Error Examples of the GeoE3 Quality Rules

This documentation contains error examples of the quality rules implemented in GeoE3 Quality Software. The application implementation and instructions can be found in the [GeoE3 GitHub repository](https://github.com/opengeospatial/GEOE3).

Most quality rules used in GeoE3 Quality Software are pre-built directly to the FME environment. For that reason, most error descriptions are copied from [GeometryValidator documentation](http://docs.safe.com/fme/html/FME_Desktop_Documentation/FME_Transformers/Transformers/geometryvalidator.htm).

And the rules in FME are largely based on the Val3dity-application. The Val3dity has more comprehensive [documentation](https://val3dity.readthedocs.io/_/downloads/en/2.2.0/pdf/) about the errors. The Val3dity documentation is also directly used as a source for this documentation.

# GLOSSARY

|  |  |
| --- | --- |
| **Vertex** | Vertex is the smallest part of a 3D model i.e. one point of intersection containing coordinates. |
| **Orientation** | Order of the connected vertices. |
| **Normal** | A direction vector that is perpendicular to a given surface/solid. Vertices can also have normals. |
| **Solid** | A three-dimensional cohesive geometrical object that has a volume. Usually, solids should be water-tight and uniform, like cubes. A solid consists of surfaces as boundaries (so-called boundary surfaces). It can also contain inner boundaries. |
| **Surface** | Geometries that have a planar area. Surfaces are 2D features in 3D space, and they can be simple or composite. Boundaries of the surfaces are areas. Surfaces consist of curves or lines, which themselves consist of points. |
| **Face** | One of the most basic parts of a 3D polygon, in which three or more vertices (points) connect to make a visible and planar surface. The geometric realization of a face is a surface. |
| **OGC** | Open Geospatial Consortium is an organization focused on developing and defining open standards for the geospatial community. |
| **Feature** | The feature is an individual item that can be processed. Abstraction of real-world phenomena. The feature can be a surface, a solid or a face, as example. |
| **Area** | The boundary of the surface that has an area. The measure of the physical extent of the object. Areas can be 2D or 3D. |
| **Void** | A solid can have zero or more empty regions i.e. voids, which are defined by an interior boundary. Voids are also called cavities. |
| **Shell** | Shells are used to define the closed boundaries of a solid. Every solid must have a shell. Also called outer boundaries. |
| **Polygon** | A polygon is a planar surface. |
| **Interior** | The set of direct positions that are on a geometric object, but not in its boundary. |
| **Exterior** | Difference between the universe and the closure |
| **Geometry** | Geometry is the root class of the geometric object taxonomy and supports the interfaces common to all geographically references geometric objects. |

Source: ISO 19107:2017

[GLOSSARY 2](#_Toc95470650)

[Non-Planar Surfaces 4](#_Toc95470651)

[Self-Intersections in 2D 5](#_Toc95470652)

[Duplicate Consecutive Points in 3D 6](#_Toc95470653)

[Missing Vertex Normals 7](#_Toc95470654)

[Incorrect Surface Orientation 8](#_Toc95470655)

[Incorrect Area Orientation 9](#_Toc95470656)

[Incorrect Solid Orientation 10](#_Toc95470657)

[Missing Measures and Elevations 11](#_Toc95470658)

[Mismatched Dimensions 12](#_Toc95470659)

[AreaTooSmall 13](#_Toc95470660)

[VolumeTooSmall 14](#_Toc95470661)

[GroundOverlappingIn2D 15](#_Toc95470662)

[MinGroundHigherThanMinRoof 16](#_Toc95470663)

[MaxGroundHigherThanMaxRoof 17](#_Toc95470664)

[FeatureHasSpikes 18](#_Toc95470665)

[SurfaceWronglyOriented 19](#_Toc95470666)

[Degenerate or Corrupt Geometries 20](#_Toc95470667)

[Invalid Solid Voids 21](#_Toc95470668)

[**a)** **Duplicate Shells** 21](#_Toc95470669)

[**b)** **Shells Intersect** 22](#_Toc95470670)

[**c)** **Inner Shell Outside Outer** 23](#_Toc95470671)

[**d)** **Interior of Shell Not Connected** 24](#_Toc95470672)

[Invalid Solid Boundaries 25](#_Toc95470673)

[**a)** **Surface not closed** 25](#_Toc95470674)

[**b)** **Vertices Not Used** 26](#_Toc95470675)

[**c)** **Not Enough Faces** 27](#_Toc95470676)

[**d)** **Face Wrong Orientation** 28](#_Toc95470677)

[**e)** **Not a Valid 2-Manifold** 29](#_Toc95470678)

[**f)** **Surface Self Intersects** 30](#_Toc95470679)

[**g)** **Surface Wrong Orientation** 31](#_Toc95470680)

[**h)** **Multiple Connected Components** 32](#_Toc95470681)

[Fails OGC Valid (or Compliant) 33](#_Toc95470682)

[**a)** **Self-Intersection** 33](#_Toc95470683)

[**b)** **Hole Outside Shell** 34](#_Toc95470684)

[**c)** **Nested Holes** 35](#_Toc95470685)

[**d)** **Disconnected Interior** 36](#_Toc95470686)

[**e)** **Ring Self-Intersection** 37](#_Toc95470687)

[**f)** **Nested Shells** 38](#_Toc95470688)

[**g)** **Duplicated Rings** 39](#_Toc95470689)

[**h)** **Too Few Points** 40](#_Toc95470690)

