OPEN CASCADE 学习笔记
——拓扑和几何

著: **Roman Lygin**

译：**George Feng**

这是一篇关于开源三维建模软件 OPEN CASCADE 内核的博文：ROMAN LYGIN 是 OPEN CASCADE 的前程序开发员和项目经理，曾经写过许多关于该开源软件开发包的深入文章，可以在他的博客(HTTP://OPENCASCADE.BLOGSPOT.COM)上面找到这些文章。

序

在 **OpenCascade** 的论坛上知道了 **Roman Lygin** 在他的博客上写了 **Open CASCADE　notes** 系列文章，但是却无法访问他的博客，幸而百度文库已经收录了 **Topology and Geometry** 和 **Surface Modeling** 两篇文章，拜读之后获益良多。如果大家发现文中翻译有错误或不足之处，望不吝赐教，可以发到我的邮箱 **fenghongkui@sina.com.cn**，十分感谢。

2012 年 **6** 月 **28** 日星期四

第 1 节　概述

下面首先介绍一下 Open CASCADE 的基本概念。关于这些基本概念的问题经常在论坛中出现，我希望这篇文章能够帮助大家理解这些基本概念。如果你要使用或者开发自己的建模算法，就更要理解这些基本概念。或许你应该重新读一读 Modeling Data User's Guide 的各个章节，从而加深记忆。

拓扑抽象类

Open CASCADE 的拓扑结构是在参考 STEP ISO-10303-42 标准的基础上设计的。也许读一读 STEP 标准也是有益的(我自己曾经在 1997 年读过一次)。

Open CASCADE 的结构是面向单向图的(an oriented one-way graph)，在该单向

图中父类可以引用(refer to)他们的子类，而不能反向引用(reference)。抽象结构在 TopoDS 包中的 C++类中实现。图 1 是由 Doxygen 生成的继承关系图。

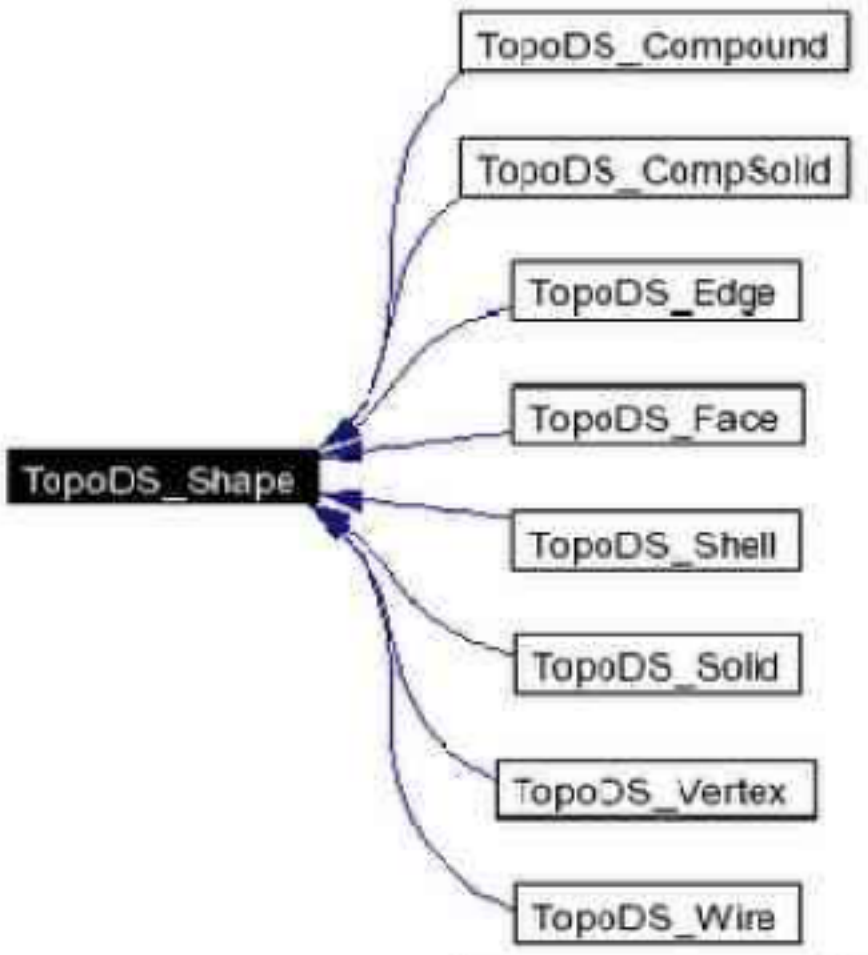

图 1 由 Doxygen 生成的抽象类继承关系图

TopoDS_Shape 包含三个成员变量，–位置、朝向以及一个 TopoDS_TShape 的 myTShape 句柄(handle)–如下图所示(该图中各类仅包含非常重要的成员变量):

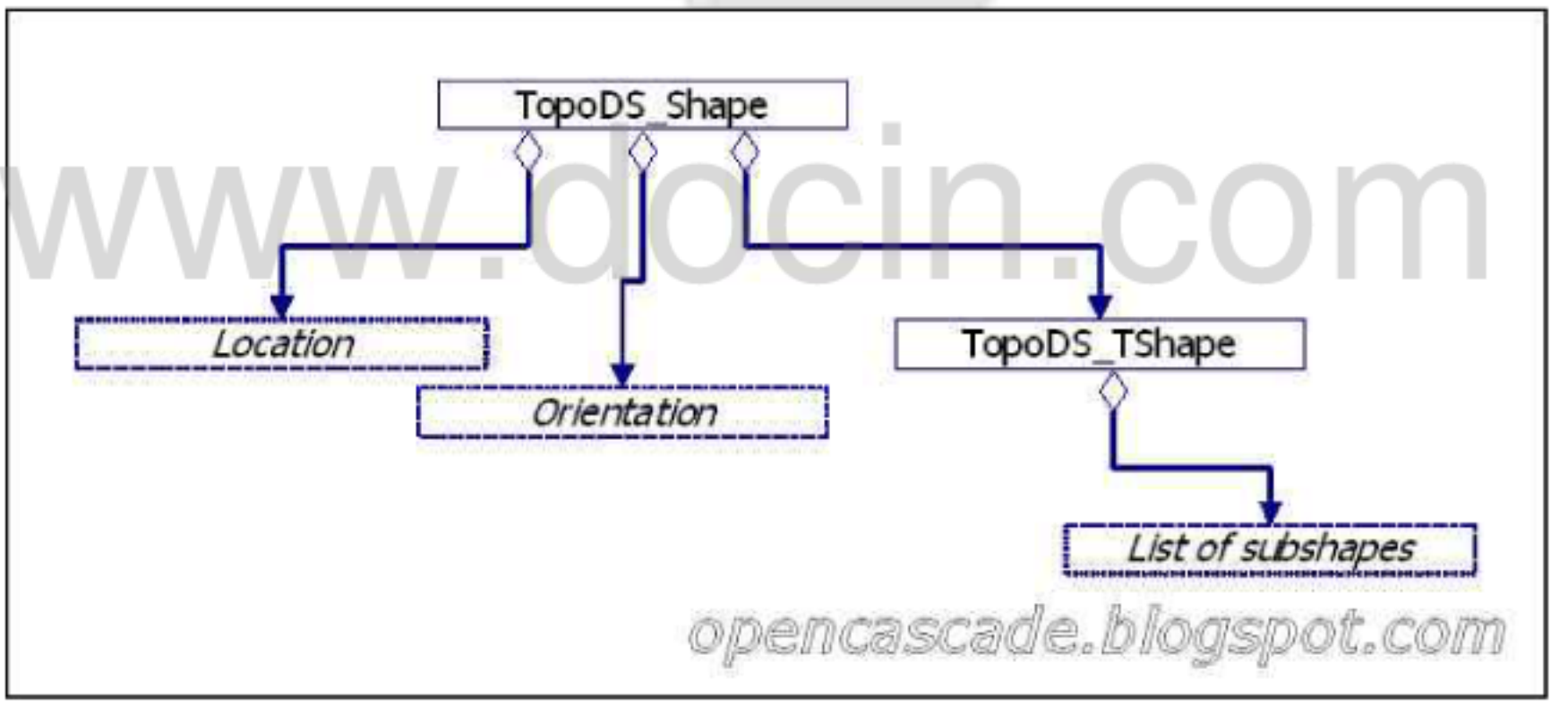

图 2 TopoDS_Shape 的成员变量

各种不同的形体(shape)可以共享拓扑形体和位置(myTShape and Location)信息，从而节省大量的内存空间。例如，一条边属于两个面，具有相同的拓扑形体和位置，但是不同的朝向(Orientations)(在一个面中朝向向前，在另一个面中朝向相反).

