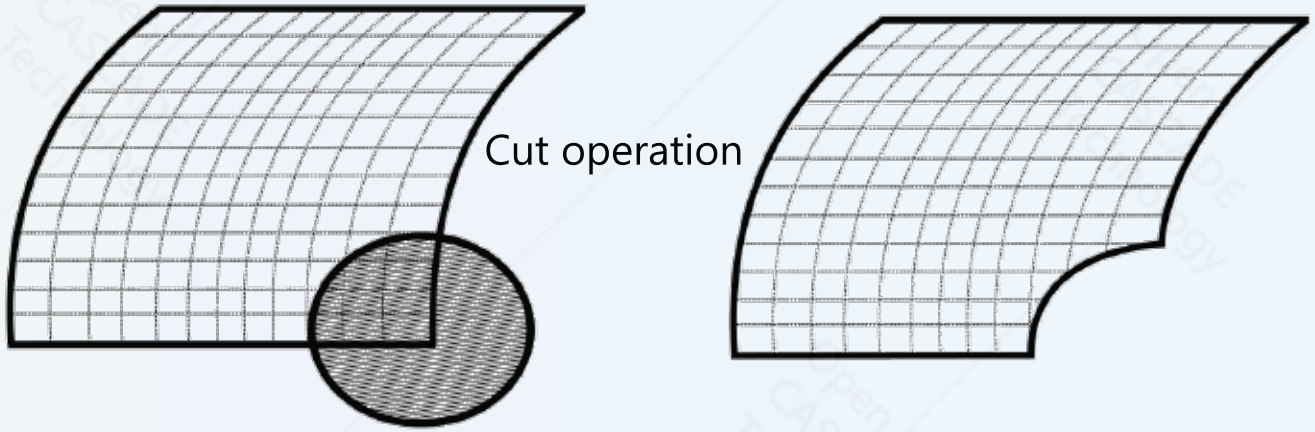
Open CASCADE TECHNOLOGY

Training: Topology


TRAINING
Open CASCADE TECHNOLOGY
Topology: Introduction

Geometry limitations

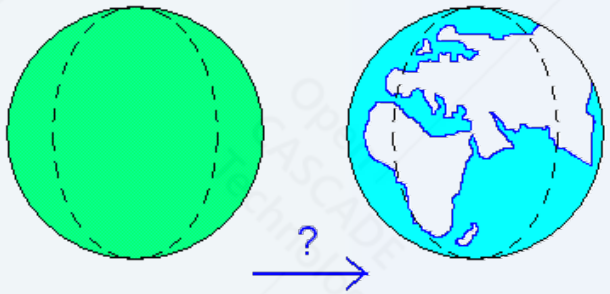
OCCT surfaces support rectangular trimming. A non-rectangular domain may arise after the Boolean operation.



How to store the result of the cut operation?

Purpose of topology

In general, Topology is a means to describe the limitation of an object.



Open CASCADE Topology is used to describe:

- Boundaries of objects.
- Connectivity between objects (via common boundaries).

Open CASCADE topological entities are called shapes.

Definition of topology

Topological shapes are defined following these two concepts:

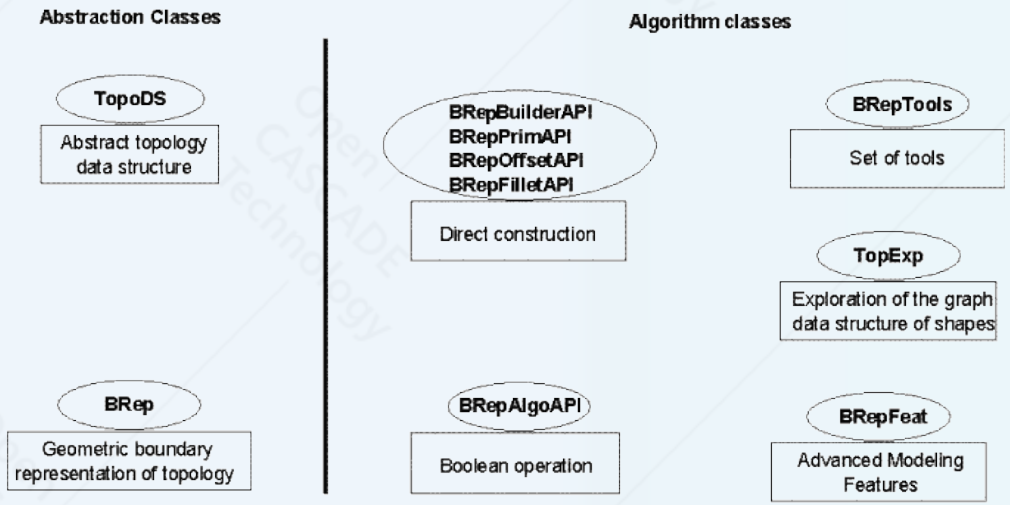
- Abstract Topology (TopoDS): defines the data structure by describing the relation between bounded and bounding objects.

Example: an edge is described by its boundaries which are vertices.

- Boundary representation (B-Rep): completes the definition of an object by associating topological and geometric information.

Example: an edge lies on a curve and is bounded by points.

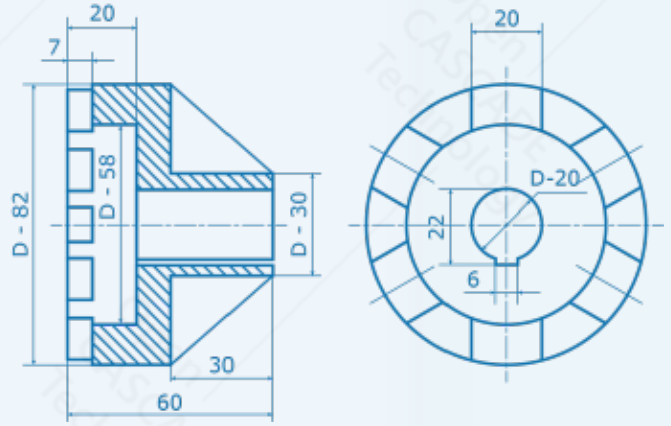
Abstract, boundary representation and algorithm classes are grouped in different packages.

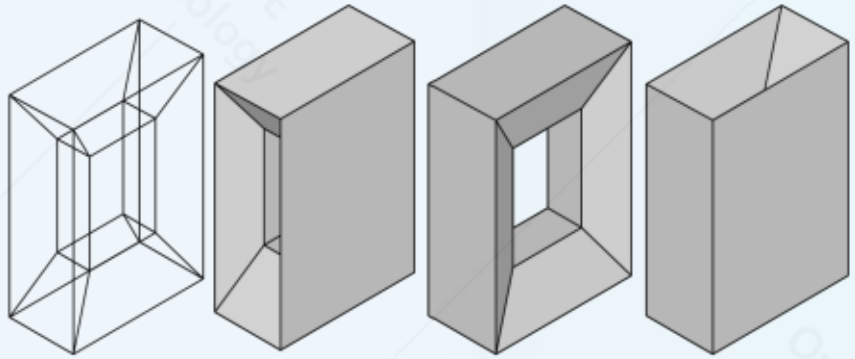


Why boundary representation?

There are several alternatives to B-Rep:

- Constructive solid geometry (CSG). CSG does not allow to model arbitrary figures.
- Drawing. Drawings are not suitable for downstream engineering operations.
- Wireframe. Several solids may correspond to a single wireframe model.
- Mesh. Meshes do not support curved geometry.





B-Rep has some drawbacks. It is glassy, verbose, and complex. B-Rep model uses vertices, edges, and faces as geometry carriers but also has special topological types. Model complexity causes fragility.

