Topology and Geometry in OpenCascade

Location and Orientaion

eryar@163.com

摘要 Abstract：本文简要介绍了几何造型中的边界表示法（BRep），并结合程序说明OpenCascade 中的边界表示的具体实现，即拓朴与几何的联系。拓朴结构中的位置（Location）和朝向（Orientation）进行了详细说明。

关键字 Key Words：OpenCascade、BRep、Topology、Geometry、Location、Orientation

一、引言 Introduction

OpenCascade 中的拓朴（topology）是根据 STEP 标准 ISO-10303-42 设计的。也许读一下这个标准中的有关概念还是很有帮助的。STEP ISO-10303-42 的相关资源：

http://www.steptools.com/support/stdev_docs/express/step_irs/index.html

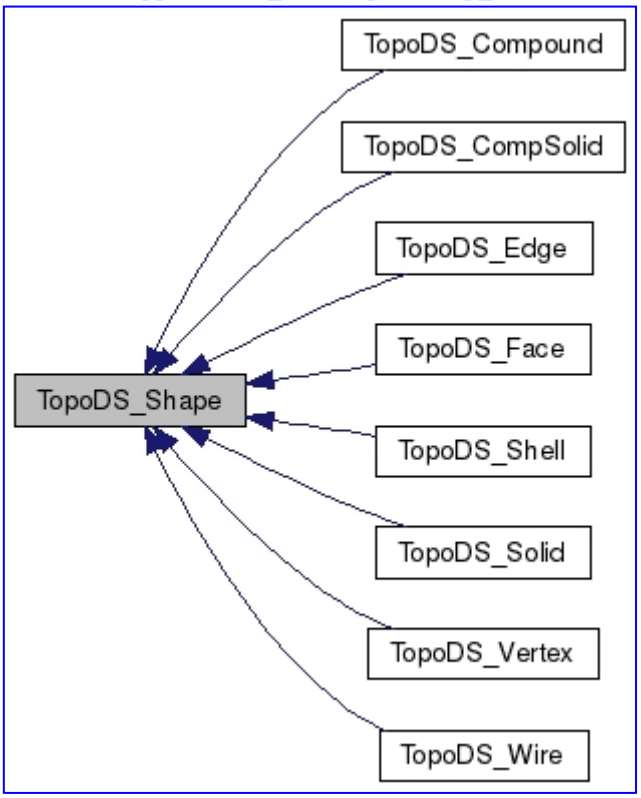
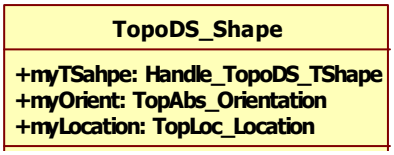