与几何数据**(Geometry)**的关系
现在考虑该抽象拓扑类是如何与几何信息关联起来的。

这是通过继承 TopoDS 包中的抽象拓扑类实现的，在 BRep 包中通过这些类实现了对具有边界的模型描述。只有三种类型的拓扑体具有几何特征，顶点(vertex)、边(edge)、面(face)，如图 3 所示。

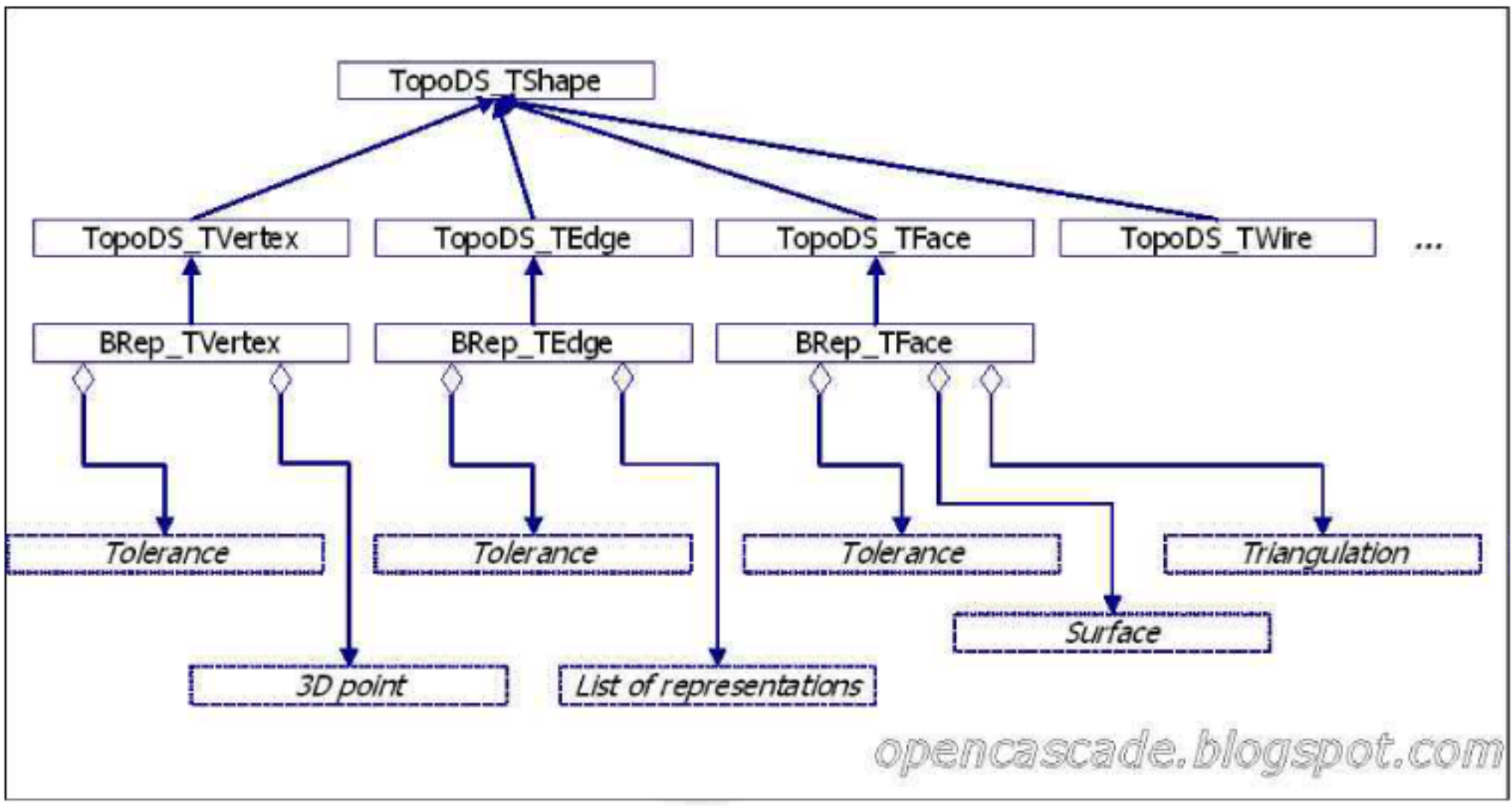

图 3 抽象拓扑类与几何信息的关系

下面将会更为详细的描述它们。

待续... ...

第 2 节 顶点

接上节... ...

顶点**(Vertex)**的基本概念
顶点具有基本的几何表示，其几何表示是在三维空间中用类 gp_Pnt 描述的一个点。它也有其他的表示方式，但是这些表示方式在实际中几乎不会用到。

顶点的另一个重要的属性是误差(tolerance)，用来描述其位置的精确度。顶点误

差的几何意义是在顶点精确位置处具有半径为 T 的圆球。这些误差圆球必须通过所有在该点相连接的曲线端部。

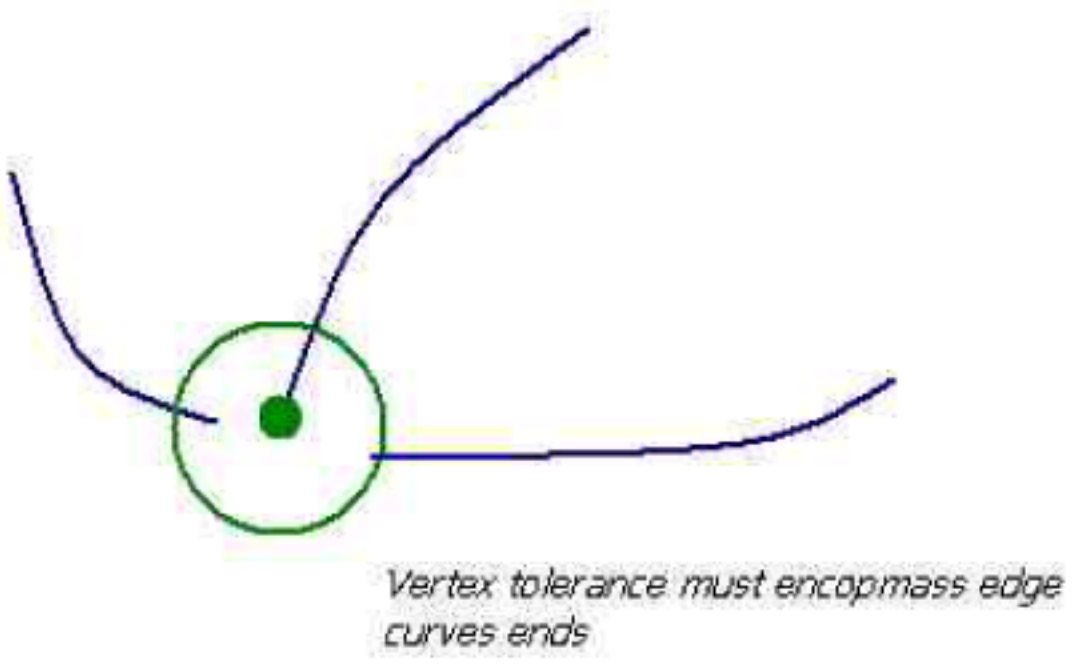

图 4 误差圆球必须通过所有在该点相连接的曲线端部

误差(Tolerances)

下面将更为详细的介绍误差。

不像其他的几何库，一般都具有全局的精度误差(例如，模型包围盒大小的 1/1000), Open CASCADE 将误差作为一种局部属性(local properties)。正如在第一节的图 3 中介绍的，误差是顶点(vertex)、边(edge) 、面(face)的成员变量。一般来说，这种方法能够以更高的精度描述模型。如图 5 所示：

