GEOM Module manual: I User guide

Frédéric Boudon, Christophe Nouguier, Christophe Godin

GEOM Module Manual.
I. User guide

F. Boudon, C. Nouguier, C. Godin, C. Pradal

8th December 2003

About this document

The GEOM module is part of the AMAPmod software and consists of a 3D objects description language. Based on the MTG model, this language provides a simple and flexible mechanism to describe a hierarchical 3D scene as a collection of objects arranged into a graph structure, called *Scene Graph.*

In addition to this module, AMAPmod includes a Viewer, which allow the user to examine the scenes he has created and to export them into various 3D file formats. This way it is possible to perform additional operations on the scenes such as ray tracing, walk through, hemispherical snapshots and so on.

Although, this language has been designed to be used by non specialist and do not require strong backgrounds in 3D computer graphics, it is recommended to consult books introducing basic concepts on 3D graphics to have a better understanding.

This document contains the following chapters:

- The chapter 1 explains how to represent 3D scenes using AMAPmod.
- The chapter 2 forms a reference to the GEOM's file formats.
- The chapter 3 forms a reference to the objects available within the GEOM module.

Notes about the development

The GEOM module is written in C++ and is running on Linux platform. Based on standard libraries, GEOM is set-up to be highly portable.

The module is still being developed and unfortunately is still subject to bugs. Thus users are invited to submit bugs report by mailing to the AMAPmod forum: `amapmod@cirad.fr`. Meanwhile as the library is beginning to be used by several researchers in different institutes, it is getting more and more improved. Basic shortcomings are expected to be treated in the near future, and extensions are currently being developed.

Acknowledgments

Our thanks go to C. Godin who contributed so much to the development and content of this module, and to the integration within the AMAPmod software.

Thanks to N. Dones and B. Adam to the development of the Viewer.

Contents

List of Figures

List of Tables

Chapter 1

How to build 3D scenes ?

The GEOM module allows the description of a scene in the 3D space, by adding geometrical and appearance features to a MTG.

The first section of this chapter introduces the formal model adopted to represent a scene.

The second section presents the language used to describe geometrical and appearance features.

1.1 Scene representation model

The scene representation model is based on both the concept of *Multiscale Tree Graph* (MTG) and on the concept of *Scene Graphs*.

The objectives of this section are to review some basics about scene graphs and to present how MTG are related to such graphs.

1.1.1 Scene Graphs basics

One of the most commonly-used method to represent a 3D scene is to use a *Directed Acyclic Graph* [**?**], whose nodes hold the information about the elements that make up the scene. Directed means that the parent-child relationship is one-way, Acyclic means that the graph can't contain any loops, i.e. child can't be one of its own ancestor. As child nodes may have more than one parent, this enables to share nodes within the graph. Such graphs are called *Scene Graphs* [**?**, **?**]. Figure 1.1 shows a simple scene graph.

Nodes within a scene graph, or *scene objects*, are refined into two categories:

1. *Grouping objects*, which collect one or more objects into a graph. They generally correspond to an operation used in order to build more complex objects such as geometric transformations, unions...
2. *Leaf objects*, which contain their own parameters and do not contain other objects. They generally correspond to basic models such as geometric primitives, colors...

The hierarchy of the scene graph allows a natural partitioning of the elements within the space. That is especially important in computer graphics for optimization purposes. It leads to efficient implementation of algorithms such as rendering, picking, searching, collision detection and so forth.

1.1.2 Multiscale Scene Graphs

In the AMAPmod software, 3D scenes are represented using the concept of scene graphs. The core of the scene graph is given by a MTG. To each vertex within the MTG is attached additional

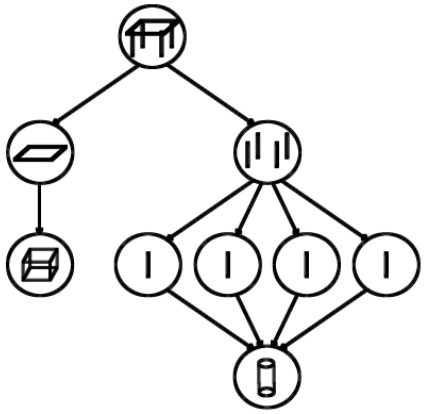


Figure 1.1: A Scene Graph.

Each foot of the table is represented using a cylinder node which is located and oriented in space using transformation nodes. In the same way the table top is represented using a scaled and located box node.

features concerning its geometry and its material. These features are also represented by the mean of scene graphs specified using the GEOM modelling language.

Combining GEOM scene objects with a MTG forms a *Multiscale Scene Graph*.

Such structure allows to model a scene at different level of representation. Effectively, at a given scale, the MTG is represented by a tree graph whose vertices may point to GEOM scene graphs. In that case, this tree graph is a model of a scene graph as introduced in the previous section and it represent the scene at the specified scale.

Here's an example of such a formal representation. Supposing the following MTG modelling a flower (F) composed of a petiole (S) and four petals (P):

```
/F1
^/S1
    +P1
    +P2
    +P3
    +P4
```

To each entity is attached a formal scene graph representing its geometries : FG, SG, P_1G, P_2G, P_3G and P_4G. The resulting multiscale scene graph is illustrated figure 1.2.

The scene graph at the global scale appears in dot line, while the scene graph at the second scale appears in solid line.

1.2 GEOM modelling language

GEOM modelling language (GML) is a powerful tool for describing the required features, i.e the geometry and the appearance (i.e.: material), to represent elements in the 3D space. As introduced in the previous section, GEOM collects these features into scene graphs.

This section overviews main features of the GEOM modelling language. It first introduces basics about GEOM files needed to describe scene objects and the way objects are specified within

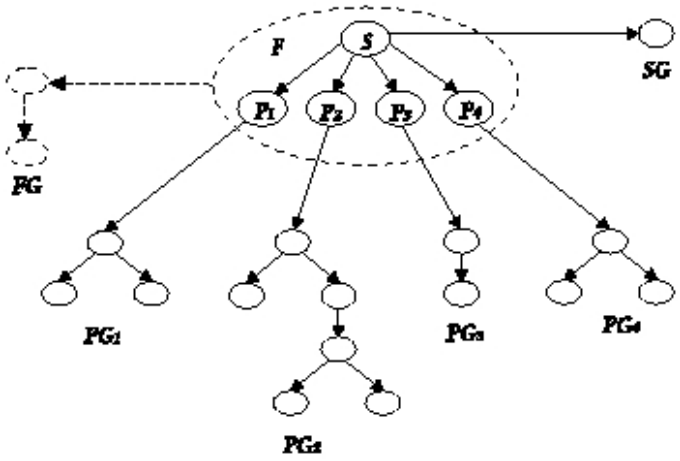


Figure 1.2: A Multiscale Scene Graph

these files. Then semantic of various scene objects types is presented. For a better understanding, scene objects are divided into three families: GEOM objects which are used to describe geometrical features, APP objects which are used to represent appearance features, and SHAPE objects used to combine the GEOM objects with the APP objects and identify them.

1.2.1 Scene objects

Scene objects are externally specified by the mean of ASCII files. GML introduces two different types of file: `geom` file which are used to specify GEOM objects, and `app` files which are used to specify APP objects. Each of these files consists of a list of description of scene objects. According to their semantic, these objects may be leaf objects or grouping objects. Therefore, a GEOM description file may be considered as a catalog of scene graphs representing a more or less complex object as illustrated figure 1.3.

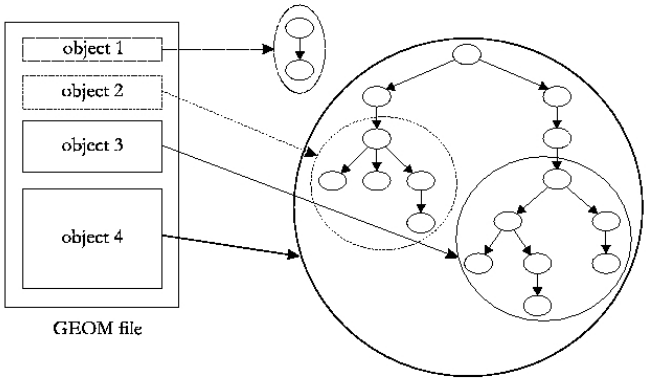


Figure 1.3: GEOM file objects and associated scene graphs.

Each objects are specified by its *class*, an optional *label* and finally the *fields* description.

- The class specifies the type of the object, such as `Box`, `Cylinder`, `FaceSet`, `Material`, `Shape`... Subsequent sections introduce the semantic of various classes of objects available

within GML.

- The label specifies a symbolic name which allows to identify the object. This key is required when describing an object at the root. Elsewhere it is optional. Naming is useful when it is needed to share the object within different objects declaration. Moreover, this label is used as the key which allows to bind the vertices within a MTG to the scene objects.

- The fields description specifies the parameters and their associated data held by the object.

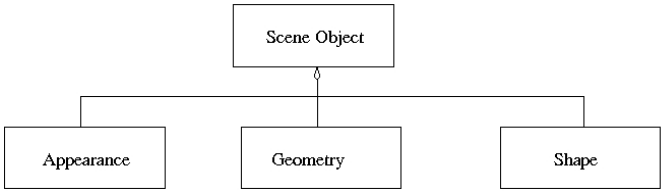


Figure 1.4: Scene Object hierarchy

1.2.2 GEOM objects

3D geometric objects are represented using GEOM objects. Three families of GEOM objects have to be considered: *Primitives*, *Transformed* and *Group*.

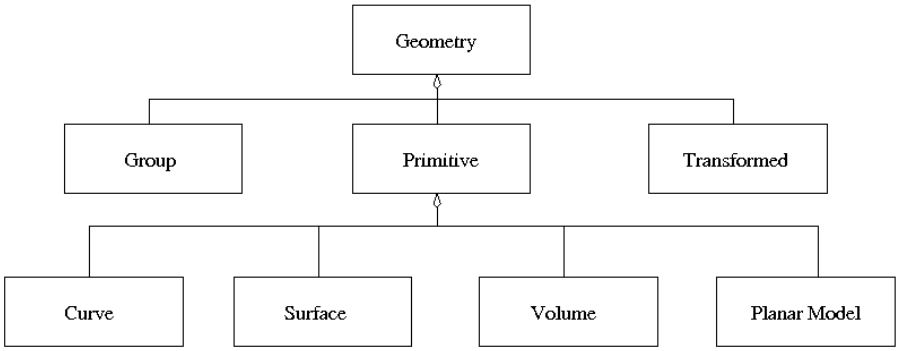


Figure 1.5: Geometry hierarchy

1.2.2.1 Primitives

Primitives correspond to a common 3D geometric models. Actually, twenty six primitives are provided (see table 1.1).

Primitives are described in their own local coordinate system, known as *local reference system*.

Among these classes of objects, four categories are distinguished:

- Points are represented using the object `PointSet`. are explicitly specified by giving their coordinates in the 3D space.

- Curves are represented by:

Primitives
AmapSymbol
AsymmetricHull
BezierCurve
BezierCurve2D
BezierPatch
Box
Cone
Cylinder
Disc
ElevationGrid
ExtrudedHull
Extrusion
FaceSet
Frustum
NurbsCurve
NurbsCurve2D
NurbsPatch
Paraboloid
PointSet
Polyline
Polyline2D
QuadSet
Revolution
Sphere
Swung
TriangleSet

Table 1.1: Primitives.

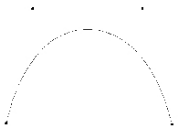


Figure 1.6: A parametric curve.

1. *Parametric curves.* A parametric curve is a mathematical expression of a curve defined by few parameters such as the control points. It is a convenient method to fit a curve by passing through a given set of points as illustrated figure 1.6.
2. *Polyline.* A Polyline is a sequence of connected segments, representing a linear approximation of a general parametric curve.

Note: Curves are often used to generate surfaces or to represents the *skeleton* of an object. The skeleton of a geometric object generally corresponds to the medial axis of this object.

- Surfaces are represented by:

  1. *Parametric surfaces* or *Patches.* In the same way as the parametric curves, they provide a convenient method to fit a smooth surface by specifying only the control points. The `ElevationGrid` is a particular case of such surfaces and is mainly used for terrain modelling.
  2. *Surfaces of revolution* (SOR). A surface of revolution is constructed by rotating one or a set of 2D curves around the `z`-axis.
  3. *Polygon meshes.* A mesh is a collection of connected polygons, known as *faces*, defining an object in the 3D space (see figure 1.7).

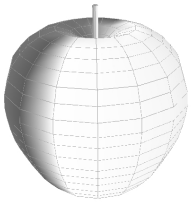


Figure 1.7: A polygon mesh representing an apple.

