Topology Shapes of OpenCascade BRep

eryar@163.com

摘要 Abstract：通过对 OpenCascade 中的 BRep 数据的读写，理解边界表示法的概念及实现。理解了拓朴形状的数据结构，就对 ModelingData 模块有了清晰认识，方便 OpenCascade 其他模块如 ModelingAlgorithms 和 Visiualization 模块的理解。

关键字 Key Words：OpenCascade, BRep, Topology, BRep Format

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表示，而每个面又由它所在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。如下图 1.1 所示，曲面的汇合处形成曲线，而曲线的汇合处形成点。所以点、线、面是描述一个形状所需要的基本组成单元。

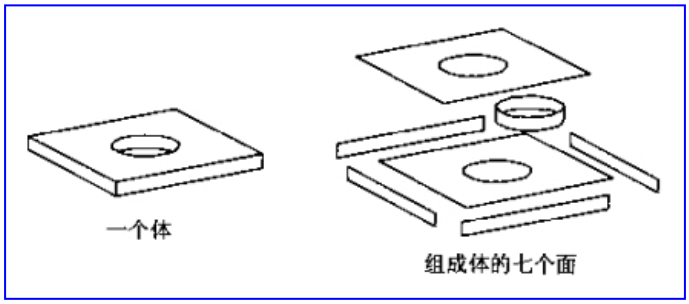

Figure 1.1 BRep Shape demo

边界表示的一个重要特点是描述形状的信息包括几何信息（geometry）和拓朴（topology）信息两个方面。拓朴信息描述形状上的顶点、边、面的连接关系，它形成物体边界表示的“骨架”。形状的几何信息犹如附着在“骨架”上的肌肉。在 OpenCascade 中，形状的几何信息包含曲线和曲面的参数解析表示 Geom_Curve/Geom_Surface。

这样我们就可以用平面方程和柱面方程来描述曲面，用直线或圆弧方程来描述曲线。这时会出现一个问题，即代数表达式只能定义无边界的几何体。除了单个点、圆以及球体，经典的解析几何仅能表示无限延伸的曲线和曲面。为了解决这个问题，边界表示法按下述方法明确地定义曲线或曲面的边界：

- 曲线的边界由位于曲线上的一对点来确定；
- 曲面的边界由位于曲面上的一组曲线来确定；

通过这个方法，就可以定义一段曲线或一片曲面。这时，不同几何元素之间的关系的组织问题就出现了，为此我们将记录如下信息：

- 哪些点界定哪些曲线；
- 哪些曲线界定哪些曲面；

这些关于谁关联谁的信息，就是几何造型系统经常提到的拓朴。在边界表示法中，理论上表示一个物理模型只需要三个拓朴体（顶点 TopoDS_Vertex、边 TopoDS_Edge 和面 TopoDS_Face），但在实际应用中，为了提高计算机处理的速度或提供高级的操作功能，还要引入其他一些概念，如环 TopoDS_Wire、壳 TopoDS_Shell、复合体 TopoDS_Compound 等。

