Topology and Geometry in OpenCascade-Topology

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。对具有几何信息的拓朴结构顶点（vertex）、边（edge）、面（face）进行了详细说明。本文通过 ACIS 与 OpenCascade 进行对比来对拓朴（Topology）的概念进行说明。并通过示例程序，说明如何在 OpenCascade 中取得与一个拓朴对象相连的其他拓朴对象，包括父对象和子对象。

关键字 Key Words：OpenCascade、ACIS、BRep、Topology、Geometry

一、引言 Introduction

边界表示（Boundary Representation）也称为 BRep 表示，它是几何造型中最成熟、无二义的表示法。实体的边界通常是由面的并集来表示，而每个面又由它所在的曲面的定义加上其边界来表示，面的边界是边的并集，而边又是由点来表示的。

边界表示的一个重要特征是描述形体的信息包括几何信息（Geometry）和拓朴信息（Topology）两个方面。拓朴信息描述形体上的顶点、边、面的连接关系，它形成物体边界表示的“骨架”。形体的几何信息犹如附着在“骨架”上的肌肉。例如，形体的某个面位于某一个曲面上，定义这一曲面方程的数据就是几何信息。此外，边的形状、顶点在三维空间中的位置（点的坐标）等都是几何信息，一般来说，几何信息描述形体的大小、尺寸、位置和形状等。

在边界表示法中，边界表示就按照体－面－环－边－点的层次，详细记录构成形体的所有几何元素的几何信息及其相互连接的拓朴关系。这样，在进行各种运算和操作中，就可以直接取得这些信息。

拓朴是指一个模型中的不同实体之间的关系，它描述了几何实体之间的连接方式。拓朴定义了一个空间位置不固定的浮动模型。当拓朴实体与几何信息关联在一起时，它的空间位置才确定。

拓朴可以是有边界的、没边界的和半封闭的，它允许实体是完全实体，也可以是不完全实体。例如，实体可以没有面，面可以没有边，实体也可以从内部将它分割成壳的内部面。这种实体在物理世界中是不存在的，但在几何造型内核中可以表现出来。

二、ACIS 中的拓朴结构 Topology of ACIS

ACIS 模型的边界表示（B-Rep）是将模型的拓朴结构按层次分解成下述对象：

1. 体（Body）：是实体对象的最高层次，是块（lump）的集合。体可以是线、面、或实心体；
2. 块（Lump）：空间一维、二维或三维点连接而成的集合，与其他块（lump）不关联，其边界由壳（shell）组成；
3. 壳（Shell）：互联的线或面的集合，它可以界定实体的外部或内部区域；
4. 子壳（Subshell）：壳的进一步分解，用于处理内部处理算法的效率；
5. 面（Face）：被一个或多个边（edge）组成的环（loop）界定的曲面中的连通域。面可以是“双向”的，这时它的厚度趋于无穷小。面也可以是“单向”的，这时面的法线指向面的外部，另一边侧是面的内部；
6. 环（Loop）：面（face）的边界中互相连接的部分，它由一系列的有向边（coedge）组成。通常环是封闭的，没有实际的开始和结束点，但是 ACIS 中的环可以是开环；
7. 线（Wire）：没有附着在面上的，连接在一起的有向边（coedge）；
8. 有向边（Coedge）：表示面（face）或线（wire）中对某个边的引用；
9. 边（Edge）：与曲线关联的拓朴，由顶点（vertex）界定。
10. 顶点（Vertex）：点是几何造型中的最基本元素。用计算机存储、管理、输出形体的实质就是对点集及其连接关系的处理。