  There are several rules to follow when building a polygonal mesh. Within a mesh, each faces must be planar, convex and have the same orientation. The orientation is specified by the order the vertices are described : *clockwise* or *counter-clockwise* The orientation defines the normal to the polygon. The normal to a surface is important as it is used when lighting the scene (see figure 1.8).

- Volumes describe a surface or a set of planes bounding a closed volume. There are generally represented by:

  1. Surfaces of type of Mesh or SOR with the special attribute *solid* set to `true`. This means that the surface is closed.
  2. *Hulls.* A Hull represent a 3D envelop. These geometric representation derive from tree crown models used in ecology modelling [**?**].

- Planar Model describes 2D geometric objects. By default, there are placed on *xy plane.*

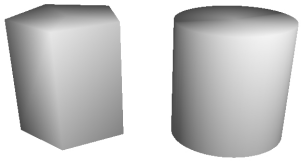
(a) (b)

Figure 1.8: Clockwise and counter-clockwise (CCW) orientation.

(a) The normal is backward oriented. (b) the normal is frontward oriented.

1. Curves with *Polyline2D*, *BezierCurve2D* and *NurbsCurve2D*
2. Surfaces with *Disc*.

Note: Some of these models required to be discretized before they can be drawn as most of the graphics hardwares are able to draw very simple primitives such as points, segments, triangles, quadrilaterals or polygons. GML provides a convenient way to allow the user to control this process by adding specific fields to the object definitions. It is important to keep in mind that the simpler the discretization is, the quicker the object is rendered. Figure 1.9 illustrates a cylinder rendered with two different level of subdivisions.



(a) (b)

Figure 1.9: A cylinder subdivided into slices.

(a) 5 slices are used to represent the cylinder. (b) The same cylinder drowned with 64 slices.

1.2.2.2 Transformed

The transformed classes allows you to specify the position, the orientation and the shape of a GEOM object in 3D space. It binds a *transformation* to a GEOM object in order to create a new object. A transformation changes an object from its local coordinate system to the global coordinate system known as the *global coordinate space.*

If a Transformed geometry contains another Transformed geometry, they are said to be *nested* and their settings have a cumulative effect. Consequently you need to take care about the order of transformation.

In total, seven Transformed are provided (table 1.2).

As a Transformed contains another GEOM objects, the resulting object is a graph. The root node correspond to the transformed object.

If a Transformed object contains another Transformed object, they are said to be *nested* and their settings have a cumulative effect. Consequently the order of transformation has to be considered carefully.

Transformed
AxisRotated
EulerRotated
IFS (Iterated Function System)
Oriented
Scaled
Tapered
Translated

Table 1.2: Transformed.

1.2.2.3 Group

A Group allows to combine a collection of GEOM objects in order to build more complex objects. It may contain Transformed objects, Primitives and Groups. As the same manner as for the Transformed objects, the result correspond to a graph of objects, whose root is the created object.

1.2.3 APP objects

Appearance objects are represented using APP objects. Two families of APP objects have to be considered: *Colors* and *Physical models*.

- Colors
- Physical models

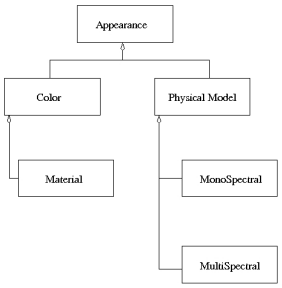


Figure 1.10: Appearance hierarchy

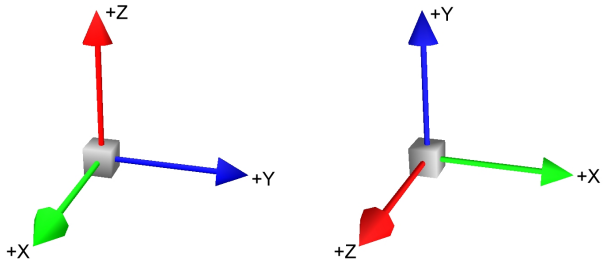
1.2.4 SHAPE objects

At the first level of the GEOM Scene Graph, the objects are of type `SHAPE` and associate a 3D geometric object with an appearance. Technically, it associates a `GEOM` object with an `APP` object.

There is two way to instantiate the shape objects. The first one is, in AMAPmod, to link a geometry and an appearance with an entity. AMAPmod will create automatically a shape and associate it to the entity. The second way is to describe them in GML, generally in the `GEOM` file. When reading `GEOM` file, the viewer will display only the shape objects. If no shape is defined, it will take all GEOM object and create shapes with a default appearance.

1.3 GEOM's coordinate system

To have an intuitive use of the library for representing plant architecture, the coordinate system use the z-coordinate to define the height of geometric shapes. Some traduction on other standard



(a) (b)

Figure 1.11: (a) GEOM's Coordinate System (b) VRML, PovRay Coordinates System

geometric format of the GEOM objects are available like PovRay and VRML in the module. Theses formats use other coordinate systems. The traduction transform the shapes into the local coordinate system.

Chapter 2

File formats reference

2.1 GEOM's file syntax

The following section outlines the syntax for the `geom` and `app` ASCII file format.

2.1.1 Writing scene objects

Whatever the type of the objects are, the syntax follows the same writing rules. An object is written with the following structure:

1. The type of the object. Each word within the type name begins with an uppercase letter.
2. A name for the object. This label is required when describing a root object, and is optional in the other cases. As mentioned as above, naming is useful when you wish to refer to the object later in the file. Names can not begin with a digit (*0-9*) and can contain only alphabetical characters (`a-z A-Z`), digits (*0-9*) and the underscore (_). Types of objects and fields name are reserved keywords and cannot be used for naming.
3. Fields within the object if any. A Field appears as a pair consisting of a name and a certain type of value (see table 2.1). Fields description is enclosed in braces (`{ }`) and fields can

Type	**Description**
Boolean	A boolean value : `True` or `False`
Color*3*	It consists of 3 integer numbers. It must appears within '<' '>': `<160, 160,` 160>. Numbers domain is 0 to 255. For Grey Level, a Color*3* can be instantiated with a single integer number i.e. 160 correspond to Color*3* <160,160,160>.
Index	A list of integer numbers. All numbers must be positives and must be enclosed by '[' ']' : [1,2,5,3,6,1]
Integer	An integer number : `5`
Real	A floating point number : `1.2`
String	A sequence of characters. It must appears within double quotes : `"string"`
Vector*d*	According the dimension *d*, it consists of 2,3 or 4 floating point numbers. It must appears within '<' '>': `<1.0,2.3>`
Appearance	An Appearance description or a reference to an Appearance.
Geometry	A Geom description or a reference to a Geom

Table 2.1: Basic field types.

be specified in any order. There are 2 types of fields : mandatory fields and optional fields.

Unspecified optional fields are set to their default values.
A field is written with the following elements:

(a) The name of the field. Each words within the field name begins with an uppercase letter.

(b) The value(s) of the field. According to the field, it can be:

- a single value;
- an expression that can be enclosed by parenthesis ( ) .
- an array of values. It is expressed as a series of single values separated by commas and enclosed in square brackets `[ ]`.
- a matrix of values. It is expressed as a series of arrays separated by commas and enclosed in square brackets `[ ]`.

This example creates a leaf (see figure 2.1) using the `BezierPatch` primitive:

```
BezierPatch leaf {
    CtrlPointMatrix [  [ <0,0,0>, <0,0,0> ],
                       [ <2,-1,0.8>, <2,1,1.2> ],
                       [ <4,-2,2.1>, <4,2,1.8> ],
                       [ <6,-1,0.3>, <6,1,0.5> ],
                       [ <7,0,-1>, <7,0,-1> ]
    ]
    UStride 10
    VStride 10
}
```

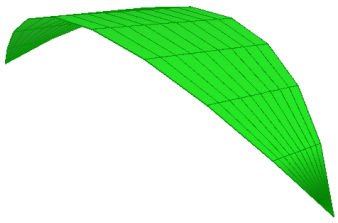


Figure 2.1: The `leaf` object.

For convenience, some fields can be defined as expression. Some classical arithmetic operators have been defined on basic type. For example, `+,-,*,/` can be used on Integer and Real. So a Real can be defined as a mathematical expression composed of Real, and Integer as a mathematical expression of Integer.

```
RealField   ((3.6-1.2)*(1.7-0.3))/2  # RealField will be equal to 1.2
IntegerField  5 / 2                  # IntegerField will be equal to 2
```

Some other operators are defined on Vector$d$:

```
Vectord + Vectord
Vectord - Vectord
Vectord * Real
Vectord / Real
```

And finally, a string can be defined as a concatenation of string using the operator + :

```
StringField  ''a''+''b''+''c''
```

All the expressions defined in a GEOM file are interpreted at the parsing time.

2.1.2 Sharing objects

Using multiple instance of an object is often used when it is needed to reuse objects definitions. Instead of repeating an object's description, it is possible to refer to it by just naming it. Shared instancing allows to save time and space when writing GEOM files.

Here's an example of how to use the object `a_leaf` in order to create a petal (see figure 2.2):

```
Scaled a_petal {
    Scale <0.5,0.8,0.2>
    Geometry leaf
}
```

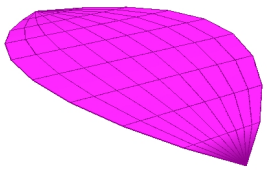


Figure 2.2: The `a_petal` object.

In the following example the `a_petal` object is used to create a more complex object representing a flower, illustrated figure 2.3. The flower is composed of five petals. Each petal is represented by the `petal` object which is rotated about the z-axis.

```
Group flower {
    GeometryList [
        a_petal,          # the first petal
        AxisRotated {     # the second petal
            Axis <0,0,1>
            Angle 72
            Geometry a_petal
        },
        AxisRotated {     # the third petal
            Axis <0,0,1>
            Angle 144
            Geometry a_petal
        },
        AxisRotated {     # the fourth petal
            Axis <0,0,1>
            Angle 216
            Geometry a_petal
```

```
        },
        AxisRotated {     # the fifth petal
            Axis <0,0,1>
            Angle 288
            Geometry a_petal
        }
    ]
}
```

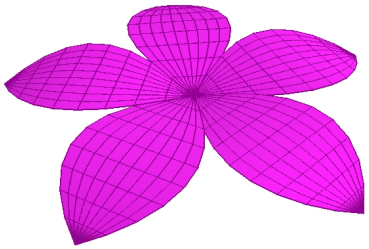


Figure 2.3: The `flower` object.

2.1.3 Miscellaneous

2.1.3.1 Adding comments

Extra white space created by spaces, tabs and new lines is ignored. Comments serve as an aid to the user. There are two comments delimiters.

- The sharp sign `#` serves to delimit a single-line comment. It can be placed anywhere on a line and everything to the right of the delimiter on this line is treated as a comment and consequently is ignored.

  ```
  # This is an example of a single line comment
  ```

- The pair `(#,#)` serves to delimit a block comment. Every character that falls between the `(#` and the `#)` is part of a comment. A comment pair can be placed anywhere. A tab, space, or new-line is permitted and can span multiple lines. For example:

  ```
  (#
  This is an example of a block comment
  #)
  ```

2.1.3.2 Including files

The directive `:include` allows to include other files of same types. It is useful when user-defined objects have to be reused.

2.2 MTG syntax

Geometry and material features are linked to a MTG using the two feature types : `GEOMETRY` and `APPEARANCE`. When defining such a feature, it is needed to specify the name of the GEOM

file containing the objects that can be attached to the vertices. To the `GEOMETRY` feature must correspond a `geom` file, and to the `APPEARANCE` feature must correspond a `app` file.

This binding allows to attach to each entities the desired scene object by just writing its label on the same line as the entity and of course within the corresponding feature column.

Heres is an example of a coding file.

```
CODE: FORM-A
CLASSES:
SYMBOL    SCALE  DECOMPOSITION  INDEXATION DEFINITION
$         0      FREE           FREE       IMPLICIT
F         1      FREE           FREE       EXPLICIT
S         2      FREE           FREE       EXPLICIT
P         2      FREE           FREE       EXPLICIT
DESCRIPTION:
LEFT   RIGHT  RELTYPE  MAX
F      F      +        ?
S      P      +        ?
P      P      +        ?
FEATURES:
NAME TYPE
Geom GEOMETRY my_flower.geom
App APPEARANCE flower.app
MTG:
TOPO         Geom             App
/F1
^/S1         my_petiole       dark_green
    +P1      a_petal   lila
    +P2      my_second_petal  lila
    +P3      my_third_petal   lila
    +P4      my_fourth_petal  lila
    +P5      my_fifth_petal   lila
```

2.3 Dressing File Syntax

The MTG entities can be extend with GEOM and APP object defined in the Dressing File and linked with an AML function. In the Dressing file, you must define the `geom` file and the `app` file.