TRAINING

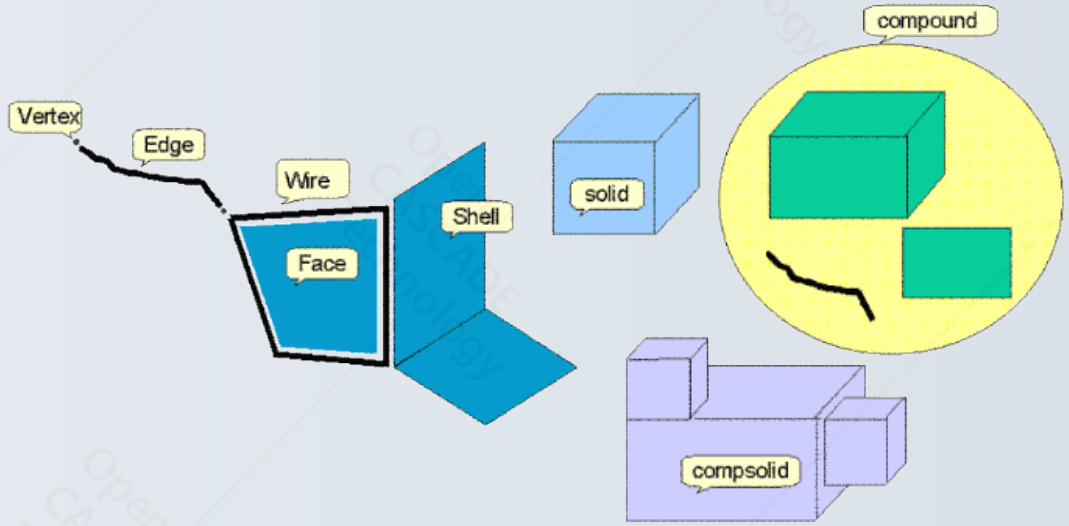
Open CASCADE TECHNOLOGY

Topology: Abstract topology

Topological shapes

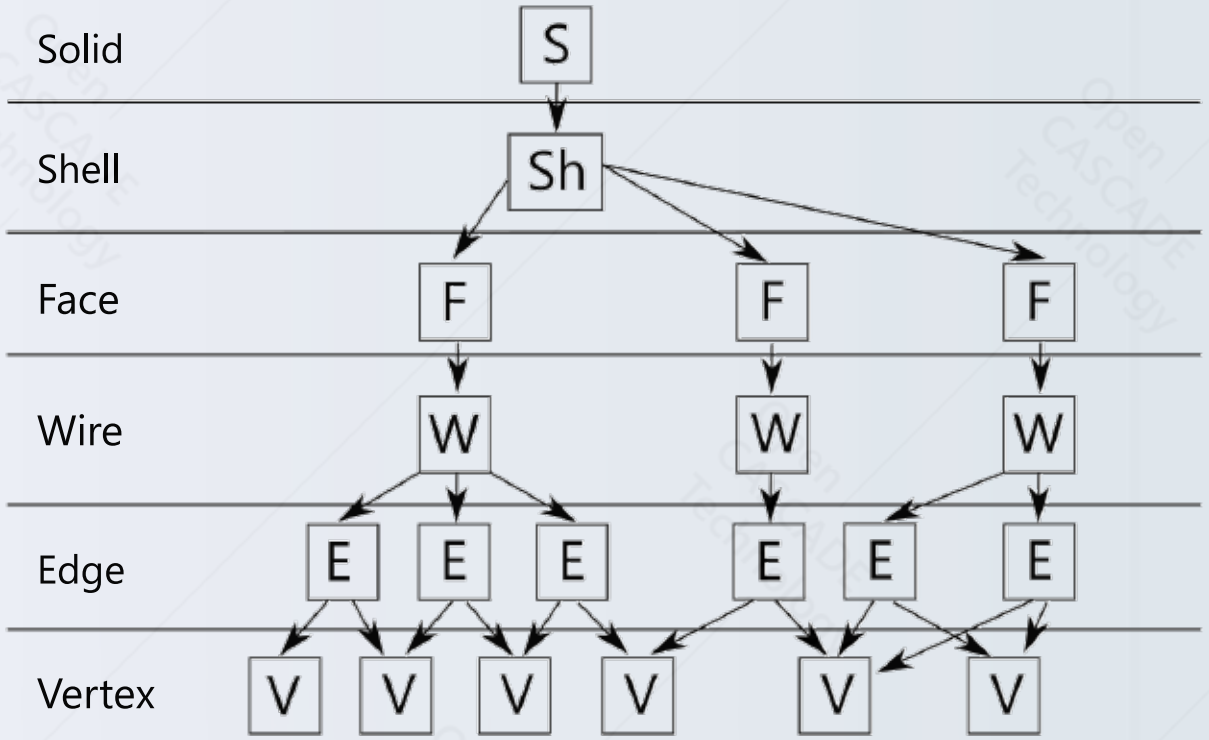
Open CASCADE Technology defines the following types of topological shapes:

- Vertex: a point.
- Edge: a part of a curve limited by vertices.
- Wire: a set of edges (connected by their vertices).
- Face: a part of a surface limited by wires.
- Shell: a set of faces (connected by their edges).
- Solid: a part of space limited by shells.
- Compsolid: a set of solids connected by their faces.
- Compound: a group of any topological shapes.



Graph structure

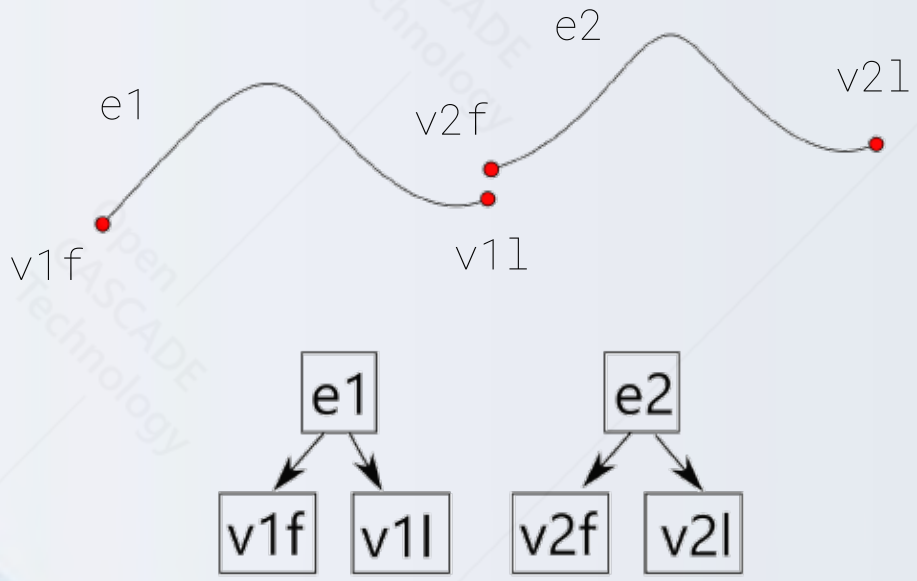
The following graph shows an example of relations between sub-shapes of a complex shape (a solid in this case):

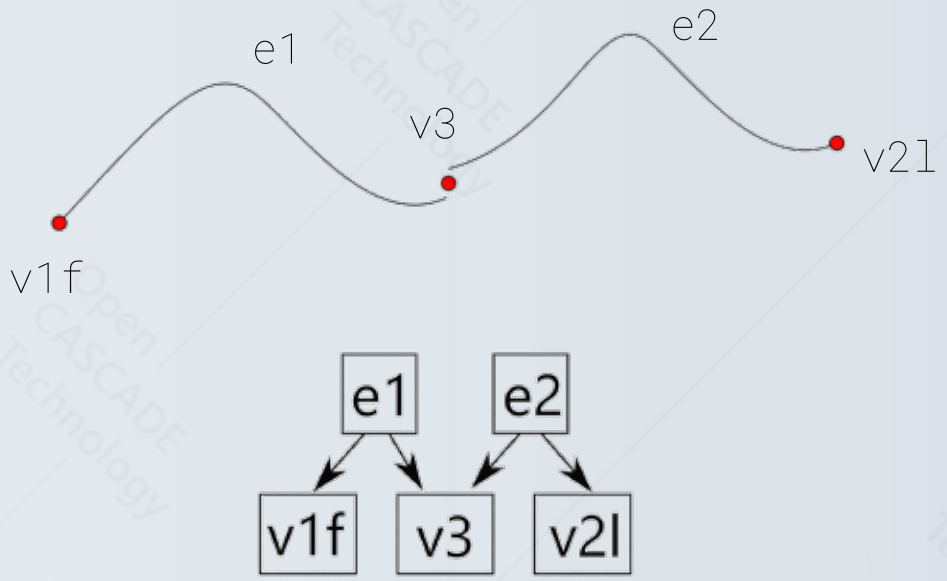


Connectivity of shapes

Two shapes are connected if they share some bounding sub-shape(s).