二、边界表示形状中的几何数据 Geometry of BRep shapes

对形状数据的读写主要是由类 BRepTools_ShapeSet 来完成的，其中在类的函数 AddGeometry 中对拓朴形状中的几何数据进行了处理，代码如下所示：

```
//=======================================================================
//function : AddGeometry
//purpose  :
//=======================================================================
void BRepTools_ShapeSet::AddGeometry(const TopoDS_Shape& S)
{
  // Add the geometry

  if (S.ShapeType() == TopAbs_VERTEX) {

    Handle(BRep_TVertex) TV = Handle(BRep_TVertex)::DownCast(S.TShape());
    BRep_ListIteratorOfListOfPointRepresentation itrp(TV->Points());

    while (itrp.More()) {
      const Handle(BRep_PointRepresentation)& PR = itrp.Value();

      if (PR->IsPointOnCurve()) {
        myCurves.Add(PR->Curve());
      }

      else if (PR->IsPointOnCurveOnSurface()) {
        myCurves2d.Add(PR->PCurve());
        mySurfaces.Add(PR->Surface());
      }

      else if (PR->IsPointOnSurface()) {
        mySurfaces.Add(PR->Surface());
      }

      ChangeLocations().Add(PR->Location());
      itrp.Next();
    }

  }
  else if (S.ShapeType() == TopAbs_EDGE) {

    // Add the curve geometry
    Handle(BRep_TEdge) TE = Handle(BRep_TEdge)::DownCast(S.TShape());
    BRep_ListIteratorOfListOfCurveRepresentation itrc(TE->Curves());

    while (itrc.More()) {
      const Handle(BRep_CurveRepresentation)& CR = itrc.Value();
      if (CR->IsCurve3D()) {
        if (!CR->Curve3D().IsNull()) {
          myCurves.Add(CR->Curve3D());
          ChangeLocations().Add(CR->Location());
        }
      }
      else if (CR->IsCurveOnSurface()) {
        mySurfaces.Add(CR->Surface());
        myCurves2d.Add(CR->PCurve());
```

```
            ChangeLocations().Add(CR->Location());
            if (CR->IsCurveOnClosedSurface())
              myCurves2d.Add(CR->PCurve2());
          }
          else if (CR->IsRegularity()) {
            mySurfaces.Add(CR->Surface());
            ChangeLocations().Add(CR->Location());
            mySurfaces.Add(CR->Surface2());
            ChangeLocations().Add(CR->Location2());
          }
          else if (myWithTriangles) { // for XML Persistence
            if (CR->IsPolygon3D()) {
              if (!CR->Polygon3D().IsNull()) {
                myPolygons3D.Add(CR->Polygon3D());
                ChangeLocations().Add(CR->Location());
              }
            }
            else if (CR->IsPolygonOnTriangulation()) {
              myTriangulations.Add(CR->Triangulation());
              myNodes.Add(CR->PolygonOnTriangulation());
              ChangeLocations().Add(CR->Location());
              if (CR->IsPolygonOnClosedTriangulation())
                myNodes.Add(CR->PolygonOnTriangulation2());
            }
            else if (CR->IsPolygonOnSurface()) {
              mySurfaces.Add(CR->Surface());
              myPolygons2D.Add(CR->Polygon());
              ChangeLocations().Add(CR->Location());
              if (CR->IsPolygonOnClosedSurface())
              myPolygons2D.Add(CR->Polygon2());
            }
          }
          itrc.Next();
        }
      }

      else if (S.ShapeType() == TopAbs_FACE) {

        // Add the surface geometry
        Handle(BRep_TFace) TF = Handle(BRep_TFace)::DownCast(S.TShape());
        if (!TF->Surface().IsNull())    mySurfaces.Add(TF->Surface());

        if (myWithTriangles) { // for XML Persistence
          Handle(Poly_Triangulation) Tr = TF->Triangulation();
          if (!Tr.IsNull()) myTriangulations.Add(Tr);
        }

        ChangeLocations().Add(TF->Location());
      }
    }
```

根据上述代码可知，OpenCascade 在保存和读写 BRep 表示的形状时，只保存了顶点、边和面的信息，因为只有这三个拓朴结构中包含了几何信息及显示用的离散点和三角网格数据。有了这些信息，就可以生成一个边界表示的形状了。

几何之间的联系也保存起来了，这也是拓朴数据的一种形式，在下节详细说明。

三、边界表示形状中的拓朴数据 Topology of BRep shapes

关于拓朴顶点 TopoDS_Vertex、边 TopoDS_Edge、面 TopoDS_Face 更详细的信息，请参考博客：

- **Topology and Geometry in OpenCascade-Vertex;**
- **Topology and Geometry in OpenCascade-Edge;**
- **Topology and Geometry in OpenCascade-Face;**
- **Topology and Geometry in OpenCascade-Topology;**

本文只对 OpenCascade 拓朴结构中的几何数据的关联信息进行分析。

3.1 顶点 TopoDS_Vertex

结合《BRep Format Description White Paper》中对<vertex data>的描述，及程序代码中对顶点数据的读取，分析 OpenCascade 的 BRep 表示中的顶点。