Here is an example of dressing file :

```
Geometry = my_flower.geom
Appearance = flower.app
```

2.4 Plot Arguments

The function `Plot` has two new optionals arguments `Geometry(Function)` and `Appearance(Function)` which allow to pass functions which define the geometry and the appearance of each entities.

Her is an example of AML code based on the same MTG without the `Geom` and `App` features :

```
AML>
AML> mtg = MTG("flower.mtg")
<MTG> : vtxnb=8, edgenb=5, levelnb=2, featurenb=12, bytesize=1493
AML> drf = DressingData("flower.drf")
<DRESSING_DATA> : Dressing data read from file ‘flower.drf’.
```

```
AML> pf = PlantFrame(0,Scale->2,DressingData->drf)
<PLANTFRAME> : Standard PlantFrame
AML> petal(_x) = Switch ((Index(_x) Mod 5)) \
AML> Case 0 : fifth_petal \
AML> Case 1 : a_petal \
AML> Case 2 : second_petal \
AML> Case 3 : third_petal \
AML> Case 4 : fourth_petal \
AML> Default : Undef
<FUNC> : Function
AML> geom_func(_x) = If (Class(_x)==’S’) Then my_petiole Else petal(_x)
<FUNC> : Function
AML> appe_func(_x) = If(Class(_x)==’S’) Then "dark_green" Else "lila"
<FUNC> : Function
AML> Plot(pf, Appearance->appe_func, Geometry->geom_func)
```

Chapter 3

Scene objects reference

The following chapter lists all GEOM and APP objects in alphabetical order.

Each object description begins with a general explanation. Then each field composing the object are listed within a table, as well as its name, its type, its default values (if any) and a brief comment explaining its role.

3.1 AmapSymbol

Class

Geometry - Primitive - Surface/Volume - Mesh

Description

The **AmapSymbol** describes an object of class of *Mesh* stored in the `SMB` file format of the Amap software. This is provided for ascendant compatibility.

Fields description

Name	Type	Defaults	Description
`FileName`	String	None	Specifies the name of the `SMB` file to bind to the symbol. It must contain the full path and `.smb` extension if needed. The corresponding file must exist.
`Solid`	Boolean	`False`	Specifies whether the symbol represents a closed surface.

Example

```
# An example of AmapSymbol Object
AmapSymbol enpomf {
    FileName "/usr/local/AMAPmod/databases/SMBFiles/enpomf.smb"
}
```

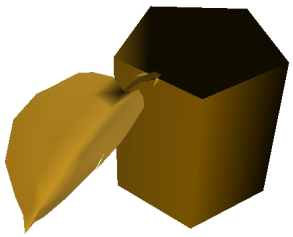


Figure 3.1: The `AmapSymbol` object.

3.2 AsymmetricHull

Class

Geometry - Primitive - Volume - Hull

Description

An **Asymmetric Hull** describes an object of class of *Mesh* defined by 6 morphological points (see figure 3.3). This is an implementation of the asymmetric crowns introduced by Cescatti in [**?**] and Koop in [**?**]. The two first morphological points are the bottom and top points of the hull. The four other points are used to defined the peripheral line of the hull (P_1,P_2,P_3,P_4 on figure 3.3). The two first points are located along the x-axis (P_1,P_2) and the two other along the y-axis (P_3,P_4) . Finally, the shape coefficients are versatile index which describe the curvature of the hull above and below the peripheral line.

Fields description

Name	Type	Defaults	Description
PosXRadius	Real	0.5	The Positive X Radius. y-coordinate of the first morphological point (P_1).
PosXHeight	Real	0	The Positive X Height. z-coordinate of the first morphological point (P_1).
NegXRadius	Real	0.5	The Negative X Radius. y-coordinate of the second morphological point (P_2).
NegXHeight	Real	0	The Negative X Height. z-coordinate of the second morphological point (P_2).
PosYRadius	Real	0.5	The Positive Y Radius. x-coordinate of the third morphological point (P_3).
PosYHeight	Real	0	The Positive Y Height. z-coordinate of the third morphological point (P_3).
NegYRadius	Real	0.5	The Negative Y Radius. x-coordinate of the fourth morphological point (P_4).
NegYHeight	Real	0	The Negative Y Height. z-coordinate of the fourth morphological point (P_4).
Bottom	Vector3	<0,0,-0.5>	The bottom point of the hull.
Top	Vector3	<0,0,0.5>	The top point of the hull.
BottomShape	Real	2	The bottom shape factor.
TopShape	Real	2	The top shape factor.
`Slices`	Integer	4	Specifies the number of subdivisions around the z-axis when discretizing the hull. It must be strictly positive.
`Stacks`	Integer	4	Specifies the number of subdivisions along the z-axis when discretizing the hull. It must be strictly positive.

Example

```
# A cyprus crown represented by an asymmetric hull
AsymmetricHull cyprus {
    PosXRadius 2 PosYRadius 2 NegXRadius 2 NegYRadius 2.5
    PosXHeight 1 PosYHeight 1 NegXHeight 1.5 NegYHeight 1.2
    Top <0,0.5,8> Bottom <0.5,0,0> TopShape 1.5 BottomShape 2
    Slices 5 Stacks 10
}
```

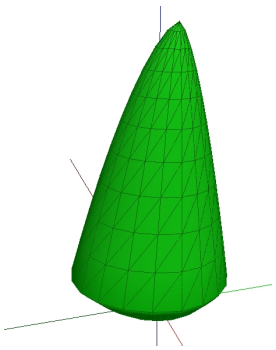


Figure 3.2: The `AsymmetricHull` object.

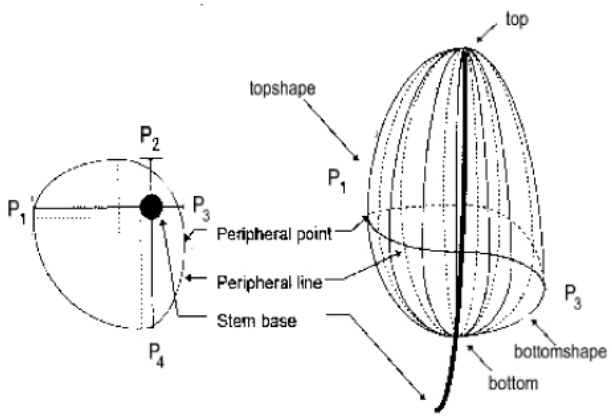


Figure 3.3: Geometry and Parameters of the AsymmetricHull

3.3 AxisRotated

Class

Geometry - Transformed

Description

The **AxisRotated** describes an object to which it has been applied a rotation of a specified angle about a specified axis. The rotation is given by the matrix:

$$M = \begin{bmatrix} (1-c)x^2 + c & txy - sz & (1-c)xy + sy \\ (1-c)xy + sz & (1-c)y^2 + c & (1-c)yz - sx \\ (1-c)xz - sy & (1-c)yz + sx & (1-c)z^2 + c \end{bmatrix},$$

where $s = sin(angle)$, $c = cos(angle)$, x the `x` coordinate of $axis$, y the `y` coordinate and z the `z` coordinate.

Fields description

Name	Type	Defaults	Description
`Axis`	Vector3	`<0,0,1>`	Specifies the axis of the rotation.
`Angle`	Real	`0`	Specifies the angle of the rotation. It must be in degrees.
`Geometry`	Geometry	None	Specifies the GEOM object which is affected by the rotation.

Example

```
# A AxisRotated Cylinder of 45 degrees about the x-axis
Cylinder cylinder {
    Height 8 Radius 1 Slices 16
}
AxisRotated axisrotated {
    Axis <1,0,0> Angle 45 Geometry cylinder
}
```

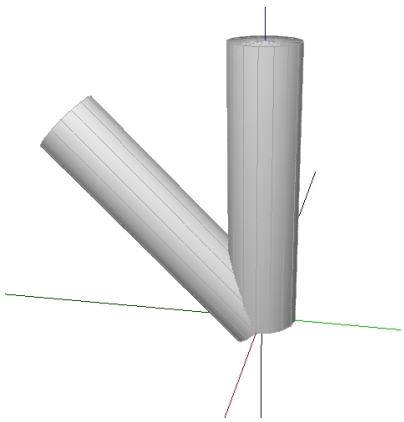


Figure 3.4: The `AxisRotated` object.

3.4 BezierCurve

Class

Geometry - Primitive - Curve

Description

The **BezierCurve** describes rational and non rational Bezier curves defined from the parametric equation

$$C(u) = \sum_{i=0}^{n} B_{i,n}(u) P_i \qquad 0 \leq u \leq 1$$

where n is called the degree of the curve, the $B_{i,n}(u)$ the classical n-th degree Bernstein polynomials and the geometric coefficients P_i the control points. For more information on this object, you could read [**?**].

Fields description

Name	Type	Defaults	Description
`Degree`	Integer	Computed	Specifies the degree of the curve. If not defined, This field is computed from the number of control points - 1. In the other case, a verification of the value's coherence is made.
`CtrlPointList`	Vector$d$[ ]	None	Specifies the control points of the curve. It could be points in 3 or 4 dimensions. With 4D control points, the curve is a Rational Bezier Curve.
`Stride`	Integer	30	Specifies the number of point to computed when discretizing the curve.

Example

```
# A bezier curve of degree 4
BezierCurve a_beziercurve {
CtrlPointList [ <0,0,0>, <4,-4,4>, <8,8,8>, <-12,12,12>, <-16,-16,16> ]
}
```

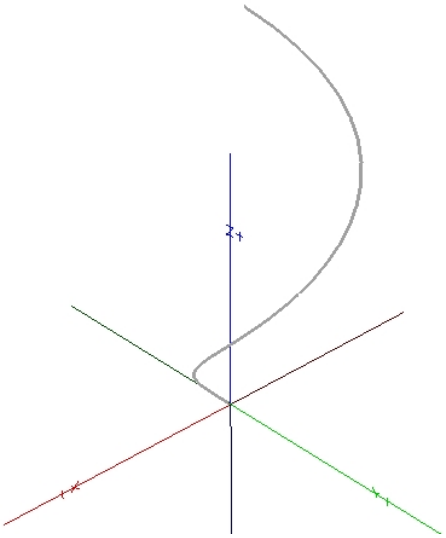


Figure 3.5: The `BezierCurve` object.

3.5 BezierCurve2D

Class

Geometry - Primitive - Planar Model - Planar Curve

Description

The **BezierCurve2D** describes 2D rational and non rational Bezier curves.

Fields description

Name	Type	Defaults	Description
`Degree`	Integer	Computed	Specifies the degree of the curve. If not defined, This field is computed from the number of control points - 1. In the other case, a verification of the value's coherence is made.
`CtrlPointList`	Vector$d$[ ]	None	Specifies the control points of the curve. It could be points in 2 or 3 dimensions. With 3D control points, the curve is a Rational Bezier Curve 2D.
`Stride`	Integer	30	Specifies the number of point to computed when discretizing the curve.

Example

```
# A bezier curve of degree 3
BezierCurve2D a_beziercurve {
    CtrlPointList [ <-4,-4>, <-2,4>, <2,-4>, <4,4> ]
}
```

Figure 3.6: The `BezierCurve2D` object.

3.6 BezierPatch

Class

Geometry - Primitive - Surface - Patch

Description

The **BezierPatch** describes rational and non rational Bezier surface defined from the parametric equation

$$S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_{i,p}(u) B_{j,q}(v) P_{i,j}$$

which is a bivariate equation where n and m are called the degrees of the surface, the $B_{i,p}(u)B_{j,q}(v)$ is the product of classical univariate p-th and q-th degrees Bernstein polynomials and the geometric coefficients $P_{i,j}$ a bidirectional net of control points. For more information on this object, you could read [**?**].

Fields description

Name	Type	Defaults	Description
`UDegree`	Integer	Computed	Specifies the degree of the surface. If not defined, This field is computed from the number of control points in the rows of the net -1. In the other case, a verification of the value's coherence is made.
`VDegree`	Integer	Computed	Specifies the degree of the surface. If not defined, This field is computed from the number of control points in the columns of the net - 1. In the other case, a verification of the value's coherence is made.
`CtrlPointMatrix`	Vector$d$[ ][]	None	Specifies the control points net of the surface. It could be points in 3 or 4 dimensions. With 4D control points, the surface is a Rational Bezier Surface.
`UStride`	Integer	30	Specifies the number of points in the first direction to compute when discretizing the surface.
`VStride`	Integer	30	Specifies the number of points in the second direction to compute when discretizing the surface.