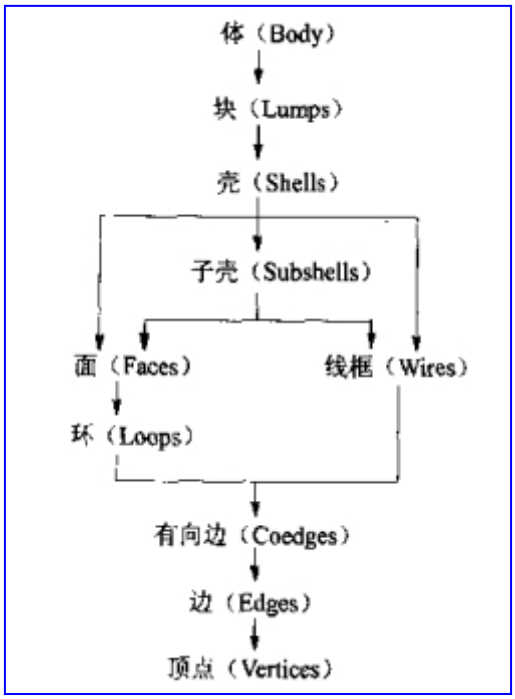


Figure 2.1 Topology of ACIS

上图说明了概念上的拓朴对象之间的关系，这些对象组成了 ACIS 边界表示方法的基础。它们在 ACIS 中分别用类 BODY、LUMP、SHELL、SUBSHELL、FACE、LOOP、WIRE、COEDGE、EDGE 和 VERTEX 实现，这些类派生于类 ENTITY。

ACIS 通过在它的数据结构中整合了线框、曲面和实体这三种表示方法，将这三种不同的几何体联系在一起。线框实体可以和实体（solid）与面实体共享，它们可以是共享边、有向边与顶点。由于这种共存的实现，使 ACIS 具有了表示混合维度模型与各种各样不封闭模型的能力，如一个平面可以只在 3 个方向上有边界边，另外一方向没有边界。

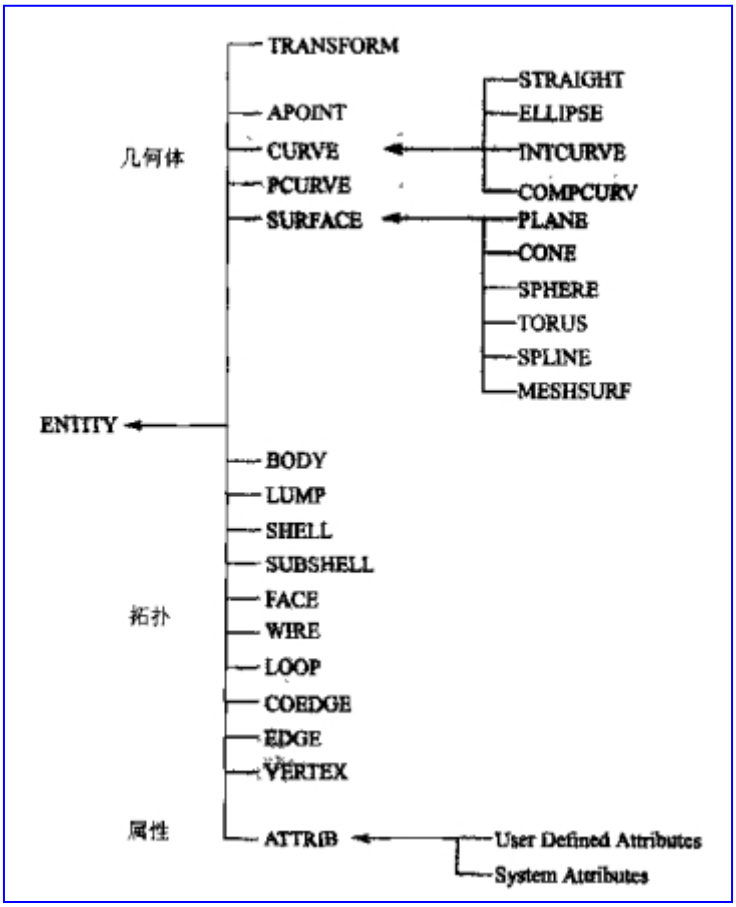


Figure 2.2 Class Diagram of ACIS Topology

ACIS 的对象都有包围盒（Bound），这在求交运算中很有用，可以提高效率。如果两个对象的包围盒相交，那么这两个对象则可能相交，然后再执行更精确的求交运算。如果两个对象的包围盒不相交，则这两个对象一定不相交，这样进一步的精确求交运算就不需要了。

