OpenCascade Shape Representation in OpenSceneGraph

eryar@163.com

摘要 Abstract：本文通过程序实例，将 OpenCascade 中的拓朴数据（边、面）离散化后在 OpenSceneGraph 中进行显示。有了这些离散数据，就可以不用 OpenCascade 的显示模块了，可以使用其他显示引擎对形状进行显示。即若要线框模式显示形状时，就绘制离散形状拓朴边后得到的多段线；若要实体渲染模式显示形状时，就绘制离散形状拓朴面得到的三角网格。理解这些概念也有助于理解显示模块的实现，及拓朴数据中包含的几何数据的意义。

关键字 Key Words：OpenCascade, polygon curve, triangulation,discrete edge, discrete face, OpenSceneGraph, OSG

一、引言 Introduction

“实体造型技术主流的是边界表达 BRep，就是模型由面和边组成，这些面和边都是参数化的解析曲面和曲线，当拉伸或切割实体操作时候，就是用生成的实体和已有的实体进行实体布尔运算，其实是进行的面和边的相交运算，从而算出到新的面或者边。比如圆柱面和平面相交，以前的圆柱面分成了两个，同时产生出一条相交的空间椭圆曲线段，这些解析面/线边要通过三角化算法离散成三角网格或者线段条作为逼近表达，才能用 OpenGL 画出来。”以上内容来自博客：http://yrcpp.blog.163.com/blog/static/126045259201310199515969/ ，感谢网友的分享，言简意赅地把造型的核心进行了说明。

以前看《计算机图形学》相关的书时，从数学概念到具体实现的桥梁总是无法衔接。现在，通过学习 OpenCascade，终于把这些都串起来了。正如上面网友所说，面和边要在 OpenGL 中显示出来就需要离散化，即把边离散为多段线，把面离散为三角网格。这样就可以把用参数精确表示的几何数据在计算机布满像素点的屏幕上逼近显示了。

本文通过程序实例，将 OpenCascade 中的拓朴数据（边、面）离散化后在 OpenSceneGraph 中进行显示。有了这些离散数据，就可以不用 OpenCascade 的显示模块了，可以使用其他显示引擎对形状进行显示。即若要线框模式显示形状时，就绘制离散形状拓朴边后得到的多段线；若要实体渲染模式显示形状时，就绘制离散形状拓朴面得到的三角网格。理解这些概念也有助于理解显示模块的实现，及拓朴数据中包含的几何数据的意义。

