OpenCascade 中网格的数据结构

Mesh Data Structure in OpenCascade

eryar@163.com

摘要 Abstract：本文对网格数据结构作简要介绍，并结合使用 OpenCascade 中的数据结构，将网格数据在 OpenSceneGraph 中可视化。

关键字 KeyWords：OpenCascade、OpenSceneGraph、Triangulation、Mesh Data Structure

一、引言 Introduction

三角网格就是全部由三角形组成的多边形网格。多边形和三角网格在图形学和建模中广泛使用，用来模拟复杂物体的表面，如建筑、车辆、人体，当然，还有茶壶等自由曲面。任意多边形网格都能转换成三角网格。三角网格以其简单性而吸引人，相对于一般多边形网格许多操作对三角网格列容易。

常用的网格数据文件有：

1. Wavefront OBJ(*.obj)
2. 3D Max(*.max, *.3ds)
3. VRML(*.vrl)
4. Inventor(*.iv)
5. PLY(*.ply, *.ply2)
6. STL(*.stl)
7. Off(*.off) in CGAL library

有些文件以文本方式保存，有些可以以二进制方式保存。如下图所示为 OBJ 文件的格式：

```
v 1.0 0.0 0.0
v 0.0 1.0 0.0
v 0.0 -1.0 0.0
v 0.0 0.0 1.0
f 1 2 3
f 1 4 2
f 3 2 4
f 1 3 4
```

Figure 1.1 Wavefront OBJ File Format

- Vertices
  - 以‘V’开始；
  - 其后为坐标值（x,y,z）；
- Faces
  - 以‘F’开始；
  - 其后为面的顶点索引值；
- Other properties
  - Normal, texture coordinates, material, etc.

二、三角网格的表示 Mesh Data Structure

三角网格为一个三角形列表，所以最直接的表示方法是用三角形数组：

```
struct Triangle
{
    Vector3 p[3];
};

struct TriangleMesh
{
    int triCount;
    Triangle* triList;
};
```

对于某些应用程序，这种表示方法已经足够。然而，术语“网格”隐含的相邻三角形的连通性未在这种简单表示中有任何体现。实际应用中出现的三角网格，每个三角形都和其他三角形共享边。于是三角网格需要存储三类信息：

- 顶点。每个三角形有三个顶点，各顶点都有可能和其他三角形共享；
- 边。连接两个顶点的边，每个三角形有三条边；
- 面。每个三角形对应一个面。我们可以用顶点或边列表表示面；

根据应用程序的不同，有多种有效的网格表示方法。常用的一种标准的存储格式为索引三角网格。

在索引三角网格中，我们维护了两个列表：顶点表与三角形表。每个顶点包含一个 3D 位置，也可能有表面法向量、纹理映射坐标、光照值附加数据。每个三角形由顶点列表的三个索引值组成。通常顶点列出的顺序是非常重要的，因为我们必须考虑面的“正面”和“反面”。从前面看时，我们将用顺时针方向列出顶点。

在 OpenCascade 中，分别用类 TColgp_Array1OfPnt 和 Poly_Array1OfTriangle 表存储顶点表和三角形表。注意到索引三角形列表中的邻接信息是隐含的，即边信息没有存储，但我们可以通过搜索三角形表找出公共边。和前面“三角形数组”方式相比，这种方式确实能节省不少空间。原因是信息存于顶点级别，它的整数索引比之三角形数组里存储的顶点重复率要小得多。实践中，三角网里确实有大量的连接性问题。

简单索引三角网格对于基本应用已经足够了。但为更加高效地实现某些操作还可以进一步改进。主要的问题是邻接信息没有显式表达，所以必须从三角形列表中搜索。另一种表达方法可以常数时间内取得这种信息。方法是显式维护一个边列表，每边由两个端点定义，同时维护一个共享该边的三角形列表。这样三角形可视为三条边而非三个点的列表，也就是说它是边列表的索引。该思想的一个扩展称作“Winged Edge”模型（翼边模型），对每一顶点，存储使用该点的边的索引。这样三角形和边都可以通过定位点列表快速查找。

大多数显卡并不直接支持索引三角网。渲染三角形时，一般是将三个顶点同时提交。这样，共享顶点会多次提交，三角形用到一次就提交一次。因为内存和图形硬件间的数据传输是瓶颈，所以许多 API 和硬件支持特殊三角网格式以减少传输量。基本思想是排序点和面，使得显存中已有的三角形不需要再次传输。