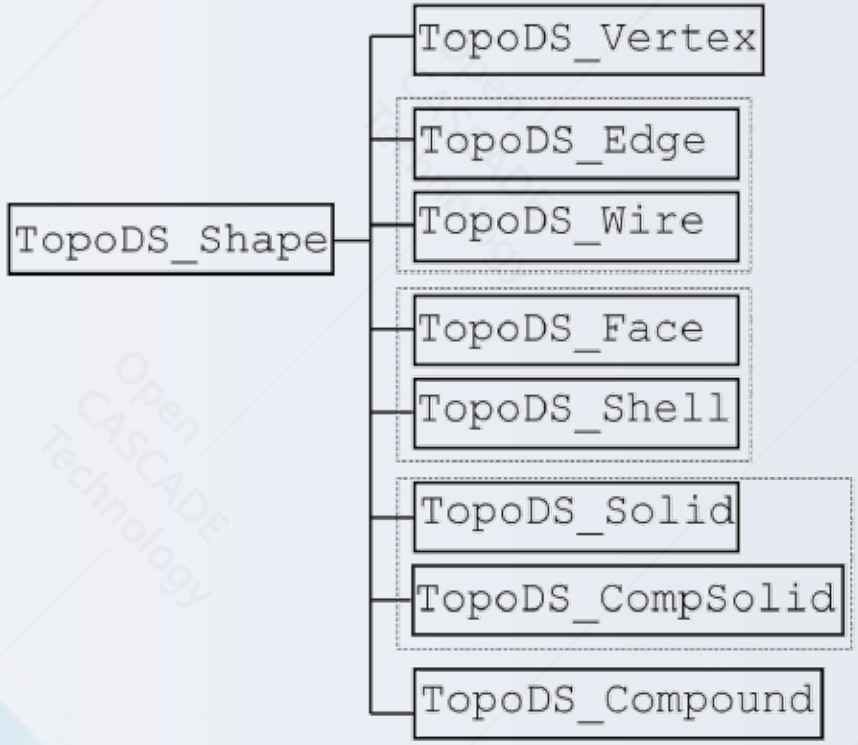
Example: Let's consider two edges – `e1` and `e2`. Each of them is limited by its boundaries, which are vertices (`v1f` and `v1l` for `e1` and `v2f` and `v2l` for `e2`). When these two edges share a common vertex `v3`, they are connected.





Hierarchy of shapes

`TopoDS_Shape` is the root class for all classes of topological shapes.



`TopoDS_Vertex` keeps information about point (zero-dimensional object).

`TopoDS_Edge` keeps information about curves (one-dimensional object). `TopoDS_Wire` is a collection of edges.

`TopoDS_Face` keeps information about surfaces (two-dimensional object). `TopoDS_Shell` is a collection of faces.

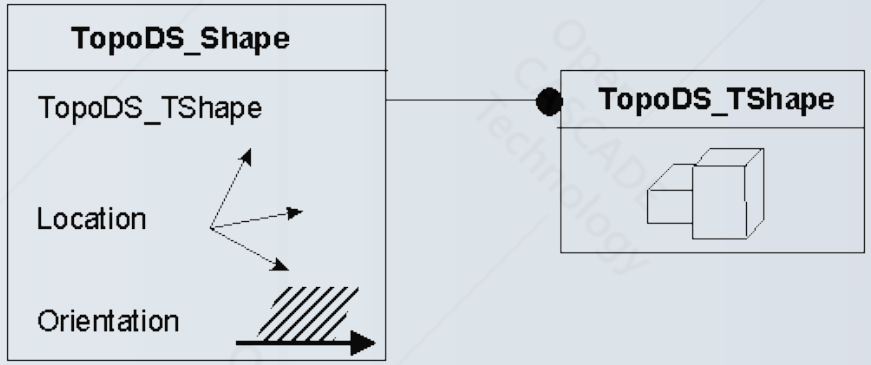
`TopoDS_Solid`, `TopoDS_Compsolid` keeps information about solids.

`TopoDS_Compound` represents a shape which is a collection of shapes.

Structure of shape

The TopoDS_Shape class defines a shape by:

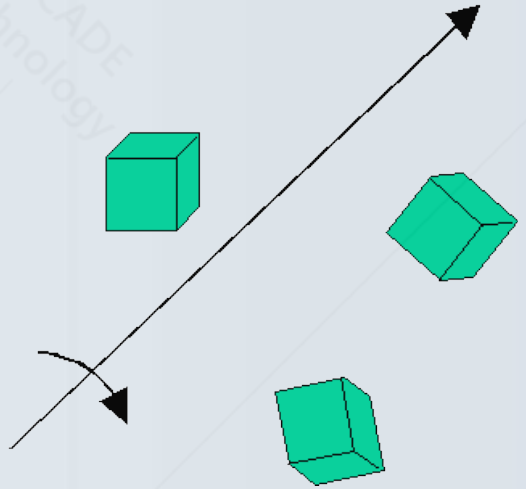
- A TopoDS_TShape handle (TopoDS package).
- A local coordinate system (TopLoc package).
- An orientation (TopAbs package).



TopoDS_TShape: a handle class which describes the object in its default coordinate system. This class is never used directly, TopoDS_Shape is used.

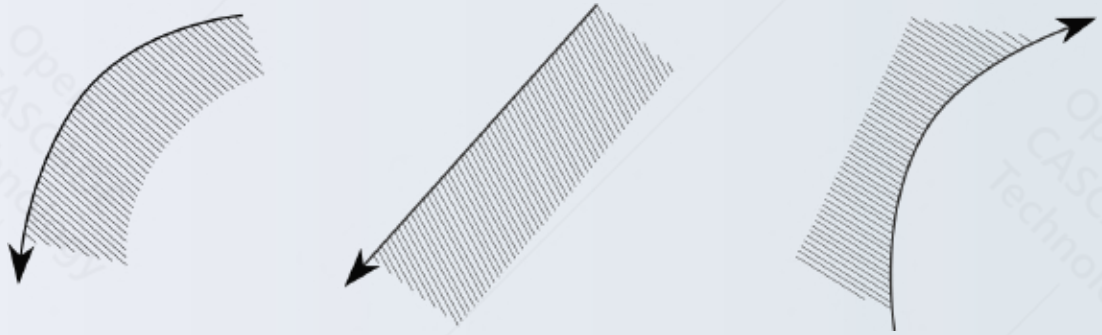
TopLoc_Location: defines a local coordinate system which places a shape at a different position from that of its definition.

Example: all these boxes share the same TShape but have different locations.



Structure of shape

`TopAbs_Orientation`: describes how a shape delimits a geometry in terms of material (or inner and outer regions).

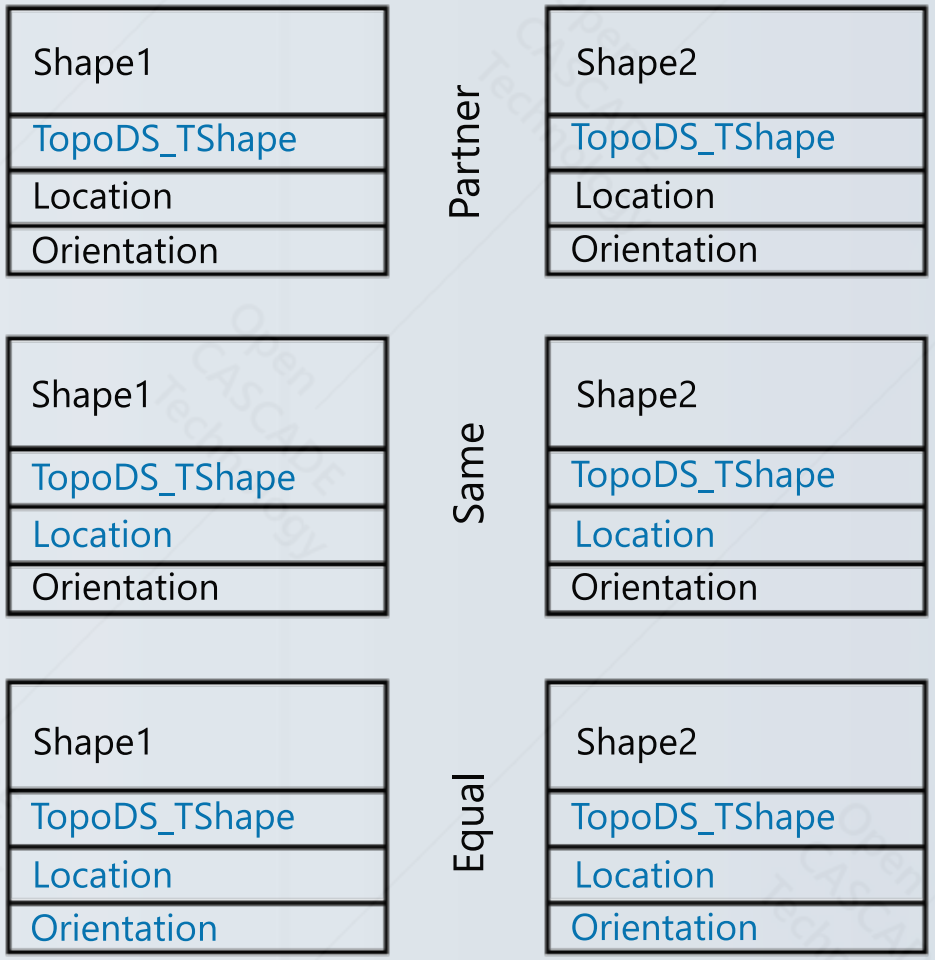


The orientation and location parameters of the shape are also assumed to affect its sub-shapes when considered in the context of that shape. When a shape is explored to sub-shapes, orientation, and location of the sub-shape is combined with that of the main shape. This ensures consistent interpretation of parameters of the sub-shape in the context of each shape that refers to it. For example, the edge shared by two connected faces will have opposite orientations when explored in the context of those faces.