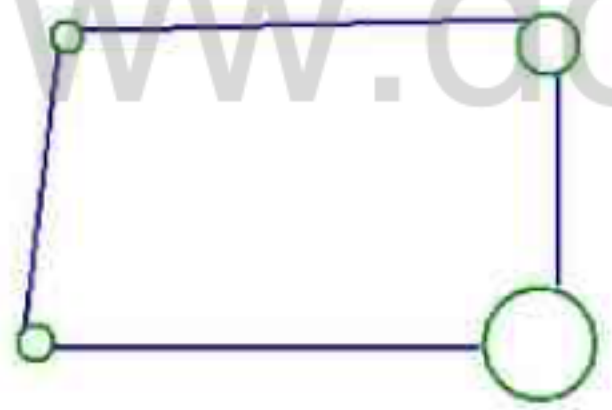

图 5 具有误差的连接边

各个拓扑类具有各自独立的误差使得它们从模型中分离出来后，仍然可以设置局部精度。假如你要从底层构造形体(shape)，最好的方法是设置最小的允许误差。误差缺省值是 Precision::Confusion()，1e-07。

一般来说，一些建模算法对误差非常的敏感，特别是当这些算法不得不使用用户设置的某个值时，就变得更加敏感。例如，从另外的三维几何内核(包括 IGES、STEP 格式，或者其他内核或者 CAD 系统的特有格式)中导入模型，这涉及到在

文件头(file header)设置的全局精度。导入程序尽力为了保持健壮而忽略了那个值，而不管这个全局误差是多大(例如，全局误差可以非常大，以至于某些小的面被它们的边界穿越了，这甚至违背了标准)。

另外一个例子是缝合(Sewing)(将不相连的面粘合到一个壳体(shell)里)。在不同的模型中，面与面之间的缝隙差别非常大。设置太小的误差会导致结果中出现许多不相连的面，设置的误差太大又会导致间距很大的面连接到一起(即使用户故意设置为不相连的面也连接到了一起，例如小的孔)。

下面讨论顶点的朝向属性。它没有直接的几何意义，但是连接的顶点具有朝向 TopAbs_FORWARD 时，必须与曲线(curve)端部的较小参数值对应(the smaller parameter，说明：三维曲线可以使用参数来描述其上的点，一般起始点参数为 First value 或者 Start value，终止点参数为 Last value 或者 End value，起始点的参数值要小于终止点的参数值，这样较小的参数值应是起始点的参数值)。相对应的，具有 TopAbs_REVERSED 属性的顶点具有较大的参数值。例如，假如有一条边(Edge)，在一个圆弧曲线之上，该圆弧半径为 1，位于平面 Z=0 上，其起始点 point (1, 0, 0)，绕着 Z 轴逆时针旋转 90 度，那么具有向前属性的点在(1, 0, 0)处，具有反向属性的点在(0, 1, 0) 处。如图 6 所示。

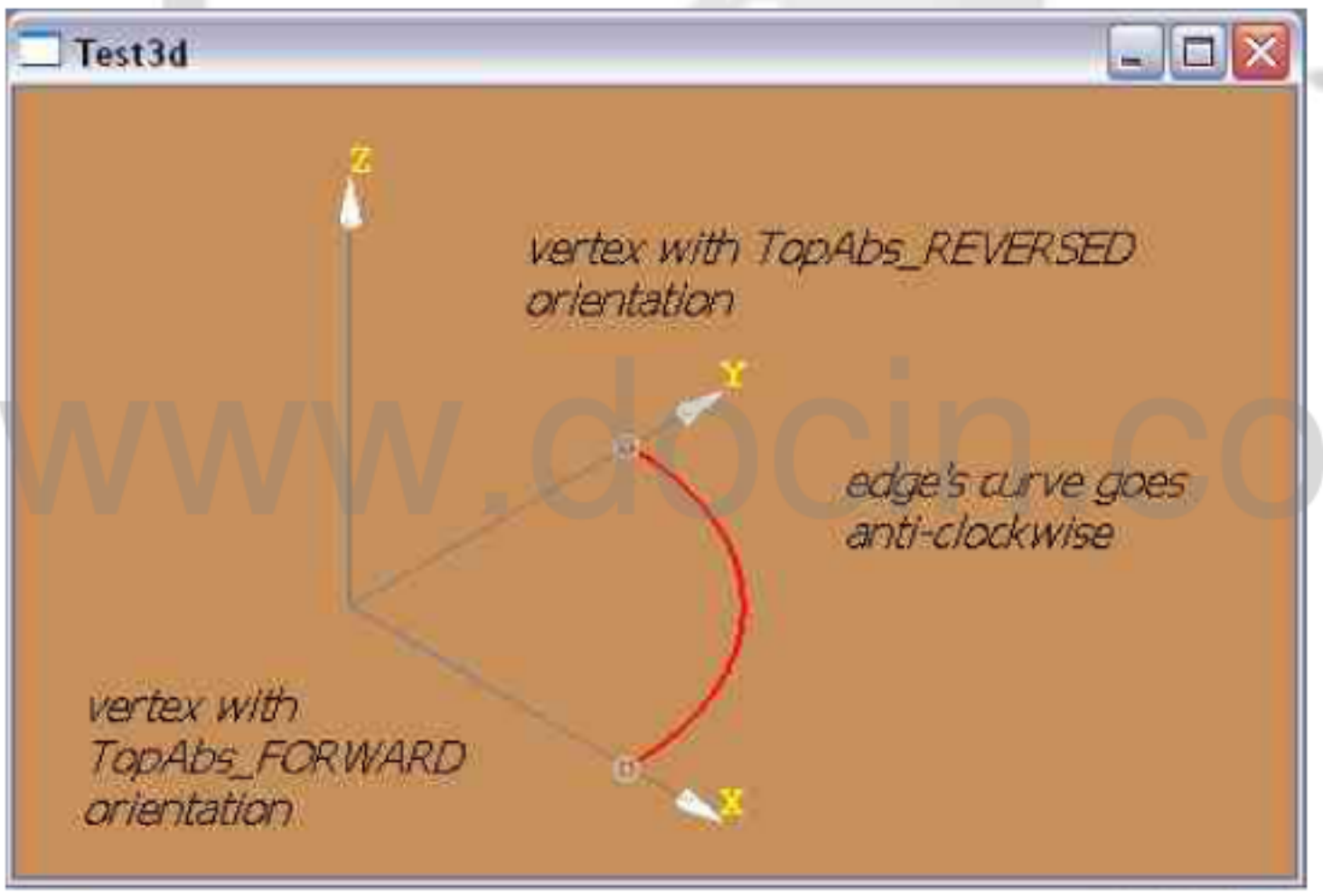

图 6 设置顶点的朝向属性值的一个示例

从底层构建顶点
BRep_Builder 实际上是一个进行底层操作的工具。

```
gp_Pnt aPoint (100., 200., 300.);
BRep_Builder aBuilder;
TopoDS_Vertex aVertex;
aBuilder.MakeVertex (aVertex, aPoint, Precision::Confusion());
```

```
aVertex.Orientation (TopAbs_REVERSED);
```

有一个非常方便的生成顶点的类 BRepBuilderAPI_MakeVertex，该类最终在内部使用 BRep_Builder 生成顶点。所以如果不得不从底层开始构造拓扑结构，你就必须对 BRep_Builder 非常熟悉。

获得顶点的信息的方法
BRep_Tool 提供了访问拓扑体的几何信息的方法，其内的很多方法都是静态(static)类型的。

```
Standard_Real aTolerance = BRep_Tool::Tolerance (aVertex);
gp_Pnt aPoint = BRep_Tool::Pnt (aVertex);
```

待续... ...

第 3 节 边

接上节... ...

好了，让我们继续一点一点的啃这个大象。下面讲边(Edge)，希望你不要觉得它太难。

边(Edge)

边是一种拓扑体，它对应于一维对象(object)——曲线(curve)。它可以用来指定面(Face)的边界(例如一个盒子有 12 条边)或者仅仅是悬空的('floating')边，即不属于任何面的边(想象一下在构建锥形体或者扫略体之前的初始轮廓线)。属于面(Face)的边可以在两个或者更多的面之间共享(例如，印章模型中它们是面之间的连接线)或者可以只属于一个面(在印章模型种这属于边界边)。我想你一定看到了这些类型的边——在缺省的视图的线框模式下，它们分别显示为红色(悬空边)、黄色(面之间的连接边)、绿色(边界边)。如图 7 所示。