Figure 2.1 Topology data structure in OpenCascade



TopoDS_Shape 由值控制，包含三个成员变量：myLocation、myOrient、myTShape。

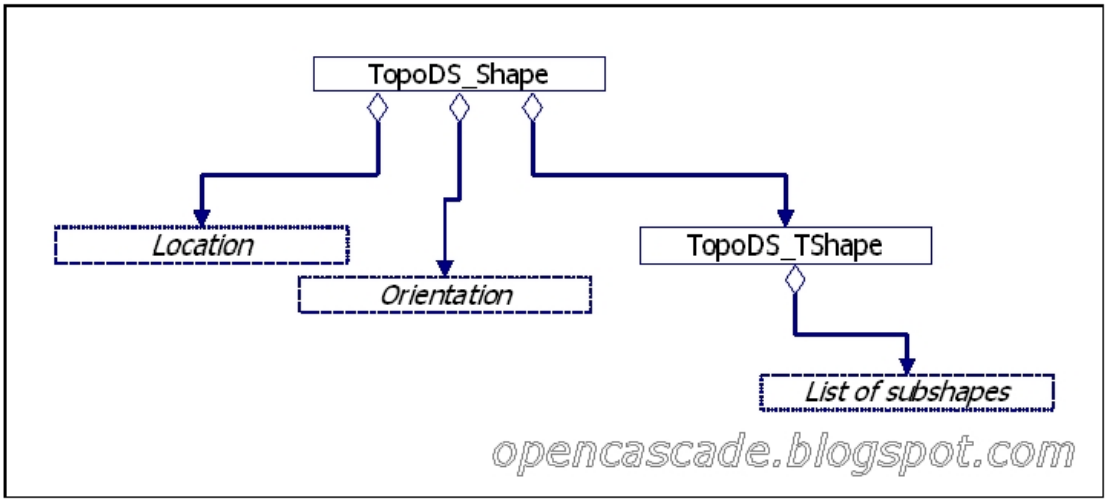


Figure 2.2 TopoDS_Shape member fields

2.2 拓朴与几何的联系 Connection with Geometry

现在我们来考虑一下拓朴结构与几何的关系。通过继承 TopoDS 包中的抽象的拓朴类实现了边界表示模型。如下图所示：

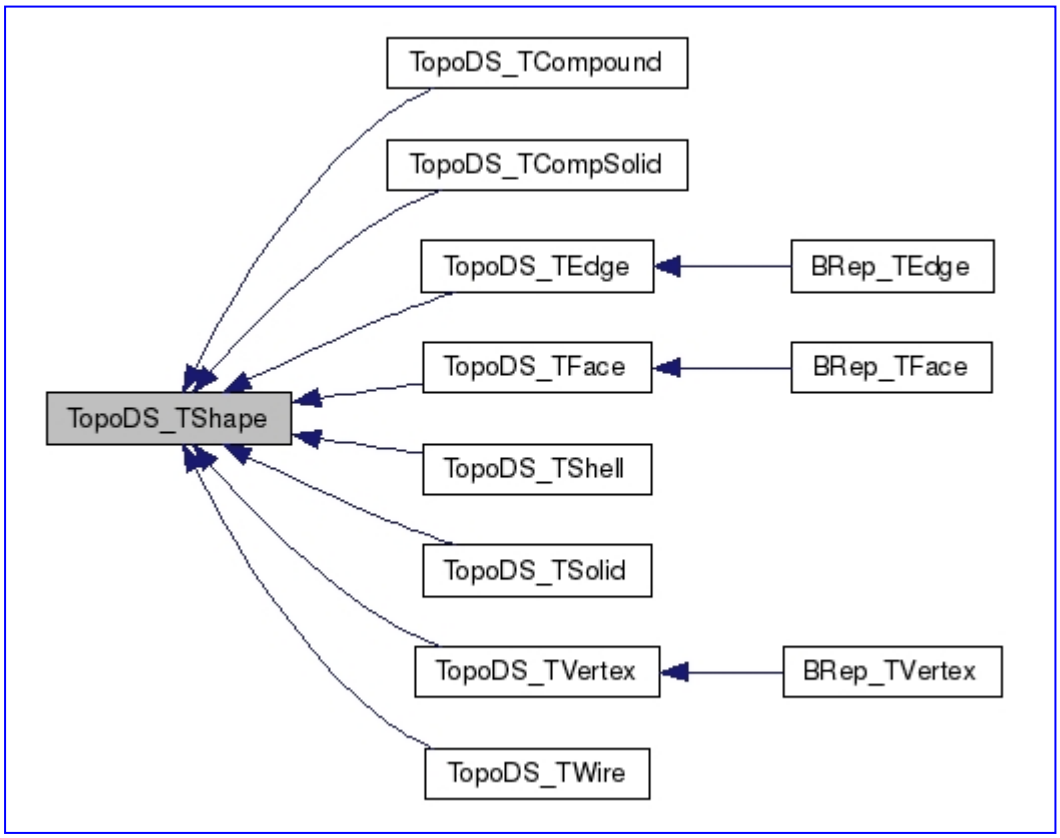


Figure 2.3 Topology data structure in OpenCascade

从上面的类图可以看出只有三种拓朴对象有几何表示：顶点（vertex）、边（edge）、面（face），分别为 BRep_TVertex、BRep_TEdge、BRep_TFace。

