General Fuse Algorithm,
Partition Algorithm,
Boolean Operations Algorithm

Backgrounds
# Table of Contents

1. Introduction .................................................................................................................. 4
2. Overview ......................................................................................................................... 4
3. Analysis of Operators .................................................................................................... 5
4. Terms and Definitions .................................................................................................... 5
   4.1. Tolerances ................................................................................................................. 5
   4.2. Interferences ............................................................................................................. 5
      4.2.1. Vertex/Vertex interference ................................................................................. 5
      4.2.2. Vertex/Edge interference .................................................................................... 5
      4.2.3. Vertex/Face interference ..................................................................................... 5
      4.2.4. Edge/Edge interference ....................................................................................... 5
      4.2.5. Edge/Face interference ....................................................................................... 5
      4.2.6. Face/Face interference ....................................................................................... 5
      4.2.7. Computation Order ............................................................................................ 5
      4.2.8. Results ............................................................................................................... 5
   4.3. PAVES ....................................................................................................................... 5
   4.4. PAVE Blocks ............................................................................................................ 5
   4.5. Shrunken Range ....................................................................................................... 5
   4.6. Common Blocks ....................................................................................................... 5
   4.7. Connectivity Chains ................................................................................................. 5
   4.8. Split Edges ............................................................................................................... 5
   4.9. Section Edges .......................................................................................................... 5
5. Main Parts of Algorithms ............................................................................................... 5
   5.1. Data Structure ......................................................................................................... 5
   5.2. Intersection Part ...................................................................................................... 5
   5.3. Building Part ........................................................................................................... 5
6. General Fuse Algorithm ................................................................................................. 5
   6.1. Arguments ............................................................................................................... 5
   6.2. Results ..................................................................................................................... 5
      6.2.1. Example 1 ........................................................................................................... 5
      6.2.2. Example 2 ........................................................................................................... 5
      6.2.3. Example 3 ........................................................................................................... 5
      6.2.4. Example 4 ........................................................................................................... 5
      6.2.5. Example 5 ........................................................................................................... 5
      6.2.6. Example 6 ........................................................................................................... 5
      6.2.7. Example 7 ........................................................................................................... 5
      6.2.8. Example 8 ........................................................................................................... 5
      6.2.9. Example 9 ........................................................................................................... 5
      6.2.10. Example 10 ...................................................................................................... 5
7. Partition Algorithm ......................................................................................................... 5
   7.1. Arguments ............................................................................................................... 5
   7.2. Results ..................................................................................................................... 5
      7.2.1. Example 1 ........................................................................................................... 5
      7.2.2. Example 2 ........................................................................................................... 5
      7.2.3. Example 3 ........................................................................................................... 5
8. Boolean Operations Algorithm ...................................................................................... 5
   8.1. Arguments ............................................................................................................... 5
   8.2. Results ..................................................................................................................... 5
      8.2.1. Example 1 ........................................................................................................... 5
      8.2.2. Example 2 ........................................................................................................... 5
      8.2.3. Example 3 ........................................................................................................... 5
      8.2.4. Example 4 ........................................................................................................... 5
9. Limitations of Algorithms .............................................................................................. 5

Copyright © 2001-2010. All rights reserved.
Page 2 of 56
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1</td>
<td>ARGUMENTS</td>
</tr>
<tr>
<td>9.1.1</td>
<td>Common requirements</td>
</tr>
<tr>
<td>9.1.2</td>
<td>Pure self-interference</td>
</tr>
<tr>
<td>9.1.3</td>
<td>Self-interferences due to tolerances</td>
</tr>
<tr>
<td>9.1.4</td>
<td>Parametric representation</td>
</tr>
<tr>
<td>9.1.5</td>
<td>Stop-gaps</td>
</tr>
<tr>
<td>9.2</td>
<td>INTERSECTION PROBLEMS</td>
</tr>
<tr>
<td>9.2.1</td>
<td>Pure intersections and common zones</td>
</tr>
<tr>
<td>9.2.2</td>
<td>Tolerances</td>
</tr>
</tbody>
</table>
1. **INTRODUCTION**

This document describes the backgrounds of the following algorithms: General Fuse Algorithm, Partition Algorithm, Boolean Operation Algorithm (hereafter Algorithms).

2. **OVERVIEW**

For the moment there are two algorithms that solve similar tasks:

- **Boolean Operations Algorithm in Open CASCADE 6x (BOA).**
  
  BOA has been designed and developed in 2000. Since 2000 the algorithm is modified to fix the problems upon requests (in general the modifications were bug fixes).

- **Partition Algorithm for SALOME platform (PA).**

  PA has been designed and developed in 2005 and uses modern (at that time) algorithms of OCC as auxiliary tools, for e.g. unbalanced binary tree of overlapped bounding boxes.

  PA has the following features:
  
  - PA is based on General Fuse Algorithm (GF).
  - PA was developed taking into account the problems faced in BOA between 2000-2005 years.
  - The architecture of PA is history-based and has been designed to support modern trends in topology science.
  - The architecture of PA is expandable, that allows to create a lot of specific algorithms upon requests.
3. **Analysis of Operators**

- The General Fuse operator can be applied to arbitrary number of arguments (in terms of TopoDS_Shape).

  The operator can be represented as the following:

  \[ R_{GF} = GF(S_1, S_2, ..., S_n) \]  
  \[(3.1.1)\]

  where

  \[ R_{GF} \] – result of the operation,
  \[ S_1, S_2, ..., S_n \] - Arguments of the operation,
  \[ n \] - Number of arguments \((n>1)\)

- The Partition operator can be applied to arbitrary number of arguments (in terms of TopoDS_Shape).

  The operator can be represented as the following:

  \[ R_{PA} = PA(S_1, S_2, ..., S_{nOB}, [T_1, T_2, ..., T_{nT}]) \]  
  \[(3.1.2)\]

  where

  \[ R_{PA} \] – result of the operation,
  \[ S_1, S_2, ..., S_{nOB} \] - Arguments of the operation (Objects),
  \[ nOB \] - number of Objects \((nOB>1)\)
  \[ T_1, T_2, ..., T_{nT} \] - Arguments of the operation (Tools),
  \[ nT \] - number of Tools

  It is evident that for \(nT=0\)

  \[ R_{PA} = PA(S_1, S_2, ..., S_{nOB}) = R_{GF} \]  
  \[(3.1.3)\]

  Thus, PA is particular case of GF.
BOA provides the operations (Common, Fuse, Cut) between two arguments (in terms of TopoDS_Shape).

The operator can be represented as the following:

\[ R_{BOA} = B_j (S_1, S_2) \]  

(3.1.4)

where

- \( R_{BOA} \) – result of the operation
- \( B_j \) – operation of type j (Common, Fuse, Cut),
- \( S_1, S_2 \) - Arguments of the operation.