BNF-like Definition

<vertex data> = <vertex data tolerance> <_\n> <vertex data 3D representation> <_\n> <vertex data representations>;

<vertex data tolerance> = <real>;

<vertex data 3D representation> = <3D point>;

<vertex data representations> = (<vertex data representation> <_\n>)* "0 0";

<vertex data representation> = <vertex data representation u parameter> <_>
<vertex data representation data> <_> <location number>;

<vertex data representation u parameter> = <real>;

<vertex data representation data> =
("1" <_> <vertex data representation data 1>) |
("2" <_> <vertex data representation data 2>) |
("3" <_> <vertex data representation data 3>);

<vertex data representation data 1> = <3D curve number>;

<vertex data representation data 2> = <2D curve number> <_> <surface number>;

<vertex data representation data 3> =
<vertex data representation v parameter> <_> <surface number>;
<vertex data representation v parameter> = <real>;

Figure 3.1.1 NBF-like definition of Vertex

详细说明：

<vertex data representation u parameter>u 的使用方法说明如下：

<vertex data representation data 1> 和参数 u 定义了三维曲线 C 上的点 V 的位置。参数 u 是曲线 C 上点 V 对应的参数：C（u）＝V。对应的类是：BRep_PointOnCurve；

<vertex data representation data 2>和参数 u 定义了曲面上的二维曲线 C 上点 V 的位置。参数 u 是曲线 C 上点 V 对应的参数：C(u)＝V。对应的类是：BRep_PointOnCurveOnSurface；

<vertex data representation data 3>和参数 u 及<vertex data representation v parameter>v 定义了曲面 S 上的点 V：S（u，v）＝V。对应的类是：BRep_PointOnSurface；

在这些类中都将顶点对应的曲线、曲面及其上点的参数都保存起来了。有了这些信息就

可以判断与顶点有联系的边或面，因为曲线、曲面属于边和面。

<vertex data tolerance>t 定义如下所示：

$$\max_{P \in R} |P - V| \le t .$$

下面结合程序示例片段，创建一个顶点并将其输出为 BRep 文件，并在 OpenCascade 中进行显示。

```
void TestVertex(void)
{
	ofstream dumpFile("vertex.brep");

	TopoDS_Vertex aVertex = BRepBuilderAPI_MakeVertex(gp_Pnt(1.0, 2.0, 3.0));

	BRepTools::Dump(aVertex, std::cout);
	BRepTools::Write(aVertex, dumpFile);
}
```

当使用 BRepTools::Dump 时，显示更易读的信息。可将数据 dump 到屏幕，也可将数据 dump 到文件；当使用 Write 时，生成的信息即是 BRep 文件格式的数据。可以通过 Read 生成形状，也可直接读入到 OpenCascade 中显示，如下图 3.1.2 所示：