既然拓朴表示了各对象之间的连接关系，那么给定一个对象时，可以容易得到其相连接其它对象。图 3 说明了 ACIS 中组成实体、面、线和混合体的所有实体类，这些类及其方法提供了一个边界表示造型器所必需的数据和方法。层次关系中向上和向下的指示说明这些类之间允许数据的快速切换，利用这种功能就可以确定两个实体是否共享边或顶点。从图中可以看出拓朴类都有指向对应的几何体类的指针。

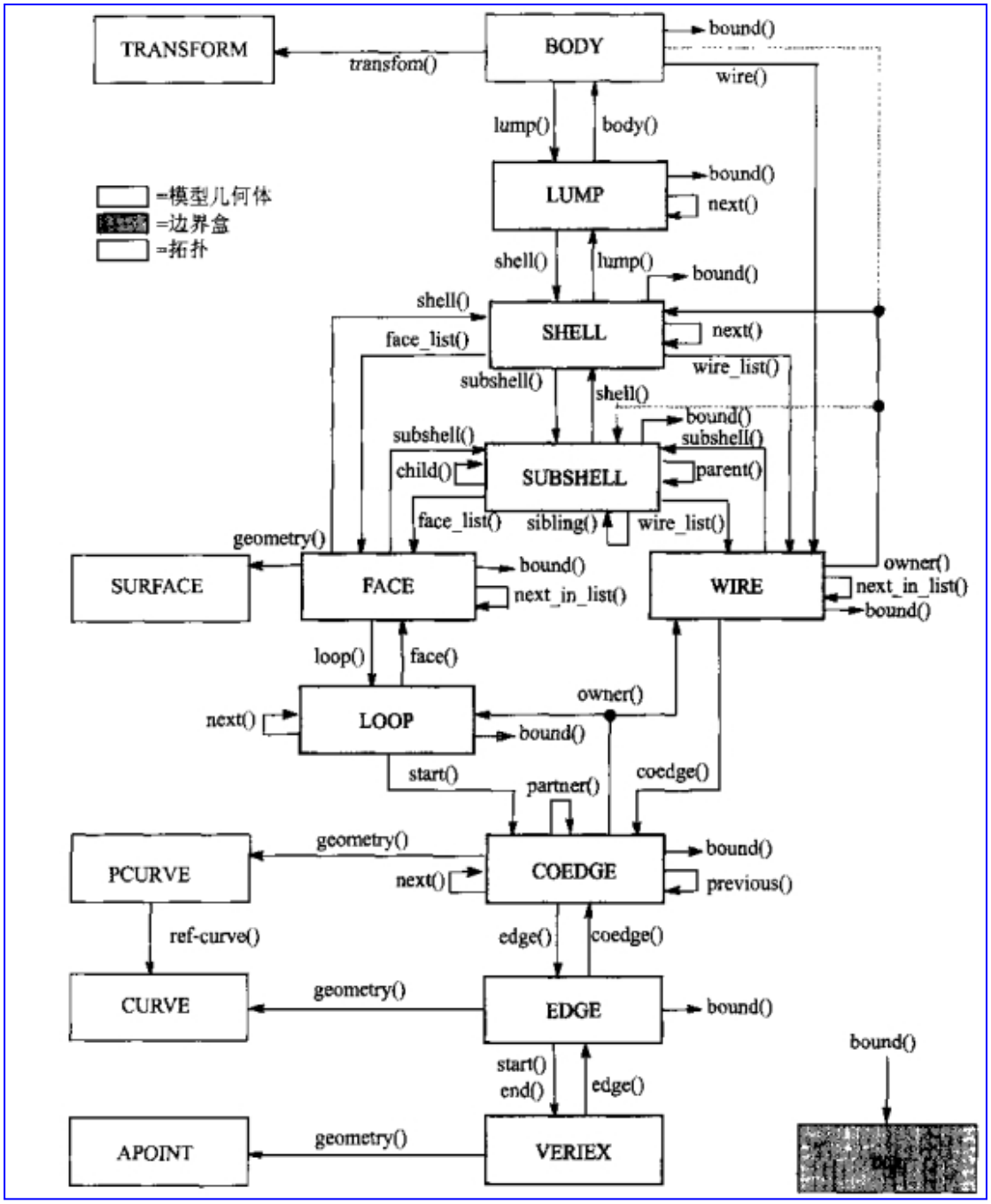


Figure 2.3 Topology structure of ACIS

从上图可以看出，任意给定一个对象，可以快速获得与其相连的其它对象，不管是向下还是向上。如给定一个 FACE 对象，可以通过 loop()向下获得 LOOP 对象；可以通过 shell()向上获得 SHELL 对象。

除了顶点以外，其他拓朴对象都有 bound()，可以取得其包围盒。

下面是统计一个模型中所有面的数量的程序，该程序就是利用拓朴类的公共成员函数来完成面的统计功能。通过这个程序来说明函数的使用方法。

```
//该程序先产生一个立方体，然后计算该立方体中面的总数
#include <stdio.h>                    //含有输入/输出函数的声明
#include "constrct/kernapi/cstrapi.hxx"   //声明了构造 API 函数
#include "kernel/kernapi/api/apimsc.hxx"  //声明了与拓扑有关的 API 函数