[**i)** **Invalid Coordinate** 41](#_Toc95470691)

[**j)** **Ring Not Closed** 42](#_Toc95470692)

[**k)** **Undetermined Error** 43](#_Toc95470693)

[**l)** **Unparsable Geometry** 43](#_Toc95470694)

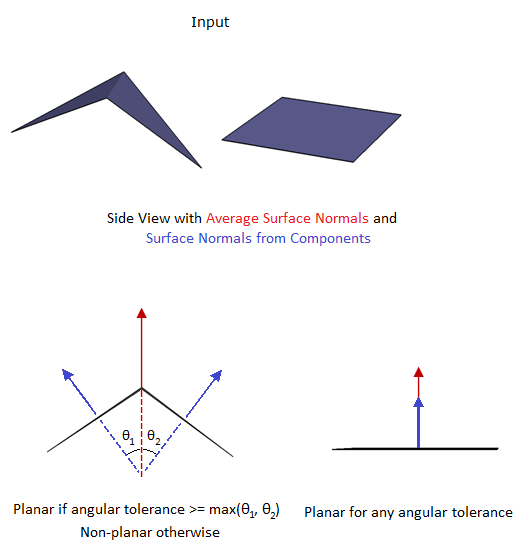
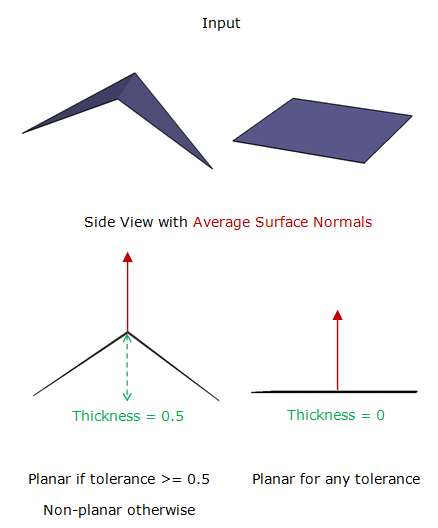
[Fails OGC Simple 44](#_Toc95470695)

[**a)** **Self-Intersection** 44](#_Toc95470696)

[**b)** **Repeated Point** 44](#_Toc95470697)

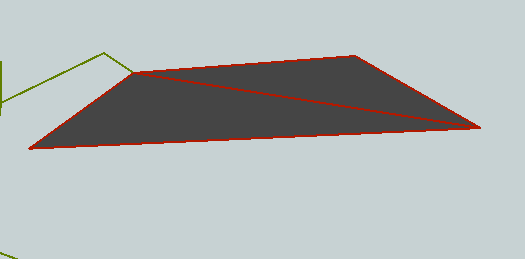
[**c)** **Unparsable Geometry** 44](#_Toc95470698)

# Non-Planar Surfaces



203 and 204 in Val3dity

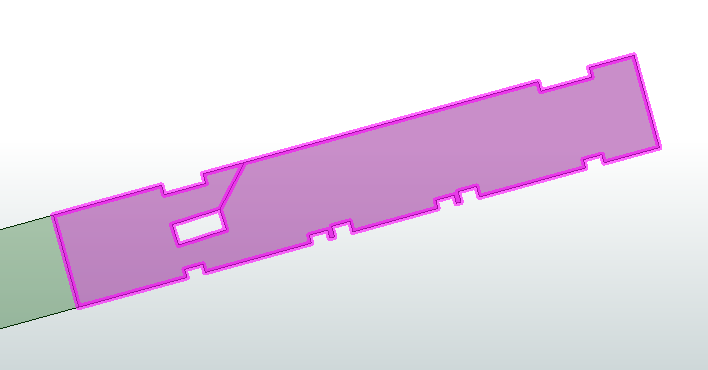
Non-planar surfaces can be difficult to see with the bare eyes, like in the example below:



# Self-Intersections in 2D

The example picture below illustrates how a surface can intersect itself in 2D. The arrow points to a location, which has a small gap of overlapping areas. This rule can also detect duplicate consecutive points if they were not removed earlier (See **Duplicate Consecutive Points in 3D**).

This rule discards the features that intersect in the GeoE3 Quality Software. That prevents **Invalid Solid Voids** and **Invalid Solid Boundaries** rules from detecting the same intersection error.



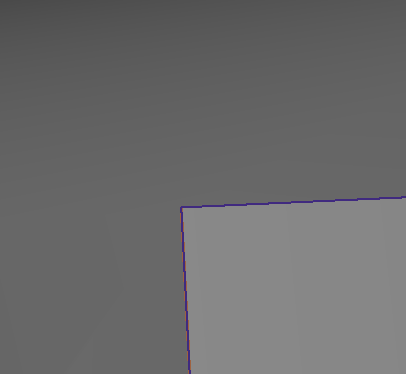
The principles of this rule are similar to **Self-Intersection**, **Ring Self-Intersection**, **Self-Intersection** and **Surface Self Intersects**, but this is a single rule.

# Duplicate Consecutive Points in 3D

Points in any feature should not be repeated in the same location. GeoE3 Quality Software removes these points automatically.

102 in Val3dity

Example:



Two consecutive coordinate points in the same location within a given tolerance (1 mm)

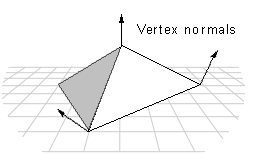
# Missing Vertex Normals

Vertex normals (sometimes called pseudo-normals) are values stored at each vertex that is most commonly used by a renderer to determine the reflection of lighting or shading models, such as phong shading. For example, the normal of a surface is used to calculate which direction light reflects off this surface. While surfaces inherently all have normals as part of their structure, vertices do not. Therefore, storing an explicit normal at each vertex is required.