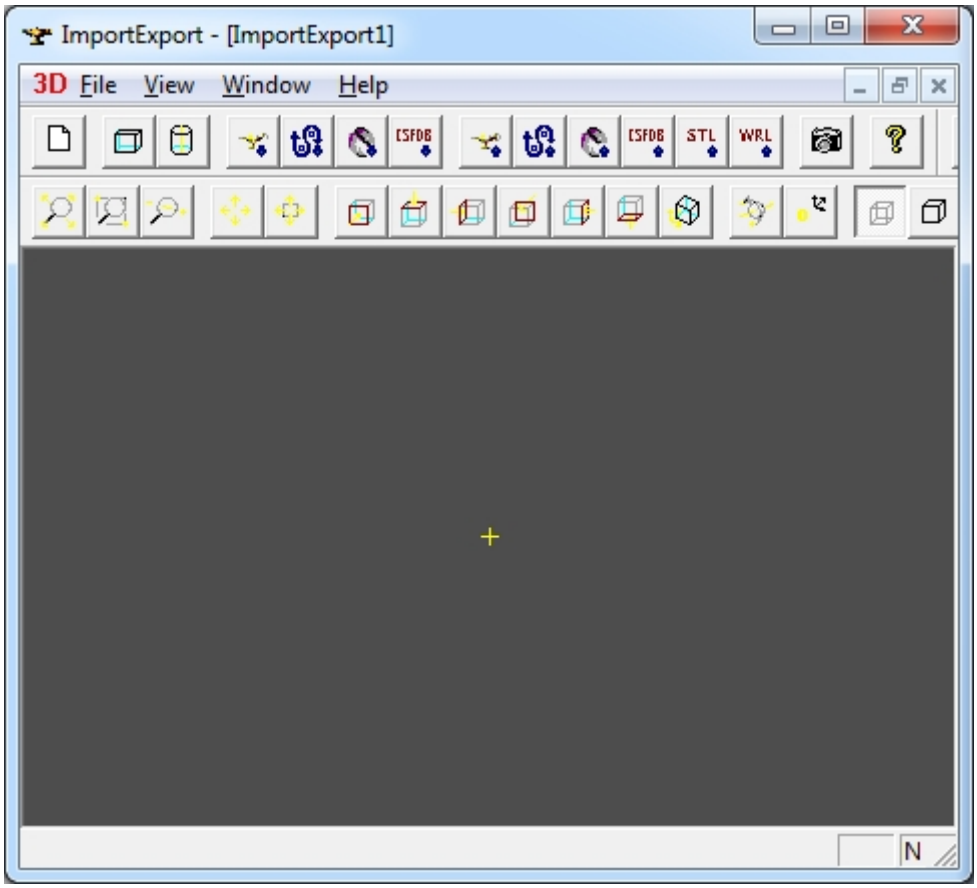

Figure 3.1.2 Import a Vertex from brep file

3.2 边 TopoDS_Edge

BNF-like Definition

<edge data> = <_> <edge data tolerance> <_> <edge data same parameter flag> <_> edge data same range flag> <_> <edge data degenerated flag> <_\n> <edge data representations>;

<edge data tolerance> = <real>;

<edge data same parameter flag> = <flag>;

<edge data same range flag> = <flag>;

<edge data degenerated flag> = <flag>;

<edge data representations> = (<edge data representation> <_\n>)* "0";

<edge data representation> =
"1" <_> <edge data representation data 1>
"2" <_> <edge data representation data 2>
"3" <_> <edge data representation data 3>
"4" <_> <edge data representation data 4>
"5" <_> <edge data representation data 5>
"6" <_> <edge data representation data 6>
"7" <_> <edge data representation data 7>;

<edge data representation data 1> = <3D curve number> <_> <location number> <_>
<curve parameter minimal and maximal values>;
<edge data representation data 2> = <2D curve number> <_> <surface number> <_>
<location number> <_> <curve parameter minimal and maximal values>
[<_\n> <curve values for parameter minimal and maximal values>];

<edge data representation data 3> = (<2D curve number> <_>) ^ 2 <continuity order> <_>
<surface number> <_> <location number> <_> <curve parameter minimal and maximal values>
<\n> <curve values for parameter minimal and maximal values>];

<continuity order> = "C0" | "C1" | "C2" | "C3" | "CN" | "G1" | "G2".

<edge data representation data 4> =
<continuity order> (<_> <surface number> <_> <location number>) ^ 2;

<edge data representation data 5> = <3D polygon number> <_> <location number>;

<edge data representation data 6> =
<polygon on triangulation number> <_> <triangulation number> <_> <location number>;

<edge data representation data 7> = (<polygon on triangulation number> <_>) ^ 2
<triangulation number> <_> <location number>;

详细说明：

标志位<edge data same parameter flag>，<edge data same range flag>，<edge data degenerated flag>有特别的用途。