从最高灵活性到最低灵活性，我们讨论三种方案：

- 顶点缓存；
- 三角带 Triangle Strip；
- 三角扇 Triangle Fan；

三、程序示例 Code Example

在安装好的 CGAL 库中发现其例子中有很多 off 文件，其格式同常见的网格文件格式基本相同，结合 OpenCascade 和 OpenSceneGraph，读取 off 文件，将其表示的网格模型显示出来。程序代码如下所示：

```
/*
*    Copyright (c) 2013 eryar All Rights Reserved.
*
*        File    : Main.cpp
*        Author  : eryar@163.com
*        Date    : 2013-08-10 18:02
*        Version : V1.0
*
*    Description : Mesh Viewer for the general mesh file format.
*                      Poly_Triangulation data structure can save vertices and
triangle index.
*
*/

// 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")

// OpenCascade library.
#include <TColgp_Array1OfPnt.hxx>
#include <Poly_Array1OfTriangle.hxx>
#include <Poly_Triangulation.hxx>

#pragma comment(lib, "TKernel.lib")
#pragma comment(lib, "TKMath.lib")

/**
* @breif Build the mesh from *.off file.
*/
osg::Node* buildMesh(const std::string& fileName)
{
    std::ifstream offFile(fileName.c_str());
    std::string strBuffer;
```

```
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();

Standard_Integer nbNodes = 0;
Standard_Integer nbTriangles = 0;

// Ignore "OFF"
offFile>>strBuffer;
offFile>>nbNodes>>nbTriangles>>strBuffer;

TColgp_Array1OfPnt nodes(0, nbNodes);
Poly_Array1OfTriangle triangles(0, nbTriangles);

// Read node coordinate and store them.
Standard_Real dx = 0.0;
Standard_Real dy = 0.0;
Standard_Real dz = 0.0;

for (Standard_Integer i = 0; i < nbNodes; i++)
{
    offFile>>dx>>dy>>dz;

    nodes(i).SetCoord(dx, dy, dz);
}

// Read the triangles
Standard_Integer ni = 0;
Standard_Integer n1 = 0;
Standard_Integer n2 = 0;
Standard_Integer n3 = 0;

for (Standard_Integer i = 0; i < nbTriangles; i++)
{
    offFile>>ni>>n1>>n2>>n3;

    triangles(i).Set(n1, n2, n3);
}

// Construct the mesh data by Poly_Triangulation.
gp_Pnt node1;
gp_Pnt node2;
```

```
    gp_Pnt node3;
    Poly_Triangle triangle;
    Handle_Poly_Triangulation T = new Poly_Triangulation(nodes, triangles);

    for (Standard_Integer i = 0; i < nbTriangles; i++)
    {
        triangle = triangles.Value(i);

        triangle.Get(n1, n2, n3);

        node1 = nodes.Value(n1);
        node2 = nodes.Value(n2);
        node3 = nodes.Value(n3);

        gp_XYZ vector12(node2.XYZ() - node1.XYZ());
        gp_XYZ vector13(node3.XYZ() - node1.XYZ());
        gp_XYZ normal = vector12.Crossed(vector13);
        Standard_Real rModulus = normal.Modulus();

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

        vertices->push_back(osg::Vec3(node1.X(), node1.Y(), node1.Z()));
        vertices->push_back(osg::Vec3(node2.X(), node2.Y(), node2.Z()));
        vertices->push_back(osg::Vec3(node3.X(), node3.Y(), node3.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();
```

```
}

int main(int argc, char* argv[])
{
    osgViewer::Viewer myViewer;

    std::string strFile;

    (argc > 1) ? strFile = argv[1] : strFile = "ChineseDragon-10kv.off";

    myViewer.setSceneData(buildMesh(strFile));

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

    return myViewer.run();
    }
```