The result \( R_{BOA} \) (3.1.4) can be obtained from \( R_{GF} \) (3.1.1).

For e.g. for the two arguments \( S_1, S_2 \) (see the Figure 1) the result \( R_{GF} \) will be

\[ R_{GF} = GF (S_1, S_2) = S_{p1} + S_{p2} + S_{p12} \]  

(3.1.5)
On the other hand

\[
B_{\text{common}} (S_1, S_2) = S_{p12} \\
B_{\text{cut12}} (S_1, S_2) = S_{p1} \\
B_{\text{cut21}} (S_1, S_2) = S_{p2} \\
B_{\text{fuse}} (S_1, S_2) = S_{p1} + S_{p2} + S_{p12} = R_{GF}
\]

\[
R_{GF} = GF (S_1, S_2) = B_{\text{fuse}} = B_{\text{common}} + B_{\text{cut12}} + B_{\text{cut21}}
\]

(3.1.6)

(3.1.7)

The fact that the \( R_{GF} \) contains the components of \( R_B \) (3.1.7) allows to consider that GF is the general case of BOA. Thus it is possible to implement BOA, PA as subclasses of GF using C++ inheritance mechanism.
4. **TERMS AND DEFINITIONS**

The chapter contains the background, terms, definitions that are necessary to understand how the algorithms work.

4.1. **TOLERANCES**

Each shape that has boundary representation in the Open CASCADE contains its own internal value of geometrical tolerance.

The meaning of the tolerance is different for different shapes (Table 1).

<table>
<thead>
<tr>
<th>No</th>
<th>Picture</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1  | ![Vertex](image1) | Vertex  
Value of the tolerance for a vertex $V$ is the value of radii of a imaginary circumscribed sphere around the 3-D point of the vertex $V$ |
| 2  | ![Edge](image2)  | Edge  
Value of the tolerance for an edge $E$ is the value of radii of a imaginary circumscribed pipe around the 3-D curve of the edge $E$ |
| 3  | ![Face](image3)  | Face  
Value of the tolerance for a face $F$ is the value of thickness of a imaginary offset surfaces to underlying surface of the face $F$ |
Tolerance is a way of stating how much precision is needed or conversely, how much error can be accepted.

The tolerances in Open CASCADE are absolute values (in some units: mm, in, etc). Absolute tolerance defines the maximum permissible distance apart that two shapes (or sub-shapes) are permitted to be and still be considered to be “close enough”, i.e. that these two shapes will be capable of being joined.

There are the following rules for the values of tolerances:

- For any edge $E$ with vertices $V_i$:

  $$\text{Tol} \ (E) \leq \max \ (\text{Tol} \ (V_i)), \ i=1, \ 2\ldots \text{Nb}_V$$

  where $\text{Nb}_V$ – Number of vertices of the edge $E$.

- For any edge face $F$ with edges $E_i$:

  $$\text{Tol} \ (F) \leq \max \ (\text{Tol} \ (E_i)), \ i=1, \ 2\ldots \text{Nb}_E$$

  where $\text{Nb}_E$ - Number of edges of the face $F$.
4.2. INTERFERENCES

The shapes interfere between each other in terms of their tolerances.

Shapes are interfered when there is a part of 3D space where the distance between the underlying geometry of shapes is less or equal to the sum of tolerances of the shapes.

For the three types of shapes (vertex, edge, face) there are six types of interferences.

4.2.1. Vertex/Vertex interference

A vertex $V_i$ and a vertex $V_j$ have Vertex/Vertex interference when $V_i$ and $V_j$ have the distance between corresponding 3D points that is less than the sum of tolerances $Tol(V_i)$ and $Tol(V_j)$ of the vertices (Figure 2).

![Figure 2](image)

Result:

New vertex $V_n$ with 3D point $P_n$ and tolerance value $Tol(V_n)$ that can be computed by the formulas:

$$P_{new} = 0.5 \cdot (P_i + P_j)$$

$$Tol(V_n) = \max (Tol(V_i), Tol(V_j)) + 0.5 \cdot D$$

where

$$D = \text{distance}(P_i, P_j)$$
4.2.2. Vertex/Edge interference

A vertex $V_i$ and an edge $E_j$ have Vertex/Edge interference when for the vertex $V_i$ and the edge $E_j$ the distance $D$ between 3D point of the vertex and its projection on the 3D curve of the edge $E_j$ is less or equal than sum of tolerances of the vertex $\text{Tol}(V_i)$ and the edge $\text{Tol}(E_j)$ (Figure 3).

![Figure 3](image)

**Result:**
- Vertex $V_i$ with the corresponding tolerance value.

\[
\text{Tol}(V_i) = \max (\text{Tol}(V_i), \text{Tol}(E_j)) + 0.5 \cdot D
\]

where

\[
D = \text{distance} (P_i, PP_i)
\]

- Parameter $t_i$ of the projected point $PP_i$ on the 3D curve $C_j$ of the edge $E_j$. 


4.2.3. Vertex/Face interference

A vertex $V_i$ and a face $F_j$ have Vertex/Face interference when for the vertex $V_i$ and the face $F_j$ the distance $D$ between 3D point of the vertex and its projection on the surface of the face is less or equal than sum of tolerances of the vertex $\text{Tol}(V_i)$ and the face $\text{Tol}(F_j)$ (Figure 4).

![Figure 4](image)

Result:
- Vertex $V_i$ with the corresponding tolerance value.
- Parameters $u_i$, $v_i$ of the projected point $PP_i$ on the surface $S_j$ of the face $F_j$.

\[
\text{Tol}(V_i) = \max (\text{Tol}(V_i), \text{Tol}(E_j)) + 0.5 \cdot D
\]

where

\[
D = \text{distance} \ (P_i, PP_i)
\]
4.2.4. Edge/Edge interference

An edge \( E_i \) and an edge \( E_j \) have Edge/Edge interference when for two of edges \( E_i \) and \( E_j \) (with corresponding 3D curves \( C_i, C_j \)) there is (are) some place(s) where the distance between the curves is less than (or equals to) sum of tolerances of the edges.

Case 1: Two edges have common part(s) of 3D curves in terms of tolerance (Figure 5).

Result:
- Parametric range \([t_{i1}, t_{i2}]\) for the 3D curve \( C_i \) of the edge \( E_i \).
- Parametric range \([t_{j1}, t_{j2}]\) for the 3D curve \( C_j \) of the edge \( E_j \).
Case 2: Two edges have common point(s) in terms of tolerance (Figure 6).

