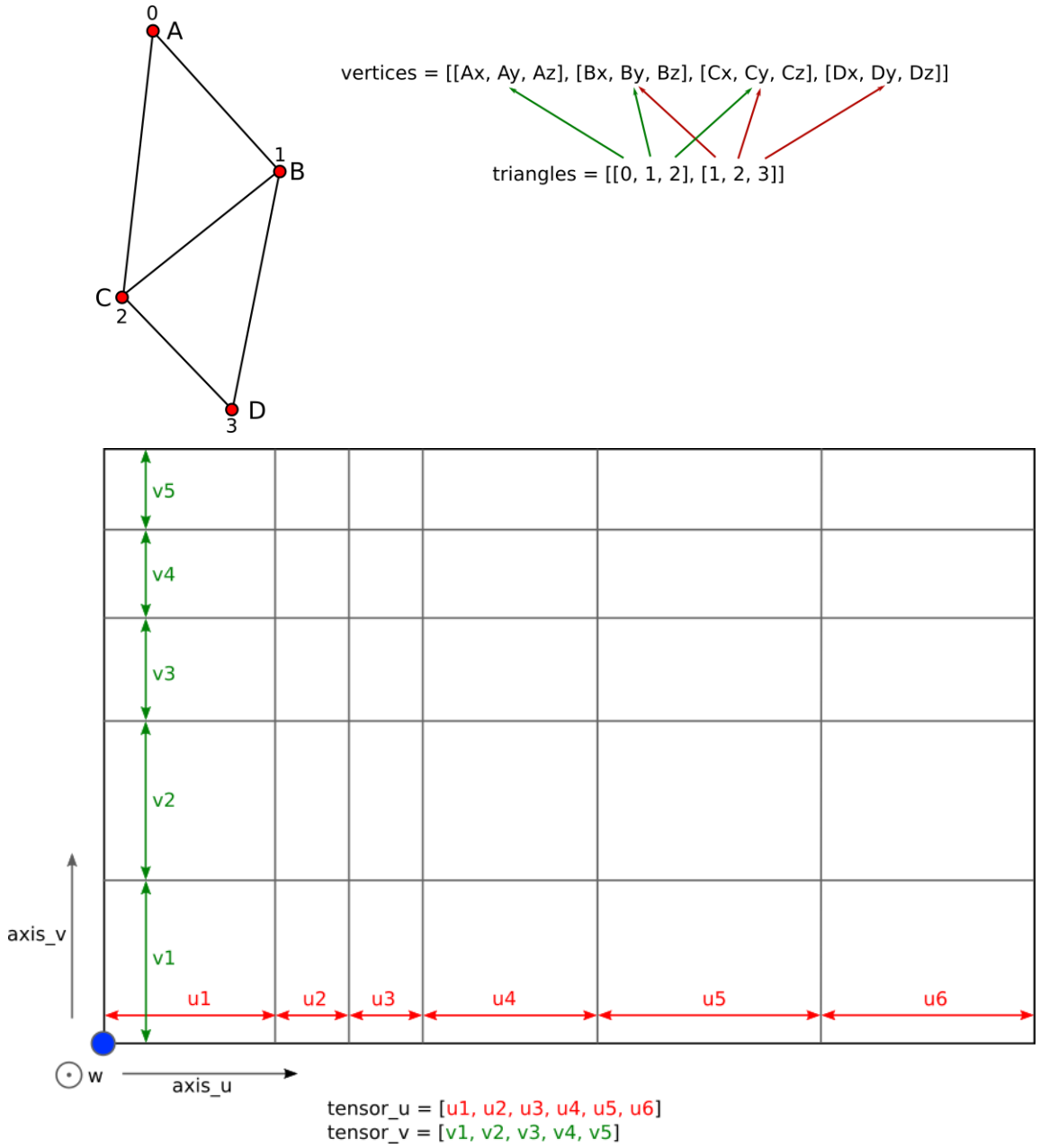
Attributes

Attributes is a list of *attributes*. For LineSets, `location='vertices'` and `location='segments'` are valid.