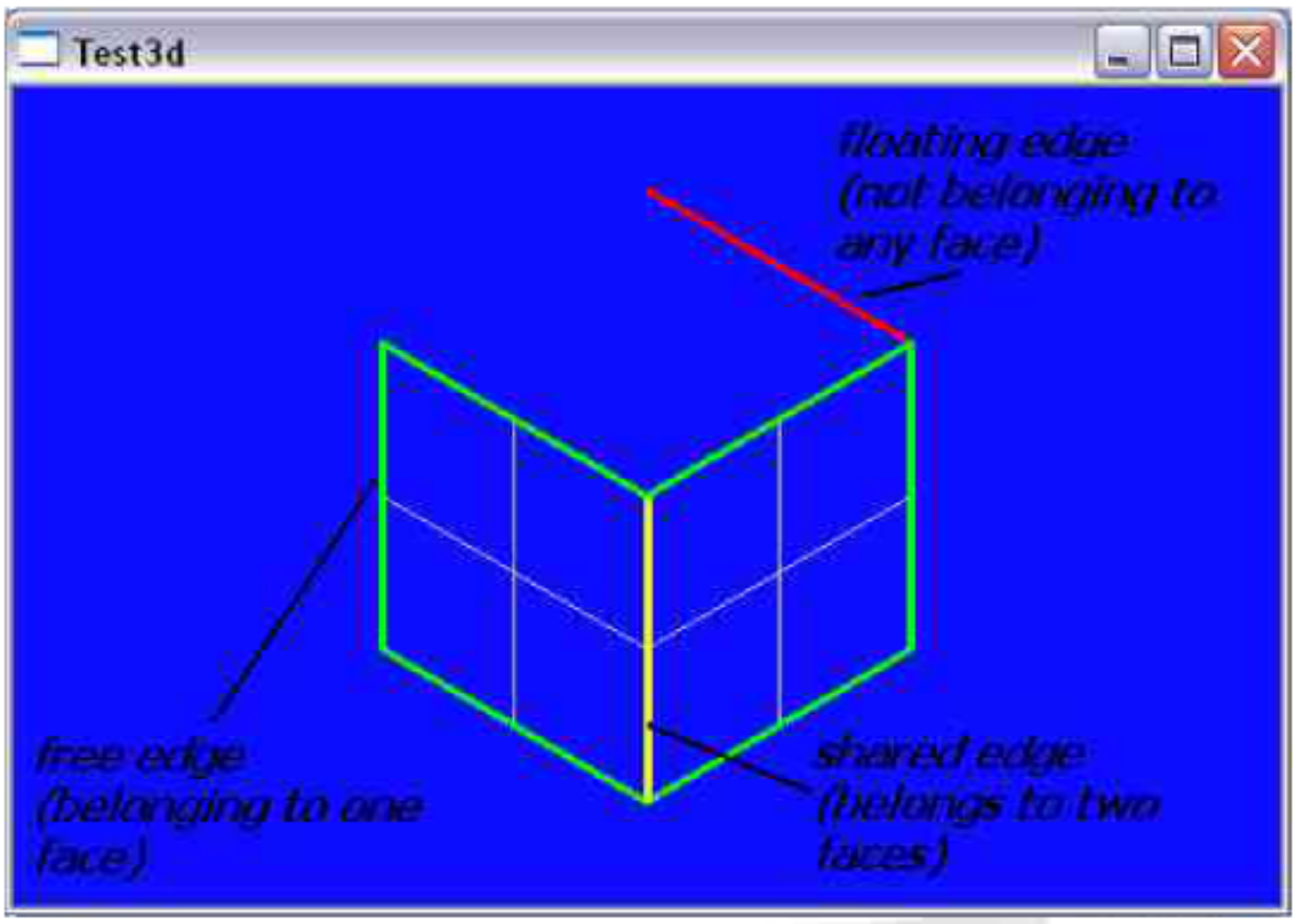

图 7 印章模型的线框模式

边有几种几何表示方式(参考第一节中的图 3，BRep_TEdge 之下有一个节点 List Of Edge Represention)：
(1) 三维空间的曲线 C(t)，用类 Geom_Curve 实现。这是一种基础表示方式。
(2) 曲线 P(t) 为二维空间的参数曲线，用于描述位于曲面上的曲线。它们经常被称为 pcurves，在类 Geom2d_Curve 来实现。
(3) 多边形表示(Polygonal representation)，使用一组三维点来描述，在类 Poly_Polygon3D 实现。
(4) 多边形表示(Polygonal representation) ，也可以使用一组三维点的编号来描述，在类 Poly_PlygonOnTriangulation 实现。

后两种表示方式在前面两种的帮助下用面片描述模型。

这些表示方式可以使用 BRep_Tool 来获取必要的信息，例如：

```
Standard_Real aFirst, aLast, aPFirst, aPLast;
Handle(Geom_Curve) aCurve3d = BRep_Tool::Curve (anEdge, aFirst, aLast);
Handle(Geom2d_Curve) aPCurve = BRep_Tool::CurveOnSurface (anEdge,
aFace, aPFirst, aPLast);
```

每一个边必须有其所在的曲面上的所有参数曲线(pcurves)，只有在平面上可以没有参数曲线，因为在平面上参数曲线可以在悬空状态下计算。

边的所有曲线表示方式必须相互之间保持一致，例如，C(t)和 P(t)表示的曲线在朝向必须一致。这样边上的点可以使用任意表示方式计算得到，可以使用 C(t)

在[first, last]取值得到；也可以 u 在[first1, last1] 取值得到曲面 S1 (P1x (u), P1y(u))上的点 Pi，这里 Pi 是曲面 Si 上的参数曲线 pcurve 的一点。

边的标志位**(Edge flags)**

边具有两种比较特殊的标志：
(1) "same range" (BRep_Tool::SameRange())，相同的取值区间，当 first = first_i and last = last_i 时，该标志位为 true，例如，所有的几何表示都在相同的表示区间内。
(2) "same parameter" (BRep_Tool::SameParameter())，相同的参数，当 C(t) = S1(P1x(t), P1y(t))时，即对于同样的参数 t，C(t)和曲面 S1 上的点 P1(t)能够得到相同的点，此时该位为 true，也就是说，任何边上的点都对应参数曲线上相同的参数值的点。

许多算法假定设置了这两个标志位，因此建议你注意这种情况，一定要设置这些标志位。

误差**(Tolerance)**
边的误差是其三维曲线和其他任何表示方式之间的最大偏差。这样，它的几何意义就是一根沿着三维曲线的具有半径的管子，这个管子将从其他表示方式中获取的曲线包围。

图 8 具有孔隙(void)的体

特殊类型的边
有两种特殊类型的边。分别是：

-缝合边(seam edge)。一种被同一个面使用两次的边(例如，在同一个面上具有 2 个参数曲线)。
- 退化边(degenerated edge)。这种边位于曲面的奇异点位置处，在三维空间中为一个点。
球面中这两种类型的边都有。缝合边位于经线 (U iso-lines)，参数为 0 和 2*PI。退化边位于南北极，对应于纬线(V iso-lines)，参数为–PI/2 和 PI/2。