Example

```
(#
   A bezier patch
#)

BezierPatch a_bezierpatch {
    CtrlPointMatrix [
        [ <5,0,2>, <5,2,3>, <5,4,0> ],
        [ <1,0,3>, <1,2,3>, <1,4,2> ],
        [ <-1,0,3>, <-1,2,3>, <-1,4,1> ],
        [ <-5,0,3>, <-5,2,4>, <-5,4,2> ]
    ]
}
```

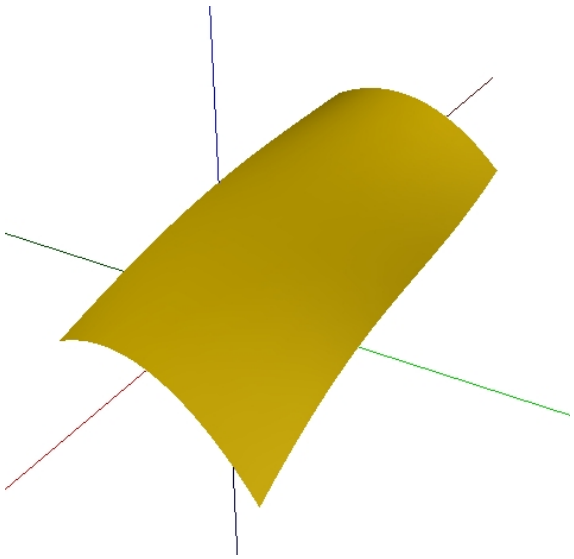


Figure 3.7: The `BezierSurface` object.

3.7 Box

Class

Geometry - Primitive - Volume - Mesh

Description

The **Box** describes a rectangular axis-aligned box centered at $(0, 0, 0)$ and whose extension along the `x`, `y` and `z`-axis is specified with a 3D vector.

Fields description

Name	Type	Defaults	Description
`Size`	Vector3	`<.5,.5,.5>`	Specifies the `x`, `y` and `z` extents of the box's width, depth and height. Each coordinates must be positive.

Example

```
# A Box
Box box {
    Size <2,1,6>
}
```

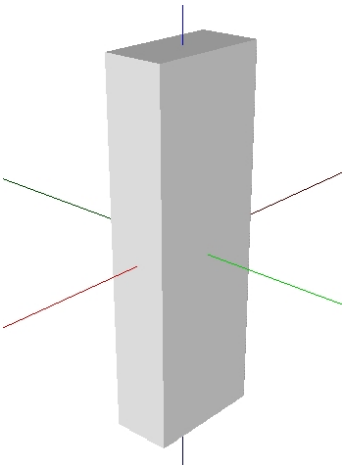


Figure 3.8: The `Box` object.

3.8 Cone

Class

Geometry - Primitive - Surface/Volume - SOR

Description

The **Cone** describes a cone whose base lies into the x-y plane and central axis is the z-axis.

Fields description

Name	Type	Defaults	Description
`Radius`	Real	`0.5`	Specifies the radius of the cone. It must be strictly positive.
`Height`	Real	`1`	Specifies the height of the cone. It must be strictly positive.
`Solid`	Boolean	`True`	Specifies whether the bottom side is visible.
`Slices`	Integer	`8`	Specifies the number of subdivisions around the z-axis when discretizing the cone.

Example

```
# A Cone
Cone cone {
    Radius 8
    Height 6
    Solid True
    Slices 32
}
```

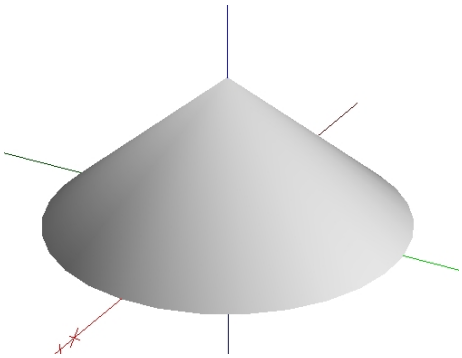


Figure 3.9: The `Cone` object.

3.9 Cylinder

Class

Geometry - Primitive - Surface/Volume - SOR

Description

The **Cylinder** describes a cylinder whose base lies into the x-y plane and central axis is the z-axis.

Fields description

Name	Type	Defaults	Description
`Radius`	Real	`0.5`	Specifies the radius of the cylinder. It must be strictly positive.
`Height`	Real	`1`	Specifies the height of the cylinder. It must be strictly positive.
`Solid`	Boolean	`True`	Specifies whether the bottom and top sides are visible.
`Slices`	Integer	`8`	Specifies the number of subdivisions around the z-axis when discretizing the cylinder.

Example

```
# A Cylinder
Cylinder cylinder {
    Radius 5
    Height 8
    Solid True
    Slices 64
}
```

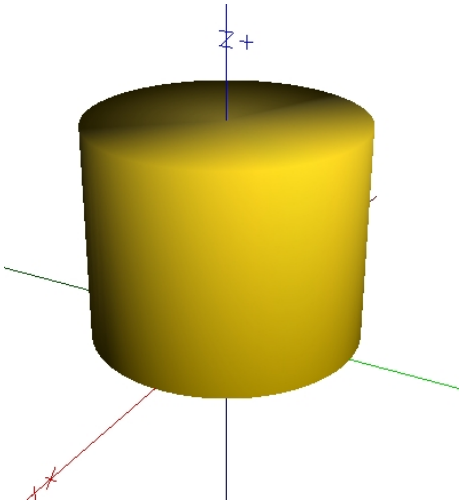


Figure 3.10: The `Cylinder` object.

3.10 Disc

Class

Geometry - Primitive - Planar Model - Planar Surface

Description

The **Disc** represents a disc centered at $(0, 0, 0)$ and lying into the x-y plane.

Fields description

Name	Type	Defaults	Description
`Radius`	Real	`0.5`	Specifies the radius of the disc. It must be strictly positive.
`Slices`	Integer	`8`	Specifies the number of subdivisions around the z-axis when discretizing the disc.

Example

```
# A Disc
Disc disc {
    Radius 4
    Slices 16
}
```

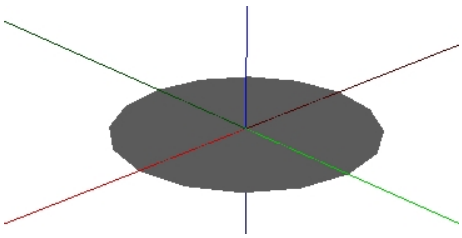


Figure 3.11: The `Disc` object.

3.11 ElevationGrid

Class

Geometry - Primitive - Surface - Patch

Description

The **ElevationGrid** describes a regular grid of a specified number of rows and columns and the elevation on each points of that grid. Heights are described row-major order (along the x-axis first), left to right and top to bottom.

It is mainly used for terrain modelling.

Fields description

Name	Type	Defaults	Description
`HeightMatrix`	Real[ ][ ]	None	specify the elevation associated to each points of the grid. Each arrays must contain exactly the same number of values and must contain at leat two values.
`XSpacing`	Real	`1`	Specifies the distance between two consecutive points in the x-direction. It must be strictly positive.
`YSpacing`	Real	`1`	Specifies the distance between two consecutive points in the y-direction. It must be strictly positive.

Example

```
# A digital terrain represented with an ElevationGrid.
ElevationGrid elevation_grid {
    HeightMatrix [
        [0.0, 1.5, 2.0, 1.5],
        [0.5, 1.8, 2.2, 1.7],
        [0.8, 2.2, 2.8, 2.2],
        [1.2, 2.6, 3.2, 2.8],
        [1.0, 2.4, 2.5, 2.1],
        [0.8, 1.8, 2.0, 1.6]
    ]
    XSpacing 5
    YSpacing 5
}
```

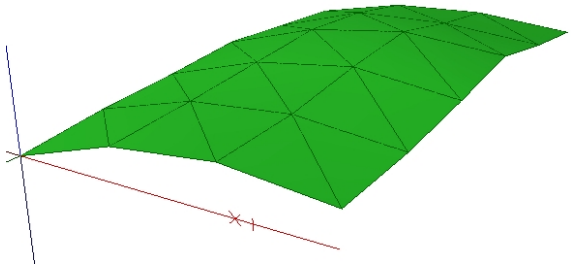


Figure 3.12: The `ElevationGrid` object.

3.12 EulerRotated

Class

Geometry - Transformed

Description

The **EulerRotated** describes an object to which it has been applied a composition of rotations by *roll* about the x-axis, by *elevation* about the rotated y-axis and by *azimuth* about the rotated z-axis. The equivalent rotation is given by the matrix:

$$M = \begin{bmatrix} ca * ce & ca * se * sr - sa * cr & ca * se * cr + sa * sr \\ sa * ce & ca * cr + sa * se * sr & sa * se * cr - ca * sr \\ -se & ce * sr & ce * cr \end{bmatrix},$$

where $cr = cos(roll)$, $sr = sin(roll)$, $ce = cos(elevation)$, $se = sin(elevation)$, $ca = cos(azimuth)$ and $sa = sin(azimuth)$.

Fields description

Name	Type	Defaults	Description
`Azimuth`	Real	`0`	Specifies the angle of the rotation about the z-axis.
`Elevation`	Real	`0`	Specifies the angle of rotation about the rotated y-axis.
`Roll`	Real	0	Specifies the angle of rotation about the rotated x-axis.
`Geometry`	Geometry	None	Specifies the GEOM object which is affected by the rotation.

Example

```
# A EulerRotated Box
Box box {

    Size <2,0.2,4>

}
EulerRotated axisrotated {

    Azimuth 45
    Elevation 0
    Roll 60
    Geometry box
```

```
}
```

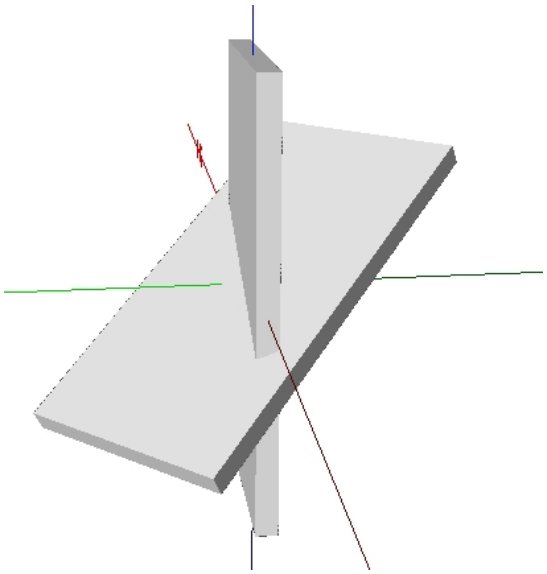


Figure 3.13: The `EulerRotated` object

3.13 ExtrudedHull

Class

Geometry - Primitive - Volume - Hull

Description

The **Extruded Hull** describes an object of class of *Hull* extruded by a vertical and an horizontal profiles. For more information on this model, see Birnbaum Thesis [**?**].

Fields description

Name	Type	Defaults	Description
Vertical	Planar Curve	None	Specifies the vertical profile of the hull. There must be at least one point.
Horizontal	Planar Curve	None	Specifies the horizontal profile of the hull. There must be at least one point.
`CCW`	Boolean	`True`	Indicates whether each polygon's points are listed in counterclockwise (`True`) or clockwise (`False`) order when viewed from the front.

Example

```
ExtrudedHull hull {
    Vertical Polyline2D {
        PointList [ <0,-3.5>, <-1,-3>, <-3,-1>,
           <-3,1>, <-1,3.2>, <1,3.2>,
           <3,1>, <3,-1>, <1,-3>,
           <1,-3.5>, <0.5,-4>, <0,-5> ]
    }
    Horizontal Polyline2D {
        PointList [ <-3.8,0>, <-4,-1>, <-2,-3>,
           <0,-4>, <0,-4>, <1,-3>,
           <3,-3>, <5,-2>, <5,0>,
           <4,1>, <3,1>, <2,2>,
           <2,3>, <1,4>, <-1,4>,
           <-2,3.75>, <-3,2.8>, <-3,1> ]
    }
}
```

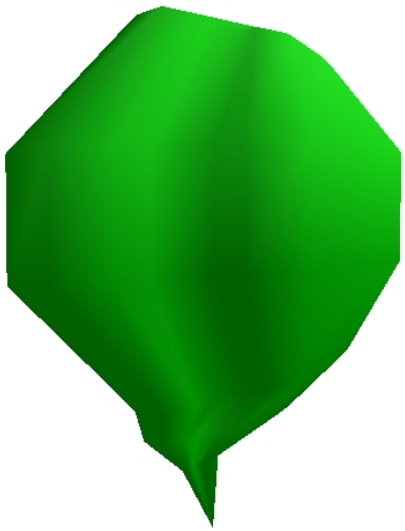


Figure 3.14: The `ExtrudedHull` object.