7.1.4 Surfaces



Elements



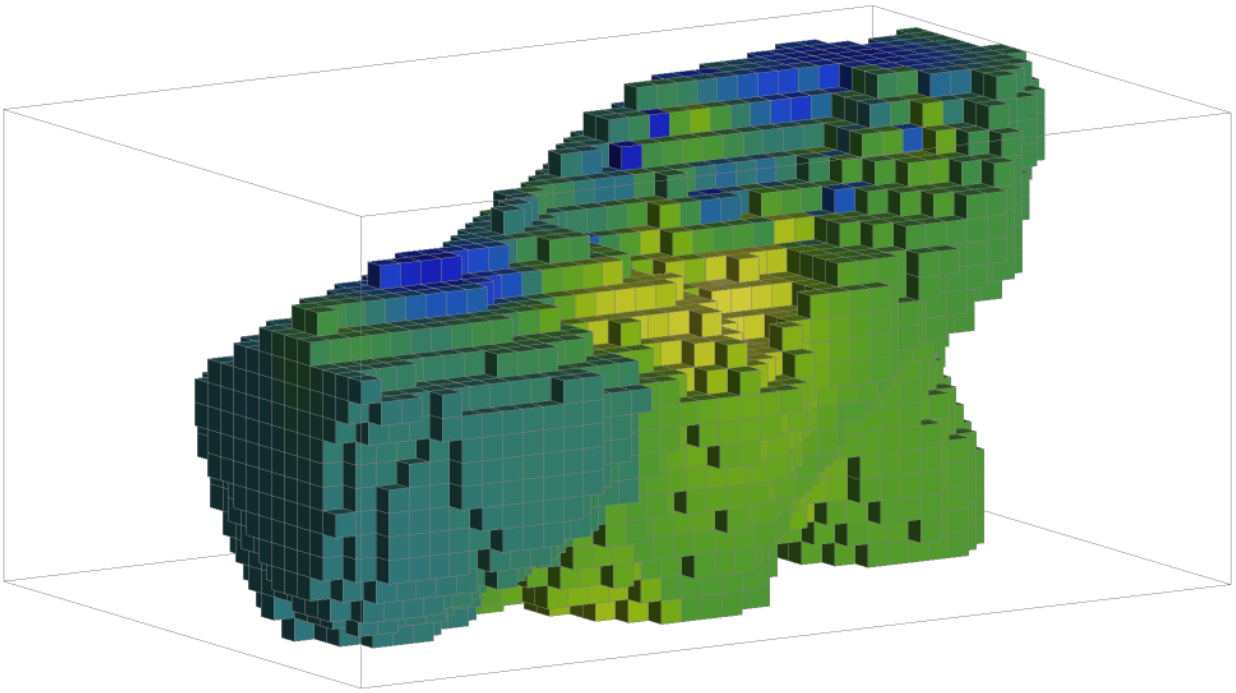
Attributes

Attributes is a list of *attributes*. For Surfaces, `location='vertices'` and `location='faces'` are valid.