Vertex normals are 3D coordinates, with each coordinate within the range [-1,1].

Most FME geometries (except meshes and point clouds) store vertex normals as specially-named measures on the vertices of the geometry. The names of these measures are:

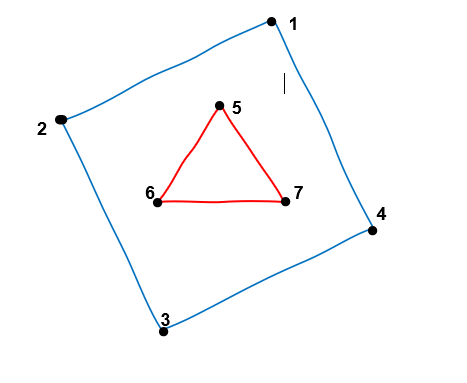
* fme\_vertex\_normal\_x
* fme\_vertex\_normal\_y
* fme\_vertex\_normal\_z



Source: <https://docs.safe.com/fme/html/FME_Desktop_Documentation/FME_ReadersWriters/!FME_Geometry/Vertex_Normals.htm>

# Incorrect Surface Orientation

Surfaces have orientations, which can be invalid. Faces can have an invalid orientation if they have a donut area where the normal of any inner boundary has the same direction as the normal of the outer boundary. Composite surfaces can have an invalid orientation if they have two parts that share an edge and those parts are not consistently oriented concerning each other.



208 in Val3dity

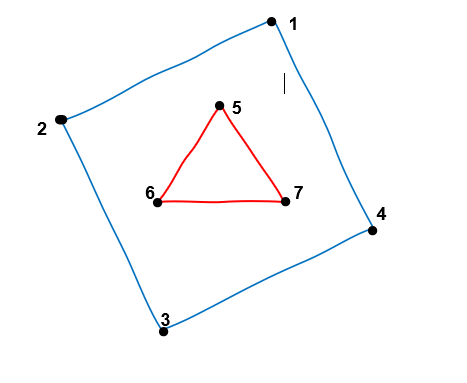
Similar to the **Incorrect Area Orientation**, if the area/polygon is used as a surface of a shell.

Same as the **Surface Wrong Orientation**, but as an independent rule.

# Incorrect Area Orientation

Areas, such as polygons, ellipses, and donuts, have an orientation. Their orientation can be either left-handed, right-handed, or invalid. A left-handed orientation means the area's outer boundary has its vertices arranged in a counterclockwise direction, and the holes have their vertices in a clockwise direction. In a right-handed orientation, the area's outer boundary has its vertices arranged in a clockwise direction, and the holes have their vertices in a counterclockwise direction.

An example of an invalid area would be a donut, whose inner and outer rings are all right-handed, or all left-handed (see image below).



208 in Val3dity

Similar to the **Incorrect Surface Orientation**, when the area/polygon is used as a bounding surface of a shell.

# Incorrect Solid Orientation

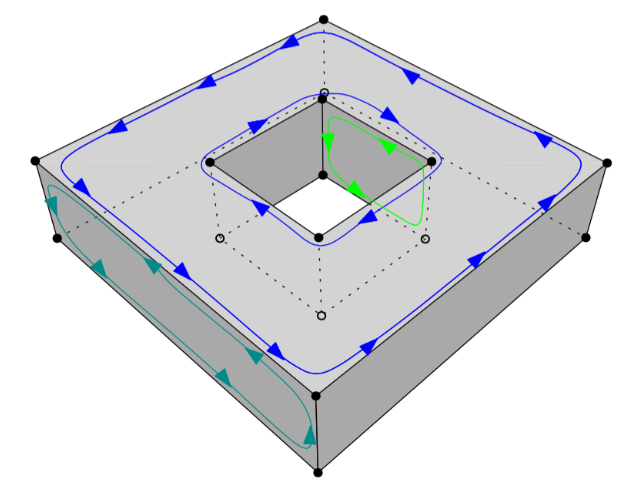
Solids, such as boxes, BRep solids, CSG solids, extrusions, and composite solids, have an orientation. This orientation is considered invalid by the GeometryValidator if the normals of outer shells do not face out or the normals of inner shells do not face in. Their orientation is also invalid if they are inconsistent.

The direction of normals is defined in the following way: if a right-hand system is used (see **Incorrect Area Orientation**), the normals must point outwards. If the left-handed system is used, normals point inwards.

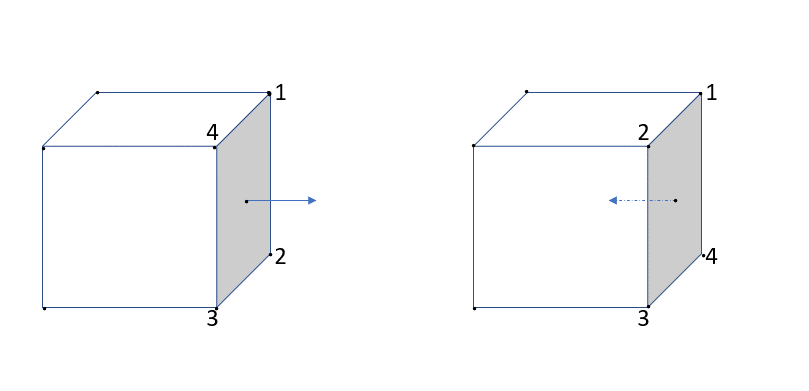
This is an error that you can’t see directly by looking at the data when normals or orientations are not visualized.