![Diagram showing two edges and their tolerance](image)

**Figure 6**

**Result:**
- New vertex \( V_n \) with 3D point \( P_n \) and tolerance value \( \text{Tol}(V_n) \) that are calculated by the formulas:
  \[
P_n = 0.5 \cdot (P_i + P_j)
  \]
  \[
  \text{Tol}(V_n) = \max (\text{Tol}(E_i), \text{Tol}(E_j)) + 0.5 \cdot D
  \]
  where
  \[
  D = \text{distance } (P_i, P_j)
  \]
  \( P_i, P_j \) – the nearest 3D points for the curves \( C_i, C_j \)
- Parametr \( t_i \) of \( P_i \) for the 3D curve \( C_i \).
- Parametr \( t_j \) of \( P_j \) for the 3D curve \( C_j \).
4.2.5. Edge/Face interference

An edge $E_i$ and a face $F_j$ have Edge/Face interference when for the edge $E_i$ and the face $F_j$ (with corresponding 3D curve $C_i$, and surface $S_j$) there is (are) some place(s) in 3D space where the distance between the $C_i$ and the surface $S_j$ is less than (or equal to) sum of tolerances of the edge $E_i$ and the face $F_j$.

Case 1: Edge $E_i$ and Face $F_j$ have common part(s) in terms of tolerance (Figure 7).

Result:

- Parametric range $[t_{i1}, t_{i2}]$ for the 3D curve $C_i$ of the edge $E_i$.
Case 2: Edge $E_i$ and Face $F_j$ have common point(s) in terms of tolerance (Figure 8).

![Diagram showing common points and tolerance](image)

Result:

- New vertex (-ices) $V_n$ with 3D point $P_n$ and tolerance value $Tol(V_n)$ that are calculated by the formulas:

$$P_n = 0.5 \cdot (P_i + P_j)$$

$$Tol(V_n) = \max \{ Tol(E_i), Tol(F_j) \} + 0.5 \cdot D$$

where

$D=$distance $(P_i, P_j)$

$P_i, P_j$ – the nearest 3D points for the curve $C_i$ and surface $S_j$

- Parameter $t_i$ of $P_i$ for the 3D curve $C_i$.
- Parameters $u_i, v_i$ of the projected point $PP_i$ on the surface $S_j$ of the face $F_j$. 
4.2.6. Face/Face Interference

A face $F_i$ and a face $F_j$ have Face/Face interference when for the face $F_i$ and the face $F_j$ (with corresponding surfaces $S_i$, $S_j$) there are some place(s) in 3D space where the distance between the surfaces is less than (or equal to) sum of tolerances of the faces.

*Case 1:* Face $F_i$ and face $F_j$ have common curve(s) $C_{ijk}$ ($k=0, 1, 2...k_n$) in terms of tolerance (Figure 9).

![Figure 9](image.png)

**Result:**

- Intersection curves $C_{ijk}$ ($k=0, 1, 2...k_n$, $k_n$ - number of intersection curves) with correspondent values of tolerances $\text{Tol}(C_{ijk})$. 
Case 2: Face $F_i$ and face $F_j$ have common point(s) in terms of tolerance (Figure 10).

![Figure 10](image)

Result:

- New vertices $V_{ijm}$ ($m=0, 1, 2, m_n$, number of intersection points).
- New vertex (-ices) $V_{ijm}$ with 3D point $P_n$ and tolerance value $Tol(V_n)$ that are calculated by the formulas:

$$P_n = 0.5 \cdot (P_i + P_j)$$

$$Tol(V_{ijm}) = \max\{Tol(F_i), Tol(F_j)\} + 0.5 \cdot D$$

where

$$D=\text{distance } (P_i, P_j)$$

- Parameters $u_i, v_i$ of the projected point $PP_i$ on the surface $S_i$ of the face $F_i$
- Parameters $u_j, v_j$ of the projected point $PP_j$ on the surface $S_j$ of the face $F_j$

4.2.7. Computation Order

The interferences between shapes are computed on the basis of increasing of the dimension value of the shape in the following order: Vertex/Vertex, Vertex/Edge, Edge/Edge, Vertex/Face, Edge/Face, Face/Face. The reason is to avoid the computation of redundant interferences between upper shapes $S_i, S_j$ when there are interferences between lower sub-shapes $S_{ik}, S_{jm}$.

Copyright © 2001-2010. All rights reserved.
4.2.8. Results

- The result of the interference is a shape that can be either interfered shape itself (or its part) or a new shape.

- The result of the interference is a shape with the dimension value that is less or equal to the minimal dimension value of interfered shapes. For e.g. the result of Vertex/Edge interference is a vertex, but not an edge.

- The result of the interference splits source shapes on parts each time as it can do that.

4.3. PAVES

The result of interferences of the type Vertex/Edge, Edge/Edge, Edge/Face in a number of cases is a vertex (new or old) lying on the edge.

The result of interferences of the type Face/Face in a number of cases is intersection curves. These curves go through the vertices lying on the faces.

The position of the vertex \( V_i \) on the curve \( C \) is determined by a value of a parameter \( t_i \) of the 3D point of the vertex on the curve.

A Pave \( PV_i \) on a curve \( C \) is a structure containing the vertex \( V_i \) and correspondent value of the parameter \( t_i \) of the 3D point of the vertex on the curve. The curve \( C \) can be 3D or 2D curve. (Figure 11)

![Diagram of PAVES](image)

**Figure 11**

Two paves \( PV_1, PV_2 \) on the one curve \( C \) can be compared using the parameter value. \( PV_1 > PV_2 \) if \( t_1 > t_2 \).

The usage of paves allows making the difference between the one vertex \( V \) shared between arbitrary number of edges \( E_1, E_2, ... E_i \).

The usage of paves allows to bind the vertex to the curve (or any structure that contains a curve: edge, intersection curve).
4.4. Pave Blocks

A set of paves \( PV_i \) (\(i=1, 2...nPV, nPV\)-number of paves) of the curve \( C \) can be sorted in increasing order using the value of parameter \( t \) on the curve \( C \).

A pave block \( PB_i \) is a part of the object (edge, intersection curve) between neighboring paves. (Figure 12)

![Figure 12](image)

Any finite source edge \( E \) has the one bounding pave block that contains two paves \( PV_b, PV_e \). The pave \( PV_b \) corresponds to the vertex \( V_b \) with minimal parameter \( t_b \) on the curve of the edge. The pave \( PV_e \) corresponds to the vertex \( V_e \) with maximal parameter \( t_e \) on the curve of the edge.
4.5. **SHRUNK RANGE**

A pave block \( PV \) (Figure 13) of a curve \( C \) is bounded by vertices \( V_1, V_2 \) with the values of tolerances \( \text{Tol}(V_1), \text{Tol}(V_2) \).

The curve \( C \) has a value of tolerance \( \text{Tol}(C) \). In case of edge the value of tolerance is tolerance of the edge. In case of intersection curve the value of tolerance is obtained from an intersection algorithm.