二、程序示例

以下通过一个具体程序实例，来对 OpenCascade 中的拓朴边和拓朴面进行离散化。

```
/*
*       Copyright (c) 2013 eryar All Rights Reserved.
*
*                  File : Main.cpp
*                Author : eryar@163.com
*                  Date : 2013-12-03 18:09
*               Version : 1.0v
*
*       Description : Draw OpenCascade polygon Curves of the edge
*                             and triangulations of the face in OpenSceneGraph.
*                             When you want to display the shape in the computer,
*                             you can not display the geometry exactly, the only
*                             way to show them is in the approximation form.
*
*          Key Words : OpenCascade, polygon curve, triangulation,
*                             discrete edge, discrete face, OpenSceneGraph, OSG
*
*/

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

#include <Poly_Polygon3D.hxx>
#include <Poly_Triangulation.hxx>

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

#include <BRep_Tool.hxx>
#include <BRepMesh.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")

// OpenSceneGraph library.
#include <osgDB/ReadFile>
#include <osgViewer/Viewer>
#include <osgGA/StateSetManipulator>
#include <osgViewer/ViewerEventHandlers>

#pragma comment(lib, "osgd.lib")
#pragma comment(lib, "osgDBd.lib")
#pragma comment(lib, "osgGAd.lib")
#pragma comment(lib, "osgViewerd.lib")
```

```
/*
* @breif Descret the shape: edge.
*              For Edge will be discreted to polylines; (GCPnts_TangentialDeflection)
*              To get the polyline of the edge, use BRep_Tool::Polygon3D(Edge, L);
*/
osg::Node* BuildPolyline(const TopoDS_Edge& edge, double deflection = 0.1)
{
	osg::ref_ptr<osg::Geode> geode = new osg::Geode();
	osg::ref_ptr<osg::Geometry> linesGeom = new osg::Geometry();
	osg::ref_ptr<osg::Vec3Array> pointsVec = new osg::Vec3Array();

	TopLoc_Location location;

	BRepMesh::Mesh(edge, deflection);
	Handle_Poly_Polygon3D polyline = BRep_Tool::Polygon3D(edge, location);

	for (int i = 1; i < polyline->NbNodes(); i++)
	{
		gp_Pnt point = polyline->Nodes().Value(i);

		pointsVec->push_back(osg::Vec3(point.X(), point.Y(), point.Z()));
	}

	// Set the color of the polyline.
	osg::ref_ptr<osg::Vec4Array> colors = new osg::Vec4Array;
	colors->push_back(osg::Vec4(1.0f, 1.0f, 0.0f, 0.0f));
	linesGeom->setColorArray(colors.get());
	linesGeom->setColorBinding(osg::Geometry::BIND_OVERALL);

	// Set vertex array.
	linesGeom->setVertexArray(pointsVec);
	linesGeom->addPrimitiveSet(new  osg::DrawArrays(osg::PrimitiveSet::LINE_LOOP,  0,
pointsVec->size()));

	geode->addDrawable(linesGeom.get());

	return geode.release();
}

/*
* @breif Descret the shape: face.
*              For Face will be discreted to triangles; (BRepMesh_FastDiscret)
*              To get the triangles of the face, use BRep_Tool::Triangulation(Face, L);
*/
osg::Node* BuildMesh(const TopoDS_Face& face, double deflection = 0.1)
{
	osg::ref_ptr<osg::Geode> geode = new osg::Geode();
	osg::ref_ptr<osg::Geometry> triGeom = new osg::Geometry();
	osg::ref_ptr<osg::Vec3Array> vertices = new osg::Vec3Array();
	osg::ref_ptr<osg::Vec3Array> normals = new osg::Vec3Array();

	TopLoc_Location location;
	BRepMesh::Mesh(face, deflection);
```

```
Handle_Poly_Triangulation triFace = BRep_Tool::Triangulation(face, location);

Standard_Integer nTriangles = triFace->NbTriangles();

gp_Pnt vertex1;
gp_Pnt vertex2;
gp_Pnt vertex3;