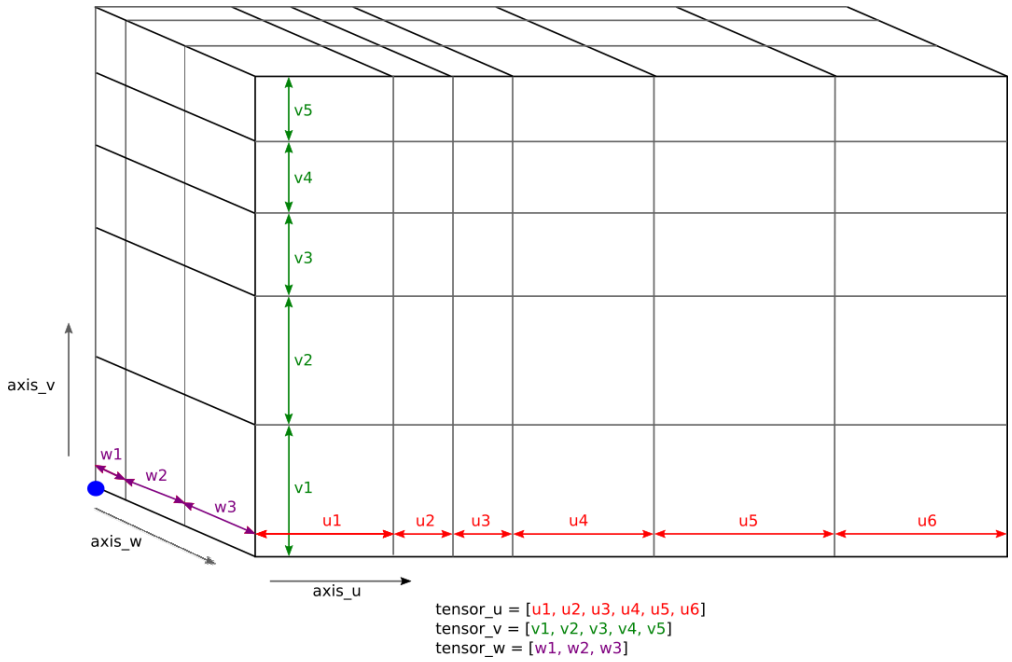
Textures

Textures is a list of *Textures* mapped to the Surface.

7.1.5 Block Models



Element



Attributes

Attributes is a list of *attributes*. For block models, `location='parent_blocks'` and `location='sub_blocks'` are valid.

7.1.6 Composites

Composites are used to compose multiple other elements into a single, more complex, grouped object.

Element

Attributes

Attributes is a list of *attributes*. For Composite Elements, only `location='elements'` is valid. However, attributes may also be defined on the child `elements`

7.1.7 Attributes

ProjectElements include a list of ProjectElementAttribute. These specify mesh location ('vertices', 'faces', etc.) as well as the array, name, and description. See class descriptions below for specific types of Attributes.

Mapping attribute array values to a mesh is straightforward for unstructured meshes (those defined by vertices, segments, triangles, etc); the order of the attribute array corresponds to the order of the associated mesh parameter. For grid meshes, however, mapping 1D attribute array to the 2D or 3D grid requires correctly ordered ijk unwrapping.

NumericAttribute

VectorAttribute

StringAttribute

CategoryAttribute

ContinuousColormap

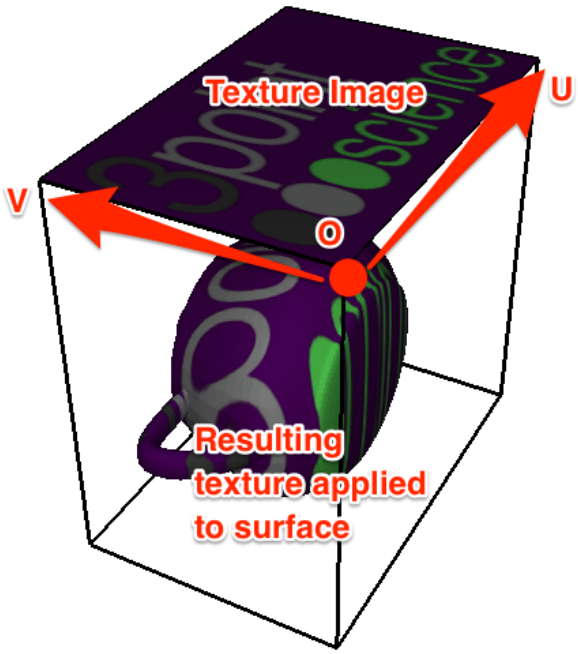
DiscreteColormap

CategoryColormap

7.1.8 Textures

Projected Texture

Projected textures are images that exist in space and are mapped to their corresponding elements. Unlike attributes, they do not need to correspond to mesh nodes or cell centers. This image shows how textures are mapped to a surface. Their position is defined by a corner and axis vectors then they are mapped laterally to the element position.