![Figure 13](image)

The theoretical parametric range of the pave block is \([t_{1C}, t_{2C}]\).

The positions of the vertices \( V_1, V_2 \) of the pave block can be different and are determined by the following conditions:

\[
\text{Distance } (P_1, P_{1c}) \leq \text{Tol}(V_1)+\text{Tol}(C) \\
\text{Distance } (P_2, P_{2c}) \leq \text{Tol}(V_2)+\text{Tol}(C)
\]

The Figure 13 shows that each tolerance sphere of a vertex can reduce the parametric range of the pave block to \([t_{1S}, t_{2S}]\). The range \([t_{1S}, t_{2S}]\) is the *shrunk range of the pave block*. At limiting state \( t_{1S}=t_{1C}, t_{2S}=t_{2C} \).

The shrunk range of the pave block is the part of 3D curve that can interfere with other shapes.
4.6. **COMMON BLOCKS**

The interferences of the type Edge/Edge, Edge/Face can produce results as common parts. In case of Edge/Edge interference the common parts are pave blocks having different edges as a base. (Figure 14)

If the pave blocks $PV_1, PV_2... PV_{NbPV}$ ($NbPV$ – number of the pave blocks) have the same bounding vertices and geometrically coincide, the pave blocks form common block $CB$.

In case of Edge/Face interference the common parts are pave block lying on a face(s) (Figure 15).

If the pave blocks $PV_i$ geometrically coincide with a face $F_j$, the pave block form common block $CB$.

In general case a common block $CB$ contains:
- Pave blocks $PV_i$ ($i=2, 3... NbPV$)
- A set of faces $F_j$ ($j=0,1..NbF$, $NbF$ – number of faces).
4.7. CONNEXITY CHAINS

The interfering shapes can produce connexity chains.

The term connexity chains can be explained by the following examples

- *Interfering vertices.* (Figure 16)

![Figure 16](image)

The pairs of interfered vertices are: \( (nV_{11}, nV_{12}), (nV_{11}, nV_{13}), (nV_{12}, nV_{13}), (nV_{13}, nV_{15}), (nV_{13}, nV_{14}), (nV_{14}, nV_{15}), (nV_{21}, nV_{22}), (nV_{21}, nV_{23}), (nV_{22}, nV_{23}) \)

The pairs produce the two connexity chains: \( (nV_{11}, nV_{12}, nV_{13}, nV_{14}, nV_{15}), (nV_{21}, nV_{22}, nV_{23}) \).

Each connexity chain is used to create new vertex: \( VN_1, VN_2 \).

- *Interfering edges* (Figure 17)

![Figure 17](image)

The pairs of coincided pave blocks are: \( (PB_{11}, PB_{12}), (PB_{11}, PB_{13}), (PB_{12}, PB_{13}), (PB_{21}, PB_{22}), (PB_{21}, PB_{23}), (PB_{22}, PB_{23}) \).
The pairs produce the two connexity chains: (PB₁₁, PB₁₂, PB₁₃), (PB₂₁, PB₂₂, PB₂₃).

- **Interfering faces** (Figure 18)

![Diagram showing interfering faces](image)

The pairs produce the three connexity chains: (F₁₁, F₂₁), (F₂₂, F₃₁), (F₄₁, F₅₁, F₆)

The pairs of same domain faces are: (F₁₁, F₂₁), (F₂₂, F₃₁), (F₄₁, F₅₁), (F₄₁, F₆), (F₅₁, F₆)

The pairs produce the three connexity chains: (F₁₁, F₂₁), (F₂₂, F₃₁), (F₄₁, F₅₁, F₆)

A connexity chain contains shapes that are *Same Domain* Shapes (same domain vertices, same domain edges, same domain faces).

### 4.8. **SPLIT EDGES**

The set of pave blocks PB₁, PB₂... PBₙPB (nPB – number of pave blocks) for the edge E is used to build split edges $S_{pi}$ (i=1, 2...nPB).

### 4.9. **SECTION EDGES**

The set of pave blocks PB₁, PB₂... PBₙPB (nPB – number of pave blocks) for the intersection curve C is used to build section edges $S_{si}$ (i=1, 2...nPB).
5. **Main Parts of Algorithms**

The Algorithms (GF, PA, BOA) consist of three parts:

5.1. **Data Structure**

Data Structure (DS) is used to:

- Store information for data and intermediate results
- Provide the access to the information
- Provide the links between chunks of information

The contents of the information are:

- Arguments
- New shapes
- Interferences
- Pave Blocks
- Common Blocks

5.2. **Intersection Part**

Intersection Part (IP) is used to:

- Initialize DS
- Compute interferences between the arguments (and their sub-shapes)
- Compute same domain vertices, edges
- Build split edges
- Build section edges
- Build p-Curves
- Store all obtained information in DS

IP uses DS as input data.

5.3. **Building Part**

Building Part (BP) is used to:

- Build sub-shapes of the result of the operation
- Build the result of the operation
• Provide history information (in terms of `BRepBuilderAPI_MakeShape::Generated()`, `::Modified()`, `::IsDeleted()`)

BP uses DS as input data.

6. **GENERAL FUSE ALGORITHM**

6.1. **ARGUMENTS**

• The arguments are shapes (in terms of `TopoDS_Shape`).

• Number of arguments is non-limited

• Each argument is valid shape (in terms of `BRepCheck_Analyzer`);

• Each argument can be one of the following types (see the Table 2)

<table>
<thead>
<tr>
<th>No</th>
<th>Type</th>
<th>Index of Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>COMPOUND</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>COMPSOLID</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>SOLID</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>SHELL</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>FACE</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>WIRE</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>EDGE</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>VERTEX</td>
<td>7</td>
</tr>
</tbody>
</table>

• The argument of type 0 (COMPOUND) can include any number of shapes of any type (0, 1…7).

• The argument should not be self-interfered, i.e. all sub-shapes of the argument that have geometrical coincidence through any topological entities (vertices, edges, faces) must share these entities.
6.2. RESULTS

- During the operation the argument Si can be split on parts \(S_{i1}, S_{i2}, \ldots S_{iNbSp}\), \(NbSp\) - number of the parts). The set \((S_{i1}, S_{i2}, \ldots S_{iNbSp})\) is *image* of the argument Si.

- The result of GF operation is a compound. Each sub-shape of the compound (resulting shape) corresponds to the concrete argument shape \(S_{i1}, S_{i2}, \ldots S_{in}\) and has shared sub-shapes in accordance with interferences between the argument and the others.

- For the arguments of the type EDGE, FACE, SOLID the result contains split parts of the argument.