<edge data representation data 1>表示一个三维曲线，对应类：Geom_Curve；

<edge data representation data 2>表示曲面上的一个二维曲线，

对应类 Geom_Curve/Geom_Surface；

<curve values for parameter minimal and maximal values>只在 2 版本中使用；

<edge data representation data 3>表示闭合曲面上的一个二维曲线；

对应类 Geom_Curve/Geom_Surface；

<curve values for parameter minimal and maximal values>只在 2 版本中使用；

<edge data representation data 4>表示 Regularity 的边，使用到的类有：

Geom_Curve/Geom_Surface；

<edge data representation data 5>表示一个三维的多段线（3D polyline）；

对应的类：Poly_Polygon3D，是边的近似表示，主要用来显示；

<edge data representation data 6>表示三角剖分上一条多段线；

对应的类：Poly_PolygonOnTriangulation，也是边在三角剖分上的近似表示；

<edge data tolerance> t 的定义如下所示：

$$\max_{C\in R}\max_{P\in E}\min_{Q\in C}|Q-P|\le t\,.$$

下面的示例程序片段将圆的边导出为 BRep 文件，并在 OpenCascade 中显示。程序代码如下所示：

```
void TestEdge(bool bSubdivision = false)
{
	ofstream dumpFile("edge.brep");

	TopoDS_Edge anEdge = BRepBuilderAPI_MakeEdge(gp_Circ(gp::XOY(), 6.0));

	if (bSubdivision)
	{
		BRepMesh::Mesh(anEdge, 1.0);
	}

	BRepTools::Dump(anEdge, std::cout);
	BRepTools::Write(anEdge, dumpFile);
}
```

其中参数 bSubdivision 用来生成显示用的离散多段线数据，这里会生成<edge data representation data 5>的 Poly_Polygon3D，将生成的 BRep 文件导入进行显示如下图所示：

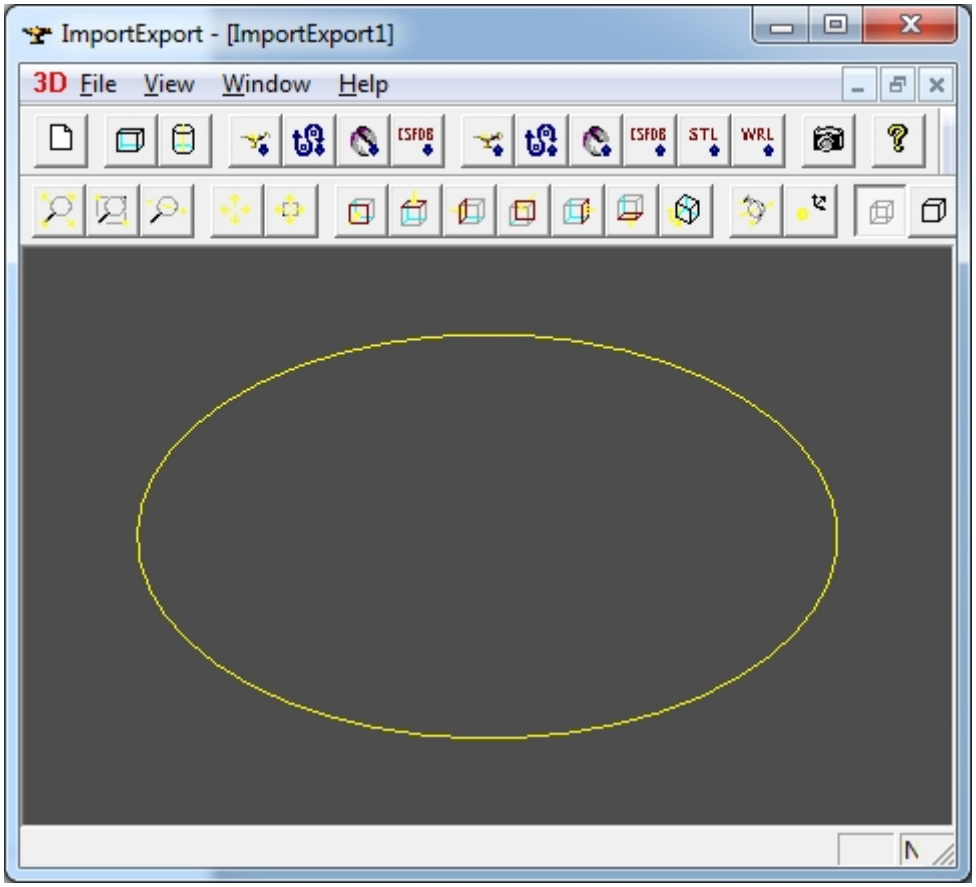

Figure 3.2.1 Import a Face from brep file

3.3 面 TopoDS_Face

BNF-like Definition

<face data> = <face data natural restriction flag> <_> <face data tolerance> <_> <surface number> <_> <location number> <\n> [“2” <_> <triangulation number>];