#include "kernel/kerndata/top/alltop.hxx" //声明了所有拓扑类
void main()
{
    api_start_modeler(0);
    api_initialize_constructors();
    BODY *block;
    api_make_cuboid(100,150,200,block);
    FACE *ff = block->lump()->shell()->face();  //调用类的直接接口
    int count=0;
    while(ff!=NULL)
    {
       count++;
       ff=ff->next();
    }
    printf("The block has %d faces/n",count);
    api_terminate_constructors();
    api_stop_modeler(0);
}
```

三、OpenCascade 中的拓朴结构 Topology of OpenCascade

3.1 OpenCascade 拓朴简介 Introduction of OpenCascade Topology

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

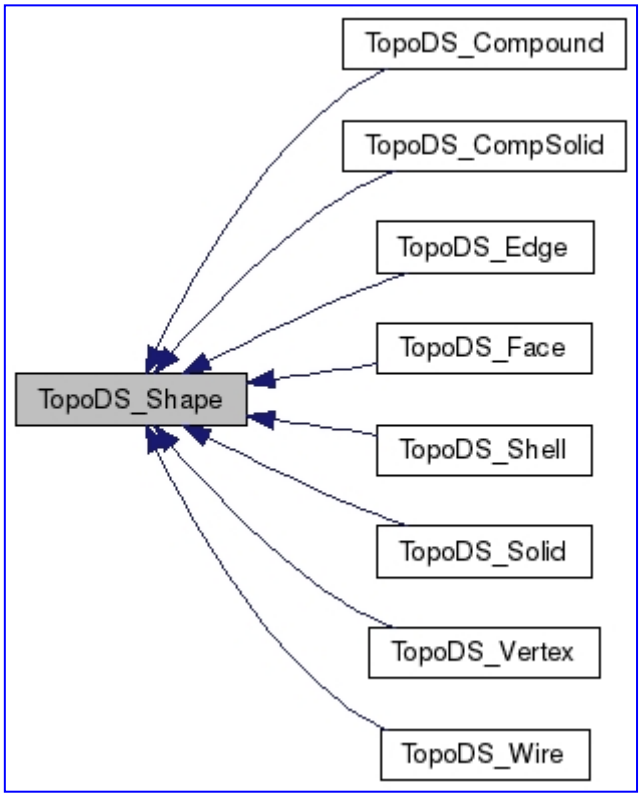
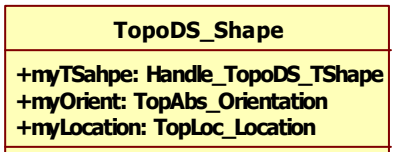