3.14 Extrusion

Class

Geometry - Surface/Volume

Description

The **Extrusion** differs from straight Cylinder in that a space curve may act as the central axis and the cross section may vary in size or shape. Because of this variability, the Extrusion can represent a wide variety of forms. This model is one type of *swept surface*. For more information on this model, see Bloomenthal Thesis [**?**].

Fields description

Name	Type	Defaults	Description
`CrossSection`	Curve2D	None	The Cross Section of the Extrusion.
`Axis`	Curve	None	The Axis of the Extrusion.
`Scale`	Vector2[]	[<1,1>]	The scaling factors for 2D scaling transformation. The cross Section is scaled using theses factors
`Orientation`	Real[]	[0]	The orientation angles for 2D orientation transformation.The cross Section is oriented using thoses angles
`KnotList`	Real[]	[0,1]	We consider the scaling function to be linear between two knots. Must have the same size than Scale
`CCW`	Boolean	`True`	Indicates whether cross section's points are listed in counterclockwise (`True`) or clockwise (`False`) order when viewed from the front.
`Solid`	Boolean	`False`	Specifies whether the bottom and top sides are visible.

Example

```
# Definition of the Cross Section.
```

```
BezierCurve2D _crossSection {
    CtrlPointList [ <-2,0>, <-2,-2>, <2,-2>,
                    <2,2>,  <-2,2>,  <-2,0> ]
    Stride 10
}
# Definition of the axis
BezierCurve _axis {
    CtrlPointList [ <0,0,0>, <3,3,3>, <0,0,6>,
                    <-3,-3,9>, <0,0,12> ]
    Stride 10
}
# The Extrusion
Extrusion an_extrusion {
    Axis _axis
    CrossSection _crossSection
    Scale [ <5,5>,<2,2>,<0.2,0.2> ]
    KnotList [ 0, 0.2,1 ]
}
```

Figure 3.15: The `Extrusion` object.

3.15 FaceSet

Class

Geometry - Primitive - Surface/Volume - Mesh

Description

A **FaceSet** describes a surface formed by connected faces. Polygons are specified using indices into a list of vertices located at the specified coordinates.

Fields description

Name	Type	Defaults	Description
`PointList`	Vector3[ ]	None	Specifies the coordinates of the constructing points.
`IndexList`	Integer[ ][ ]	None	Specifies the faces, each specified as a series of indices into the list of points. Be sure that the indices are not out of range and each series must contain at least 3 elements.
`CCW`	Boolean	`True`	Indicates whether each polygon's points are listed in counterclockwise (`True`) or clockwise (`False`) order when viewed from the front.
`Solid`	Boolean	`False`	Specifies whether this mesh represent a closed surface.
`Skeleton`	Geometry	`Null`	Specifies the skeleton of this mesh. It must be a GEOM object of type of Polyline.

Example

```
FaceSet my_faceset {
    PointList [ <2.5,0,0>, <2.5,0,4.5>, <1.25,2.16506,0>,

        <1.25,2.16506,4.5>, <-1.25,2.16506,0>, <-1.25,2.16506,4.5>,
        <-2.5,0,0>, <-2.5,0,4.5>, <-1.25,-2.16506,0>,
        <-1.25,-2.16506,4.5>, <1.25,-2.16506,0>, <1.25,-2.16506,4.5>,
        <0,0,-1.5>, <0,0,6> ]
    IndexList [ [0,2,3,1], [1,3,13], [0,12,2],
        [2,4,5,3], [3,5,13], [2,12,4],
        [4,6,7,5], [5,7,13], [4,12,6],
        [6,8,9,7], [7,9,13], [6,12,8],
        [8,10,11,9], [9,11,13], [8,12,10],
        [10,0,1,11], [11,1,13], [10,12,0]
    ] Solid True Skeleton Polyline {
        PointList [ <0,0,0>, <0,0,4.5> ]
    }
}
```

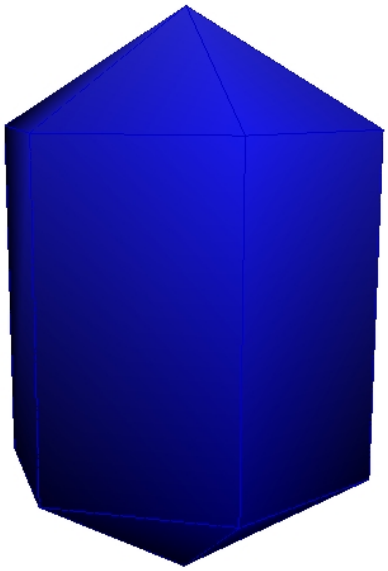


Figure 3.16: The `FaceSet` object.

3.16 Frustum

Class

Geometry - Primitive - Surface/Volume - SOR

Description

The **Frustum** describes a cone frustum whose base lies into the x-y plane and central axis is the z-axis. The ratio between the top radius and the base radius is defined by a taper rate such as: $t = \frac{r_{top}}{r_{base}}$, where t denotes the taper rate, r_{top} the top radius and r_{base} the base radius.

Fields description

Name	Type	Defaults	Description
`Radius`	Real	`0.5`	Specifies the radius of the frustum. It must be strictly positive.
`Height`	Real	`1`	Specifies the height of the frustum It must be strictly positive.
`Taper`	Real	`0`	Specifies the rate of taper. It must be positive.
`Solid`	Boolean	`True`	Specifies whether the bottom and top sides are visible.
`Slices`	Integer	`8`	Specifies the number of subdivisions around the z-axis when discretizing the frustum.

Example

```
# An empty cone frustum
Frustum frustum {

    Radius 4
    Height 8
```

```
        Taper 0.25
        Slices 64
        Solid False # empty
    }
```

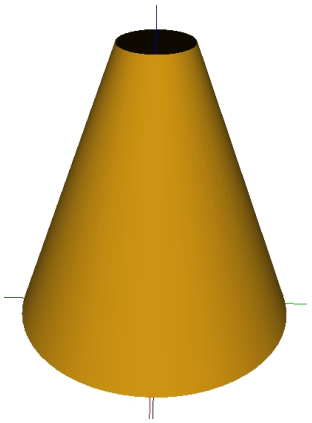


Figure 3.17: The `Frustum` object.

3.17 Group

Class

Geometry - Group

Description

A **Group** combines a list of GEOM objects in order to build a more complex object.

Fields description

Name	Type	Defaults	Description
`GeometryList`	Geometry[ ]	None	Specifies the list of Geoms which are to be grouped.
`Skeleton`	Geometry	`Null`	Specifies the skeleton of this group. It must be a GEOM object of type of Polyline.

Example

```
(#
   A conifer represented by a Group. The trunk is represented using a
   Cylinder, and the crown is represented using a Translated Cone.
#)
Group group {
    GeometryList [
        Cylinder { # the trunk
           Height 3
           Radius 0.8
           Slices 32
        },
        Translated { # the crown
           Translation <0,0,3>
           Geometry Cone {
             Radius 3
             Height 7
             Solid True
             Slices 32
           }
        }
    ]
}
```

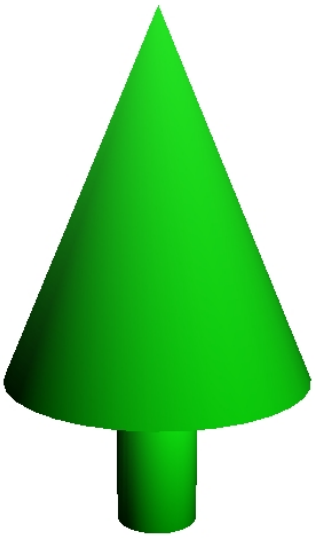


Figure 3.18: The `Group` object.

3.18 Iterated Function System (IFS)

Class

Geometry - Transformed

Description

The **IFS** describes an object to which it has been applied iteratively a set of affine transformations. It is a method for generating fractals and the complexity is exponential with regard to numbers of iterations. The function is computed in the following way:

Given a set of affine transformations $T_1, T_2, ..., T_n$.

Then $F = T_1 \bigcup T_2 \bigcup ... \bigcup T_n$.

We iterate the process *depth* time, then $IFS = F^{depth} = F \circ F \circ ... \circ F$.

Fields description

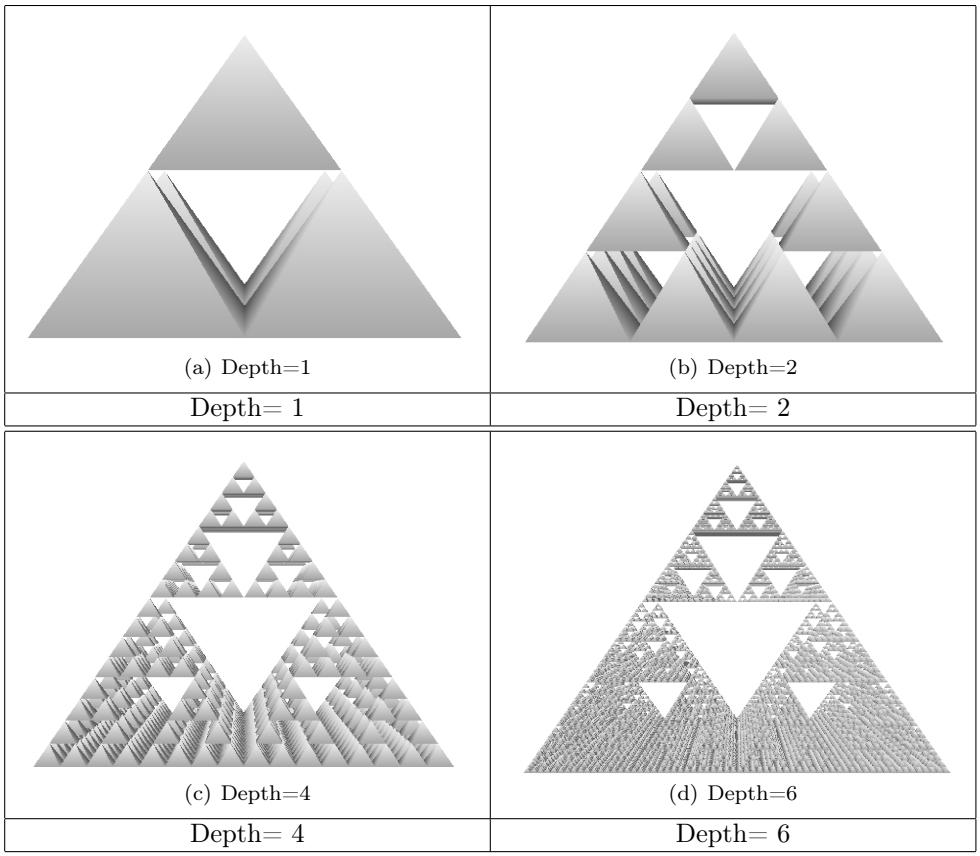
Name	Type	Defaults	Description
`Depth`	Integer	`1`	Specifies the number of iterations. It must be positive.
`TransformationList`	Transformation[]	None	Specifies the list of affine transformations which are to be applied at each iteration.
`Geometry`	Geometry	None	Specifies the GEOM object which is affected by the IFS.

Fields description of a transformation

Name	Type	Defaults	Description
`Translation`	Vector3	`<0,0,0>`	Specifies translation vector.
`Scale`	Vector3	`<1,1,1>`	Specifies the scaling factors along the `x`-axis, the `y`-axis and the `z`-axis. Each of the coordinates must be different from 0.
`Rotation`	Rotation	Identity	Specifies the ortho-normal transformation: EulerRotation (see EulerRotated), AxisRotation (see AxisRotated) or BaseOrientation (see Oriented).

Example

```
(#
   The Spiersinki pyramide
#)
Tapered pyramid {
    BaseRadius 1.0
    TopRadius 0
    Primitive Box {
        Size <1.0, 1.0, 1.0>
    }
}
IFS _ifs {
    Depth 5
    Geometry pyramid
    TransfoList [
        Transfo {
           Translation <0,0,0.5>
           Scale <1/2,1/2,1/2>
        },
        Transfo {
           Translation <.5,.5,-0.5>
           Scale <1/2,1/2,1/2>
        },
        Transfo {
           Translation <-0.5,0.5,-0.5>
           Scale <1/2,1/2,1/2>
        },
        Transfo {
           Translation <0.5,-0.5,-0.5>
           Scale <1/2,1/2,1/2>
        },
        Transfo {
           Translation <-0.5,-0.5,-0.5>
           Scale <1/2,1/2,1/2>
        }
    ]
}
```



(a) Depth=1 (b) Depth=2

Depth= 1 Depth= 2

(c) Depth=4 (d) Depth=6

Depth= 4 Depth= 6

Figure 3.19: The `IFS` object.