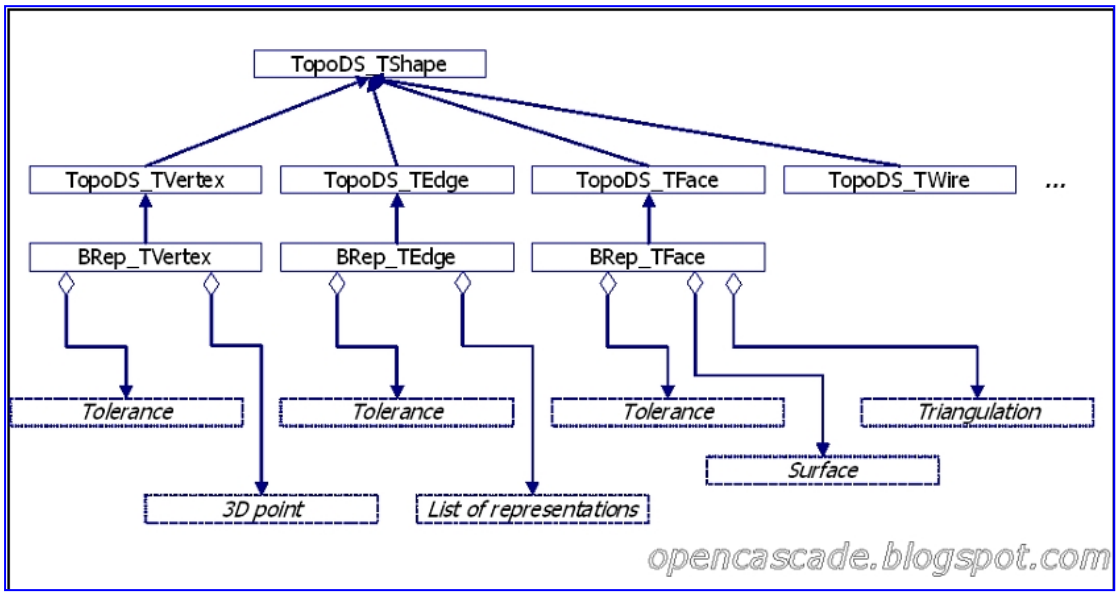


Figure 2.4 TopoDS_TShape class diagram

二、位置 Location

ToposDS_Shape 有个 TopLoc_Location 的成员变量 myLocation，该变量定义了子形状相对于该形状的偏移量。例如，环（wire）有一个位置变量，该变量沿着向量{0,0,10}移动，意味着环所有的边都沿着 Z 轴移动 10 个单位。下面以一个程序来具体说明：

```
/*
*    Copyright (c) 2013 eryar All Rights Reserved.
*
*        File    : Main.cpp
*        Author  : eryar@163.com
*        Date    : 2013-09-26
*        Version : V1.0
*
*    Description : Shape location.
*
*/

#define WNT
#include <Geom_Circle.hxx>

#include <TopoDS_Edge.hxx>
#include <TopoDS_Wire.hxx>
#include <TopoDS_Iterator.hxx>

#include <BRepBuilderAPI_MakeEdge.hxx>
#include <BRepBuilderAPI_MakeWire.hxx>

#pragma comment(lib, "TKernel.lib")
#pragma comment(lib, "TKMath.lib")
#pragma comment(lib, "TKG3d.lib")
#pragma comment(lib, "TKBRep.lib")
#pragma comment(lib, "TKTopAlgo.lib")

const std::string dumpShapeType(const TopAbs_ShapeEnum& type)
{
    std::string strType("Shape");

    switch (type)
    {
    case TopAbs_COMPOUND:
        strType = "COMPOUND";
        break;

    case TopAbs_COMPSOLID:
        strType = "COMPSOLID";
        break;

    case TopAbs_SOLID:
        strType = "SOLID";
        break;
```

```
    case TopAbs_SHELL:
        strType = "SHELL";
        break;

    case TopAbs_FACE:
        strType = "FACE";
        break;

    case TopAbs_WIRE:
        strType = "WIRE";
        break;

    case TopAbs_EDGE:
        strType = "EDGE";
        break;

    case TopAbs_VERTEX:
        strType = "VERTEX";
        break;

    default:
        break;
    }

    return strType;
}

void dumpShapeLocation(const TopoDS_Shape& shape)
{
    std::cout << "Shape Type: " << dumpShapeType(shape.ShapeType()) << std::endl;
    shape.Location().ShallowDump(std::cout);

    TopoDS_Iterator anItr(shape);

    for (; anItr.More(); anItr.Next())
    {
        const TopoDS_Shape& aChild = anItr.Value();

        dumpShapeLocation(aChild);
    }
}

int main(void)
{
    Handle_Geom_Curve aCircle = new Geom_Circle(gp::XOY(), 5.0);

    TopoDS_Edge anEdge = BRepBuilderAPI_MakeEdge(aCircle);
    TopoDS_Wire aWire = BRepBuilderAPI_MakeWire(anEdge);

    std::cout << "Before transformation: " << std::endl;
    dumpShapeLocation(aWire);
```

```
    gp_Trsf trsf;
    trsf.SetTranslation(gp_Vec(0, 0, 10));
    TopLoc_Location location(trsf);

    aWire.Location(location);

    std::cout << "After transformation: " << std::endl;
    dumpShapeLocation(aWire);

    return 0;
}
```