程序效果图如下所示：

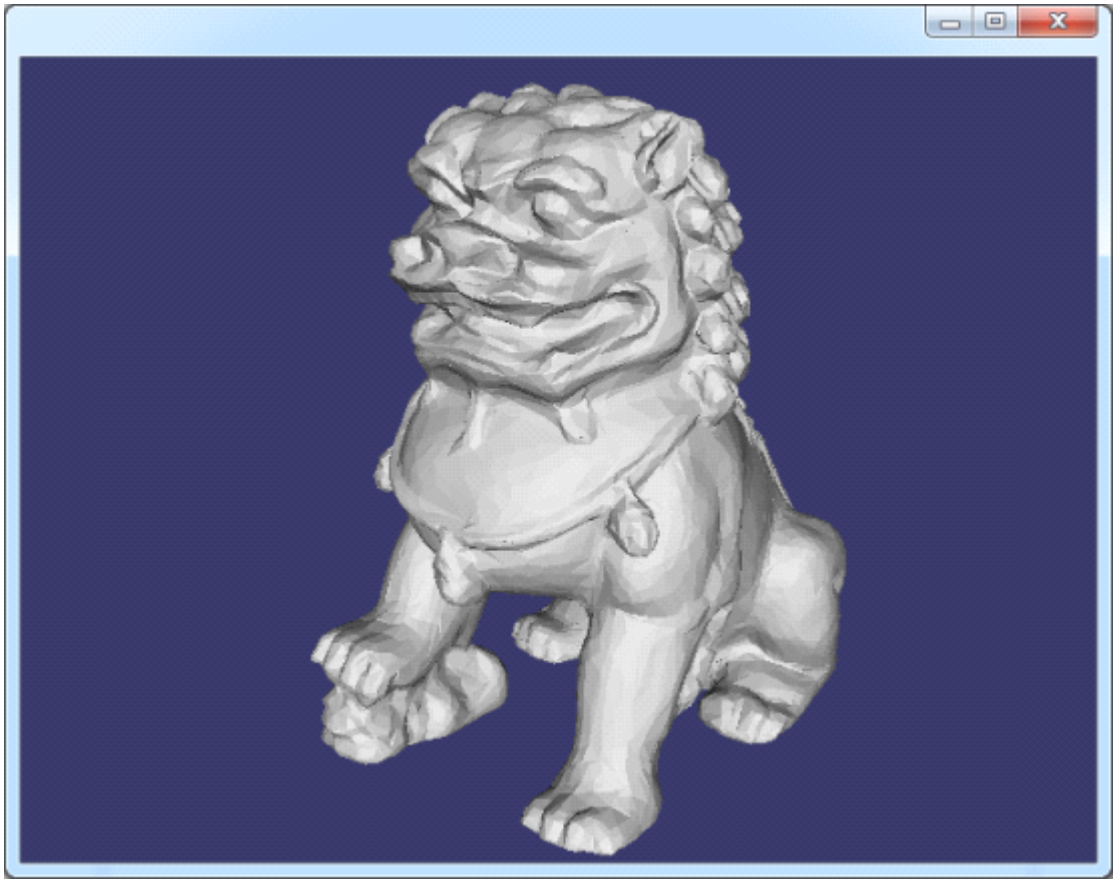

Figure 3.1 ChineseDragon-10kv.off