<face data natural restriction flag> = <flag>;

<face data tolerance> = <real>;

详细说明：

<face data>描述了面 F 的曲面 S 和三角剖分 T。曲面 S 可能为空：<surface number>＝0.

<face data tolerance> t 的定义如下所示：

$$\max_{P\in F}\min_{Q\in S}|Q-P|\le t\,.$$

标志位<face data natural restriction flag>有特别的用途。

面中的数据比较简单，有参数表示的曲面的索引号。若曲面已经被三角剖分，将会把剖分后的网格数据也保存起来。下面的示例程序片段将一个球面导出为 brep 文件：

```
void TestFace(bool bSubdivision = false)
{
	ofstream dumpFile("face.brep");

	TopoDS_Face aFace = BRepBuilderAPI_MakeFace(gp_Sphere(gp::XOY(), 6.0));

	if (bSubdivision)
	{
		BRepMesh::Mesh(aFace, 1.0);
	}

	BRepTools::Dump(aFace, std::cout);
	BRepTools::Write(aFace, dumpFile);
}
```

其中参数 bSubdivision 用来生成显示用的网格数据，这里会生成 Poly_Triangulation，将生成的 BRep 文件导入进行显示如下图所示：

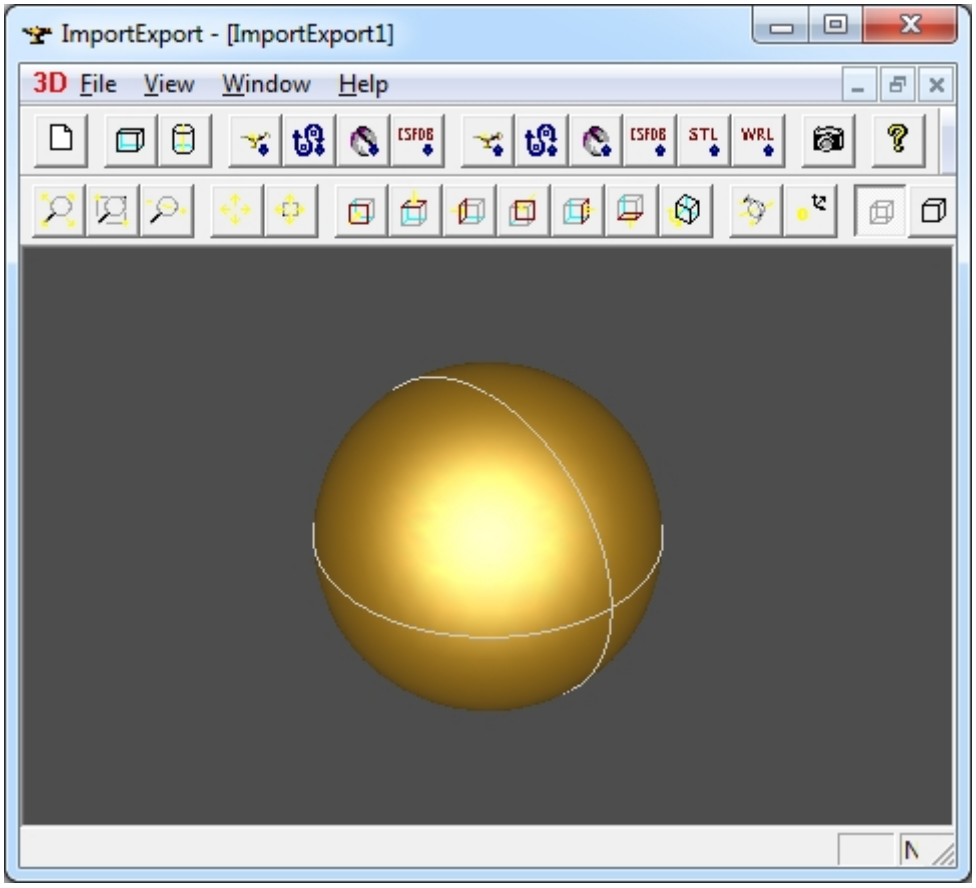

Figure 3.3.1 Import a Face from brep

四、示例程序 Example Code

将上述代码放在一起，完整的程序代码如下所示：

```
/*
*       Copyright (c) 2013 eryar All Rights Reserved.
*
*                  File : Main.cpp
*                Author : eryar@163.com
*                  Date : 2013-12-21 21:18
*               Version : 1.0v
*
*       Description : Use BRepTools to dump and write BRep files.
*
*          Key Words : OpenCascade, BRep, Vertex, Edge, Face
*
*/

// OpenCascade library.
#define WNT