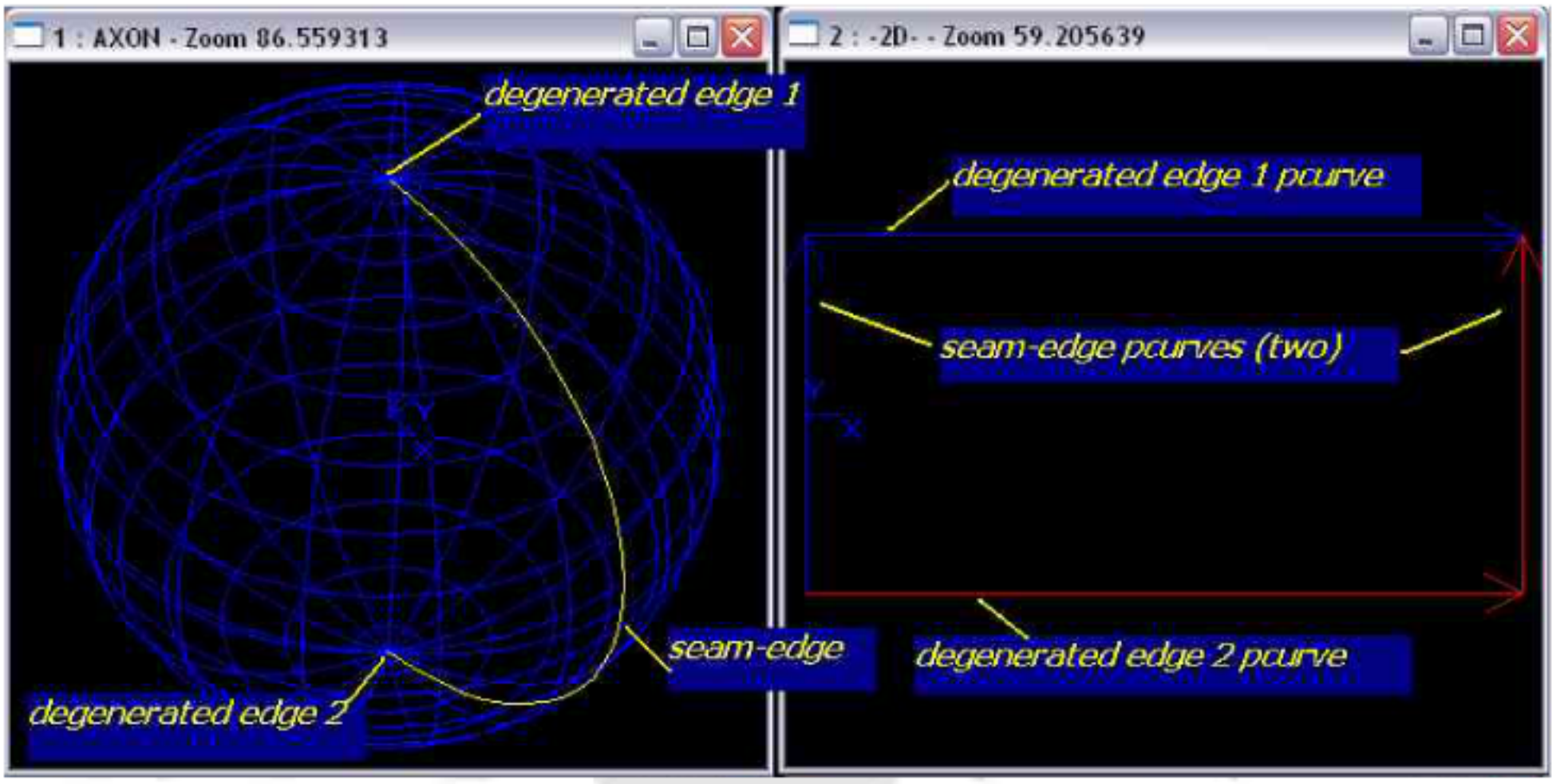

图 9 具有孔隙(void)的体

其他例子——环形体(torus)、圆柱体(cylinder)、锥体(cone)。环形体有两条缝合边，对应于它的参数空间的边界；圆柱体有一条缝合边。退化宾出现在锥体的顶点。

要检测边是缝合边或者退化边，使用函数 BRep_Tool::IsClosed() 和 BRep_Tool::Degenerated()。

边的朝向**(Edge orientation)**
边的方向向前，意味着边(edge)的逻辑方向与曲线(curve)的方向相同。反向意味着逻辑方向与曲线方向相反。因此，缝合边在一个面中总是有 2 个朝向——一个相反，另一个向前。

待续... ...

第 4 节 面

接上节... ...

虽然在边之后的拓扑类型是环，但是下面将先讲面，它是包含几何信息(geometry)的最后一种拓扑体。我们将在之后的章节中把环和其他拓扑类型放到一起讨论。坦白说，我本来没有打算讲环和其他拓扑类型，但是有好几个人在博客中要求讲它们。

面
面是一种拓扑体，用于描述三维实体的边界部分。面是由底层的曲面以及一个或者多个环来描述的。例如，圆柱体包含 3 个面——底面、顶面和侧面。每一个面的底层曲面都是无限大的(Geom_Plane 和 Geom_CylindricalSurface)，同时每一个面由一个环限定其在曲面上的范围，其中两个环是位于 Geom_Circle 上一条边，侧面包含 4 条边，其中两条边与顶面与底面共享，剩下的两条边是缝合边(参见之前对于边的讨论)，例如面在环中包含缝合边两次(两条缝合边具有不同的朝向)。

曲面**(Surface)**
让我们简要回想一下什么是曲面。假如你在高中的时候上过数学分析(mathematical analysis)，你可能记得很清楚，假如你忘了，就自学一下。在维基百科(wikipedia)上关于曲面的第一部分给出了参数曲线的简单例子。

曲面将二维参数空间{U,V}映射到三维空间的物体上(虽然还是二维)。与压模(一块平面铁皮可以被压成弯曲的)相比，参数空间可以是有界的也可以是无界的，或者仅仅在某个方向上有界。例如，平面参数空间是无界的，但是 NURBS 曲面是有界的，圆柱曲面在 U 方向是有界的(从 0 到 2*PI)，在 V 方向上是无界的(-infinity, + infinity)。

Geom_Surface::Value()返回参数空间中(U, V)的三维点(X, Y, Z)。例如，地球上的任意一点都可以由纬度(V)和经度(U)表示，但是在世界坐标系中可以看作三维点(假如地球的中心定义为原点)。

让我们回想一下，边必须具有三维曲线(a 3D curve)和曲面空间中的参数曲线(a pcurve)。Open CASCADE 要求面的边界(环)在三维和二维空间中必须是闭合的。因此，圆柱的侧面是采用前面我们讨论的那样描述的。

并非所有的建模核心模块要求这样。我记得从 Unigraphics (通过 STEP 格式) 导出的圆柱，它的侧面是由两个环(wire)描述的，每一个环包含一个圆形边(circular edge)。我与来自数据交换(Data Exchange)组的同事讨论过好几次这个问题，怎么样才能识别出这种情况，怎么样才能将它们转换到 Open CASCADE 中并具有合法的环。作为讨论的结果，在 Shape Healing 的 ShapeFix_Face 中添加了 FixMissingSeam()。

朝向**(Orientation)**
面的朝向表示面的法向与曲面的法向之间的关系(Face orientation shows how face normal is aligned with its surface normal)。假如面的朝向是 TopAbs_FORWARD (向前的)，则面的法向与曲面的法向相同，假如面的朝向是 TopAbs_REVERSED (相反)，则面的法向与曲面的法向相反。