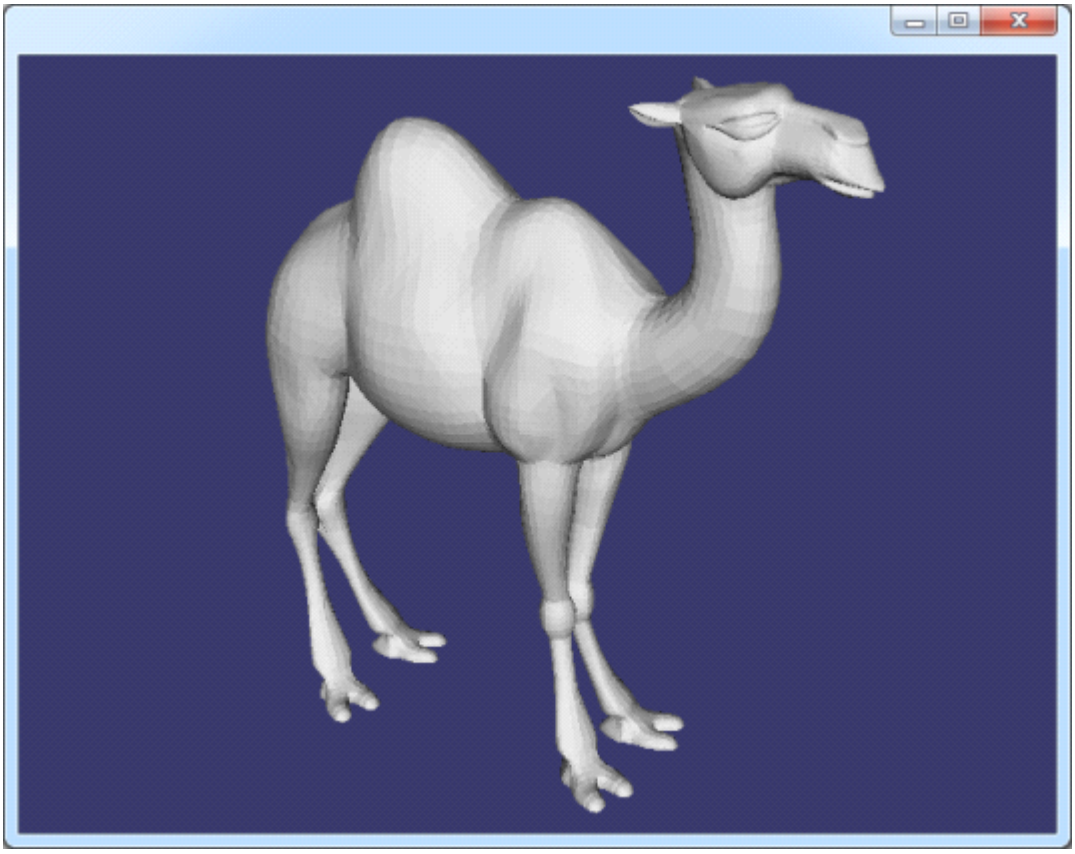

Figure 3.2 Camel.off

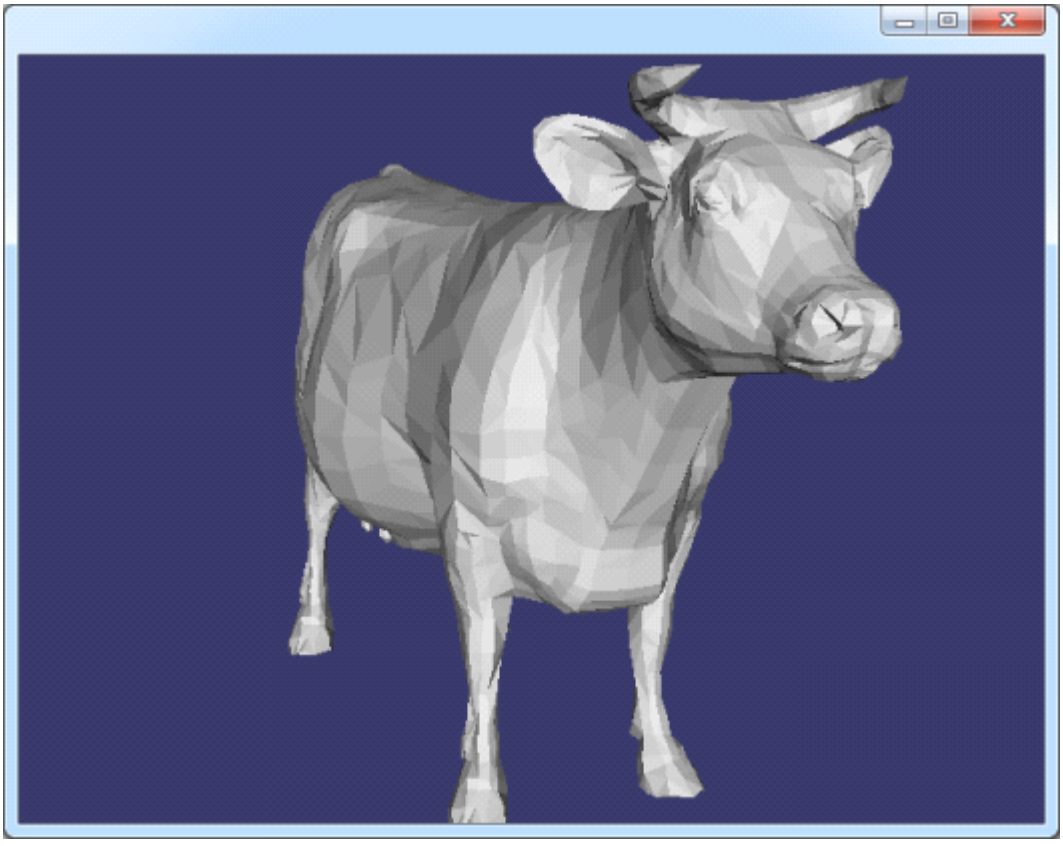