3.19 Material

Class

Appearance - Color

Description

A **Material** defined the Geometry material properties. The fields in the Material object determine the way light reflect off an object to create color.

Fields description

Name	Type	Defaults	Description
Ambient	Color3	<160,160,160>	Specifies how much ambient light the surface must reflect. Depending on the ambient light, this field define the color of the object.
Diffuse	Real	1.0	Specifies the diffuse color, which reflects all light sources depending on the angle of the surface with respect to the light source. The more directly the surface faces the light, the more diffuse light is reflected. The diffuse color is computed as *Ambient * Diffuse* and must be a valid Color*3*.
Specular	Color3	<0,0,0>	Specifies the color of an object 's highlights.
Emission	Color3	<0,0,0>	Specifies the light produced by a glowing object. Emissive color is useful for displaying radiosity-based models (where the light energy of the scene is computed explicitly).
Shininess	Real	0	degree of shininess of an object, ranging from 0.0 for a diffuse surface with no shininess to 1.0 for a highly polished surface.
Transparency	Real	0	degree of transparency of an object, ranging from 0.0 for a completely opaque surface to 1.0 for a completely clear surface.

Example

```
#  Some materials
Material RED7 {
    Ambient <224,0,0>
}
Material RED {
    Ambient Red
}
```

Color3 Constants Values

```
Black = <0,0,0>
White = <255,255,255>
Red   = <255,0,0>
Green = <0,255,0>
Blue  = <0,0,255>
Cyan  = <0,255,255>
Magenta = <255,0,255>
Yellow  = <255,255,0>
```

Default Material

The default material used if none is specified :

```
Material Default {
    Ambient <160,160,160> Diffuse 1.0 Specular <0,0,0> Shininess 0
    Transparency 0
}
```

3.20 MonoSpectral

Class

Appearance - Physical Model

Description

Fields description

Name	Type	Defaults	Description
Reflectance	Real	0.8	Must be in [0,1].
Transmittance	Real	0.0	Must be in [0,1].

Example

3.21 MultiSpectral

Class

Appearance - Physical Model

Description

Fields description

Name	Type	Defaults	Description
Reflectance	Real[]	None	Each value must be in [0,1].
Transmittance	Real[]	None	Each value must be in [0,1].
Filter	Index3	[1,1,1]	

Example

3.22 NurbsCurve

Class

Geometry - Primitive - Curve

Description

The **NurbsCurve** describes rational and non rational B-Spline curves defined from the parametric equation

$$C(u) = \sum_{i=0}^{n} R_{i,p}(u) P_i \qquad a \leq u \leq b$$

where p is called the degree of the curve, the $R_{i,p}(u)$ a p-th degree rational basis functions define on a clamped knot vector $U = \{ \underbrace{a, ..., a}_{p+1}, u_{p+1}, ..., u_{n-p-1}, \underbrace{b, ..., b}_{p+1} \}$ and the geometric coefficients P_i the control points. For more information on this object, you could read [**?**].

Fields description

Name	Type	Defaults	Description
`Degree`	Integer	3 or Computed	Specifies the degree of the curve. If not specified and if the KnotList isn't defined too, the degree value is assign to 3. If KnotList specified, the degree value is assign to the number of knots - the number of control points -1.
`CtrlPointList`	Vector$d$[ ]	None	Specifies the control points of the curve. It could be points in 3 or 4 dimensions. With 4D control points, the curve is a Rational B-Spline Curve.
`KnotList`	Real[ ]	Computed	Specifies the knot value of the curve. If not defined, This field take the value of an adapted clamped uniform vector and the curve is a Uniform B-Spline Curve. The KnotList must be clamped and increasing. If specified and if the degree is specified, the number of knot must be equal to the number of control points + degree +1.
`Stride`	Integer	50	Specifies the number of point to compute when discretizing the curve.

Example

```
# A uniform B-Spline Curve of degree 11
NurbsCurve five {
    CtrlPointList [ <0,0,0>, <-4,0,0>, <-4,4,4>,
                    <0,4,4>, <0,4,4>,  <0,4,4>,
                    <0,8,8>, <0,8,8>,  <0,8,8>,
                    <-4,8,8>,<-4,8,8>, <-4,8,8>]
}
```

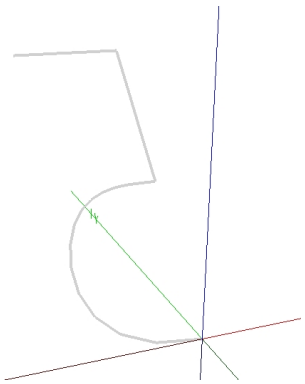


Figure 3.20: The `NurbsCurve` object.

3.23 NurbsCurve2D

Class

Geometry - Primitive - Planar Model - Planar Curve

Description

The **NurbsCurve2D** describes 2D rational and non rational B-Spline curves.

Fields description

Name	Type	Defaults	Description
`Degree`	Integer	3 or Computed	Specifies the degree of the curve. If not specified and if the KnotList isn't defined too, the degree value is assign to 3. If KnotList specified, the degree value is assign to the number of knots - the number of control points -1.
`CtrlPointList`	Vector$d$[ ]	None	Specifies the control points of the curve. It could be points in 2 or 3 dimensions. With 3D control points, the curve is a 2D Rational B-Spline Curve.
`KnotList`	Real[ ]	Computed	Specifies the knot value of the curve. If not defined, This field take the value of an adapted clamped uniform vector and the curve is a Uniform B-Spline Curve. The KnotList must be clamped and increasing. If specified and if the degree is specified, the number of knot must be equal to the number of control points + degree +1.
`Stride`	Integer	50	Specifies the number of point to compute when discretizing the curve.

Example

```
# A 2D uniform B-Spline Curve of degree 15
NurbsCurve2D spade {
```

```
        Degree 15
        Stride 100
        CtrlPointList [ <0,5>, <-2,5>, <-2,5>, <-2,5>,
            <0,5>, <0,1>, <0,1>, <0,4>, <-3,3.5>,<-4,2>, <-2,-1.5>,
            <0,-4>, <0,-4>, <0,-4>, <0,-4>, <0,-4>, <0,-4>,
            <2,-1.5>, <4,2>, <3,3.5>, <0,4>, <0,1>, <0,1>, <0,5>,
            <2,5>,<2,5>,<2,5>,<0,5>
        ]
    }
```

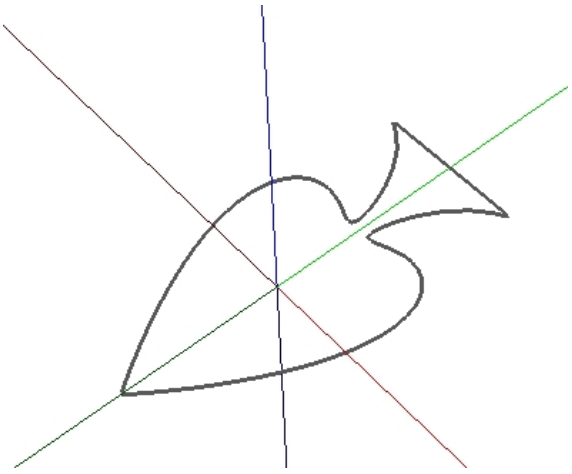


Figure 3.21: The `NurbsCurve2D` object.

3.24 NurbsPatch

Class

Geometry - Primitive - Surface - Patch

Description

The **NurbsPatch** describes rational and non rational B-Spline surface defined from the parametric equation

$$S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} R_{i,p}(u) R_{j,q}(v) P_{i,j}$$

which is a bivariate equation where p and q are called the degrees of the surface, the $R_{i,p}(u)R_{j,q}(v)$ is the product of univariate p-th and q-th degrees rational basis functions, defined on $U = \{ \underbrace{a,...,a}_{p+1}, u_{p+1},...,u_{n-p-1}, \underbrace{b,...,b}_{p+1} \}$ and $V = \{ \underbrace{c,...,c}_{q+1}, v_{q+1},...,v_{m-q-1}, \underbrace{d,...,d}_{q+1} \}$, and the geometric coefficients $P_{i,j}$ a bidirectional net of control points. For more information on this object, you could read [**?**].

Fields description

Name	Type	Defaults	Description
`UDegree`	Integer	3 or Computed	Specifies the degree of the surface in the first direction. If not specified and if the UKnotList isn't defined too, the udegree value is assign to 3. If UKnotList specified, the udegree value is assign to the number of knots of UKnotList - the number of control points in a row - 1.
`VDegree`	Integer	3 or Computed	Specifies the degree of the surface in the second direction. If not specified and if the VKnotList isn't defined too, the vdegree value is assign to 3. If VKnotList specified, the vdegree value is assign to the number of knots of VKnotList - the number of control points in a column - 1.
`CtrlPointMatrix`	Vector$d$[ ]	None	Specifies the control points net of the curve. It could be points in 3 or 4 dimensions. With 4D control points, the curve is a Rational B-Spline Surface.
`UKnotList`	Real[ ]	Computed	Specifies the knot value of the surface in the first direction. If not defined, This field take the value of an adapted clamped uniform vector. The UKnotList must be clamped and increasing. If specified and if the udegree is specified, the number of knot must be equal to the number of control points in a row + udegree +1.
`VKnotList`	Real[ ]	Computed	Specifies the knot value of the surface. If not defined, This field take the value of an adapted clamped uniform vector. The VKnotList must be clamped and increasing. If specified and if the vdegree is specified, the number of knot must be equal to the number of control points in a column + vdegree +1.
`UStride`	Integer	50	Specifies the number of point to compute when discretizing the curve in the first direction.
`VStride`	Integer	50	Specifies the number of point to compute when discretizing the curve in the second direction.

Example

```
# A uniform B-Spline Patch of degree 3
NurbsPatch a_nurbspatch {
    CtrlPointMatrix [

        [ <5,-3,2,1>, <5,-1,2,1>, <5,0,-2,1>, <5,2,-2,1>, <5,3,0,1> ],
        [ <0,-3,2,1>, <0,-1,2,5>, <0,0,-2,5>, <0,2,-2,1>, <0,3,0,1> ],
        [ <-3,-3,4,1>, <-3,-1,4,5>, <-3,0,0,5>, <-3,1,0,1>, <-3,2,2,1> ],
        [ <-6,-3,4,1>, <-6,-1,4,1>, <-6,0,0,1>, <-6,1,0,1>, <-6,2,2,1> ],
        [ <-6,-3,6,1>, <-6,-1,6,1>, <-6,0,2,1>, <-6,1,2,1>, <-6,2,4,1> ] ]
}
```

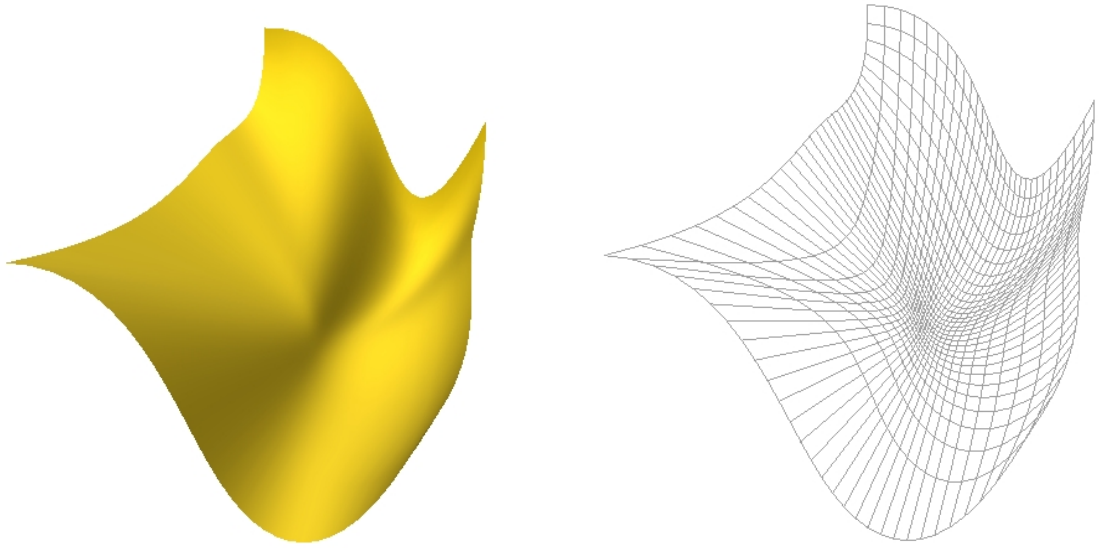


Figure 3.22: The `NurbsPatch` object.