Shape manipulations

The TopoDS_Shape class and it descendants provide various useful methods such as:

- Access to TShape
  - IsNull() – checks whether TShape is null or not.
  - Nullify() – nullifies TShape smart pointer.
- Access to location
  - Location() – returns existing location.
  - Move() – applies transformation to actual shape.
  - Moved() – returns new shape with applied transformation.
- ShapeType() – returns the type of the TopoDS_Shape
- Shapes comparison:
  - IsPartner() – the same TShape
  - IsSame() – the same TShape and location.
  - IsEqual() – the same TShape, location and orientation.



Shape downcasting

`TopoDS_Shape` objects are manipulated by value. That is why special methods are implemented to supply downcast functionality:

- `TopoDS::Vertex()` Returns a `TopoDS_Vertex`
- `TopoDS::Edge()` Returns a `TopoDS_Edge`
- `TopoDS::Wire()` Returns a `TopoDS_Wire`
- `TopoDS::Face()` Returns a `TopoDS_Face`
- `TopoDS::Shell()` Returns a `TopoDS_Shell`
- `TopoDS::Solid()` Returns a `TopoDS_Solid`
- `TopoDS::CompSolid()` Returns a `TopoDS_CompSolid`
- `TopoDS::Compound()` Returns a `TopoDS_Compound`

Note: exception is raised when an inappropriate conversion is done.

Example: the first block is correct, but the second is rejected by the compiler.

```
// Correct.
if (aShape.ShapeType() == TopAbs_VERTEX)
{
  const TopoDS_Vertex& aV1 =
                    TopoDS::Vertex(aShape);
}
```

```
// Rejected by compiler.
if (aShape.ShapeType() == TopAbs_VERTEX)
{
  TopoDS_Vertex aV2 = aShape;
}
```


Collections of shapes

The `TopTools` package provides:

- Classes to compute hash code for a shape with or without orientation.
- Instantiation of collections for shapes.

```
BRep_Builder BB;
TopTools_MapOfShape anEmap = anItl.Value();
TopTools_ListIteratorOfListOfShape anItl(anEdges);
for (;anItl.More();anItl.Next())
  BB.Remove(aWire, anItl.Value());

for (anItl.Initialize(edges); anItl.More();anItl.Next())
{
  TopoDS_Shape anEdge = anItl.Value();
  if (anEmap.Contains(anEdge))
    anEdge.Reverse();
  BB.Add(aWire, anEdge);
}
```


Exploration tools

Exploring a topological shape means finding its sub-shapes, possibly matching specific criterion.

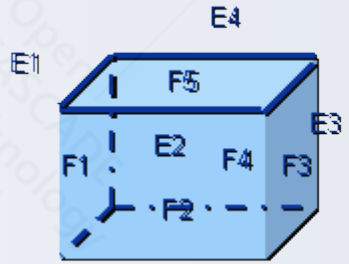
- `TopoDS_Iterator` class explores the first level sub-shapes of the given shape (from the list in its `TShape`).
- `TopExp_Explorer` class explores all sub-shapes in the given shape, with a possibility to select the kind of entities (for example, faces only).

```
TopExp_Explorer anExp(aShape, TopAbs_EDGE);
for (; anExp.More(); anExp.Next())
{
  TopoDS_Edge anEdge = TopoDS::Edge(anExp.Current());
}
```

- `TopExp::MapShapes()` method explores sub-shapes and puts them in a map (thus detecting the same elements).

Exploration tools

- `TopExp::MapShapesAndAncestors()` method returns all the entities that reference another one.



E1		E2		E3		E4	
F1	F5	F2	F5	F3	F5	F4	F5

In Open CASCADE Technology, there are no back pointers from a sub-shape to its ancestor shapes. Instead, `TopExp::MapShapesAndAncestors()` may be used to restore this information. For example, if you want to find all faces that contain a given vertex or an edge, you may use this method.

TRAINING

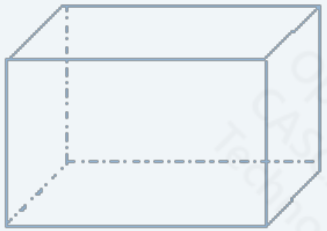
Open CASCADE TECHNOLOGY

Topology: Boundary representation

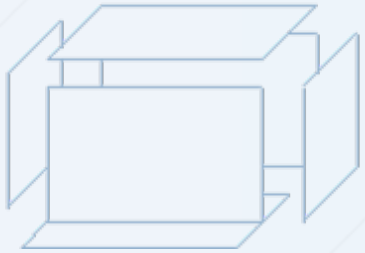
Boundary representation

The Boundary Representation (B-Rep) describes the model objects in three dimensions.

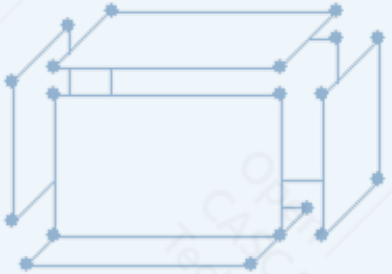
In B-Rep modeling, entities are represented by their boundaries.



Model in 3D



Faces bound model



Edges bound faces

B-Rep immerses Geometry into Topology:

- Geometry: a face lies on a surface, an edge lies on a curve, and a vertex lies on a point.
- Topology: connectivity of shapes.

Thus the description of a model object becomes complete.

B-Rep description is based on:

- `TopoDS` package - to describe the topological structure of objects.
- `Geom` and `Geom2d` packages - to describe the geometry of these objects.

B-Rep entities

`BRep_TVertex`, `BRep_TEdge`, and `BRep_TFace` are defined to add geometric information to a topological model.

`BRep_TVertex`, `BRep_TEdge`, and `BRep_TFace` inherit `TopoDS_TShape`.

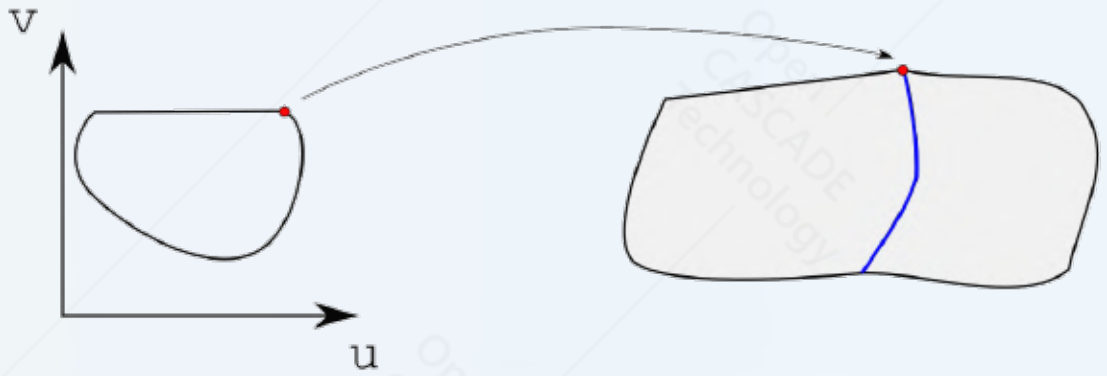
The geometric information is stored in a different way according to the topological entity. Entities that store geometric information allows to describe:

- An edge: a curve limited by vertices.
- A face: a surface limited by edges.
- A solid: space limited by faces.

Geometry in BRep_TVertex

`BRep_TVertex` geometry is stored as:

- A 3D point (`gp_Pnt`) - for all vertices
- A list of point representations that can be:
  - A point on a curve (`Geom_Curve`, parameter) - if a vertex bounds an edge.
  - A point on a curve on a surface (`Geom_Surface`, `Geom2d_Curve`, parameter) - if a vertex bounds an edge lying on a surface.
  - A point on a surface (`Geom_Surface`, Uparameter, Vparameter) - if a vertex bounds a face.