Figure 3.1 Topology data structure in OpenCascade



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

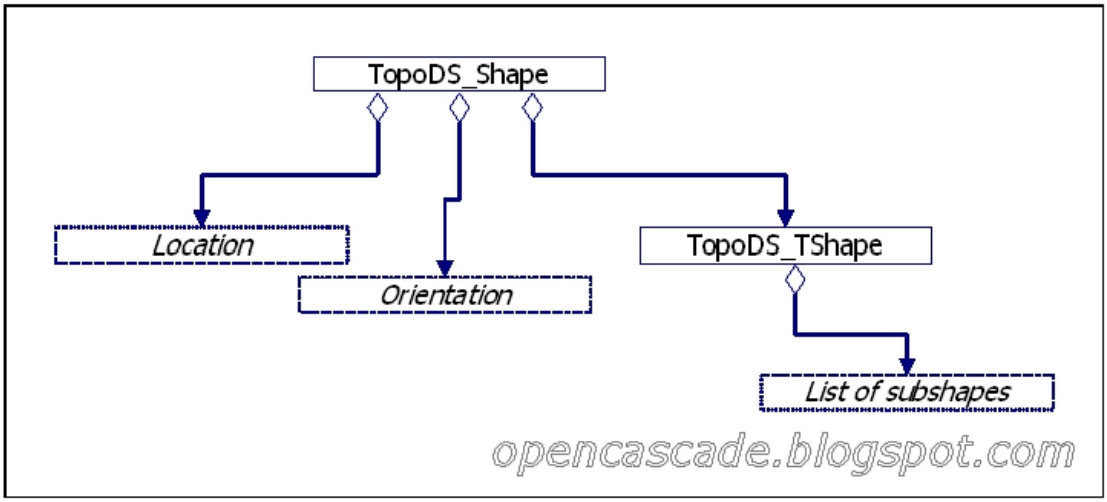


Figure 3.2 TopoDS_Shape member fields

其中 TopoDS_TShape 中包含与此对象相连接的**子对象**。

下图所示为由一条边连接的两个面组成的壳（shell）：

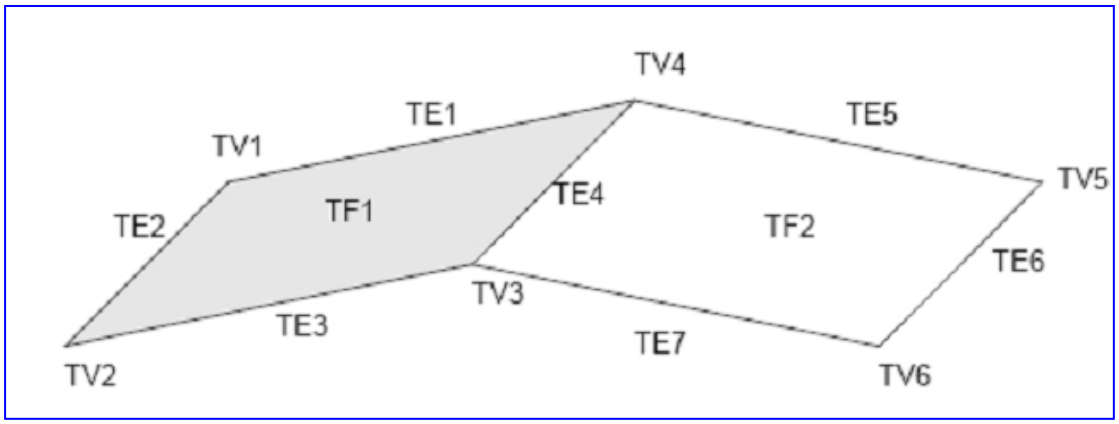


Figure 3.3 Structure of a shell formed from two faces