- For the arguments of the type WIRE, SHELL, COMPSOLID, COMPOUND the result contains the image of the shape with the corresponding type (i.e. WIRE, SHELL, COMPSOLID, and COMPOUND).

The types of resulting shapes depend on the type of the corresponding argument (Table 3).

<table>
<thead>
<tr>
<th>No</th>
<th>Type of argument</th>
<th>Type of resulting shape</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>COMPOUND</td>
<td>COMPOUND</td>
<td>The resulting COMPOUND is built from</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Images of sub-shapes of type COMPOUND</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- COMPSOLID, SHELL, WIRE, VERTEX</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Sets of split sub-shapes of type SOLID, FACE,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EDGE</td>
</tr>
<tr>
<td>2</td>
<td>COMPSOLID</td>
<td>COMPSOLID</td>
<td>The resulting COMPSOLID is built from split SOLID-s</td>
</tr>
<tr>
<td>3</td>
<td>SOLID</td>
<td>Set of split SOLID-s</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>SHELL</td>
<td>SHELL</td>
<td>The resulting SHELL is built from split FACE-s</td>
</tr>
<tr>
<td>5</td>
<td>FACE</td>
<td>Set of split FACE-s</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>WIRE</td>
<td>WIRE</td>
<td>The resulting WIRE is built from split</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EDGE-s</td>
</tr>
<tr>
<td>7</td>
<td>EDGE</td>
<td>Set of split EDGE-s</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>VERTEX</td>
<td>VERTEX</td>
<td></td>
</tr>
</tbody>
</table>

The following set of examples is to clarify the definition above.
6.2.1. Example 1

The arguments are three edges $E_1$, $E_2$, $E_3$ (Figure 19), that intersect in one 3D point.

![Figure 19](image)

The result of GF operation is compound.

The compound contains 6 new edges $E_{11}$, $E_{12}$, $E_{21}$, $E_{22}$, $E_{31}$, and $E_{32}$. These edges have one shared vertex $V_{n_1}$.

In this case:
- the argument edge $E_1$ has resulting split edges $E_{11}$, $E_{12}$ (image of $E_1$),
- the argument edge $E_2$ has resulting split edges $E_{21}$, $E_{22}$ (image of $E_2$),
- the argument edge $E_3$ has resulting split edges $E_{31}$, $E_{32}$. (image of $E_3$).
6.2.2. Example 2

The arguments are two wires \( W_1 (E_{w11}, E_{w12}, E_{w13}) \), \( W_2 (E_{w21}, E_{w22}, E_{w23}) \) and the edge \( E_1 \) (Figure 20).

![Diagram showing two wires and an edge, with a transformation to a compound structure](image)

The result of GF operation is compound.

The compound consists of 2 wires \( W_{n1} (E_{w11}, E_{n1}, E_{n2}, E_{n3}, E_{w13}) \), \( W_{n2} (E_{w21}, E_{n2}, E_{n3}, E_{n4}, E_{w23}) \) and edges \( E_{11}, E_{12} \).

In this case

- the argument \( W_1 \) has image \( W_{n1} \),
- the argument \( W_2 \) has image \( W_{n2} \),
- the argument edge \( E_1 \) has split edges \( E_{11}, E_{22} \). (image of \( E_1 \))

The edges \( E_{n1}, E_{n2}, E_{n3}, E_{n4} \) and the vertex \( V_{n1} \) are new shapes created during the operation so as the edge \( E_{w12} \) has split edges \( E_{n1}, E_{n2}, E_{n3} \), the edge \( E_{w22} \) has split edges \( E_{n2}, E_{n3}, E_{n4} \).
6.2.3. Example 3

The arguments are the edge \( E_1 \) and the face \( F_2 \) (Figure 21).

![Diagram of Example 3](image1)

The result of GF operation is compound. The compound consists of 3 shapes:

- Split parts of the edge \( E_{11}, E_{12} \) (image of \( E_1 \))
- New face \( F_{21} \) with internal edge \( E_{12} \) (image of \( F_2 \))

6.2.4. Example 4

The arguments are the edge \( E_1 \) and the face \( F_2 \) (Figure 22).

![Diagram of Example 4](image2)

The result of GF operation is compound. The compound consists of 5 shapes:

- Split parts of the edge \( E_{11}, E_{12}, E_{13} \) (image of \( E_1 \))
- Split parts of the face \( F_{21}, F_{22} \) (image of \( F_2 \))
6.2.5. Example 5

The arguments are the edge $E_1$ and the shell $Sh_2$ that consists of 2 faces $F_{21}$, $F_{22}$ (Figure 23).

The result of GF operation is compound. The compound consists of 5 shapes:

- Split parts of the edge $E_{11}, E_{12}, E_{13}, E_{14}$ (image of $E_1$)
- Image shell $Sh_{21}$ (that contains split parts of the faces $F_{211}, F_{212}, F_{221}, F_{222}$)

6.2.6. Example 6

The arguments are the wire $W_1$ ($E_1, E_2, E_3, E_4$) and the shell $Sh_2$ ($F_{21}, F_{22}$) (Figure 24).

The result of GF operation is compound. The compound consists of 2 shapes:

- Image wire $W_{11}$ that consist of split parts of the edges of $W_1$: $E_{11}, E_{12}, E_{13}, E_{14}$
- Image shell $Sh_{21}$ that contains split parts of the faces $F_{211}, F_{212}, F_{213}, F_{221}, F_{222}, F_{223}$
6.2.7. Example 7

The arguments are 3 faces $F_1$, $F_2$, $F_3$ (Figure 25).

![Figure 25](image.png)

The result of GF operation is compound. The compound consists of 7 shapes:

- Split parts of the faces $F_{n_1}$, $F_{n_2}$, $F_{n_3}$, $F_{n_4}$, $F_{n_5}$, $F_{n_6}$, $F_{n_7}$

6.2.8. Example 8

The arguments are the shell $Sh_1$ ($F_{11}$, $F_{12}$, $F_{13}$) and the face $F_2$ (Figure 26).

![Figure 26](image.png)

The result of GF operation is compound. The compound consists of 4 shapes:

- Image shell $Sh_{11}$ that consist of split parts of the faces of $Sh_1$: $F_{n_1}$, $F_{n_2}$, $F_{n_3}$, $F_{n_4}$, $F_{n_5}$, $F_{n_6}$
- Split parts of the face $F_2$: $F_{n_3}$, $F_{n_6}$, $F_{n_7}$
6.2.9. Example 9

The arguments are the shell Sh₁ (F₁₁, F₁₂...F₁₆) and the solid So₂ (Figure 27).

Figure 27

The result of GF operation is compound.