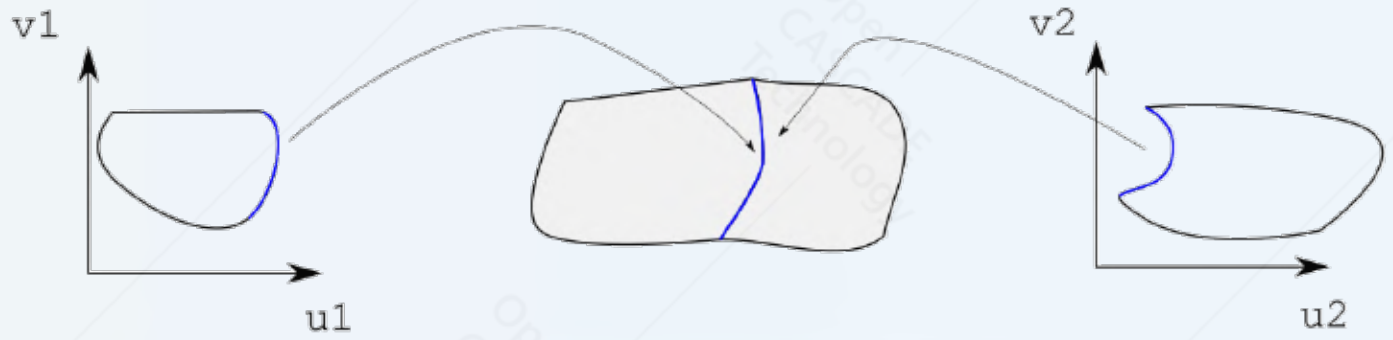


Geometry in BRep_TEdge

`BRep_TEdge` geometry is stored as a list of curve representations that can be :

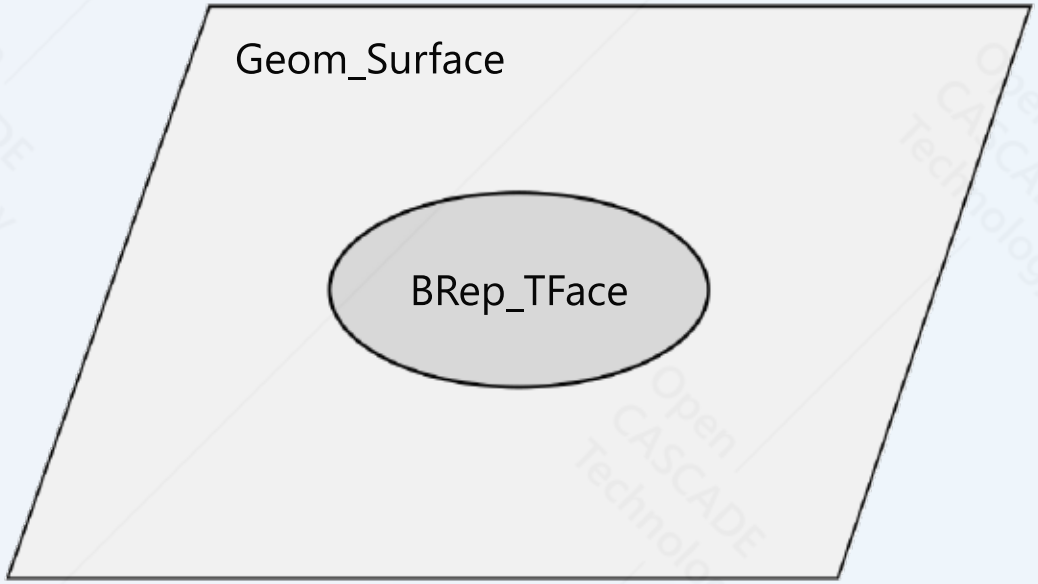
- A 3D curve and two parameters on a curve (`Geom_Curve`, `FirstParameter`, `LastParameter`).
- A curve on a surface, two parameters on a curve and two pairs of parameters on a surface (`Geom2d_Curve`, `FirstParameter`, `LastParameter`, `Geom_Surface`, `FirstUVCoord`, `LastUVCoord`).

The SameParameter property indicates whether different representations of the edge are parametrized synchronously i.e., points with the same parameter value on the 3D curve and each of 2D curves coincide (within the edge's tolerance)



Geometry in BRep_TFace

`BRep_TFace` geometry is stored as a `Geom_Surface`



Precision in B-Rep

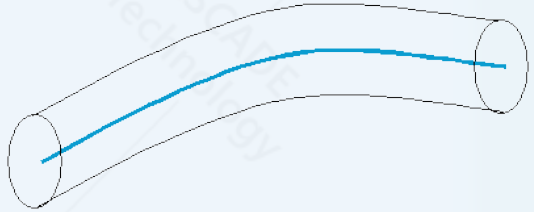
Several geometric representations may be attached to a topological (B-Rep) object. For example, a vertex can be represented by:

- 3D point;
- parameter on a curve;
- pair of parameters on a surface.

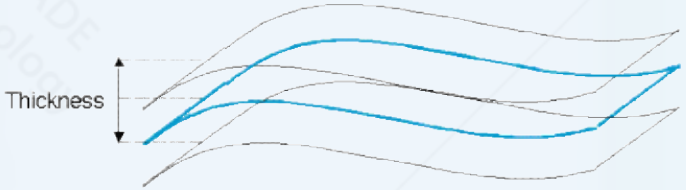
These representations are similar but rarely identical. For modeling algorithms, it is necessary to know exactly the precision associated with this approximation. The numeric value of this precision is associated with each B-Rep shape and is called `tolerance`. It defines the zone in which all geometrical representations of the object are located.

In BRep_TVertex precision defines the `radius of a sphere` around a 3D point:

In BRep_TEdge precision defines the `radius of a pipe` around a 3D curve



In BRep_TFace precision defines the `thickness` above and below a surface

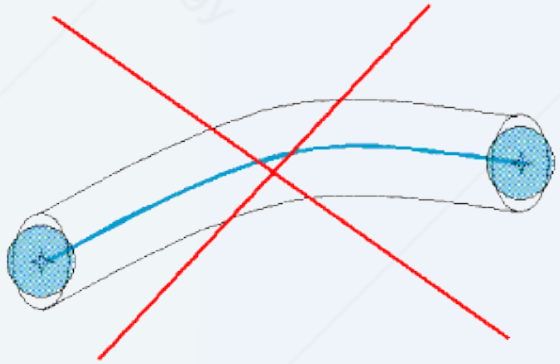


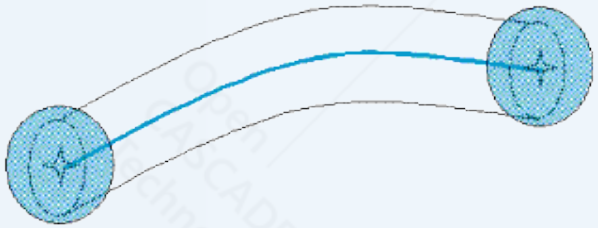
Precision in B-Rep

Since tolerance is associated with geometry carriers, it is defined by the algorithms creating or modifying the geometry in B-Rep.

Open CASCADE Technology requires that:

Tolerance(Vertex) >= Tolerance(Edge) >= Tolerance(Face)





Package: BRepAdaptor

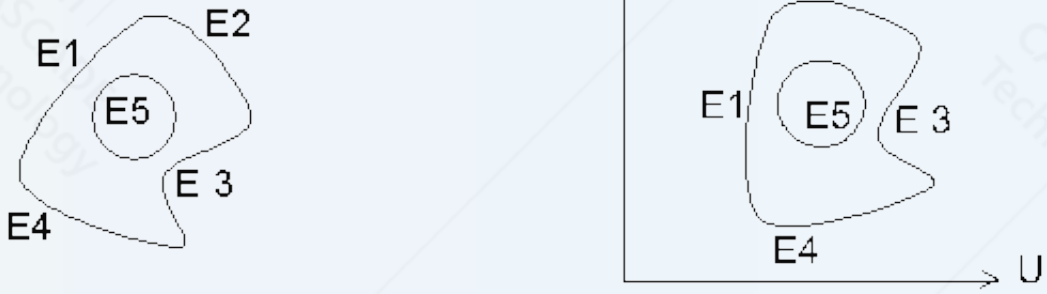
OCCT enables topological entities usage in geometric algorithms via the adapter pattern. Adapters for boundary representation work as trimmed curves and surfaces, thus eliminating the necessity of manual trimming. The following adapters are available:

- `BRepAdapter_Curve` – curve adapter accepting edge.
- `BRepAdapter_Curve2d` – curve adapter accepting edge and face.
- `BRepAdapter_CompCurve` – curve adapter accepting wire.
- `BRepAdaptor_Surface` – surface adapter accepting face.