Standard_Integer nVertexIndex1 = 0;
Standard_Integer nVertexIndex2 = 0;
Standard_Integer nVertexIndex3 = 0;

TColgp_Array1OfPnt nodes(1, triFace->NbNodes());
Poly_Array1OfTriangle triangles(1, triFace->NbTriangles());

nodes = triFace->Nodes();
triangles = triFace->Triangles();

for (Standard_Integer i = 1; i <= nTriangles; i++)
{
	Poly_Triangle aTriangle = triangles.Value(i);

	aTriangle.Get(nVertexIndex1, nVertexIndex2, nVertexIndex3);

	vertex1 = nodes.Value(nVertexIndex1).Transformed(location.Transformation());
	vertex2 = nodes.Value(nVertexIndex2).Transformed(location.Transformation());
	vertex3 = nodes.Value(nVertexIndex3).Transformed(location.Transformation());

	gp_XYZ vector12(vertex2.XYZ() - vertex1.XYZ());
	gp_XYZ vector13(vertex3.XYZ() - vertex1.XYZ());
	gp_XYZ normal = vector12.Crossed(vector13);
	Standard_Real rModulus = normal.Modulus();

	if (rModulus > gp::Resolution())
	{
		normal.Normalize();
	}
	else
	{
		normal.SetCoord(0., 0., 0.);
	}

	//if (face.Orientable() != TopAbs_FORWARD)
	//{
	//	normal.Reverse();
	//}

	vertices->push_back(osg::Vec3(vertex1.X(), vertex1.Y(), vertex1.Z()));
	vertices->push_back(osg::Vec3(vertex2.X(), vertex2.Y(), vertex2.Z()));
	vertices->push_back(osg::Vec3(vertex3.X(), vertex3.Y(), vertex3.Z()));

	normals->push_back(osg::Vec3(normal.X(), normal.Y(), normal.Z()));
}

triGeom->setVertexArray(vertices.get());
```

```
	triGeom->addPrimitiveSet(new osg::DrawArrays(osg::PrimitiveSet::TRIANGLES, 0, vertices->size()));

	triGeom->setNormalArray(normals);
	triGeom->setNormalBinding(osg::Geometry::BIND_PER_PRIMITIVE);

	geode->addDrawable(triGeom);

	return geode.release();
}

osg::Node* BuildScene(void)
{
	osg::ref_ptr<osg::Group> root = new osg::Group();

	gp_Ax2 axis;

	// 1. Test circle while deflection is default 0.1;
	TopoDS_Edge circleEdge1 = BRepBuilderAPI_MakeEdge(gp_Circ(axis, 6.0));
	root->addChild(BuildPolyline(circleEdge1));

	// 2. Test circle while deflection is 0.001.
	axis.SetLocation(gp_Pnt(8.0, 0.0, 0.0));
	axis.SetDirection(gp_Dir(1.0, 1.0, 1.0));

	TopoDS_Edge circleEdge2 = BRepBuilderAPI_MakeEdge(gp_Circ(axis, 6.0));
	root->addChild(BuildPolyline(circleEdge2, 0.001));

	// 3. Test ellipse while deflection is 1.0.
	TopoDS_Edge ellipseEdge = BRepBuilderAPI_MakeEdge(gp_Elips(gp::XOY(), 16.0, 8.0));
	root->addChild(BuildPolyline(ellipseEdge, 1.0));

	// 4. Test sphere face while deflection is default 0.1.
	axis.SetLocation(gp_Pnt(26.0, 0.0, 0.0));
	TopoDS_Face sphereFace1 = BRepBuilderAPI_MakeFace(gp_Sphere(axis, 8.0));
	root->addChild(BuildMesh(sphereFace1));

	// 5. Test sphere face while deflection is 2.0.
	axis.SetLocation(gp_Pnt(26.0, 18.0, 0.0));
	TopoDS_Face sphereFace2 = BRepBuilderAPI_MakeFace(gp_Sphere(axis, 8.0));
	root->addChild(BuildMesh(sphereFace2, 2.0));

	// 6. Test sphere face while deflection is 0.001.
	axis.SetLocation(gp_Pnt(26.0, -18.0, 0.0));
	TopoDS_Face sphereFace3 = BRepBuilderAPI_MakeFace(gp_Sphere(axis, 8.0));
	root->addChild(BuildMesh(sphereFace3, 0.001));

	return root.release();
}

int main(void)
{
	osgViewer::Viewer myViewer;
```

```
        myViewer.setSceneData(BuildScene());

        myViewer.addEventHandler(new
osgGA::StateSetManipulator(myViewer.getCamera()->getOrCreateStateSet()));
        myViewer.addEventHandler(new osgViewer::StatsHandler);
        myViewer.addEventHandler(new osgViewer::WindowSizeHandler);

        return myViewer.run();
}
```