405 in Val3dity

The example picture of valid orientation (order of the vertices) from Val3dity [documentation](https://val3dity.readthedocs.io/_/downloads/en/2.2.0/pdf/):



Conceptually the same as **Incorrect Area Orientation**

In the example picture below you can see two examples of different orientations. The first solid has right-handed orientations, so the normal vector points outwards (valid). The second solid has left-handed orientations, so the normal vector points inwards (invalid).

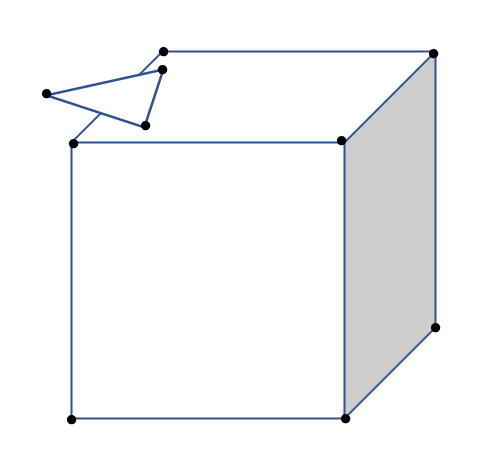
# Missing Measures and Elevations

Vector geometries, such as points, curves, and areas, may have measures (m-coordinate) and elevation (usually z-coordinate). If any vertex on a single instance of one of those geometries has a value for either a measure or an elevation, then all other vertices on that geometry must also have a value for that same measure (or elevation).

No example picture is needed.

# Mismatched Dimensions

Some features might have mismatched dimensions if they contain a mixture of both 2D and 3D parts. That causes an error.



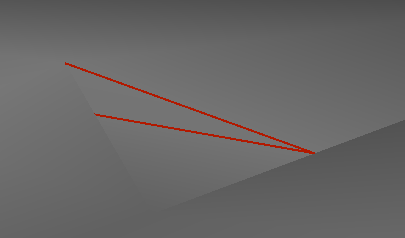
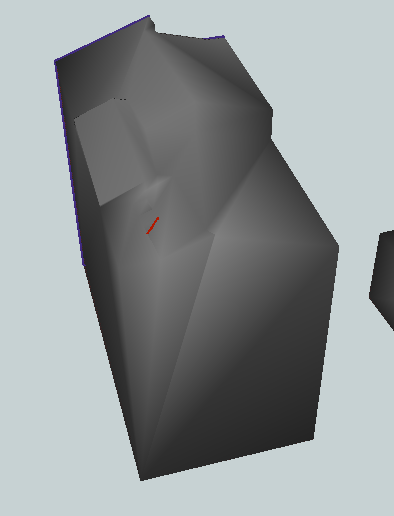
3D

2D

# AreaTooSmall

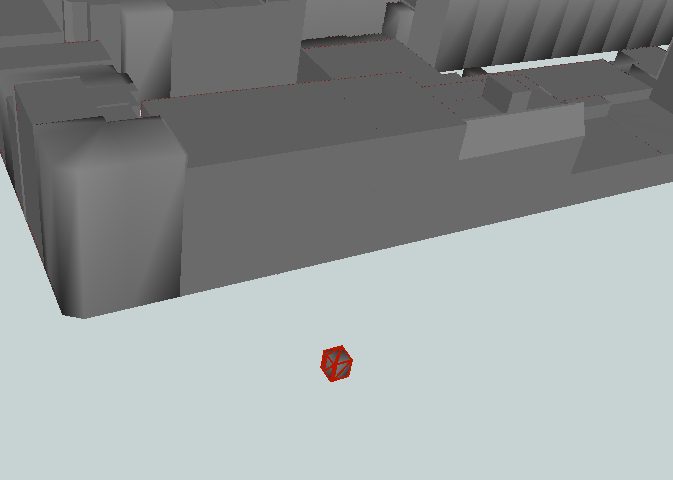
The face is too small if its area is under the given threshold. The purpose of this rule is to find faces, which are so small that they resemble more points or lines than surfaces.

In the picture below you can see an example of a too-small face, almost collapsed into a line, in the corner of the building.