Note: Handle version of adapters are available. For instance, `BRepAdaptor_HSurface` class is handled-version of the surface adapter.

B-Rep particularities

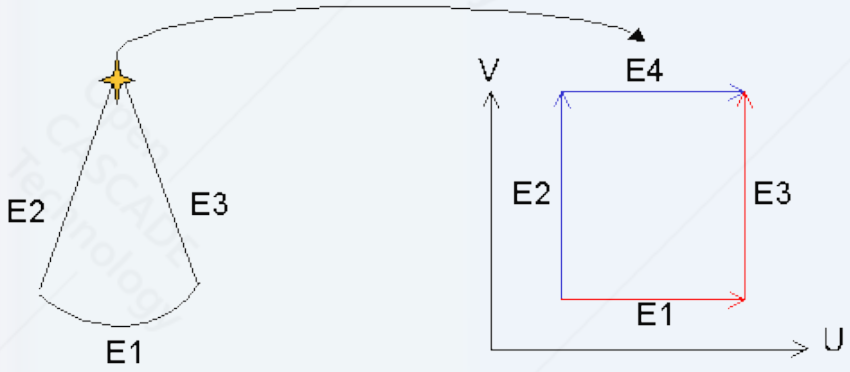
Representation of the same face in 3D space (3D topology) and in parametric space (UV topology) are usually topologically similar.

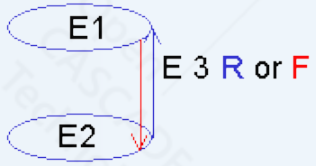


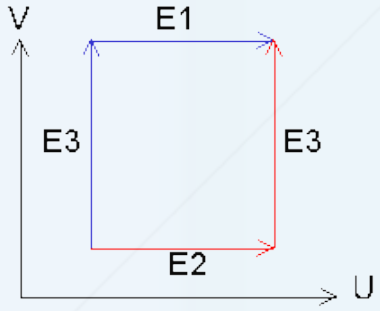
Sometimes 3D topology and UV topology are different. It is the case of seam edges and degenerated edges.

B-Rep particularities

- A seam edge is an edge, which defines a seam on a closed face (usually built on a periodic surface). In this case, one 3D topology corresponds to several UV topologies.
- An edge is said to be degenerated when one or several UV 2d curves correspond to a single 3D vertex. Such edge does not have a 3d curve representation and includes the same vertex twice (with opposite orientations).







Shapes construction

Open CASCADE Technology supplies several packages that provide a high-level API to modeling algorithms. This API is both simple and powerful:

- simple because only one call to a function is needed to create an object

```
gp_Pnt aP1(10.0, 0.0, 0.0), aP2(20.0, 0.0, 0.0);
TopoDS_Egde E = BRepBuilderAPI_MakeEdge(aP1, aP2);
```

- powerful because it includes error handling and access to extra information provided by the algorithms

```
gp_Pnt aP1(10.0, 0.0, 0.0), aP2(20.0, 0.0, 0.0);
BRepBuilderAPI_MakeEdge aME(aP1, aP2);

// Construction state.
if (!aME.IsDone())
  std::cout << "Error. MakeEdge failed:" << aME.Error() << std::endl;

// Edge extraction from algorithm.
TopoDS_Edge anEdge = aME.Edge();
```


Package: BRepBuilderAPI

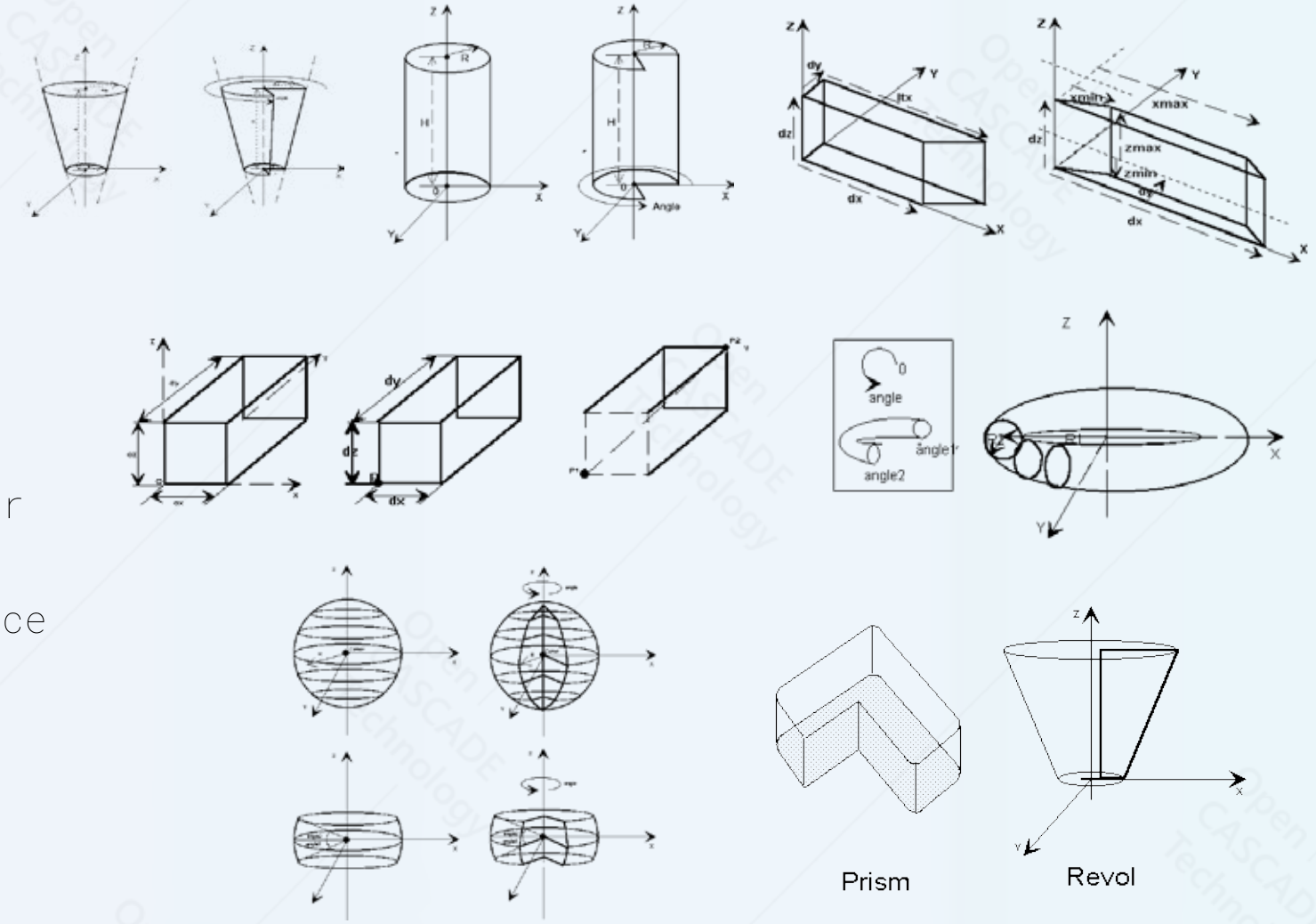
This package contains classes for:

- Direct construction of the topological objects from geometric entities or combine topological items to collection-like shapes.
  - `BRepBuilderAPI_MakeVertex`: builds a vertex from points.
  - `BRepBuilderAPI_MakeEdge`, `MakeEdge2d`: build edges from curves.
  - `BRepBuilderAPI_MakePolygon`: builds a wire from points.
  - `BRepBuilderAPI_MakeWire`: builds a wire from edges.
  - `BRepBuilderAPI_MakeFace`: builds a face from a surface.
  - `BRepBuilderAPI_MakeShell`: builds a shell from a surface (splits to C2 patches).
  - `BRepBuilderAPI_MakeSolid`: builds a solid from shells.
- Modifying objects:
  - `BRepBuilderAPI_Transform`: applies transformation to a shape.
  - `BRepBuilderAPI_Copy`: makes deep copy of a shape.
  - `BRepBuilderAPI_Sewing`: builds a shell from a set of faces by merging boundary edges.