Like attributes, multiple textures can be applied to a element; simply provide a list of textures. Each of these textures provides a corner point and two extent vectors for the plane defining where images rests. The *axis_** properties define the extent of that image out from the corner. Given a rectangular PNG image, the *corner* is the bottom left, *corner* + *axis_u* is the bottom right, and *corner* + *axis_v* is the top left. This allows the image to be rotated and/or skewed. These values are independent of the corresponding Surface; in fact, there is nothing requiring the image to actually align with the Surface.

UV Mapped Textures

Rather than being projected onto points or a surface, UV Mapped Textures are given normalized UV coordinates which correspond to element vertices. This allows arbitrary mapping of images to surfaces.

Image

7.1.9 Array Types

Array

StringList

ArrayInstanceProperty

7.1.10 Other Classes

Project Element

Available elements are *PointSets*, *LineSets*, *Surfaces*, *Block Models*, and *Composites*; *Projects* are built with elements.

Project Element Attribute

Content Model

Base Model

Metadata Classes

7.2 OMF API Example

This (very impractical) example shows usage of the OMF API.

Also, this example builds elements all at once. They can also be initialized with no arguments, and properties can be set one-by-one (see code snippet at bottom of page).

```
import datetime
import numpy as np
import os
import png
import omf

# setup sample files
dir = os.getcwd()
png_file = os.path.join(dir, "example.png")
omf_file = os.path.join(dir, "example.omf")
for f in (png_file, omf_file):
    if os.path.exists(f):
        os.remove(f)
img = ["110010010011", "101011010100", "110010110101", "100010010011"]
img = [[int(val) for val in value] for value in img]
writer = png.Writer(len(img[0]), len(img), greyscale=True, bitdepth=16)
with open(png_file, "wb") as file:
    writer.write(file, img)

proj = omf.Project(
    name="Test project",
    description="Just some assorted elements",
)
pts = omf.PointSet(
    name="Random Points",
    description="Just random points",
    vertices=np.random.rand(100, 3),
    attributes=[
        omf.NumericAttribute(
            name="rand attr",
            array=np.random.rand(100),
```

(continues on next page)

(continued from previous page)

```
            location="vertices",
        ),
        omf.NumericAttribute(
            name="More rand attr",
            array=np.random.rand(100),
            location="vertices",
        ),
    ],
    textures=[
        omf.ProjectedTexture(
            name="test image",
            image=png_file,
            origin=[0, 0, 0],
            axis_u=[1, 0, 0],
            axis_v=[0, 1, 0],
        ),
        omf.ProjectedTexture(
            name="test image",
            image=png_file,
            origin=[0, 0, 0],
            axis_u=[1, 0, 0],
            axis_v=[0, 0, 1],
        ),
    ],
    metadata={
        "color": "green",
    },
)
lin = omf.LineSet(
    name="Random Line",
    vertices=np.random.rand(100, 3),
    segments=np.floor(np.random.rand(50, 2) * 100).astype(int),
    attributes=[
        omf.NumericAttribute(
            name="rand vert attr",
            array=np.random.rand(100),
            location="vertices",
        ),
        omf.NumericAttribute(
            name="rand segment attr",
            array=np.random.rand(50),
            location="segments",
        ),
    ],
    metadata={
        "color": "#0000FF",
    },
)
surf = omf.Surface(
    name="trisurf",
    vertices=np.random.rand(100, 3),
    triangles=np.floor(np.random.rand(50, 3) * 100).astype(int),
    attributes=[
        omf.NumericAttribute(
            name="rand vert attr",
            array=np.random.rand(100),
            location="vertices",
```

(continues on next page)