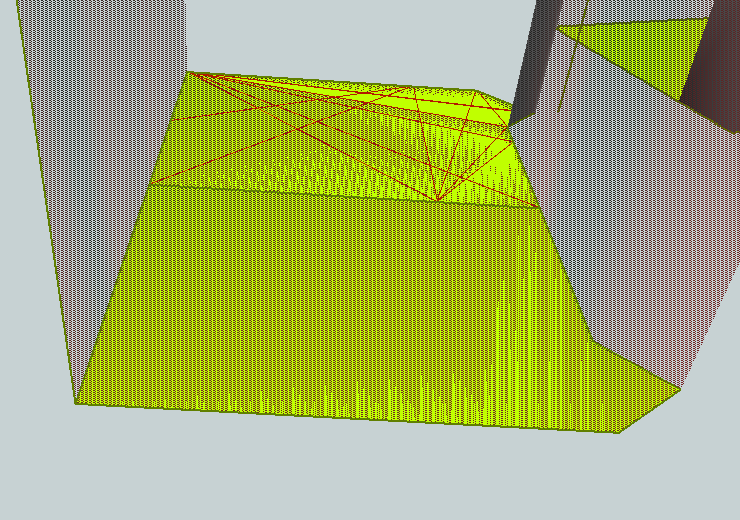
# VolumeTooSmall

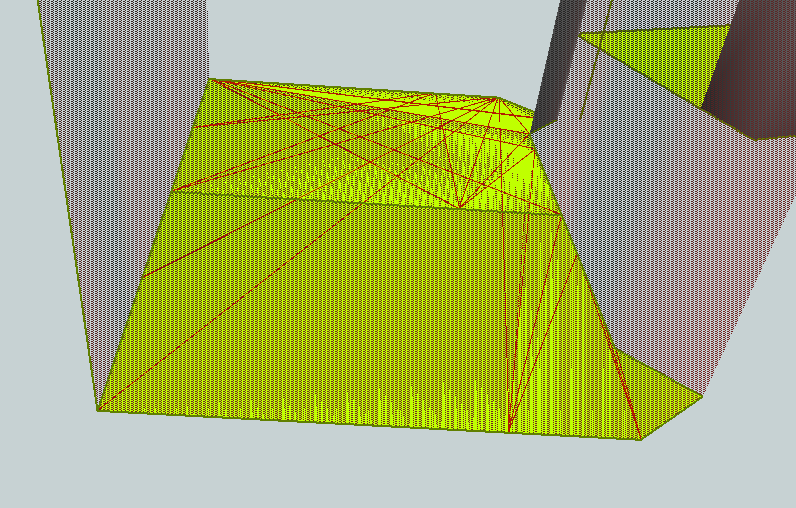
Solids can be erroneous if they have too small volume. This rule applies only to solids because other geometry types have no volume.



# GroundOverlappingIn2D

GroundSurfaces are overlapping if their area is spatially overlapping more than the given threshold. You can see an example of the two GroundSurfaces below, which have a large common overlapping area (red lines).

Area 1:

Area 2:

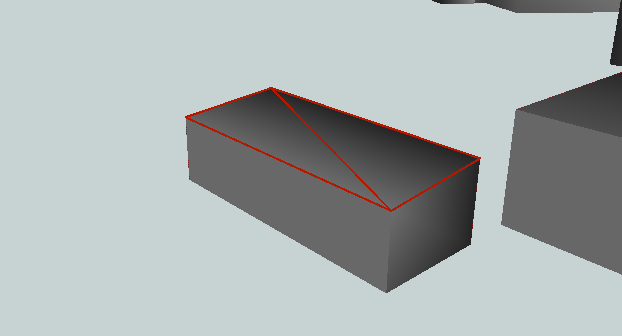
# MinGroundHigherThanMinRoof

This error occurs if the minimum height of the ground is higher than the minimum height of the roof.

In the example picture below, RoofSurface and GroundSurface have changed their places. The selected GroundSurface (represented as red lines) is higher than the RoofSurface, which is under the building.

GroundSurface on the top of the building

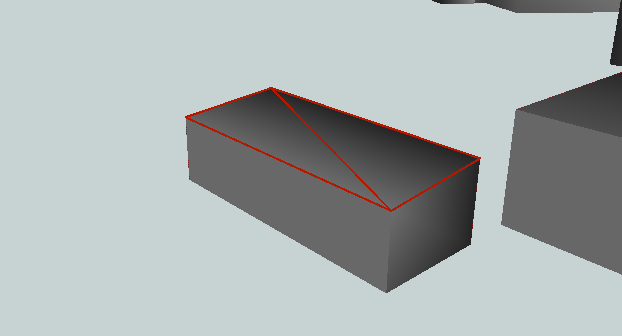
RoofSurface under the building



This is similar to **MaxGroundHigherThanMaxRoof** but based on minimum height values instead of maximum height values. Usually, both errors raise simultaneously if the surfaces are planar and horizontal. But in the other cases, only either of them is founded.

# MaxGroundHigherThanMaxRoof

This is similar to **MinGroundHigherThanMinRoof** but based on maximum height values instead of minimum height values. Usually, both errors raise simultaneously if the surfaces are planar and horizontal. But in the other cases, only either of them is founded.