3.25 Oriented

Class

Geometry - Transformed

Description

The **Oriented** describes an object to which it has been applied a change of coordinate specified by an orthonormal basis. The basis is expressed by the matrix:

$$M = \begin{bmatrix} p_x & s_x & t_x \\ p_y & s_y & t_y \\ p_z & s_z & t_z \end{bmatrix}$$

where (p_x, p_y, p_z) denotes the primary direction, (s_x, s_y, s_z) the secondary direction and (t_x, t_y, t_z) the ternary direction, which is given by: $t = p \wedge s$.

Fields description

Name	Type	Defaults	Description
`Primary`	Vector3	`<1,0,0>`	Specifies the primary direction. The vector will be automatically normalized if needed.
`Secondary`	Vector3	`<0,1,0>`	Specifies the secondary direction. It must be orthogonal to the primary direction, that is, the dot product between those 2 vectors must be 0. The vector will be automatically normalized if needed.
`Geometry`	Geometry	None	Specifies the GEOM object which is affected by the change of basis.

Example

```
# A cross represented by a Box and an Oriented Box
Box box {
    Size <10,0.5,0.5>
}
Oriented oriented {
    Primary <0,1,0> # the box is oriented along the y-axis
    Secondary <-1,0,0> Geometry box
}
```

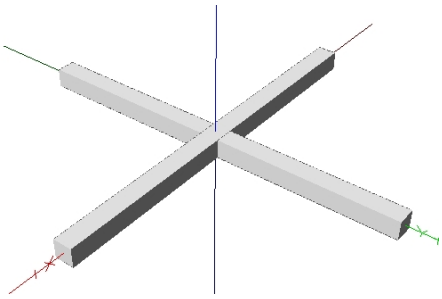


Figure 3.23: The `Oriented` object.

3.26 Paraboloid

Class

Geometry - Primitive - Surface/Volume - SOR

Description

The **Paraboloid** describes a surface of revolution whose generatrix is given by the equation $y = h - \left(\frac{x}{r}\right)^{\alpha}$, where h denotes the height, r the radius and α the shape factor.

According the shape factor value the surface represents:

- a Neiloid if $0 < \alpha < 1$
- a Cone if $\alpha = 1$
- a Paraboloid if $\alpha > 1$
- a Cylinder if $\alpha \to +\infty$

Fields description

Name	Type	Defaults	Description
`Radius`	Real	`0.5`	Specifies the radius of the paraboloid. It must be strictly positive.
`Height`	Real	`1`	Specifies the height of the paraboloid. It must be strictly positive.
`ShapeFactor`	Real	`0`	Specifies the concavity. It must be strictly positive.
`Solid`	Boolean	`True`	Specifies whether the bottom and top sides are visible.
`Slices`	Integer	`8`	Specifies the number of subdivisions around the z-axis when discretizing the paraboloid.
`Stacks`	Integer	`8`	Specifies the number of subdivisions along the z-axis when discretizing the paraboloid.

Example

```
# A neiloid represented by a Paraboloid
Paraboloid paraboloid {
    Radius 5 Height 8 ShapeFactor 0.6 Solid True Slices 25
    Stacks 25
}
```

3.27 PointSet

Class

Geometry - Primitive

Description

The **PointSet** describes a set of points, each located at the specified coordinates.

Fields description

Name	Type	Defaults	Description
`PointList`	Vector3[ ]	None	Specifies the coordinates of the constructing points of this point set. There must be at least one point.

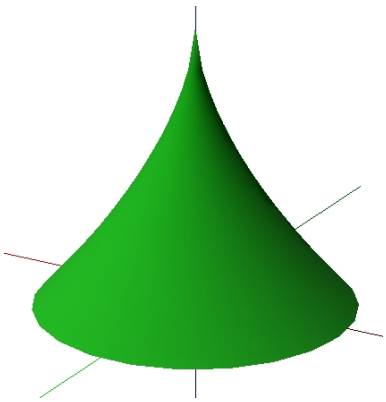


Figure 3.24: The `Paraboloid` object.

Example

```
# Corners of a cube represented by a PointSet
PointSet point_set {
    PointList [ <5,-5,-5>, <-5,-5,-5>, <-5,5,-5>, <5,5,-5>,
    <5,-5,5>, <-5,-5,5>, <-5,5,5>, <5,5,5> ]
}
```

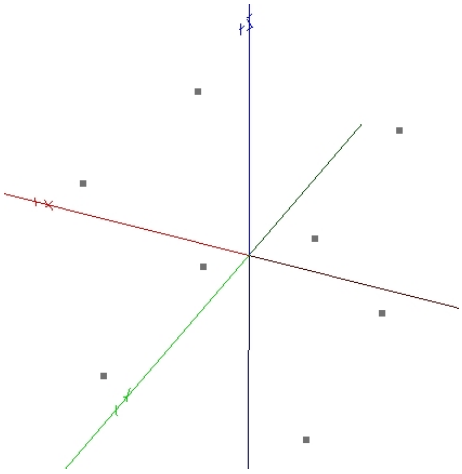


Figure 3.25: The `PointSet` object.

3.28 Polyline

Class

Geometry - Primitive - Curve

Description

A **Polyline** describes a curve formed by connected segment located at specified coordinates.

Fields description

Name	Type	Defaults	Description
`PointList`	Vector3[ ]	None	Specifies the coordinates of the constructing points of this polyline. There must be at least two points.

Example

```
# A spiral represented by a Polyline
Polyline polyline {
    PointList [ <-4,0,0>, <-3,3,0.5>, <0,4,1>, <3,3,1.5>,
        <4,0,2>, <3,-3,2.5>, <0,-4,3>, <-3,-3,3.5>,
        <-4,0,4>, <-3,3,4.5>, <0,4,5>, <3,3,5.5>,
        <4,0,6>, <3,-3,6.5>, <0,-4,7>, <-3,-3,7.5> ]
}
```

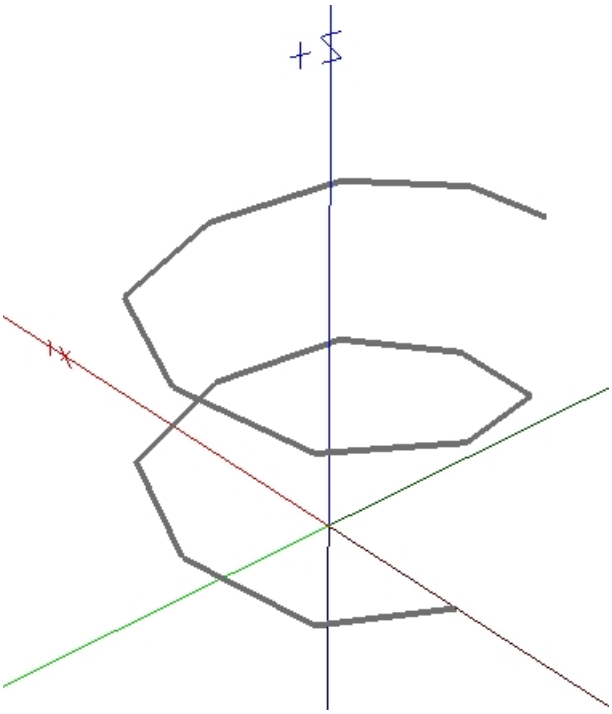


Figure 3.26: The `Polyline` object.

3.29 Polyline2D

Class

Geometry - Primitive - Planar Model - Planar Curve

Description

A **Polyline2D** describes a planar curve formed by connected segment located at specified 2D coordinates.

Fields description

Name	Type	Defaults	Description
`PointList`	Vector2[ ]	None	Specifies the coordinates of the constructing points of this polyline. There must be at least two points.

Example

```
# A spiral represented by a Polyline2D
Polyline2D polyline2D {
    PointList [  <0,0>,   <1,0>,  <1,1>, <-2,1>,
                 <-2,-2>, <3,-2>, <3,3>, <-4,3>,
                 <-4,-4>, <5,-4>, <5,5>, <-6,5>,
                 <-6,-6>, <7,-6>, <7,7> ]
}
```

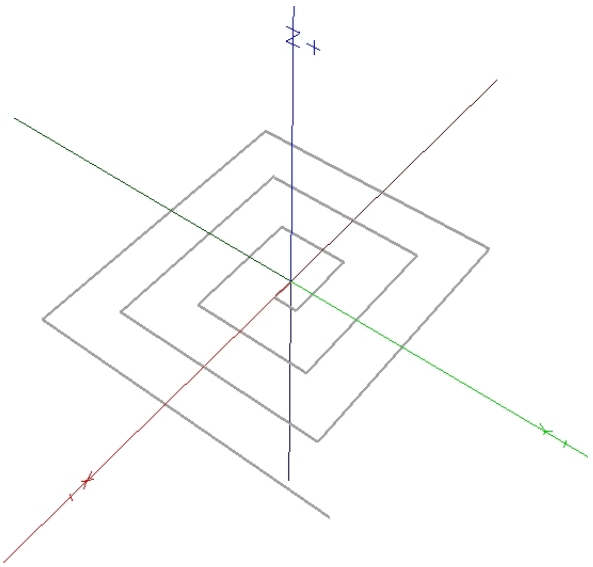


Figure 3.27: The `Polyline2D` object.

3.30 QuadSet

Class

Geometry - Primitive - Surface/Volume - Mesh

Description

A **QuadSet** describes a surface formed by quadrilaterals, four sided polygons. Quads are specified using indices into a list of vertices located at the specified coordinates.

Fields description

Name	Type	Defaults	Description
`PointList`	Vector3[ ]	None	Specifies the coordinates of the constructing points.
`IndexList`	Integer[ ][ ]	None	Specifies the faces, each specified as a series of indices into the list of points. Be sure that the indices are not out of range and each series must contain at exactly four elements.
`CCW`	Boolean	`True`	Indicates whether each quadrilateral's points are listed in counterclockwise (`True`) or clockwise (`False`) order when viewed from the front.
`Solid`	Boolean	`False`	Specifies whether this mesh represent a closed surface.
`Skeleton`	Geometry	`Null`	Specifies the skeleton of this mesh. It must be a GEOM object of type of Polyline.

Example

```
# A sheared cube represented by a QuadSet
QuadSet quad_set {
    PointList [ <2,-5,-5>, <-5,-5,-5>, <-5,5,-5>, <2,5,-5>,
                <5,-5,5>, <-2,-5,5>, <-2,5,5>, <5,5,5>
    ]
    IndexList [ [0,1,2,3], [0,3,7,4], [1,0,4,5],
                [2,1,5,6], [3,2,6,7], [4,7,6,5]
    ]
    CCW True
    Solid True
}
```

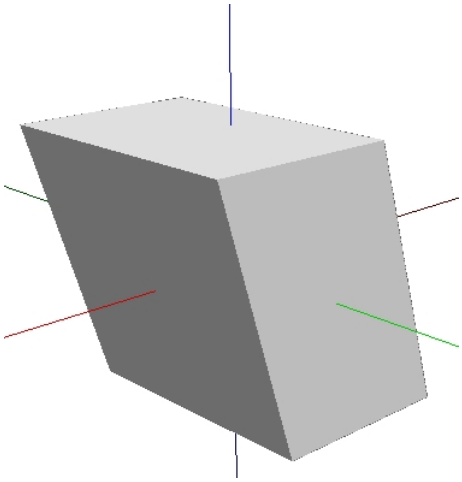


Figure 3.28: The `QuadSet` object.

3.31 Revolution

Class

Geometry - Primitive - Surface/Volume - SOR

Description

The **Revolution** primitive describes a general surface of revolution generated by the rotation of a planar curve about the z-axis.

The number of points within the generatrix curve determines the number of subdivisions along the z-axis when discretizing the object.

Fields description

Name	Type	Defaults	Description
`PointList`	Vector2[ ]	None	Specifies the coordinates of the 2D curve to be rotated.
`Slices`	Integer	`8`	Specifies the number of subdivisions around the z-axis when generating the solid.