(continued from previous page)

```
        ),
        omf.NumericAttribute(
            name="rand face attr",
            array=np.random.rand(50),
            location="faces",
        ),
    ],
    metadata={
        "color": [100, 200, 200],
    },
)
grid = omf.TensorGridSurface(
    name="gridsurf",
    tensor_u=np.ones(10).astype(float),
    tensor_v=np.ones(15).astype(float),
    origin=[50.0, 50.0, 50.0],
    axis_u=[1.0, 0, 0],
    axis_v=[0, 0, 1.0],
    offset_w=np.random.rand(11 * 16),
    attributes=[
        omf.NumericAttribute(
            name="rand vert attr",
            array=np.random.rand(11 * 16),
            location="vertices",
        ),
        omf.NumericAttribute(
            name="rand face attr",
            array=np.random.rand(10 * 15),
            location="faces",
        ),
    ],
    textures=[
        omf.ProjectedTexture(
            name="test image",
            image=png_file,
            origin=[2.0, 2.0, 2.0],
            axis_u=[5.0, 0, 0],
            axis_v=[0, 2.0, 5.0],
        ),
    ],
)
vol = omf.TensorGridBlockModel(
    name="vol",
    tensor_u=np.ones(10).astype(float),
    tensor_v=np.ones(15).astype(float),
    tensor_w=np.ones(20).astype(float),
    origin=[10.0, 10.0, -10],
    attributes=[
        omf.NumericAttribute(
            name="random attr", location="cells", array=np.random.rand(10 * 15 * 20)
        ),
    ],
)

proj.elements = [pts, lin, surf, grid, vol]

proj.metadata = {
```

(continues on next page)

(continued from previous page)

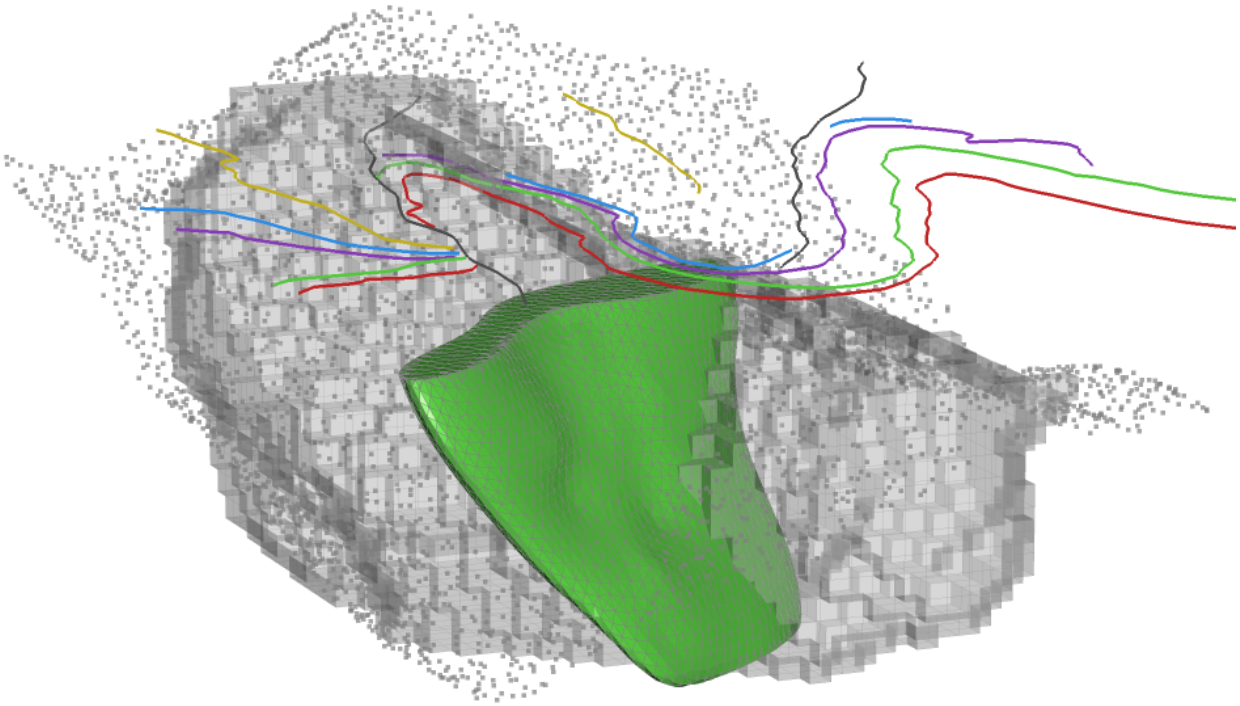
```
    "coordinate_reference_system": "epsg 3857",
    "date_created": datetime.datetime.utcnow(),
    "version": "v1.3",
    "revision": "10",
}

assert proj.validate()

omf.save(proj, omf_file)
```