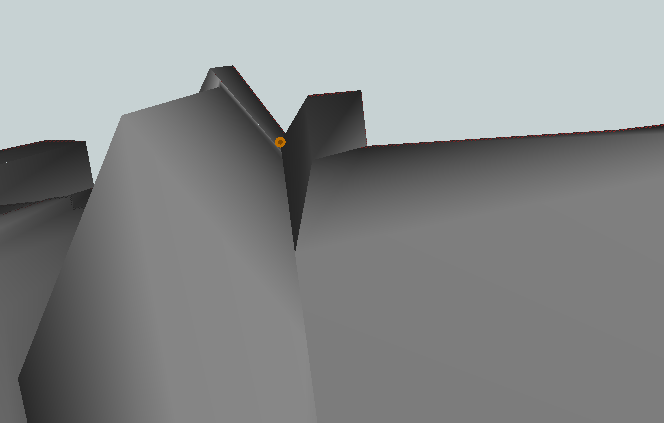


GroundSurface on the top of the building

RoofSurface under the building

# FeatureHasSpikes

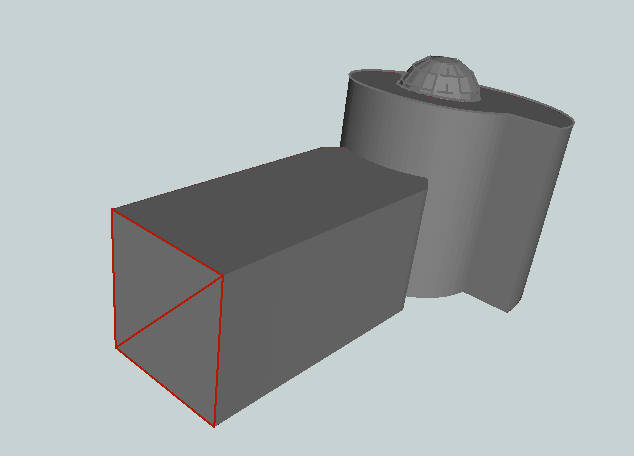
The feature might have spikes if the angle (in degrees) between the two line segments is less than or equal to the given threshold.



# SurfaceWronglyOriented

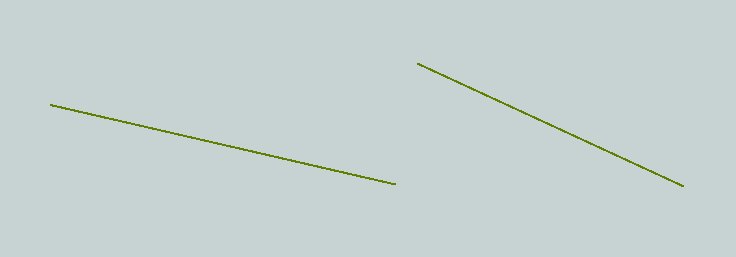
The surface can be wrongly oriented for example in the cases when the WallSurface is horizontal or the GroundSurface is upside down.

Below is an example of the RoofSurface, which is vertical instead of horizontal. Most likely the feature should be categorized as a WallSurface.



# Degenerate or Corrupt Geometries

A degenerate geometry is one whose geometry type can be simplified. For example, a polygon that has 0 areas is degenerate (image below) and it should be a line.



# Invalid Solid Voids

If the solid pass all these rules (a-d), its inner boundaries reside completely within the outer boundary and none of the boundaries intersect each other.

1. **Duplicate Shells**

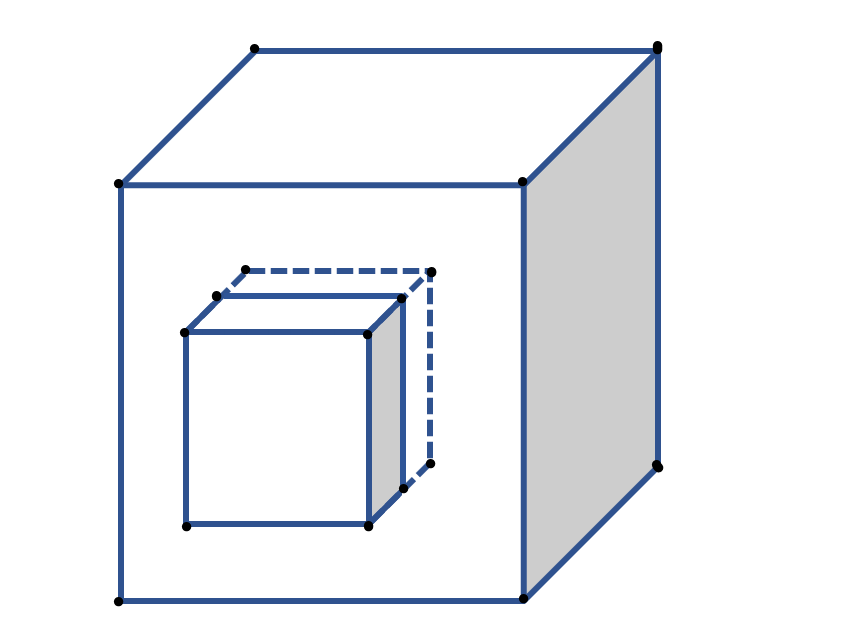
In that case, one solid has two or more identical shells.

402 in Val3dity.

At least 2 shells in the same location

1. **Shells Intersect**

This error occurs when at least two shells of the solid(s) intersect. The error should not occur, if the **Self-Intersections in 2D** errors are fixed or removed before this rule, like in the GeoE3 Quality Software.



401 in Val3dity.

1. **Inner Shell Outside Outer**

Interior shells cannot be located outside of the exterior shells. If the interior shell touches the exterior shell even in one location point, The **Shells Intersect** error will be raised.

Interior shell

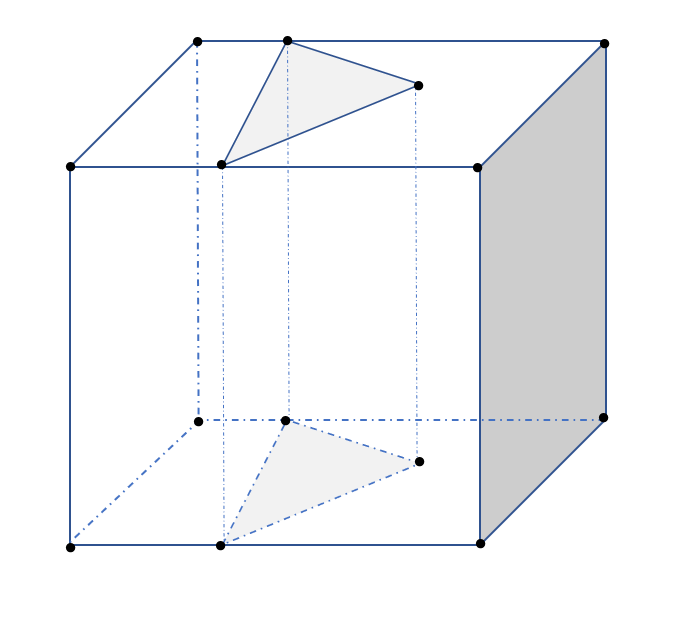
Exterior shell

403 in Val3dity

1. **Interior of Shell Not Connected**

The interior of a shell must be connected. Theoretically the same error as the **Disconnected Interior** but in the context of 3D geometry of shells.