面的法向表明材料(body material)的方位——材料位于面的后面。在正确描述的实体中，所有的面法向都是向外的('outward')，如图 10 所示。

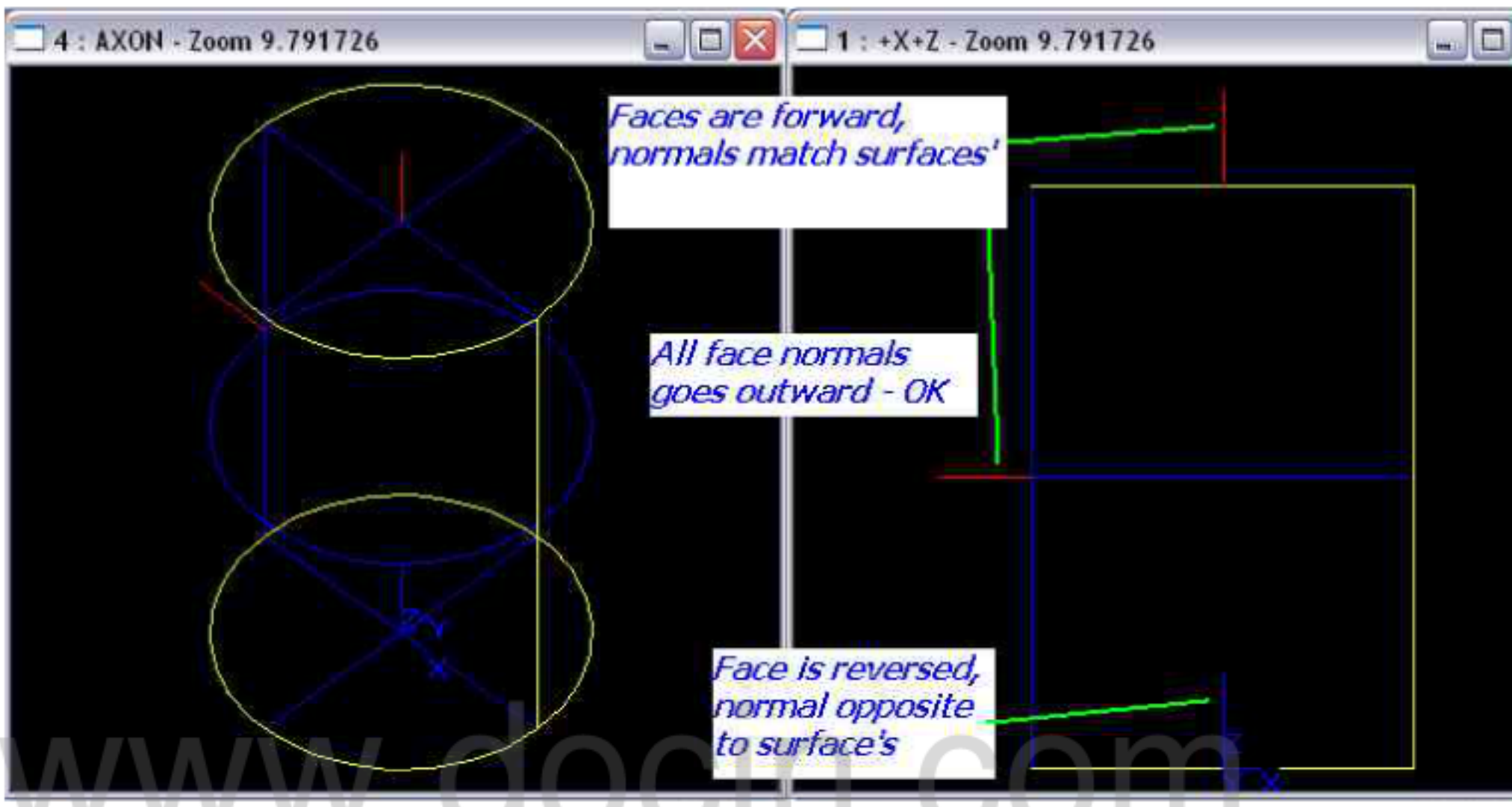

图 10 实体中所有的面法向都是向外

面上的材料是由边的朝向确定的。方位是由曲面(不是面 surface (not face!))法向和边的导数(edge derivative)的叉积确定。边的导数等于它的三维曲线的导数，假如边的朝向向前，否则，如果边的朝向相反则边的导数与它的三维曲线的导数相反。也许考虑边的参数曲线会更为容易理解：假如边是向前的，材料在它的左侧，假如相反，材料在它的右侧。如图 11 所示。这听起来很复杂，但你一旦掌握了它，还是很容易的。在后面的章节我们将更为深入的讨论朝向。

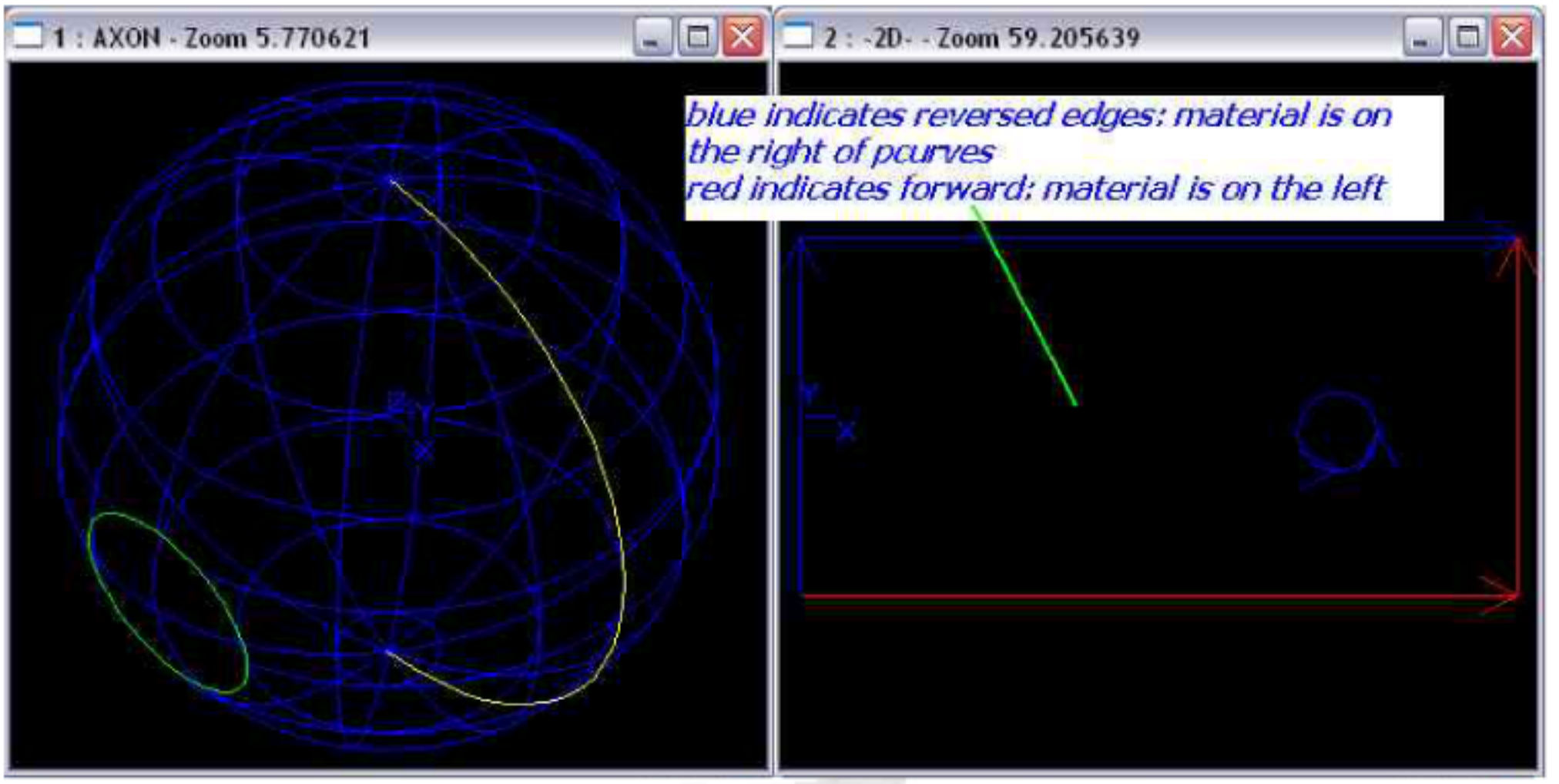

图 11 材料与边及参数曲线的关系

误差(Tolerance)

面的误差的几何意义是包围面的一个具有厚度的板。

图 12 具有孔隙(void)的体

与边和顶点的误差比较，建模算法中用到面的误差的情况相对非常少。通常都具有缺省值(Precision::Confusion())。
通常情况下，Open CASCADE 要求注意下面的情况：
面的误差 <= 边的误差 <= 顶点的误差，在此边位于面上，顶点位于边上。

三角化**(Triangulation)**
除了底层的曲面，面也包含面片化表示(tessellated representation)，可以是许多三角面片。例如，在阴影模式下显示时需要计算面片。可视化算法内部调用 BRepMesh::Mesh()，该函数为每一个面计算并添加三角面片。

附加位置**(Additional location)**
不像边和顶点，面具有附加位置(TopLoc_Location 类型)，该位置是 BRep_TFace 的一个成员。所以，在使用底层的曲面(surface)或者三角化(triangulation)时，不要忘了将其考虑进去。Stephane 最近在论坛中指出了这个问题。

采用自底向上的方式生成一个面并访问其数据
正如在顶点和边中的例子那样，你需要使用 BRep_Builder 和 BRep_Tool。

```
BRep_Builder aBuilder;
TopoDS_Face aFace;
aBuilder.MakeFace (aFace, aSurface, Precision::Confusion());
...

TopLoc_Location aLocation;
Handle(Geom_Surface) aSurf = BRep_Tool::Surface (aFace, aLocation);
gp_Pnt aPnt = aSurf->Value (aU, aV).Transformed
(aLocation.Transformation());
//or to transform a surface at once
//Handle(Geom_Surface) aSurf = BRep_Tool::Surface (aFace);
//gp_Pnt aPnt = aSurf->Value (aU, aV);

Handle(Poly_Triangulation) aTri = BRep_Tool::Triangulation (aFace,
aLocation);
aPnt = aTri->Nodes.Value (i).Transformed (aLocation.Transformation());
```

希望上面的讲的不是特别枯燥，只要做个标记，将来需要的时候再回头看看就可以了。

待续... ...

第 5 节 容器拓扑类型

接上节... ...