Figure 3.3 cow.off

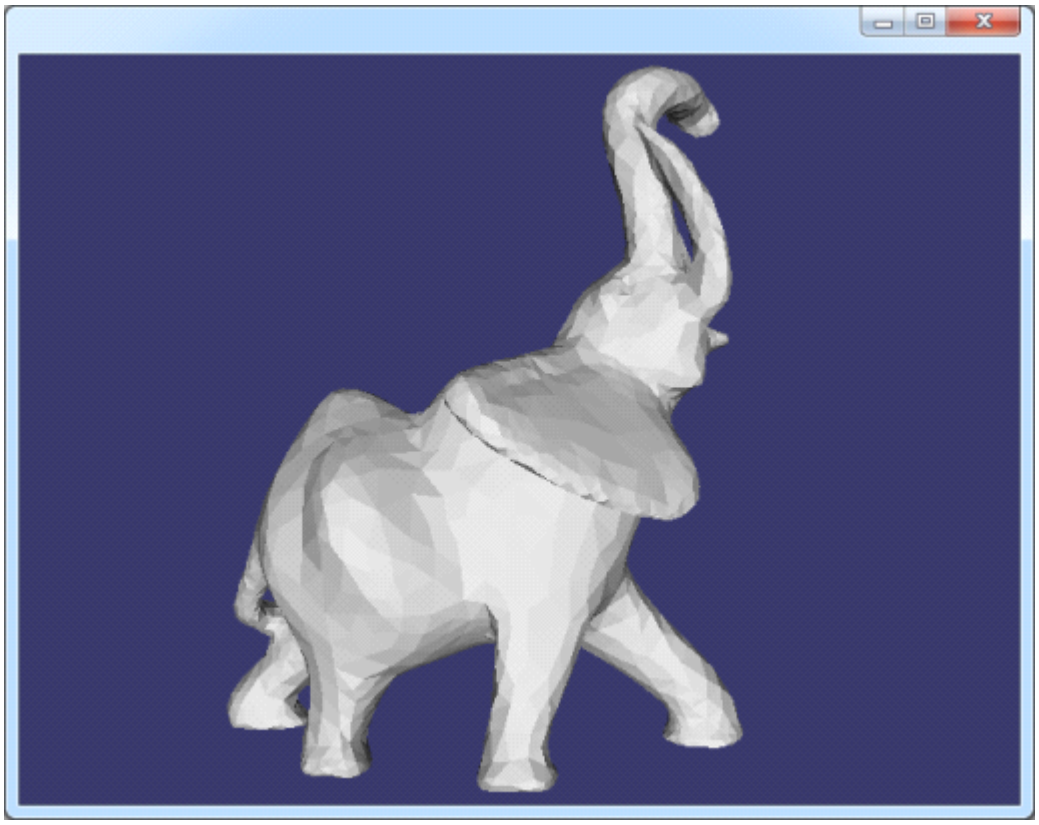

Figure 3.4 elephant.off

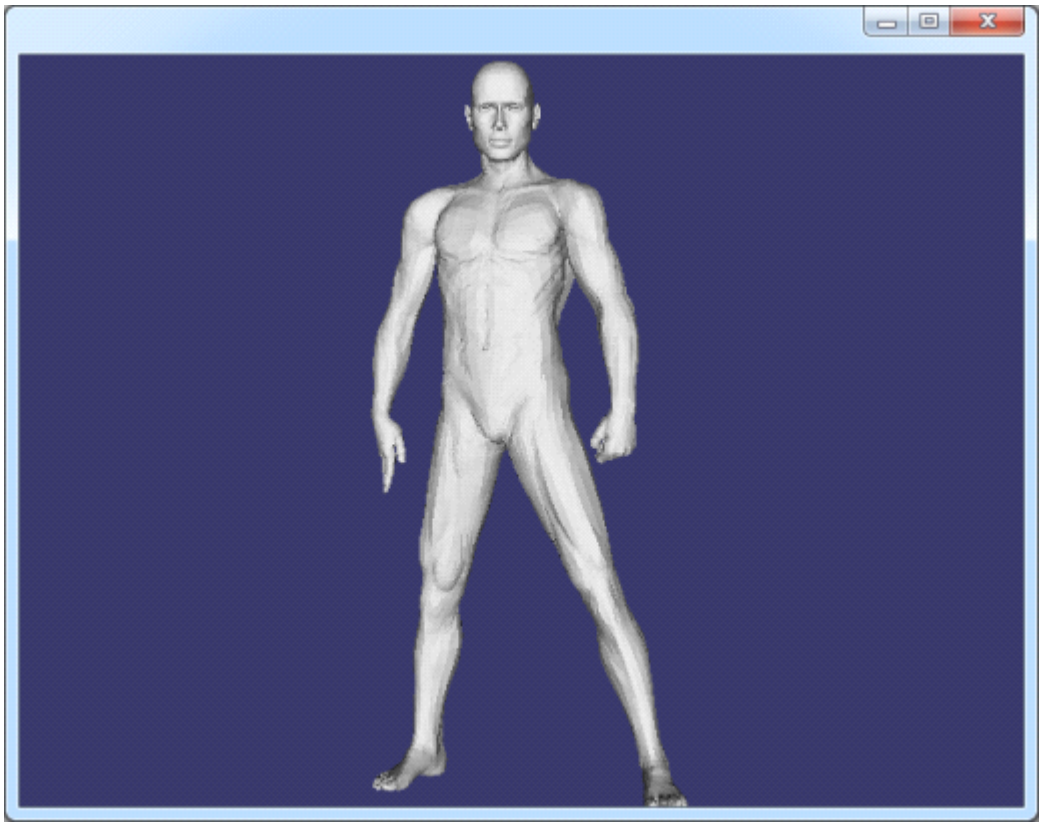