#include <gp_Pnt.hxx>
#include <gp_Circ.hxx>
#include <gp_Sphere.hxx>

#include <TopoDS_Vertex.hxx>
#include <TopoDS_Edge.hxx>
#include <TopoDS_Face.hxx>

#include <BRepMesh.hxx>
#include <BRepTools.hxx>

#include <BRepBuilderAPI_MakeVertex.hxx>
#include <BRepBuilderAPI_MakeEdge.hxx>
#include <BRepBuilderAPI_MakeFace.hxx>

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


void TestVertex(void)
{
	ofstream dumpFile("vertex.brep");

	TopoDS_Vertex aVertex = BRepBuilderAPI_MakeVertex(gp_Pnt(1.0, 2.0, 3.0));

	BRepTools::Dump(aVertex, std::cout);
	BRepTools::Write(aVertex, dumpFile);
}


void TestEdge(bool bSubdivision = false)
{
	ofstream dumpFile("edge.brep");
```

```
	TopoDS_Edge anEdge = BRepBuilderAPI_MakeEdge(gp_Circ(gp::XOY(), 6.0));

	if (bSubdivision)
	{
		BRepMesh::Mesh(anEdge, 1.0);
	}

	BRepTools::Dump(anEdge, std::cout);
	BRepTools::Write(anEdge, dumpFile);
}


void TestFace(bool bSubdivision = false)
{
	ofstream dumpFile("face.brep");

	TopoDS_Face aFace = BRepBuilderAPI_MakeFace(gp_Sphere(gp::XOY(), 6.0));

	if (bSubdivision)
	{
		BRepMesh::Mesh(aFace, 1.0);
	}

	BRepTools::Dump(aFace, std::cout);
	BRepTools::Write(aFace, dumpFile);
}


int main(void)
{
	TestVertex();

	TestEdge(true);

	TestFace(true);

	return 0;
}
```