Example

```
# A glass represented using a Revolution
Revolution revolution {
    PointList [
        <3,0>, <1,0.2>,
        <0.7,0.3>, <0.5,0.6>,
        <0.5,4>, <1,5>,
        <3,10>, <4,11>
    ]
    Slices 64
}
```

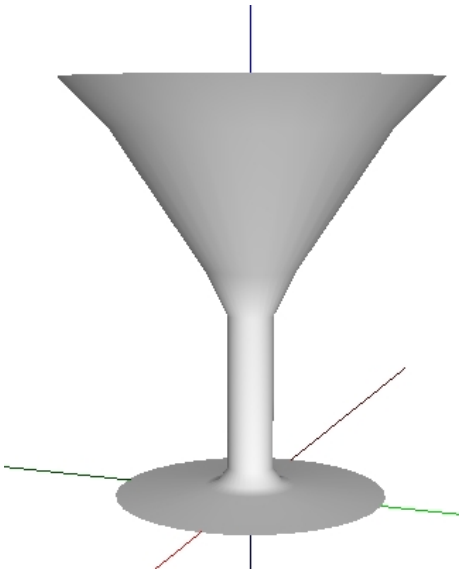


Figure 3.29: The `Revolution` object

3.32 Scaled

Class

Geometry - Transformed

Description

The **Scaled** describes an object to which it has been applied an anisotropic scale. The scaling transformation is given by the matrix:

$$M = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix},$$

where (s_x, s_y, s_z) denotes the scaling factors along the `x`, `y` and `z`-axis.

Fields description

Name	Type	Defaults	Description
`Scale`	Vector3	`<1,1,1>`	Specifies the scaling factors along the `x`-axis, the `y`-axis and the `z`-axis. Each of the coordinates must be different from 0.
`Geometry`	Geometry	None	Specifies the GEOM object which is affected by the scale.

Example

```
# A Scaled Cylinder along the x-axis and the z-axis
Cylinder cylinder {
    Height 5 Radius 2 Slices 16
}
Scaled scaled {
    Scale <3,1,2> Geometry cylinder
```

```
}
```

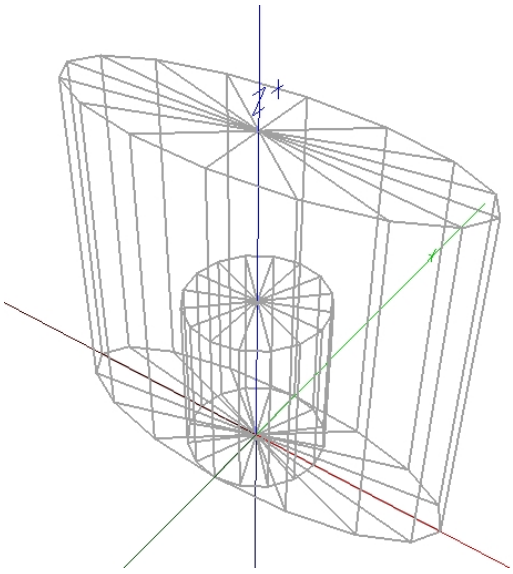


Figure 3.30: The `Scaled` object.

The original cylinder is bounded by the scaled cylinder.

3.33 Shape

Class

Shape

Description

The **Shape** object associate a geometric object with an appearance.

Fields description

Name	Type	Defaults	Description
`Geometry`	Geometry	`None`	Specifies the geometry of the shape.
`Appearance`	Appearance	`Default Material`	Specifies the appearance of the shape.

Example

```
Shape {
    Geometry crown Appearance green
}
```

3.34 Sphere

Class

Geometry - Primitive - Volume - SOR

Description

The **Sphere** describes a sphere of a specified radius and centered at $(0, 0, 0)$.

Fields description

Name	Type	Defaults	Description
`Radius`	Real	`0.5`	Specifies the radius of the sphere. It must be strictly positive.
`Slices`	Integer	`8`	Specifies the number of subdivisions around the z-axis when discretizing the sphere. It must be strictly positive.
`Stacks`	Integer	`8`	Specifies the number of subdivisions along the z-axis when discretizing the sphere. It must be strictly positive.

Example

```
# A Sphere with a low level of discretization around the z-axis
Sphere sphere {
    Radius 6
    Slices 4
    Stacks 32
}
```

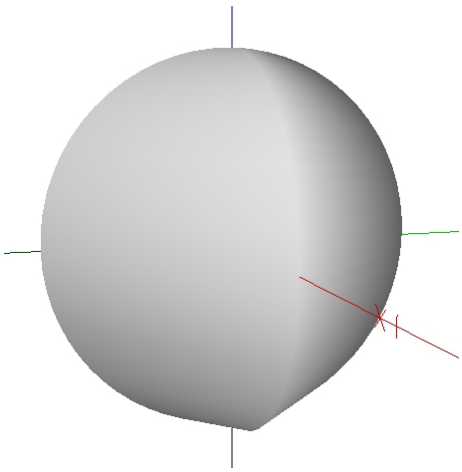


Figure 3.31: The `Sphere` object.

3.35 Swung

Class

Geometry - Primitive - Volume - SOR

Description

The **Swung** describes a surface of revolution passing by several 2D profiles along Z axis.

Fields description

Name	Type	Defaults	Description
`ProfileList`	Curve2D []	None	Sections of the surface of revolution.
`AngleList`	Real []	None	The rotation angles used to locate each section.
`Degree`	Integer	`3`	Specifies the degree of profile interpolation. It must be in [1,3]. If degree is equal to 1, the surface will be flat between each positioned profile. Otherwise, the surface will be smooth.
`Slices`	Integer	`8`	Specifies the number of subdivisions when discretizing the swung. It must be strictly positive.
`Strides`	Integer	`8`	Specifies the number of subdivisions of each profile. It must be strictly positive.
`CCW`	Boolean	`True`	Indicates whether each polygon's points are listed in counterclockwise (`True`) or clockwise (`False`) order when viewed from the front.

Example

```
# A crown represented by a swung
Polyline2D Tronc { PointList [<0,0>,<1,0>,<1,32>,<0,32>] }
BezierCurve2D f1 { CtrlPointList [ <0,30>, <20,20>, <20, 30>, <0,40> ] }
BezierCurve2D f2 { CtrlPointList [ <0,30>, <40,50>, <40, 60>, <0,40> ] }
Swung t1 {
    ProfileList [ Tronc ]
    AngleList [ 0 ]
    Slices 10
}
Swung t2 {
    ProfileList [ f1, f2, f1 ]
    AngleList [ 0, 180, 360 ]
    Slices 30
    Degree 3
    Stride 10
}
```

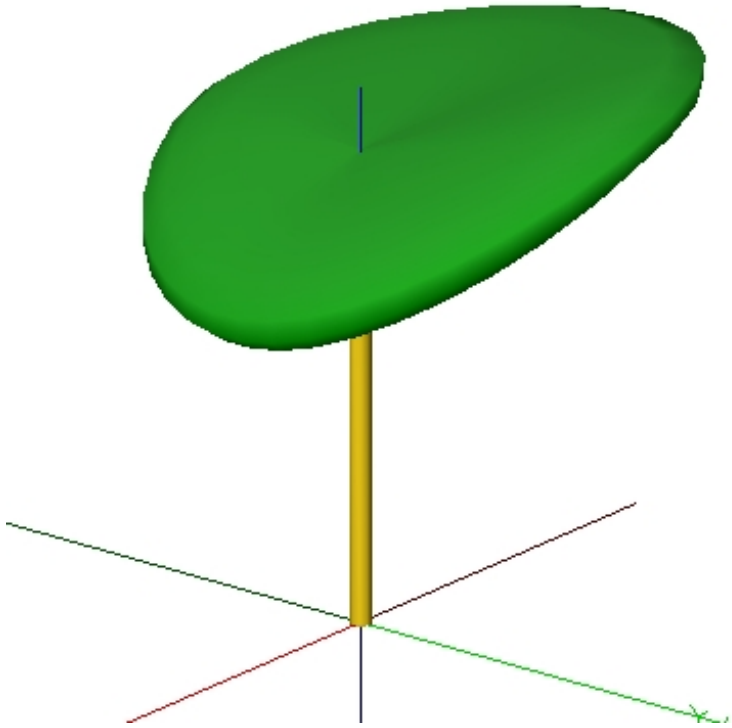


Figure 3.32: The `Swung` object.

3.36 Tapered

Class

Geometry - Transformed

Description

The **Tapered** describes an object to which it has been applied a Taper deformation. A Taper deforms an object in order to be able to bound the object within a cone frustum of a specified base radius and top radius.

For each point composing an object, a Taper scale the polar coordinates according the z-coordinate. The amplitude of the scale is given by the radii.

Fields description

Name	Type	Defaults	Description
`BaseRadius`	Real	`1`	Specifies the base radius of the cone frustum.
`TopRadius`	Real	`1`	Specifies the top radius of the cone frustum.
`Primitive`	Geometry	None	Specifies the GEOM object which is affected by the taper. It must be a GEOM object of type of Primitive.

Example

```
# A pyramid represented by a Tapered Box
```

```
Tapered tapered {
    BaseRadius 2.0
    TopRadius 0.5
    Primitive Box {
        Size <4,4,4>
    }
}
```

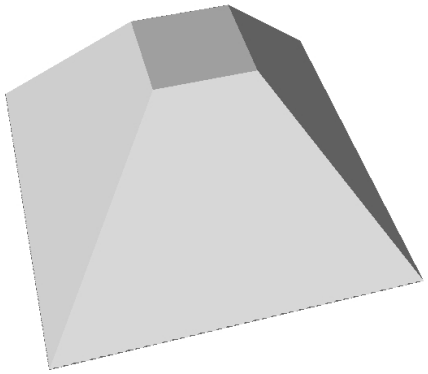


Figure 3.33: The `Tapered` object.

3.37 Translated

Class

Geometry - Transformed

Description

A **Translated** describes an object to which it has been applied a translation of a specified vector. The translation is given by the homogeneous matrix:

$$M = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where (t_x, t_y, t_z) denotes the translation vector.

Fields description

Name	Type	Defaults	Description
`Translation`	Vector3	`<0,0,0>`	Specifies translation vector.
`Geometry`	Geometry	None	Specifies the GEOM object which is affected by the translation.

Example

```
# A Translated Cylinder along the y-axis
Cylinder cylinder {
    Height 5
    Radius 2
    Slices 16
}
Translated translated {
    Translation <0,6,0>
    Geometry cylinder
}
```

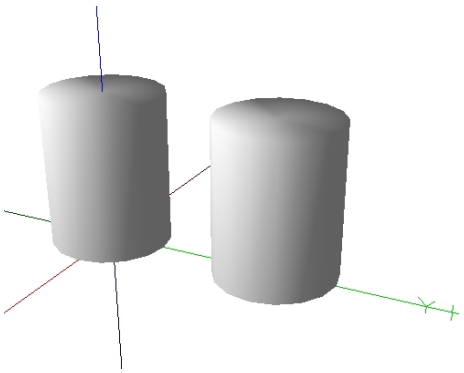


Figure 3.34: The `Translated` object.

3.38 TriangleSet

Class

Geometry - Primitive - Surface/Volume - Mesh

Description

A **TriangleSet** describes a surface formed by triangles, three sided polygons. Triangles are specified using indices into a list of vertices located at the specified coordinates.

Fields description

Name	Type	Defaults	Description
`PointList`	Vector3[ ]	None	Specifies the coordinates of the constructing points.
`IndexList`	Integer[ ][ ]	None	Specifies the faces, each specified as a series of indices into the list of points. Be sure that the indices are not out of range and each series must contain exactly three elements.
`CCW`	Boolean	`True`	Indicates whether each triangle's points are listed in counterclockwise (`True`) or clockwise (`False`) order when viewed from the front.
`Solid`	Boolean	`False`	Specifies whether this mesh represent a closed surface.
`Skeleton`	Geometry	`Null`	Specifies the skeleton of this mesh. It must be a GEOM object of type of Polyline.

Example

```
# An icosahedron represented by a TriangleSet
TriangleSet triangle_set {
    PointList [ <-5,0,8.5>, <5,0,8.5>,<-5,0,-8.5>, <5,0,-8.5>,
                <0,8.5,5>, <0,8.5,-5>, <0,-8.5,5>, <0,-8.5,-5>,
                <8.5,5,0>, <-8.5,5,0>, <8.5,-5,0>, <-8.5,-5,0>
    ]
    IndexList [ [0,1,4], [0,4,9], [9,4,5], [4,8,5], [4,1,8],
                [8,1,10], [8,10,3], [5,8,3], [5,3,2], [2,3,7],
                [7,3,10], [7,10,6], [7,6,11], [11,6,0], [0,6,1],
                [6,10,1], [9,11,0], [9,2,11], [9,5,2], [7,11,2]
    ]
    CCW False
    Solid True
}
```

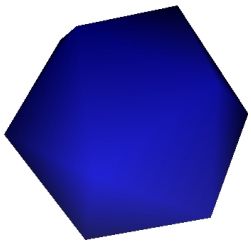


Figure 3.35: The `Triangleset` object.