Package: BRepPrimAPI

This package contains classes for:

- Creating primitive objects:
  - BRepPrimAPI_MakeBox
  - BRepPrimAPI_MakeWedge
  - BRepPrimAPI_MakeSphere
  - BRepPrimAPI_MakeCone
  - BRepPrimAPI_MakeCylinder
  - BRepPrimAPI_MakeTorus
  - BRepPrimAPI_MakeHalfSpace
- Creating sweeps:
  - BRepPrimAPI_MakePrism
  - BrepPrimAPI_MakeRevol



Location modification

Data sharing concept, available in OCCT, allows re-use topological information and instance model several times by means of the location. Location within a shape is a set of consecutive transformations which are manipulated as a single transformation.

Note: empty location and existing identical locations are considered as different. As a result, `IsSame()` check will return `false`.

Location methods are as follows:

- `Location`
- `Move`
- `Moved`

```
// Construct location.
gp_Ax1 axis(gp_Pnt(aX, aY, aZ),
            gp_Vec(aDX, aDY, aDZ));

gp_Trsf T;
T.SetRotation(axis, aR);

// Update existing location.
aShape.Move(T);
```


Orientation: vertex within an edge

OCCT orientation concept aims at the completion of the boundary representation by information about inner and outer regions. This information is a set of rules affecting shape correctness.

The orientation itself has no meaning for a vertex. Vertex orientation makes sense only when vertex bounds some edge. Edge is constructed using a pair of vertices (at least one vertex using twice in case of the periodic curve); the first vertex has `TopAbs_FORWARD` orientation, and the second one has `TopAbs_REVERSED` orientation by a convention.

Note: `BRepBuilderAPI_MakeVertex` constructs a vertex with `TopAbs_FORWARD` orientation.

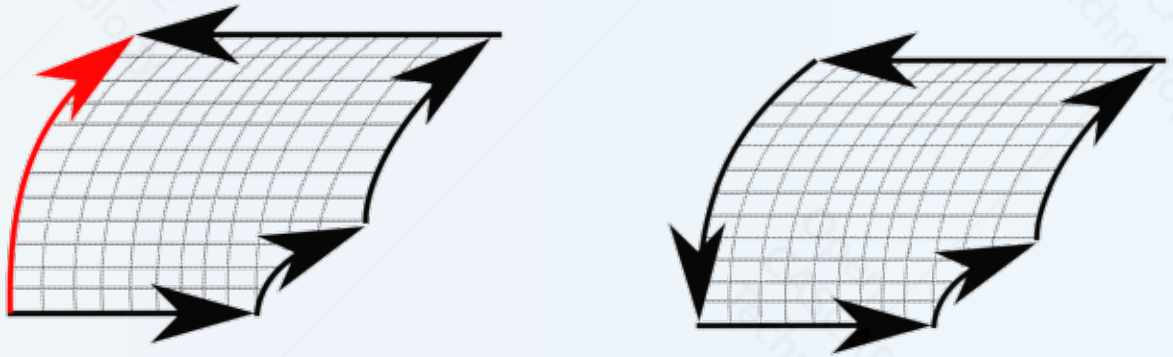
```
gp_Pnt aP1(10.0, 0.0, 0.0), aP2(20.0, 0.0, 0.0);
TopoDS_Edge anEdge = BRepBuilderAPI_MakeEdge(aP1, aP2);
TopExp_Explorer anExpEV(anEdge, TopAbs_VERTEX);
for(; anExpEV.More(); anExpEV.Next())
{
  const TopoDS_Vertex& aV =
TopoDS::Vertex(anExpEV.Current());
  const gp_Pnt& aPnt = BRep_Tool::Pnt(aV);
  if (aV.Orientation() == TopAbs_FORWARD)
  {
    std::cout << "Forward vertex is: "
              << aPnt.X() << " "
              << aPnt.Y() << " "
              << aPnt.Z() << std::endl;
  }
  else if (aV.Orientation() == TopAbs_REVERSED)
  {
    // There is inner orientation. That is why
    // else-if expression is needed.
    std::cout << "Reversed vertex is: "
              << aPnt.X() << " "
              << aPnt.Y() << " "
              << aPnt.Z() << std::endl;
  }
}
```


Orientation: edge within a wire

The face is a part surface bounded by edges. Edges are organized into wires to be able to track each loop (outer or inner) individually.

There are no limitations when a wire is free; edges can be added to wire without constraints.

The right-hand rule comes in action when wire belongs to a face; each edge in a wire should be orientated to have material on a left side according to parameter increasing direction on a curve. What to do with the wrongly oriented curve?



It is possible to rebuild a curve, but it is preferable to revert the underlying curve virtually. The second option was chosen in the OCCT.

Orientation: face orientation in a solid

Solid is a part of modeling space bounded by faces organized into a shell. Normals in the solid should point outside the material by a convention. Differential geometry states that normal to surface in point can be evaluated using the following formula ("x" stands for cross product):

$$N(u_0, v_0) = s'_u(u_0, v_0) \times s'_v(u_0, v_0)$$

Normal is calculated up to a sign, so an alternative formula exists where partial derivatives are swapped (the OCCT uses the formula presented above). How to ensure the correct normal orientations in a solid?

The face orientation determines the sign before normal:

- `TopAbs_FORWARD` – normal is evaluated according to the formula above.
- `TopAbs_REVERSED` – normal is multiplied by -1.0.

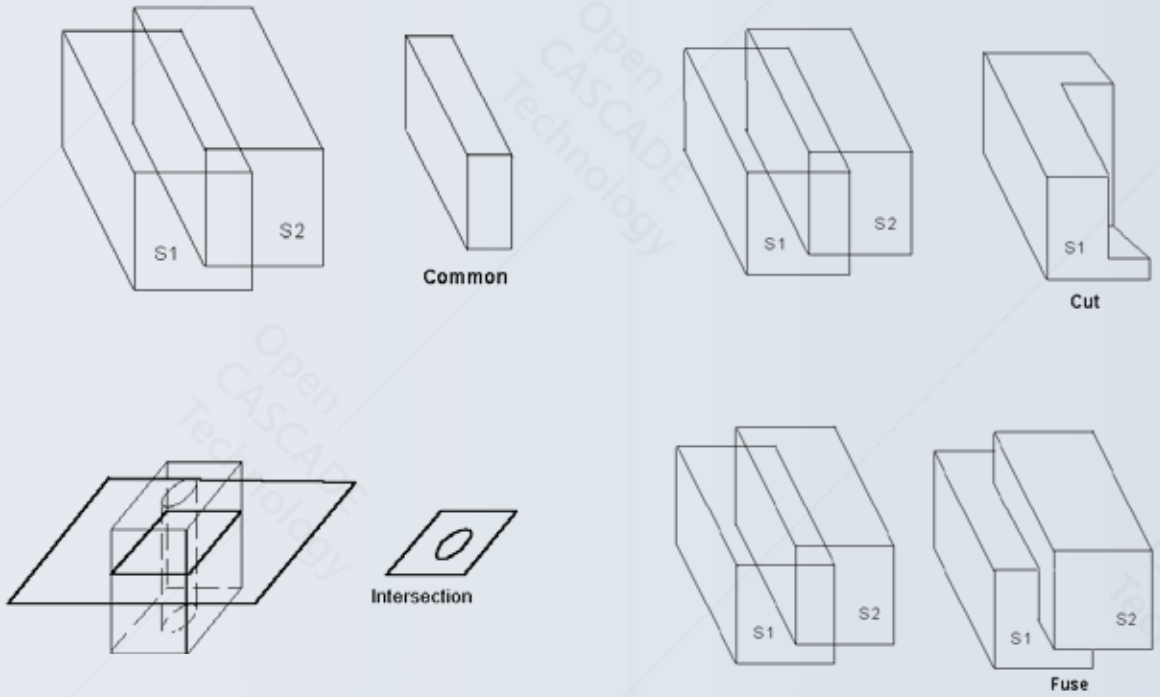
Note: solid may represent the whole modeling space except a part of space bounded by solid boundaries. Normals will point inside the bounded part of space in that case.