程序结果如下所示：

```
Before tranformation:
Shape Type: WIRE
TopLoc_Location : Identity

Shape Type: EDGE
TopLoc_Location : Identity

Shape Type: VERTEX
TopLoc_Location : Identity

Shape Type: VERTEX
TopLoc_Location : Identity

After transformation:
Shape Type: WIRE
TopLoc_Location :
        Exponent : 1
 TopLoc_Datum3D 035400E8
  (         1,          0,          0,          0)
  (         0,          1,          0,          0)
  (         0,          0,          1,         10)


Shape Type: EDGE
TopLoc_Location :
        Exponent : 1
 TopLoc_Datum3D 035400E8
  (         1,          0,          0,          0)
  (         0,          1,          0,          0)
  (         0,          0,          1,         10)


Shape Type: VERTEX
```

```
TopLoc_Location :
        Exponent : 1
 TopLoc_Datum3D 035400E8
  (          1,          0,          0,          0)
  (          0,          1,          0,          0)
  (          0,          0,          1,         10)


Shape Type: VERTEX
TopLoc_Location :
        Exponent : 1
 TopLoc_Datum3D 035400E8
  (          1,          0,          0,          0)
  (          0,          1,          0,          0)
  (          0,          0,          1,         10)


Press any key to continue . . .
```