上图所示的形状表示为 TS，面 TF1 和 TF2，有七条边 TE1~TE7 和六个顶点 TV1~TV6。环 TW1 引用边 TE1~TE4；环 TW2 引用 TE4~TE7 。边引用的顶点如下：TE1（TV1，TV4），TE2（TV1，TV2），TE3（TV2，TV3），TE4（TV3，TV4），TE5（TV4，TV5），TE6（TV5，TV6），TE7（TV3，TV6）。

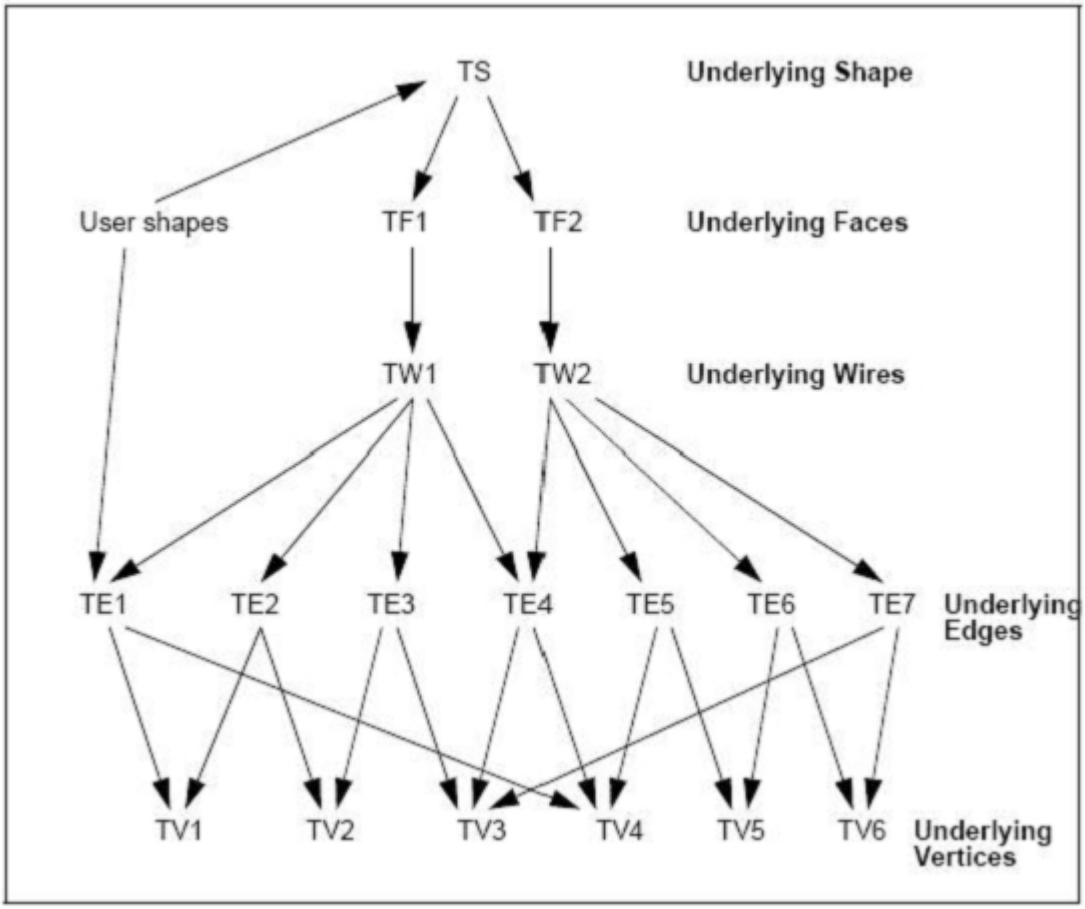


Figure 3.4 Data structure of the shell formed from two faces connected at an edge