也可以把更易懂的信息 dump 出来，这样可以更好地理解 brep 文件的格式。当将球面导出为 brep 文件时，会生成很多信息，如下所示：

```
Shape : 7, FORWARD

Dump of 7 TShapes

-----------------

Flags : Free, Modified, Checked, Orientable, Closed, Infinite, Convex

TShape # 1 : FACE          1101000 01807868
	+2
NaturalRestriction
	Tolerance : 1e-007
	- Surface : 1
```

```
TShape # 2 : WIRE           0101100 01807CE8
      -5 +4 +5 -3

TShape # 3 : EDGE           0101000 01808C10
      +6 -6
      Tolerance : 1e-007
        same parametrisation of curves
        same range on curves
        degenerated
      - PCurve : 4 on surface 1, range : 0 6.28319
   UV Points : 0, 1.5708 6.28319, 1.5708

TShape # 4 : EDGE           0101000 01807AA0
      +7 -7
      Tolerance : 1e-007
        same parametrisation of curves
        same range on curves
        degenerated
      - PCurve : 3 on surface 1, range : 0 6.28319
   UV Points : 0, -1.5708 6.28319, -1.5708

TShape # 5 : EDGE           0101000 018078C8
      +7 -6
      Tolerance : 1e-007
        same parametrisation of curves
        same range on curves
      - Curve 3D : 1, range : -1.5708 1.5708
      - PCurve : 1, 2 (C0) on surface 1, range : -1.5708 1.5708
   UV Points : 6.28319, -1.5708 6.28319, 1.5708
   UV Points : 0, -1.5708 0, 1.5708

TShape # 6 : VERTEX       0101101 018076F0

      Tolerance : 1e-007
      - Point 3D : 3.67394e-016, 0, 6

TShape # 7 : VERTEX       0101101 01807680

      Tolerance : 1e-007
      - Point 3D : 3.67394e-016, 0, -6



 -------
Dump of 4 Curve2ds
 -------

    1 : Line
   Origin :6.28319, 0
   Axis      :0, 1

    2 : Line
   Origin :0, 0
   Axis      :0, 1

    3 : Line
   Origin :0, -1.5708
   Axis      :1, 0
```

```
    4 : Line
   Origin :0, 1.5708
   Axis      :1, 0


 -------
Dump of 1 Curves
 -------

    1 : Trimmed curve
Parameters : 4.71239 7.85398
Basis curve :
Circle
   Center :0, 0, 0
   Axis      :0, -1, 0
   XAxis    :1, 0, 0
   YAxis    :-0, 0, 1
   Radius :6

 -------
Dump of 0 Polygon3Ds
 -------
 -------
Dump of 0 PolygonOnTriangulations
 -------

 -------
Dump of 1 surfaces
 -------

    1 : SphericalSurface
   Center :0, 0, 0
   Axis      :0, 0, 1
   XAxis    :1, 0, -0
   YAxis    :-0, 1, 0
   Radius :6

 -------
Dump of 0 Triangulations
 -------




 -------
 Dump of 0 Locations
 -------
```

根据上面的数据，可以很好地理解 BRep 中拓朴形状的相关数据。大部分数据还是很直观，便于理解的。其中有个数据可能需要解释即 PCurve（Parametric Curve），它是在参数（u,v）空间的曲面上的参数曲线。可能有些不好理解，结合程序代码看下 PCurve 的使用，就会 Aha! 恍然大悟的：

```
//=======================================================================/
/function : D0
//purpose    :
//=======================================================================
void BRep_CurveOnSurface::D0(const Standard_Real U, gp_Pnt& P) const
```

```
{
  // shoud be D0 NYI
  gp_Pnt2d P2d = myPCurve->Value(U);
  P = mySurface->Value(P2d.X(),P2d.Y());
  P.Transform(myLocation.Transformation());
}
```

此函数的作用是求 PCurve 上对应参数 u 的曲面上的点，即 0 次微分 D0。根据 PCurve 上的一个参数 u，可以求出对应参数 u 的 PCurve 上的点，把这个点的 x,y 分别作为参数曲面的参数 u,v，即求出了曲面上的点。

五、结论 Conclusion

通过程序代码将《BRep Format Description White Paper》中数据进行读写，深入理解OpenCascade 的边界表示法的数据结构模块 ModelingData，为理解其他模块打下基础。

在边界表示法中，理论上表示一个物理模型只需要三个拓朴体（顶点 TopoDS_Vertex、边 TopoDS_Edge 和面 TopoDS_Face），所以在对 brep 文件输出时，只处理了这三种拓朴体的信息。在生成形状时，主要也是处理这三种拓朴体，再根据他们生成其他拓朴体。

顶点、边和面的几何之间的联系在 brep 中也保存起来了，有了这些信息，就可以判断一个顶点是不是边上的点等。通过示例程序，来理解参数曲线 PCurve。

理解了 brep 表示的结构后，下一步准备来研究下造型算法模块 ModelingAlgorithms。

六、参考资料 References

1. BRepTools_ShapeSet.cpp of OpenCascade
2. TopTools_ShapeSet.cpp of OpenCascade
3. BRepTools.cpp Of OpenCascade
4. BRep Format Description White Paper of OpenCascade
5. 孙家广等. 计算机图形学. 清华大学出版社, 2000
6. 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