三、朝向 Orientation

朝向（orientation）与位置（location）的工作原理相同。当将子对象从实体中分离出来时，父对象的朝向会影响到子对象的朝向。但是有个很重要的例外就是面（Face）上边（Edge）的朝向不遵守这个规则。在讨论面的朝向时，讨论过边的参数空间曲线（pcurve）的 material 问题。这个例外说的是计算面上边的朝向时不应该受到面的朝向的影响。即若要使用边的参数空间曲线（pcurve）就按下面的方式：

```
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。让我们以一个具体例子来理解这个例外，从底层一步步的创建一个面：

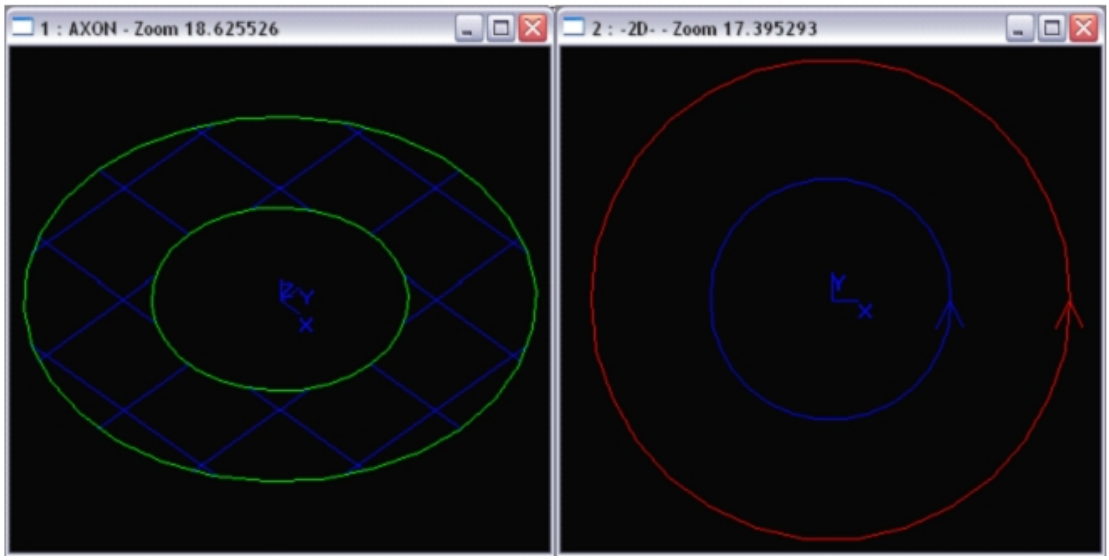
```
Handle(Geom_Surface) aSurf = new Geom_Plane (gp::XOY());
//anti-clockwise circles if to look from surface normal
Handle(Geom_Curve) anExtC = new Geom_Circle (gp::XOY(), 10.);
Handle(Geom_Curve) anIntC = new Geom_Circle (gp::XOY(), 5.);
TopoDS_Edge anExtE = BRepBuilderAPI_MakeEdge (anExtC);
TopoDS_Edge anIntE = BRepBuilderAPI_MakeEdge (anExtC);
TopoDS_Wire anExtW = BRepBuilderAPI_MakeWire (anExtE);
TopoDS_Wire anIntW = BRepBuilderAPI_MakeWire (anIntE);
BRep_Builder aB;
TopoDS_Face aFace;
aB.MakeFace (aFace, aSurf, Precision::Confusion());
aB.Update (aFace, aSurf);
aB.Add (aFace, anExtW);
//material should lie on the right of the inner wire
aB.Add (aFace, anIntW.Reversed());
```

面默认的朝向是向前的（forward），让我们来遍历面的边（edge）和参数空间曲线（pcurve）。尽管我们没有显示地来添加它们，从原来的讨论中可知平面上的参数空间曲线可以默认计算（be computed on the fly）：

```
void TraversePCurves (const TopoDS_Face& theFace)
{
    TopExp_Explorer anExp (theFace, TopAbs_EDGE);
    for (; anExp.More(); anExp.Next())
    {
        Standard_Real aF = 0.0;
        Standard_Real aL = 0.0;
        const TopoDS_Edge& anEdge = TopoDS::Edge (anExp.Current());

        Handle(Geom2d_Curve) aPCurve = BRep_Tool::CurveOnSurface (anEdge, theFace,
aF, aL);
    }
}
```

得到的参数空间曲线（pcurves）如下图所示，material 在红线的左侧，在蓝线的右侧：



一切都很正确。现在设想一下，如果把面的朝向反向（reverse），然后再遍历边和参数空间曲线，会发生什么呢？

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

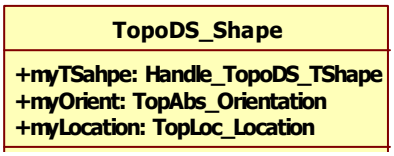
所有的边将会具有相反的朝向，对应边的参数空间曲线（pcurve）的 material 在外环的外侧，在内环的内侧，这明显是错误的！在前面讨论面的朝向时就说过面的朝向仅仅是面的逻辑朝向，而与其底层的曲面（surface）没有关系。在上面的例子中反转面 aRFace 只是一个法向为{0， 0， －1}的面。所以，要获得边的正确朝向，必须使用下面的方法来访问面中的边：

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

这样就确保面上的边具有正确的朝向，而与曲面（surface，注意在此不是 face！）的法向没有关系。OpenCASCADE 的算法对这种特殊的情况都做了处理，你也一定记得这样做。

四、结论 Conclusion

对拓朴形状（TopoDS_Shape）的位置（location）和朝向（orientation）进行深入理解，整个拓朴结构就变得清晰了，因为一个拓朴形状中除了子对象外，剩下就是位置和朝向成员变量了。



理解了拓朴结构后，对 OpenCascade 的模块 ModelingData 就有个较深刻地认识了。

五、参考资料

1. Roman Lygin, OpenCascade notes, opencascade.blogspot.com
2. 孙家广等. 计算机图形学. 清华大学出版社
3. OpenCascade source code.