TRAINING
Open CASCADE TECHNOLOGY
Topology:
Modeling API

Modeling API: BRepAlgoAPI

The OCCT has two implementations of the Boolean algorithm. Internally, they are called “old” and “new” algorithm. The old algorithm is not maintained anymore and marked obsolete. The OCCT’s “new” Boolean Algorithm is available in the `BRepAlgoAPI` package. The following algorithms are available:

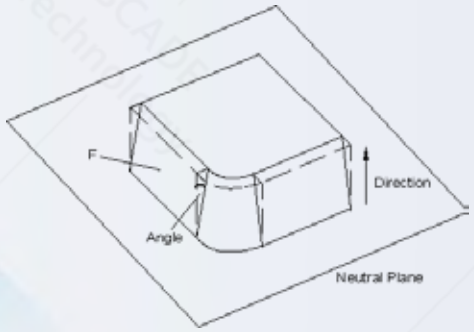
- Cut (`BRepAlgoAPI_Cut`).
- Fuse (`BRepAlgoAPI_Fuse`).
- Common (`BRepAlgoAPI_Common`).
- Section (`BRepAlgoAPI_Section`).
- Splitter (`BRepAlgoAPI_Splitter`).

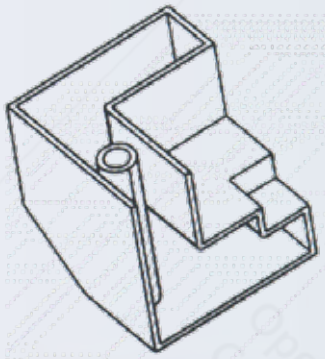


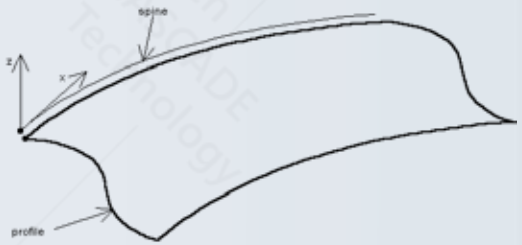
Modeling API: BRepOffsetAPI

This package provides additional tools for construction, such as:

- `BRepOffsetAPI_ThruSections` – builds a shell or a solid from a sequence of wire profiles.
- `BRepOffsetAPI_DraftAngle` – tapers a set of faces of a shape with a given angle.
- `BRepOffsetAPI_MakeOffsetShape` – builds offset shape on the given shape.
- `BRepOffsetAPI_MakeThickSolid` – builds a hollowed solid from a given solid and a set of faces to be removed.
- `BRepOffsetAPI_MakePipe` – builds a pipe shape by sweeping a base shape (profile) along a wire (spine).
- `BRepOffsetAPI_MakeEvolved` – builds an evolved shape from a planar face or wire (spine) and a wire (profile).



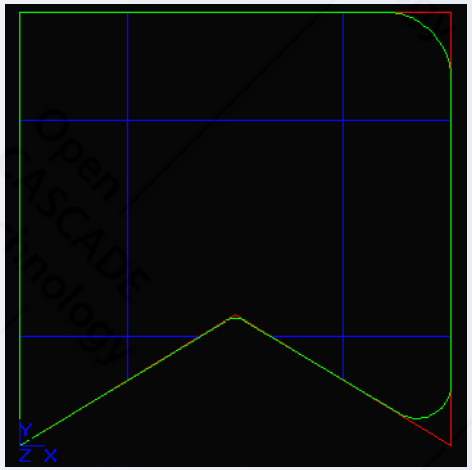


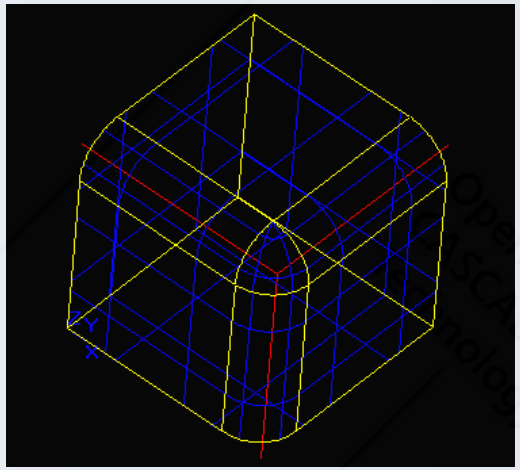


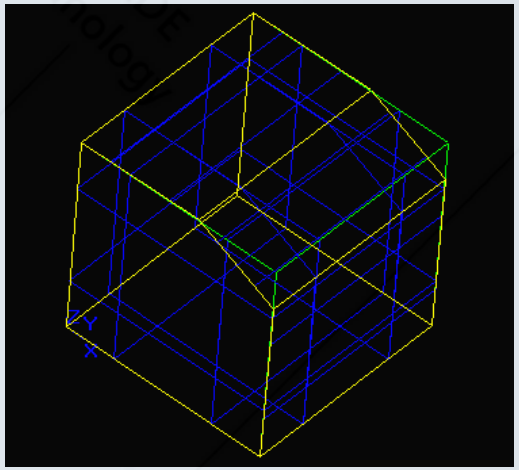
Modeling API: BRepFilletAPI

This package provides classes for making fillets and chamfers:

- `BRepFilletAPI_MakeFillet2d`
- `BRepFilletAPI_MakeFillet`
- `BRepFilletAPI_MakeChamfer`



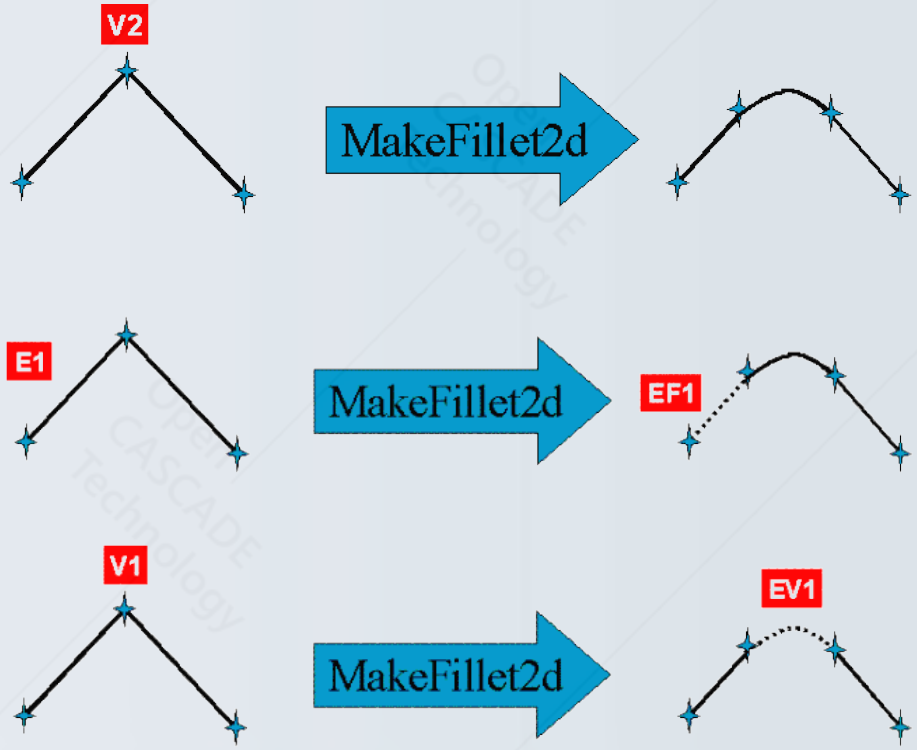




Modeling API: history of modifications

Sometimes, built-in operations are unable to solve a modeling problem. A custom modeling pipeline is used in that case. The OCCT does not have persistent indexing; topological items may change unpredictably after an operation. History support allows overcoming this issue. Modeling algorithms have three methods that allow to know the modification status of an initial shape 'S' after a topological operation:

- `IsDeleted(S)` tells if the shape 'S' has been deleted by the algorithm.
- `Modified(S)` lists shapes that represent the modified state of the shape 'S' (such shapes have the same underlying geometry with 'S').
- `Generated(S)` lists shapes generated by the algorithm from the shape 'S' (such shapes have new underlying geometry not existing in 'S')



About Open Cascade

It is a software development company which is laser-focused on digital transformation of industries through the use of 3D technologies.

Open Cascade offers a wide range of high-performance proprietary 3D software tools both open-source and commercial. The first ones have been developed, maintained and continuously improved since 2000. Whereas the second ones have been progressively aggregated in the Commercial Platform based on which the company offers creating modern tailor-made industrial solutions that meet even the most sophisticated client's requirements.

Moreover, Open Cascade expands its portfolio by offering end-user industrial software products and delivering software customization and integration services. Open Cascade provides its solutions and services worldwide. The company is a part of the Capgemini's Digital Engineering and Manufacturing Services global business line.

Learn more about Open Cascade at www.opencascade.com

OPEN CASCADE

Backing your path to digital future