其他拓扑类型**(Other topology types)**

目前为止我们讨论了顶点(vertex)、边(edge)、面(face)——这些都与几何属性有关。其他拓扑类型(包括环(wire)、壳(shell)、体(solid)、联体(compsolid)、复合体(compound)并不与几何属性直接相关，它们属于其他拓扑实体(entity)的容器(containers)：

(1) 环(wire)包含边(edge)
(2) 壳(shell)包含面(face)
(3) 体(solid)包含壳
(4) 联体(compsolid) 包含共面的体(solid)
(5) 复杂体(compound) 可以包含任意类型(包括联体 compsolid)。

下面对体(solid)简要介绍一下。体(solid)应该包含单个壳(shell)，壳(shell)中描述体的外边界。假如体(solid)中包含两个或者多个壳，就认为其为具有孔隙(void)的体。如图 13 所示。但是 Open Cascade 遇到这样的具有孔隙的体时会有问题，所以要小心。

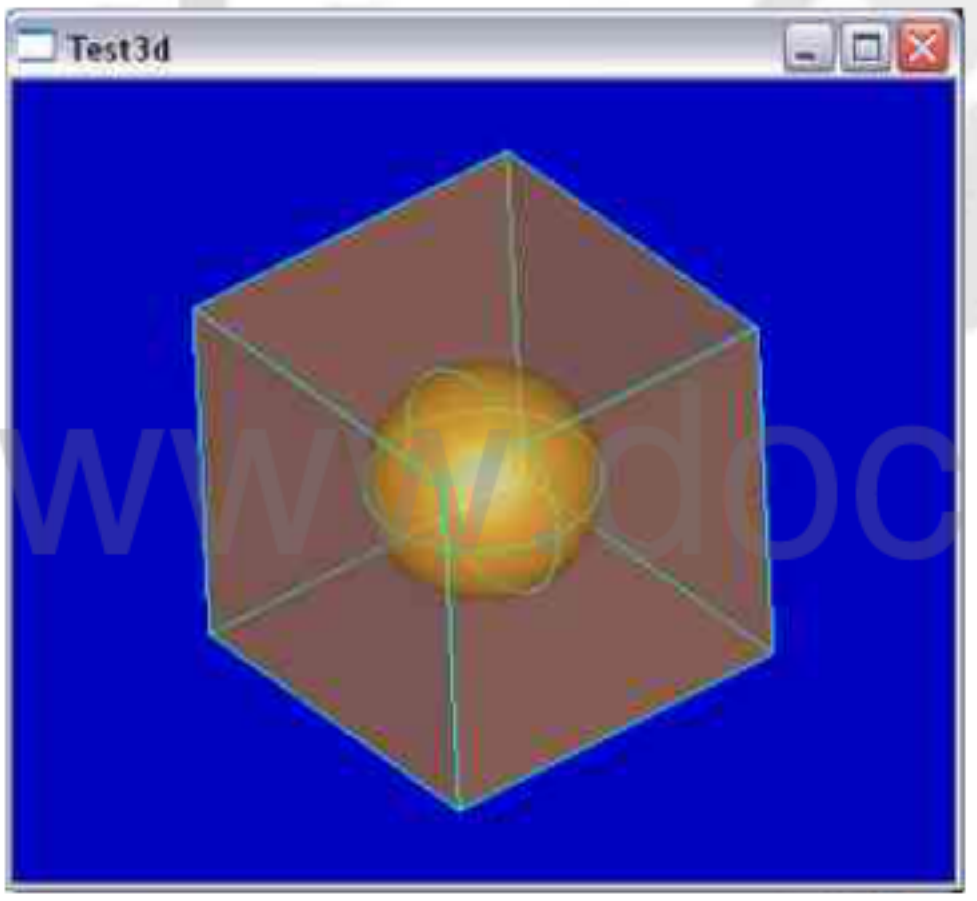

图 13 具有空隙(void)的体

迭代查询所有的子节点**(Iteration over children)**

有两种迭代查询所有的子节点的方法。

1. 直接子节点可以使用 TopoDS_Iterator 查询。

下面的函数将遍历整个实体(shape)结构：

```
void TraverseShape (const TopoDS_Shape& theShape)
{
TopoDS_Iterator anIt (theShape);
for (; anIt.More(); anIt.Next()) {
```

```
const TopoDS_Shape& aChild = anIt.Value();
TraverseShape (aChild);
}
}
```

TopoDS_Iterator 有两个标志位，用于控制在查询子节点时设定是否考虑父节点的位置和朝向。假如位置标志位设为开(on)，那么所有的字节点返回的值就像它们是独立的实体一样，而且位置是在具有全局坐标系的三维空间中的位置。(例如，用户将会看到单独取出来的边与在其父类环中显示的时候位置一样。假如朝向标志位设置为开(on)，那么返回的子节点的朝向将会变成父节点的朝向与子节点的朝向的乘积(例如，如果子节点和父节点两者都是反向或者向前，那么结果仍然向前，只要两者中有一个反向，那么结果就是反向。)
假如标志位为关(off)，那么子节点实体就只返回其自身保存的位置和朝向(参考第的节中的图 2)。缺省情况下，两者的值都设置为开(on)。

2. 访问指定类型的子节点
假如你想访问你的实体(shape)中所有边，那么可以使用下面的方法：

```
TopExp_Explorer anExp (theShape, TopAbs_EDGE);
for (; anExp.More(); anExp.Next()) {
const TopoDS_Edge& anEdge = TopoDS::Edge (anExp.Current());
//do something with anEdge
}
```

TopExp_Explorer 有一个附加的参数，可以用来指定要跳过的父节点类型。例如，你只想获取漂浮的边(floating edge，即不属于面的边，参考第 3 节)，那么你可以按照下面的方式访问边：

```
TopExp_Explorer aFloatingEdgeExp (theShape; TopAbs_EDGE,
TopAbs_FACE);
```


再深入讨论一下位置和朝向**(location and orientation)**
正如在前面章节中描述的，一个几何体(顶点 vertex，边 edge，面 face)的位置变量(Location)定义了底层几何体相对于该节点的移动大小。任意拓扑体(不论其是否包含由几何体)的位置变量(Location)也定义了其子节点相对于该节点的移动大小。例如，假如，环有一个位置变量，该变量具有沿着{0, 0, 10}向量移动的部分，就意味着它的所有边都沿着 Z 轴移动 10 个单位。

朝向与位置的工作原理相同。当将子节点从实体中分离出来时，父节点的朝向会影响子节点的朝向。但是有一个例外非常重要，面中包含的边不遵循这个规律。回顾一下第 4 节图 11，在那里我们讨论说，面的实体材料位于朝向向前的边的参数曲线的左边，在朝向向后的边的参数曲线的右边。这里计算边的反向或者前向不应该受到面的朝向的影响。也就是说，假如要使用边的参数曲线，应该按照下面的方法使用：

```
TopExp_Explorer aFaceExp (myFace.Oriented (TopAbs_FORWARD),
```

```
TopAbs_EDGE);
for (; aFaceExp.More(); aFaceExp.Next()) {
const TopoDS_Edge& anEdge = TopoDS::Edge (aFaceExp.Current());
}
```

假如你研究得更为深入，这个例外也是可以理解的。让我们从底层构造一个面：

```
Handle(Geom_Surface) aSurf = new Geom_Plane (gp::XOY());

//沿着曲面的法线方向为逆时针方向的圆，外圆
Handle(Geom_Curve) anExtC = new Geom_Circle (gp::XOY(), 10.);
//内圆
Handle(Geom_Curve) anIntC = new Geom_Circle (gp::XOY(), 5.);
// MakeEdge 外圆
TopoDS_Edge anExtE = BRepBuilderAPI_MakeEdge (anExtC);
// MakeEdge 内圆
TopoDS_Edge anIntE = BRepBuilderAPI_MakeEdge (anIntC);
//外圆环
TopoDS_Wire anExtW = BRepBuilderAPI_MakeWire (anExtE);
//内圆环
TopoDS_Wire anIntW = BRepBuilderAPI_MakeWire (anIntE);
BRep_Builder aB;
TopoDS_Face aFace;
aB.MakeFace (aFace, aSurf, Precision::Confusion());
aB.Update (aFace, aSurf);//添加曲面
aB.Add (aFace, anExtW);//添加外圆
aB.Add (aFace, anIntW.Reversed()); //材料(material)部分位于内圆环的右侧
```

面缺省情况下朝向是向前的。遍历面的所有边和参数曲线。虽然没有显式的添加参数曲线，在第 3 节中我们知道平面上的参数曲线可以默认(on the fly)计算：

```
void TraversePCurves (const TopoDS_Face& theFace)
{
TopExp_Explorer anExp (theFace, TopAbs_EDGE);
for (; anExp.More(); anExp.Next()) {
const TopoDS_Edge& anEdge = TopoDS::Edge (anExp.Current());
Standard_Real aF, aL;
Handle(Geom2d_Curve) aPCurve = BRep_Tool::CurveOnSurface (anEdge,
theFace, aF, aL);
}
}
```