Piecewise building example:

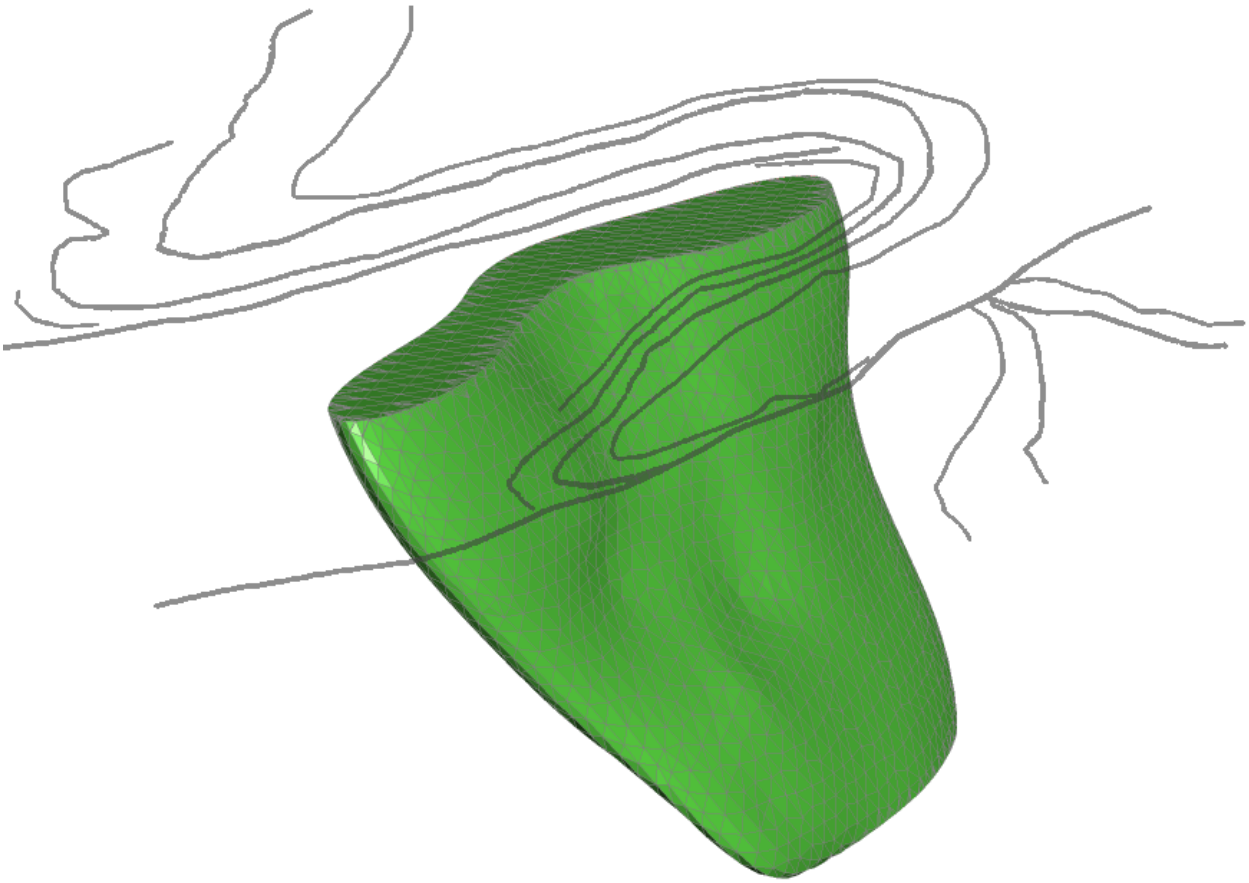
```
...
pts = omf.PointSet()
pts.name = 'Random Points',
pts.vertices = np.random.rand(100, 3)
...
```

7.3 OMF IO API

7.3.1 Save



7.3.2 Load



CHAPTER 8

Index

- genindex