In the image below is the shell, which is divided into two parts because of the interior shell.

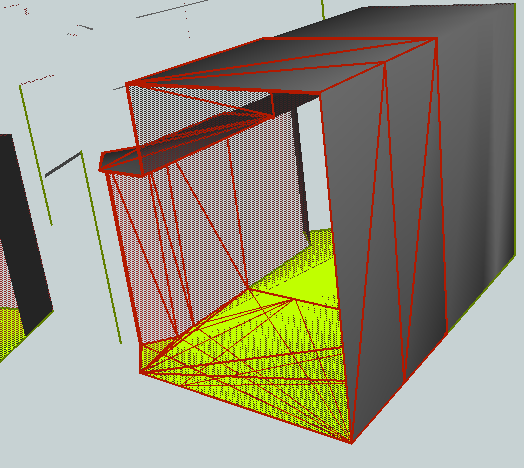


404 in Val3dity

# Invalid Solid Boundaries

If the solid pass all these rules (a-h), it is water-tight, non-self-intersecting and properly oriented.

1. **Surface not closed**

The shell of the solid cannot contain holes or missing parts. The solid must be watertight and contain volume.

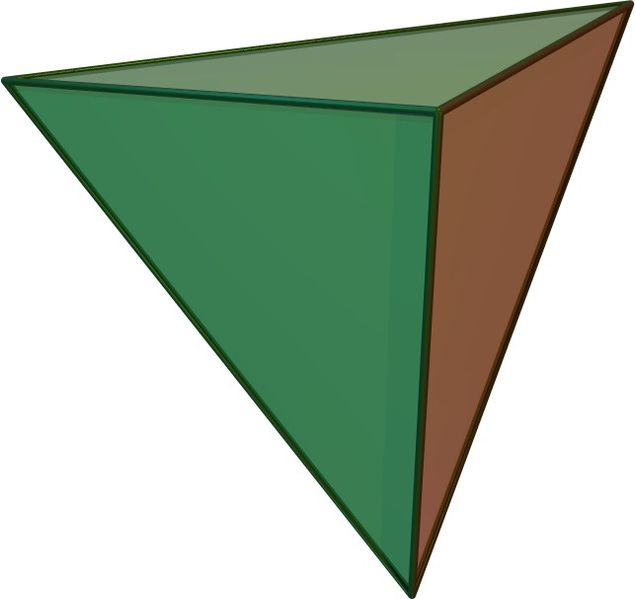
1. **Vertices Not Used**

This error occurs when the feature contains vertices that are not used in any geometry.

1. **Not Enough Faces**

This error occurs if the Solid shell has fewer than 4 faces. The simplest shape of the solid is a tetrahedron (picture below). If the shell is not closed, but it has more or equal than 4 faces, the **Surface not closed** error will be detected.

301 in Val3dity.



Tetrahedron source: https://en.wikipedia.org/wiki/File:Tetrahedron.jpg

1. **Face Wrong Orientation**

If one polygon, area or face is used to construct a shell, its exterior ring must be oriented in such a way that when viewed from outside the shell the points are ordered counterclockwise (as in the image below)

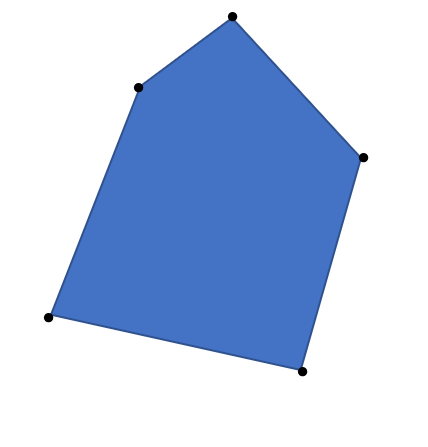
1

2

5

3

4



307 in Val3dity.

See also the image in **Incorrect Solid Orientation**

1. **Not a Valid 2-Manifold**

This error occurs when the solid is not valid, but the exact error is not known.

300 in Val3dity.

1. **Surface Self Intersects**

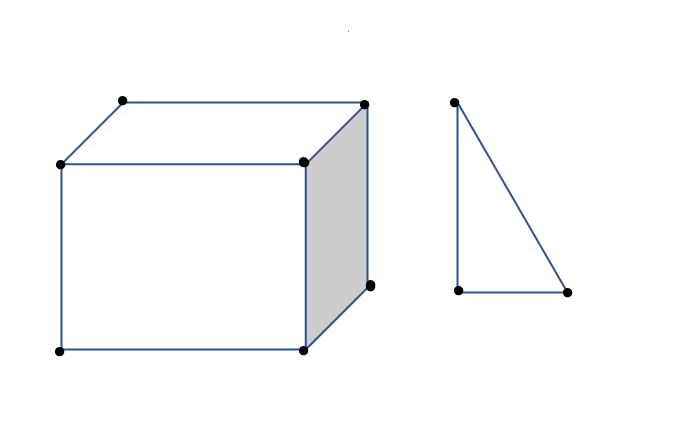
This error occurs when the boundaries of the solid intersect. The error should not occur, if the **Self Intersections in 2D** errors are fixed or removed before this rule, like in the GeoE3 Quality Software.