返回的参数曲线如图 14 所示(材料在红色曲线的左边，在蓝色曲线的右边)。

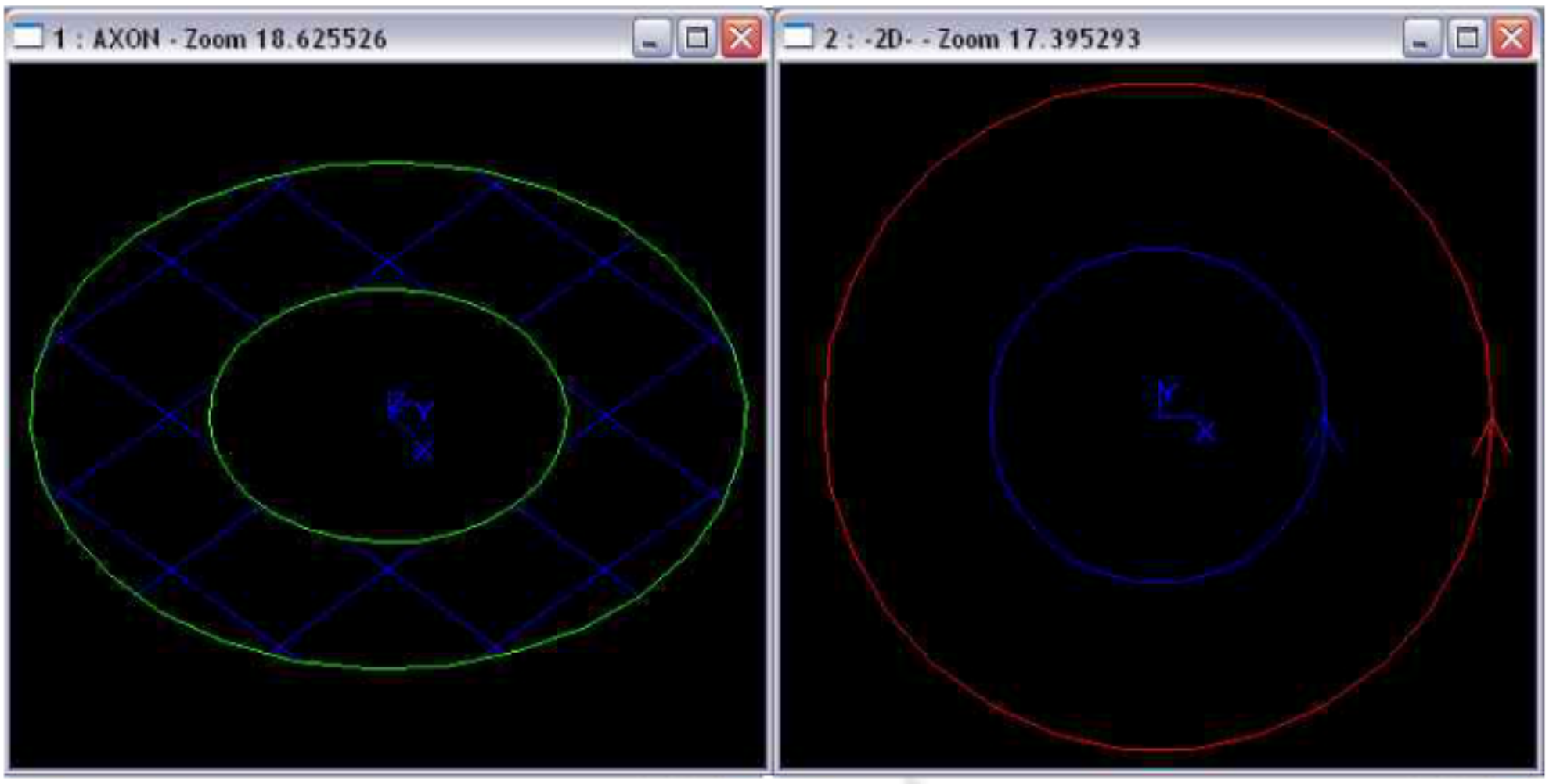

图 14 具有孔隙(void)的体

一切都是那么完美。现在想一下，将这个面的朝向反转一下，然后再遍历所有的参数曲线：

```
TopoDS_Face aRFace = TopoDS::Face (aFace.Reversed());
TraversePCurves (aRFace);
```

将会看到什么？所有的边将会具有相反的朝向，材料将位于内圆参数曲线的内侧，外圆参数曲线的外测。这样以来，虽然原始反转曲面(aRFace, a Reversed Face)是正确的，但是材料的方位明显是错误的。在第 4 节中，我们讲过面(face)的朝向仅仅是面的逻辑朝向，而与其底层的曲面(surface)没有关系。在我们这个例子中反转曲面 aRFace 只是一个法向为{0, 0, -1}的面。

因此必须使用下面的方法访问边：

```
TopExp_Explorer anExp (theFace.Oriented (TopAbs_FORWARD),
TopAbs_EDGE);
```

这样就确保边具有正确的朝向，而与曲面(surface，注意在此不是 face!)的法向没有关系。在第 4 节中我也是这样讲的。Open CASCADE 的算法对这种特殊的情形都做了这样的处理，你一定也要记得这样做。

希望朝向也变成这本书中很容易理解的一部分了。还有几块硬骨头要啃，假如你的胃口够好我们将在最后一节中结束掉拓扑和几何这个话题。加油!

待续... ...

第 6 节 转换器

接上节... ...

回顾
在使用 OpenCASCADE 时你可能也注意到了或者在第 1 节中的图中分析到了，实体包含它们的子实体，而不是相反的方式。这是可以理解的，同一个实体或者子实体可以属于不同的实体。例如任意共享边可以属于至少两个面。然后，有时候需要从子实体追踪到父实体。可以使用函数 TopExp::MapShapesAndAncestors()来完成这个任务。

```
TopTools_IndexedDataMapOfShapeListOfShape anEFsMap;
TopExp::MapShapesAndAncestors (myShape, TopAbs_EDGE, TopAbs_FACE,
anEFsMap);
```

上面的代码生成了 myShape 中的面和边之间的映射。假如 myShape 是长方体，每一条边会映射到两个面上。假如遍历同样的长方体，并在每一个面中尽力找到边的父节点，那么明显的该映射中一条边只有一个面，也就是当前正在搜索的面。

适配器**(Adaptors)**
一些 Open CASCADE 算法可以操作表示曲线的对象，然而，它们提供的 API 并不接受 Geom_Curve 类型的参数，而是 Adaptor3d_Curve 类型。例如，Extrema 就是这样，以便于该类应用于相交、映射以及其他关于几何曲线(Geom_Curve)和拓扑边(TopoDS_Edge)的算法中。
其他例子包括计算曲线长度、或者曲面面积。这种方法称为适配模式。
GeomAdaptor3d_Curve 是 Adaptor3d_Curve 的子类，该类用于适配 Geom_Curve 类型，BRepAdaptor_Curve 用于适配 TopoDS_Edge，BRepAdaptor_CompCurve 用于适配 TopoDS_Wire。对于二维曲线和曲面也有相似的类。

所以你应该按照下面的方式来计算曲线或者边的长度：

```
Handle(Geom_Curve) aCurve = ...;
Standard_Real aCurveLen = GCPoints_AbscissaPoints::Length
(GeomAdaptor_Curve (aCurve));

TopoDS_Edge anEdge = ...;
Standard_Real anEdgeLen = GCPoints_AbscissaPoints::Length
(BRepAdaptor_Curve (anEdge));
```

小结(Conclusion)

这篇关于 Open CASCADE 基本概念的论述有点长。希望现在你对它们更熟悉了，理解了几何和拓扑是什么。作为第一步，你应该正确的使用这些名词——曲线和边(curve or edge)、曲面和面(surface or face) 、点和顶点(point or vertex)。这将有助于你和其他人读懂你的问题中所说的是几何体还是拓扑体。一旦你开始使用正确的定义，并时刻牢记这些区别，部分问题可能就已经解决了。

就这些了！ 总算啃完这整头大象了！

中英文对照表

adaptor 适配器
compound 复杂体
compsolid 联体
curve 曲线
degenerated edge 退化边
edge 边
face 面
geometry 几何信息
line 线
material 材料
seam edge 缝合边
shell 壳
solid 体
surface 曲面
TEdge 拓扑边
TFace 拓扑面
topology entity 拓扑体
pcurves, parametric curve 参数曲线
polygonal representation 多边形表示
void 孔隙
wire 环