The compound consists of 2 shapes (Figure 28)

- Image shell Sh₁₁ that consist of split parts of the faces of Sh₁: F₁, F₂,...F₈.
- Solid So₂₁ with internal shell. (image of So₂)

Figure 28
6.2.10. Example 10

The arguments are the compound \( C_{m1} \) (of 2 solids \( S_{o11}, S_{o12} \)) and the solid \( S_{o2} \) (Figure 29).

The result of GF operation is compound. The compound consists of 4 shapes (Figure 30)

- Image compound \( C_{m11} \) that consists of split parts of the solids of \( S_{o11}, S_{o12} \) (\( S_{n1}, S_{n2}, S_{n3}, S_{n4} \)).
- Split parts of the solid \( S_{o2} \) (\( S_{n2}, S_{n3}, S_{n5} \)).
7. **PARTITION ALGORITHM**

7.1. **ARGUMENTS**

The arguments of the algorithm are shapes (in terms of *TopoDS*Shape). The main requirements for the arguments are described in 6.1.

The arguments in PA are separated in two groups:

- Shapes
- Tools

7.2. **RESULTS**

The main difference between GF and PA algorithms is in the rules of building the result.

- For arguments of **Shapes** group the rules are the same as it has been described in 6.2.
- For the arguments of **Tools** group there is the following rule: The result should not contain any shape \( S_{ti} \) from **Tools** group or image of the shape \( S_t \) (\( S_{t1}, S_{t2}, ..., S_{tNbt} \) - number of Tools) from **Tools** group if these shapes are not included in images of **Shapes** group.
- The result will contain shapes that have complexity (in terms of TopAbs_ShapeEnum) that is less or equal to given **Limit**. The values of Limit can have any value. In case when **Limit** = TopAbs_SHAPE the result will contain shapes with complexity that corresponds to the complexity of the arguments.
7.2.1. Example 1

The arguments are

- Shapes: edges E1, E2
- Tools: edge E3 (Figure 31)
- Limit: TopAbs_SHAPE

The result of PA operation is compound.

The compound contains 4 new edges $E_{11}$, $E_{12}$, $E_{21}$, $E_{22}$. These edges have one shared vertex $V_{n1}$.

In this case:

- the argument edge $E_3$ has resulting split edges $E_{11}$, $E_{12}$ (image of $E_1$),
- the argument edge $E_2$ has resulting split edges $E_{21}$, $E_{22}$ (image of $E_2$)
7.2.2. Example 2

The arguments are

- Shapes: wire \( W_1 \) (\( Ew_{11}, Ew_{12}, Ew_{13} \)) and the edge \( E_1 \)
- Tools: wire \( W_2 \) (\( Ew_{21}, Ew_{22}, Ew_{23} \)) (Figure 32).
- Limit : TopAbs_SHAPE

The result of PA operation is compound.

The compound consist of wire \( W_{n1} \) (\( Ew_{11}, En_1, En_2, En_3, Ew_{13} \)), and edges \( E_{11}, E_{12} \).

In this case

- the argument \( W_1 \) has image \( W_{n1} \)
- the argument edge \( E_1 \) has split edges \( E_{11}, E_{22} \). (image of \( E_1 \))

The edges \( En_1, En_2, En_3, En_4 \) and the vertex \( Vn_1 \) are new shapes created during the operation so as the edge \( Ew_{12} \) has split edges \( En_1, En_2, En_3 \), the edge \( Ew_{22} \) has split edges \( En_2, En_3, En_4 \).
7.2.3. Example 3

The arguments are

- Shapes: the face F2
- Tools: the edge E1 (Figure 33).
- Limit : TopAbs_FACE

The result of PA operation is compound.

The compound consists of 3 shapes:

- New face $F_{21}$ with internal edge $E_{12}$ (image of $F_2$)
8. **Boolean Operations Algorithm**

8.1. **Arguments**

- The arguments are shapes (in terms of `TopoDS_Shape`).
- Number of arguments: 2 (Object, Tool).
- Each argument is valid shape (in terms of `BRepCheck_Analyzer`).
- Each argument can be one of the following types (Table 4).

<table>
<thead>
<tr>
<th>No</th>
<th>Type</th>
<th>Index of Type</th>
<th>Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>COMPOUND</td>
<td>0</td>
<td>One of number 0, 1, 2, 3</td>
</tr>
<tr>
<td>2</td>
<td>COMPSOLID</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>SOLID</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>SHELL</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>FACE</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>WIRE</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>EDGE</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>VERTEX</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

- The argument of type 0 (COMPOUND) can include any number of shapes of any type (0, 1…7).
- Each argument should have constant value of the dimension to (Table 4)
- The argument should not be self-interfered, i.e. all sub-shapes of the argument that have geometrical coincidence through any topological entities (vertices, edges, faces) must share these entities.
- For Boolean operation *Fuse* the arguments should have equal dimensions.
- For Boolean operation *Cut* the dimension of `Object` should be not less then the dimension of `Tool`. 
8.2. RESULTS

- The results of BOA\(^1\) is defined by the formulas:

\[
\begin{align*}
B_{\text{common}} &= \text{Object} \cap \text{Tool} \\
B_{\text{fuse}} &= \text{Object} + \text{Tool} \\
B_{\text{cut12}} &= \text{Object} - \text{Tool} \\
B_{\text{cut21}} &= \text{Tool} - \text{Object}
\end{align*}
\]

- The result of BOA operation is a compound \(B_j\). Each component of the compound has shared sub-shapes in accordance with interferences between the \textit{Object} and \textit{Tool}.

- \(B_j\) contains components obtained by the formulas (8.2.1 - 8.2.4). The components are containers of sub-shapes having same dimensions in accordance with Table 4 (in terms of connexity for cases of edges, faces).

- The result \(B_{\text{common}}\) should have the dimension that is equal to the lower dimension of the arguments.

\(^1\) The rules are valid for Open CASCADE 5.0 and higher
8.2.1. Example 1

The arguments are (Figure 34)

- Object: edge $E_1$
- Tool: edge $E_2$

The result of BOA operation is compound.

- **$B_{\text{common}}$**
  The compound contains noting because the dimension of intersection (vertex) is less than minimal dimension of arguments.

- **$B_{\text{fuse}}$**
  The compound contains a wire of 4 new edges $E_{11}$, $E_{12}$, $E_{21}$, $E_{22}$. The edges have one shared vertex $V_{n1}$.

- **$B_{\text{cut12}}$**
  The compound contains a wire of 2 new edges $E_{11}$, $E_{12}$. The edges have one shared vertex $V_{n1}$.

- **$B_{\text{cut21}}$**
  The compound contains a wire of 2 new edges $E_{21}$, $E_{22}$. The edges have one shared vertex $V_{n1}$.
8.2.2. Example 2

The arguments are (Figure 35):

- Object: edge $E_1$
- Tool: edge $E_2$

The result of BOA operation is compound.