注：OpenCascade 中的这个数据结构中不包含“反向引用（back references）”，即所有的引用只从复杂形状到简单形状。（Note that this data structure does not contain any “back references”. All references go from more comples underlying shapes to less complex ones.）有点有向图的意思。

这个根据 OpenCascade 的拓朴结构的类图可知，访问形状的子对象是很容易的。为了获得一个对象的拓朴，不管是向上还是向下，OpenCascade 提供了专门类和函数来实现。当向下访问拓朴对象时，OpenCascacde 提供了两种方法来遍历子对象。向上遍历时，OpenCascade 提供了一个静态函数 TopExp::MapShapesAndAncestors()来实现。以下分别对其进行说明。

3.2 遍历子对象 Iteration over Children

遍历子对象是向下来访问拓朴关系。OpenCascade 中遍历一个形状的子对象的两个方法分别为：

- 直接使用类 TopoDS_Iterator 来遍历：

Direct children can be retrieved using TopoDS_Iterator，如下函数可以访问当前对象所有的子对象，不管子对象的类型。通过递归的方式来实现。

```
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，用于控制在查询子对象时设定**是否考虑父对象的位置和朝向（location and orientation）**。若位置（location）标志设置为开，那么所有子对象返回的值就像它们是独立的对象一样，而且位置是在具有全局坐标系的三维空间中的位置。（例如用户将看到单独取出来的边 edge 与其父对象环 wire 中显示的位置是一样的。）若朝向（orientation）标设置为开，则返回的子对象的朝向将会变成父对象的朝向与子对象朝向的乘积（例如，子对象和父对象两者都是反向 reversed 或是向前 forward，则结果仍然向前。向前与反向的任意组合结果都将为反向）。

若标志位为关，则子对象就只返回其自身保存的位置和朝向。默认情况下，两个标志位都设置为开。

- 使用类 TopExp_Explorer 来遍历指定类型的子对象：

若只想访问形状指定类型的子对象，可以使用类 TopExp_Explorer 来实现。如下程序所示为访问对象的边的功能。

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

类 TopExp_Explorer 还有一个附加参数，可以用来指定要跳过的父对象类型。这个参数在下面的情况下很有用。例如只想取得“悬空”边（floating edge, 即不属于面的边），就可用下面代码来实现：

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

3.3 反向引用 Back references

在使用 OpenCascade 时，你可能也注意到了，或者根据类图分析到拓朴对象包含其子对象，而不是相反的方式。这是可以理解的，同一个对象或者子对象可以属于不同的对象。例如任意共享边可以属于至少两个面。然而有时也需要从子对象追踪到与其相连的父对象。在 OpenCascade 中提供了静态函数 TopExp::MapShapesAndAncestors()来实现这个功能。

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

上面的代码生成了 myShape 中面和边之间的映射。若 myShape 是长方体，每一条边会映射到两个面上。若遍历同样的长方体，并在每一个面中尽力找到边的父对象，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。

四、**示例程序 Example Code**

1. 统计一个长方体面的数量

```
/*
*       Copyright (c) 2013 eryar All Rights Reserved.
*
*            File       : Main.cpp
*            Author     : eryar@163.com
*            Date       : 2013-09-21 11:58
*            Version    : 1.0v
*
*       Description : Count faces of a box.
*
*/

// OpenCascade library.
#define WNT
#include <BRepPrimAPI_MakeBox.hxx>
#include <TopExp_Explorer.hxx>