1. **Surface Wrong Orientation**

Same as **the** **Incorrect Surface Orientation**, but under the **Invalid Solid Boundaries** error group.

1. **Multiple Connected Components**

If the feature contains polygons which are not connected to the shell, this error occurs.



# Fails OGC Valid (or Compliant)

A geometry is “OGC Compliant” if it can be represented losslessly in the Simple Feature Access geometry model defined by the OGC. If the geometry passes all these rules (a-l), it is OGC Compliant.

**Note 1:** GeometryValidator can perform only some of the rules depending on the geometry type. For polygons as an example, it checks:

- **Invalid Coordinate**

- **Ring Not Closed**

- **Too Few Points**

- **Ring Self-Intersection**

- **Self-Intersection**

- **Nested Holes**

- **Hole Outside Shell**

- **Disconnected Interior**

For solids or multipolygons, the same checks, but also **Nested Shells**.

**Note 2:** If the feature contains arcs, they won’t pass any of these rules since OGC Geometries do not support arcs.

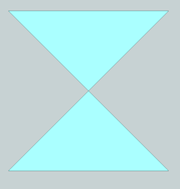
In the below, you can see 2D examples of the OGC Valid errors. The examples are similar for the 3D data when you just add one more dimension. Descriptions and images are derived from: <https://www.baooytra.com/knowledge/articles/21674/invalid-ogc-geometry-examples.html?smartspace=coordinate-systems_2>

1. **Self-Intersection**

Linear rings or polygons cannot intersect themselves.

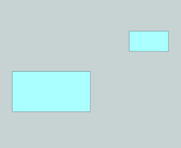
104 in Val3dity

WKT Example: POLYGON((0 0,10 10,0 10,10 0,0 0))



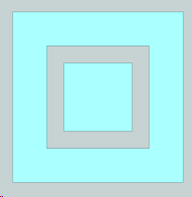
1. **Hole Outside Shell**

The hole cannot be outside of the shell.

WKT Example: POLYGON((0 0,10 0,10 10,0 10,0 0),(15日,15日15 20,20 20,20 15,15 15))

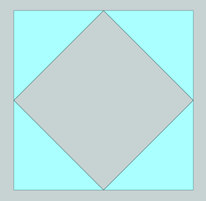
1. **Nested Holes**

The hole cannot be inside another hole.

WKT Example: POLYGON((0 0,10 0,10 10,0 10,0 0),(2 2,2 8,8 8,8 2,2 2),(3 3,3 7,7 7,7 3,3,3))

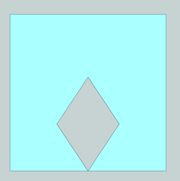
1. **Disconnected Interior**

The interior of a polygon must be connected.

WKT Example: POLYGON((0 0,10 0,10 10,0 10,0 0),(5 0,10 5,5 10,0 5,5 0))

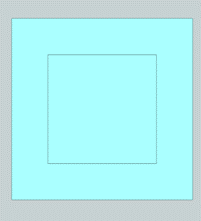
1. **Ring Self-Intersection**

The ring cannot intersect itself in any location. This case also includes the rings that are (partly) collapsed to a line.

WKT Example: POLYGON((5 0,10 0,10 10,0 10,0 0,5 0,3 3,5 6,7 3,5 0))

1. **Nested Shells**

WKT Example: MULTIPOLYGON(((0 0,10 0,10 10,0 10,0 0)),(( 2 2,8 2,8 8,2 8,2 2)))



1. **Duplicated Rings**

Two or more rings/polygons have identical coordinate points.

WKT Example: MULTIPOLYGON(((0 0,10 0,10 10,0 10,0 0))((0 0,10 0,10 10,0 10,0 0)))

1. **Too Few Points**

Every polygon must have at least three points.

101 in Val3dity

WKT Example: POLYGON((2 2,8 2))

1. **Invalid Coordinate**

Features cannot contain polygons with invalid coordinates.

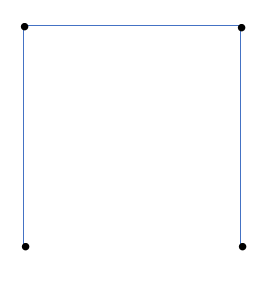
WKT Example: POLYGON((NaN 3,3 4,4 4,4 3,3 3))

1. **Ring Not Closed**

103 in Val3dity

The first and last points of a polygon or linear ring must be identical. This applies only to CityGML, not to CityJSON or other formats. FME will fix this automatically when reading the data.

WKT Example: POLYGON((0 0,0 10,10 10,10 0))



1. **Undetermined Error**

An error that cannot be determined.

1. **Unparsable Geometry**

The FME cannot parse the geometry.

# Fails OGC Simple

Errors are mainly the same as in the **Fails OGC Valid (or Compliant).**

1. **Self-Intersection**

Same as in the OGC Compliant **Self-Intersection**

1. **Repeated Point**

Same as in the **Duplicate Consecutive Points in 3D**

1. **Unparsable Geometry**

Same as in the OGC Compliant **Unparsable Geometry**