- $B_{\text{common}}$ The compound contains a wire of 1 new edge $E_{12}$.
- $B_{\text{fuse}}$ The compound contains a wire of 3 new edges $E_{11}$, $E_{12}$, $E_{13}$. The edges have shared vertices.
- $B_{\text{cut12}}$ The compound contains a wire of 1 new edge $E_{11}$.
- $B_{\text{cut21}}$ The compound contains a wire of 1 new edge $E_{13}$.
8.2.3. Example 3

The arguments are (Figure 36)

- Object: face F
- Tool: edge E

The result of BOA operation is compound.

- **B\text{common}**
  The compound contains new edge \(E_{11}\).

- **B\text{fuse}**
  The compound contains nothing (different dimensions of the arguments).

- **B\text{cut12}**
  The compound contains nothing.

- **B\text{cut21}**
  The compound contains new edge \(E_{12}\).

![Figure 36](image-url)
8.2.4. Example 4

The arguments are (Figure 37)

- Object: face \( F_1 \)
- Tool: face \( F_2 \)

![Diagram of arguments](image)

\[
\begin{align*}
B_{\text{common}} & \quad B_{\text{fuse}} \\
F_{12} & \quad F_{11} \quad F_{22}
\end{align*}
\]
The result of BOA operation is compound.

- **B_common**: The compound contains new face $F_{12}$.
- **B_fuse**: The compound contains a shell of 3 new faces $F_{11}$, $F_{12}$, $F_{22}$.
- **B_cut12**: The compound contains new face $F_{11}$.
- **B_cut21**: The compound contains new face $F_{22}$.
9. LIMITATIONS OF ALGORITHMS

The chapter is devoted to the problems that should be considered as limitations of Algorithms. In most cases the reason of failure is a superposition of different factors (self-interfered arguments, inappropriate, ungrounded values of the tolerances of the arguments, adverse mutual position of the arguments, tangency, etc).

The description below is just to illustrate the limitations and do not exhaust all problems that can be faced in practice.

Each case of failure of Algorithms is the matter of the maintenance service.

9.1. ARGUMENTS

9.1.1. Common requirements

Each argument should be valid (in terms of BRepCheck_Analyzer), or conversely, if the argument is considered as not-valid (in terms of BRepCheck_Analyzer), it can not be used as an argument of the algorithm.

The class BRepCheck_Analyzer is to check the overall validity of a shape. For a shape (or its sub-shapes) to be valid in Open CASCADE, it must respect certain criteria. If the shape is determined as not valid, the problems can be fixed by tools from ShapeAnalysis, ShapeUpgrade, ShapeFix packages.

The class BRepCheck_Analyzer is just hand-made tool that has its own problems. Examples of such problems:

- The Analyzer checks the distances between the two 3D-points $P_i$, $PS_i$ of an edge on a face F. The point $P_i$ is obtained from 3D-curve (at the parameter $t_i$) of the edge. $PS_i$ is obtained from 2D-curve (at the parameter $t_i$) of the edge on surface S of the face F. To be valid the distance should be less then Tol(E). The number of these check-points is constant value (say 23). It is supposed that edge E is valid (in terms of BRepCheck_Analyzer).

- Split the edge E onto two split edges $E_1$, $E_2$. Each split edge has 3D-curve and 2D-curve as the edge E has. Let’s check $E_1$ (or $E_2$). The Analyzer again checks the distances between the two 3D-points $P_i$, $PS_i$. The number of these check-points is again constant value (23). But there are not guarantee that the distance should be less then Tol(E), because the points for $E_1$ are not the same as for E.

- So, in case when $E_1$ is not valid, the edge E also should not be valid. But E is supposed as valid. Thus the Analyzer is wrong for E.

The fact that the argument of Algorithm is valid shape (in terms of BRepCheck_Analyzer) is necessary requirement but not sufficient to produce valid result of the Algorithms.
9.1.2. Pure self-interference

The argument should not be self-interfered, i.e. all sub-shapes of the argument that have geometrical coincidence through any topological entities (vertices, edges, faces) should share these entities.

9.1.2.1. Example 1

The compound of two edges $E_1$, $E_2$ (Figure 38) is self-interfered shape and can not be used as the argument of the Algorithms.

![Figure 38](image1.png)

9.1.2.2. Example 2

The edge $E$ (Figure 39) is self-interfered shape and can not be used as the argument of the Algorithms.

![Figure 39](image2.png)

9.1.2.3. Example 3

The face $F$ (Figure 40) is self-interfered shape and can not be used as the argument of the Algorithms.

![Figure 40](image3.png)
9.1.2.4. Example 4

The face F (Figure 42) has been obtained by revolution of the edge E around the line L (Figure 41).

In spite of the fact that the face F is valid (in terms of BRepCheck_Analyzer) the face F is self-interfered shape and can not be used as the argument of the Algorithms.
9.1.3. Self-interferences due to tolerances

9.1.3.1. Example 1
The edge E (Figure 43) is based on non-closed circle.
The distance between the vertices of E is D=0.69799.
The values of the tolerances Tol(V₁)=Tol(V₂)=0.5 (Figure 44).

![Figure 43](image1)
![Figure 44](image2)

In spite of the fact that the edge E is valid (in terms of BRepCheck_Analyzer) the edge E is self-interfered shape because its vertices are interfered. Thus the edge E can not be used as the argument of the Algorithms.

9.1.3.2. Example 2
The solid S (Figure 45) contains the vertex V. The value of the tolerance Tol(V)= 50.00075982061 (Figure 46). [pkv/901/A9]

![Figure 45](image3)
![Figure 46](image4)

In spite of the fact that the solid S is valid (in terms of BRepCheck_Analyzer) the solid S is self-interfered shape because the vertex V is interfered with a lot of sub-shapes of S without any topological connection with them. Thus the edge E can not be used as the argument of the Algorithms.
9.1.4. Parametric representation

The parametrization of some surfaces (cylinder, cone, surface of revolution) can be the cause of limitations.

9.1.4.1. Example 1

The parameterization range for cylindrical surface is:

\[ U: [0, 2\pi], \quad V: ]-\infty, + \infty[ \]  

(9.1.4)

The range on U coordinate is always restricted; meanwhile the range on V coordinate is non-restricted.

For clearness

- The face (cylinder-based, radii R=3, H=6) (Figure 47). The Figure 48 shows p-Curves for the cylinder.

![Figure 47](image1)

![Figure 48](image2)

- The face (cylinder-based, radii R=3000, H=6000, scale factor ScF=1000) (Figure 49). The Figure 50 shows p-Curves for the cylinder. Please, draw attention on Zoom value on the Figures.