示例程序测试了不同的离散精度情况下同一个形状的显示效果，程序结果如下图所示：

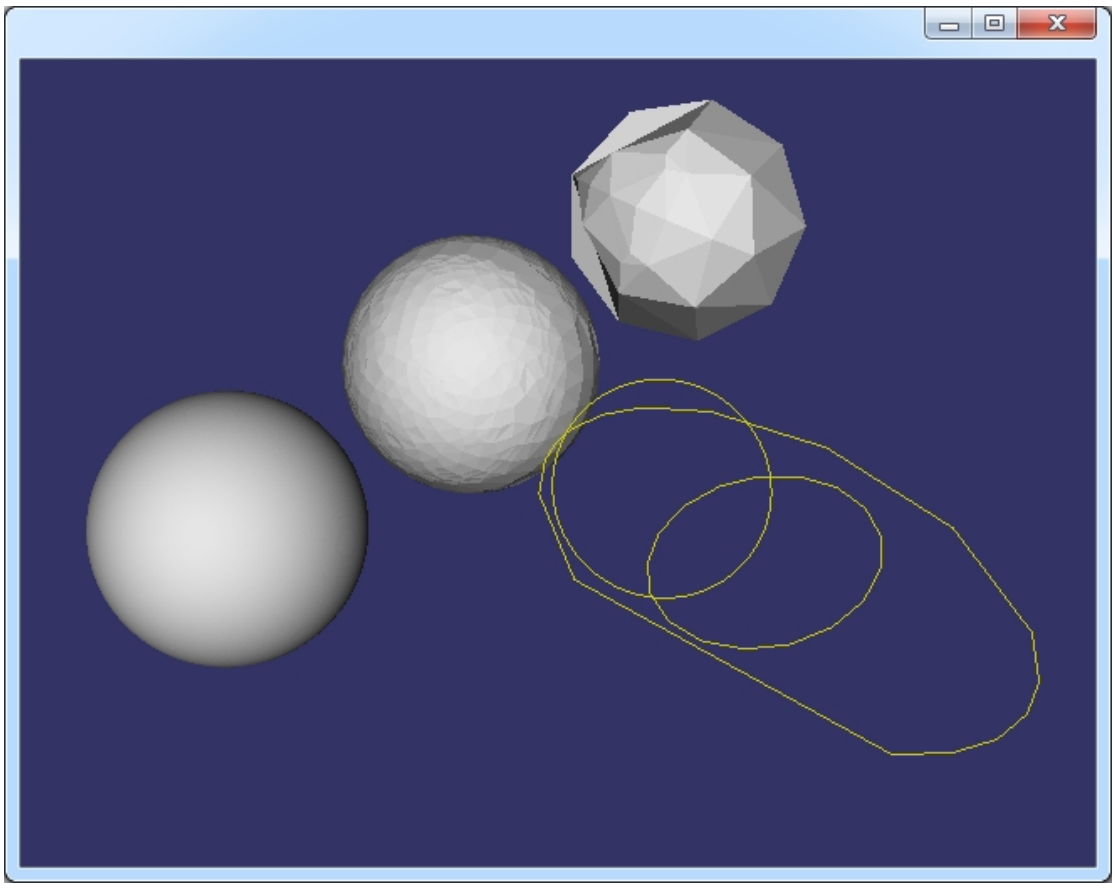

Figure 2.1 Edge and Face representation in OpenSceneGraph

从图中可知，离散精度越高，离散后得到线上的点或三角网格就越多，显示越细腻。

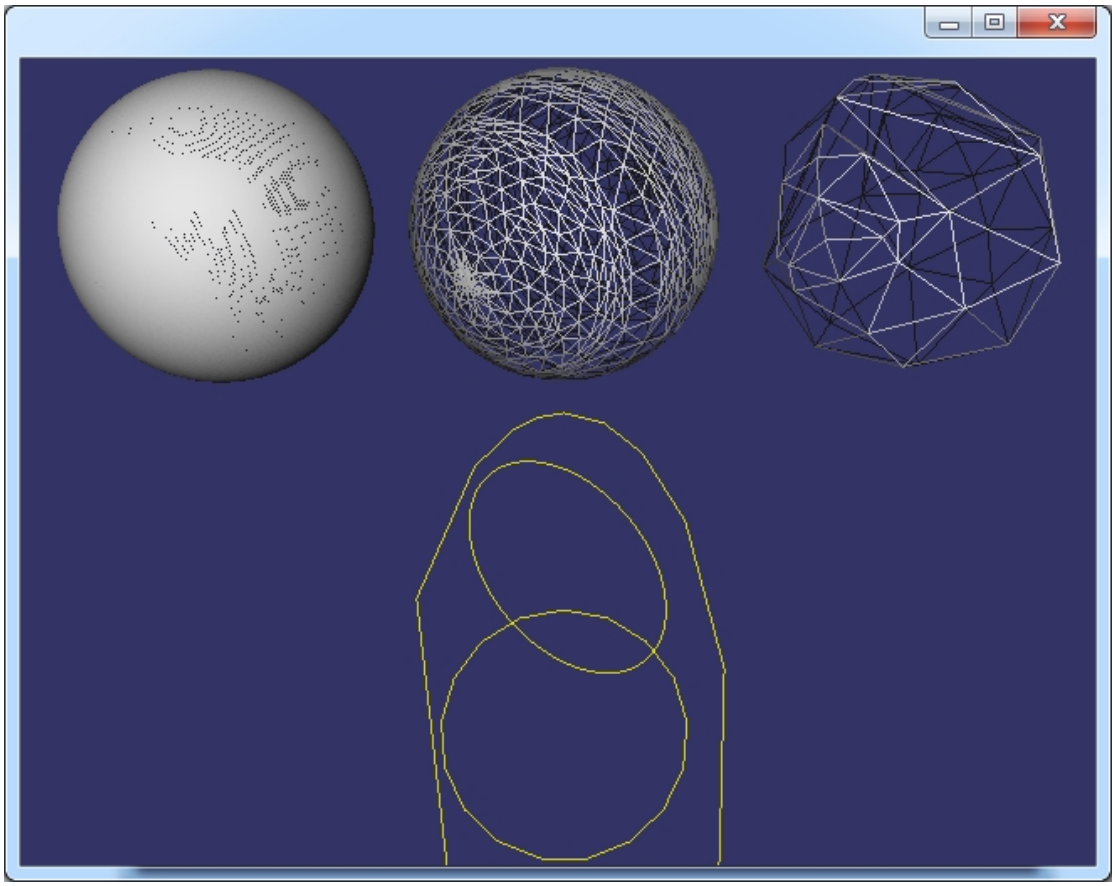

Figure 2.2 Edge and Face representation in OpenSceneGraph

其中，边的离散化使用到了类：GCPnts_TangentialDeflection；面的离散化使用到了类：BRepMesh_FastDiscret。有兴趣的读者可跟踪调试，理解其具体实现的算法。边的离散应该很好理解，面的离散使用了 Delauney 三角剖分算法。关于 Delauney 三角剖分算法的介绍可参考博客：http://www.cppblog.com/eryar/archive/2013/05/26/200605.aspx。

三、结论

通过把 OpenCascade 中的拓朴边和面离散化用 OpenSceneGraph 显示，填补了形状数学精确表示与在计算机屏幕上近似显示之间的隔阂。也有助于理解拓朴结构中包含几何数据的 BRep_TFace、BRep_TEdge、BRep_TVertex 中除了包含面、边的精确的参数表示数据外，还包含了用于近似显示的离散数据的意义。

四、参考资料

1. 博客：http://yrcpp.blog.163.com/blog/static/126045259201310199515969/
2. 博客：http://www.cppblog.com/eryar/archive/2013/05/26/200605.aspx