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

int main(void)
{
	Standard_Integer nFaceCount = 0;
	TopoDS_Shape aBox = BRepPrimAPI_MakeBox(100, 150, 200);

	for (TopExp_Explorer faceExp(aBox, TopAbs_FACE); faceExp.More(); faceExp.Next())
	{
		nFaceCount++;
	}

	std::cout << "The box has " << nFaceCount << " faces." << std::endl;

	return 0;
}
```

程序输出结果为：

```
The box has 6 faces.
```

2. 反向访问

```
/*
*    Copyright (c) 2013 eryar All Rights Reserved.
*
*        File    : Main.cpp
*        Author  : eryar@163.com
*        Date    : 2013-09-21 11:58
*        Version : 1.0v
*
*    Description : Demonstrate how to access parent and child topology data
*       for a given topology shape.
*
*/

// OpenCascade library.
#define WNT
#include <BRepPrimAPI_MakeBox.hxx>
#include <TopExp_Explorer.hxx>
#include <TopoDS.hxx>
#include <TopExp.hxx>
#include <TopTools_ListOfShape.hxx>
#include <TopTools_ListIteratorOfListOfShape.hxx>
#include <TopTools_IndexedDataMapOfShapeListOfShape.hxx>

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

void dumpVertex(const TopoDS_Vertex& vertex)
{
    gp_Pnt pnt = BRep_Tool::Pnt(vertex);

    std::cout << "(" << pnt.X() << ", " << pnt.Y() << ", " << pnt.Z() << ")" <<
std::endl;
}

int main(void)
{
    TopoDS_Shape aBox = BRepPrimAPI_MakeBox(100, 150, 200);

    TopTools_IndexedDataMapOfShapeListOfShape shapeMap;
    TopTools_ListOfShape edges;
    TopTools_ListIteratorOfListOfShape edgeItr;

    // Use TopExp_Explorer to access subshapes.
    TopExp_Explorer vertexExp(aBox, TopAbs_VERTEX);

    const TopoDS_Vertex& aVertex = TopoDS::Vertex(vertexExp.Current());
```

```
    // Use TopExp::MapShapesAndAncestors() to access parent shapes.
    TopExp::MapShapesAndAncestors(aBox, TopAbs_VERTEX, TopAbs_EDGE, shapeMap);

    edges = shapeMap.FindFromKey(aVertex);

    dumpVertex(aVertex);

    for (edgeItr.Initialize(edges); edgeItr.More(); edgeItr.Next() )
    {
        const TopoDS_Edge& anEdge = TopoDS::Edge(edgeItr.Value());

        std::cout << "Vertex belong to the Edge: " << std::endl;
        dumpVertex(TopExp::FirstVertex(anEdge));
        dumpVertex(TopExp::LastVertex(anEdge));
        std::cout << "--------------------------" << std::endl;
    }

    return 0;
}
```

程序输出结果：

```
(0, 0, 200)
Vertex belong to the Edge:
(0, 0, 0)
(0, 0, 200)
--------------------------
Vertex belong to the Edge:
(0, 0, 200)
(0, 150, 200)
--------------------------
Vertex belong to the Edge:
(0, 0, 200)
(100, 0, 200)
--------------------------
Vertex belong to the Edge:
(0, 0, 0)
(0, 0, 200)
--------------------------
Vertex belong to the Edge:
(0, 0, 200)
(0, 150, 200)
--------------------------
Vertex belong to the Edge:
(0, 0, 200)
(100, 0, 200)
--------------------------
```

五、结论 Conclusion

OpenCascade 中的拓朴关系不像 ACIS 中那样直接，但是也提供了向下访问子形状、向上访问父形状的类和函数，使用起来要涉及到好几个类，不是很方便。

当向上访问与顶点相连接的边时，有重复数据。

六、参考资料 References

1. Roman Lygin, OpenCascade notes, opencascade.blogspot.com
2. 詹海生等, 基于 ACIS 的几何造型技术与系统开发, 清华大学出版社, 2002
3. 孙家广等. 计算机图形学. 清华大学出版社
4. OpenCascade source code.