![Figure 49](image3)

![Figure 50](image4)
It is evident that starting with some value of $\text{ScF}$ (say $\text{ScF}>1000000.$) all sloped p-Curves on the Figure 48 will be almost vertical (like on the Figure 50). At least there will be not difference between the values of its angles computed by standard C Run-Time Library functions like $\text{double acos(double x)}$. The loss of accuracy in computation of angles can be the cause of failure of some sub-algorithms of BP (building faces from set of edges, building solids from set of faces).

9.1.5. Stop-gaps

Due to tolerance model it is possible to create shapes that use sub-shapes of lower order to stop gups.

9.1.5.1. Example 1

The face $F$ has two edges $E_1$, $E_2$ and two vertices, plane $(0,0,0,0,0,1)$ (Figure 51).

- The edge $E_1$ is based on line $(0,0,0,1,0,0)$ $\text{Tol}(E_1)=1.e-7$.
- The edge $E_2$ is based on line $(0,1,0,1,0,0)$ $\text{Tol}(E_2)=1.e-7$.
- The vertex $V_1$, point $(0,0.5,0)$ $\text{Tol}(V_1)=1$.
- The vertex $V_2$, point $(10,0.5,0)$ $\text{Tol}(V_2)=1$.

The face $F$ is valid (in terms of $\text{BRepCheck\_Analyzer}$).

![Figure 51](image)

The values of tolerances $\text{Tol}(V_1)$, $\text{Tol}(V_2)$ are big enough to fix the gups between the ends of the edges. But the vertices $V_1$, $V_2$ does not contain an information about trajectories of connection between corresponding ends of the edges. The trajectories are undefined. The fact will be the cause of failure of some sub-algorithms of BP. The sub-algorithms for building faces from set of edges for e.g. use information about all edges connected in a vertex. The situation when one vertex will have several pairs of edges of kind above will not be solved in right way.
9.2. INTERSECTION PROBLEMS

9.2.1. Pure intersections and common zones

9.2.1.1. Example 1

Intersection between two edges (Figure 52):

- $E_1$ – based on line: $\{0,-10,0, 1,0,0\}$, $\text{Tol}(E_1) = 2$.
- $E_2$ – based on circle: $\{0,0,0, 0,0,1\}$, $R = 10$, $\text{Tol}(E_2) = 2$.

![Diagram showing intersection of two edges](image)

Figure 52

The result of pure intersection between $E_1$ and $E_2$ is vertex $V_x \{0,-10,0\}$.

The result of intersection taking into account tolerances is the common zone $CZ$ (part of 3D-space where the distance between the curves is less than (or equals to) sum of tolerances of the edges).

As for IP of Algorithms it uses result of pure intersection $V_x$ instead of $CZ$, because of the reasons:

- The Algorithms does not produce Common Blocks between edges based on underlying curves of explicitly different type (Line / Circle for this case). The rule of thumb is the following: the different type of curves is special sign to produce the result of type “vertex”. The rule does not work for non-analytic curves (Bezier, B-Spline) and theirs combinations with analytic curves.
- The algorithm of intersection between two surfaces ($\text{IntPatch\_Intersection}$) does not compute $CZ$ of intersection curve (-s), point (-s). So even if $CZ$ was computed by Edge/Edge intersection algorithm, its result could not be treated by Face/Face intersection algorithm.
9.2.2. Tolerances

The limitations are due to modeling errors or inaccuracies.

9.2.2.1. Example 1

The arguments are vertex $V_1$ and edge $E_2$ (Figure 53).

The vertex $V_1$ interferes with the vertex $V_{12}$. The vertex $V_1$ interferes with the vertex $V_{22}$.

So the vertex $V_{21}$ should interfere with the vertex $V_{22}$.

But it is impossible because the vertices $V_{21}$, $V_{22}$ are the vertices of the edge $E_2$, thus $V_{21} \neq V_{22}$.

![Figure 53](image)

The problem cannot be solved in general, because the length can be as small as possible to provide validity of $E_2$ (in extreme case: $\text{Length } (E_2) = \text{Tol}(V_{21}) + \text{Tol}(V_{22}) + \varepsilon, \varepsilon \rightarrow 0$).

In particular case the problem can be solved by decreasing the values $\text{Tol}(V_{21})$, $\text{Tol}(V_{22})$, $\text{Tol}(V_1)$ (refinement of arguments).

It is easy to see that if the $E_2$ were slightly above than tolerance sphere of $V_1$ the problem wouldn’t appear at all.
9.2.2.2. Example 2

The arguments are two planar rectangular faces \( F_1, F_2 \) (Figure 54).

Intersection curve between the planes is the curve \( C_{12} \). The curve produces new intersection edge \( EC_{12} \).

The edge goes through the vertices \( V_1, V_2 \) thanks to big values of the tolerances of the vertices \( \text{Tol}(V_1), \text{Tol}(V_2) \). The situation is: two straight edges \( (E_{12}, EC_{12}) \) go through the two vertices. It is impossible for this case.

![Figure 54](image.png)

The problem can’t be solved in general, because the length of \( E_{12} \) can be infinite and the values of \( \text{Tol}(V_1) \) and \( \text{Tol}(V_2) \) theoretically can be infinite too.

In particular case the problem can be solved by several ways:

- Decrease if possible the values \( \text{Tol}(V_1), \text{Tol}(V_2) \) (refinement of \( F_1 \)).
- Analysis of the value of \( \text{Tol}(EC_{12}) \). Increase \( \text{Tol}(EC_{12}) \) in order to get common part between the edges \( EC_{12}, E_{12} \). The common part then will be rejected as already existing edge \( E_{12} \) for the face \( F_1 \).

It is easy to see that if the \( C_{12} \) were slightly above than tolerance spheres of \( V_1, V_2 \) the problem wouldn’t appear at all.
9.2.2.3. Example 3

The arguments are two edges E1, E2 (Figure 55):

- The edges E1, E2 have common vertices V1, V2.
- The edges E1, E2 have 3D-curves C1, C2.
- Tol(E1)=1.e-7, Tol(E2)=1.e-7

C1 practically coincides in 3D with C2. The value of deflection is Dmax (say Dmax=1.e-6)

The evident and prospective result should be Common Block between E1, E2. But the result of intersection is on the Figure 56

The result contains three new vertices Vx1, VX2, VX3 and 8 new edges (between whereas V1, VX1, VX2, VX3, V2) and no Common Blocks. The result (Figure 56) is quite correct due to source data: Tol(E1)=1.e-7, Tol(E2)=1.e-7, Dmax=1.e-6.

In particular case the problem can be solved by several ways:

- Increase if possible the values Tol(E1), Tol(E2) to get coincidence in 3D between E1, E2 in terms of tolerance.
- Replace E1 by more by the more accurate model.
The example can be extended from 1D (edges) to 2D (faces) (Figure 57).

The comments and recommendations are the same as for 1D case.