Figure 3.5 man.off

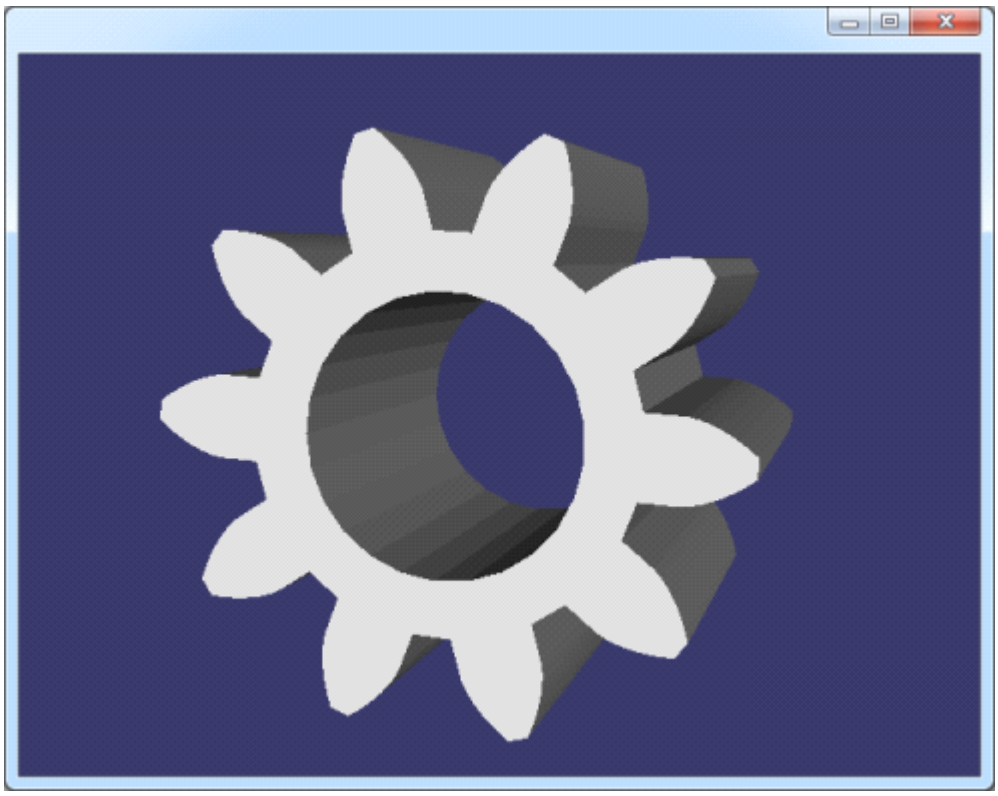

Figure 3.6 pinion.off

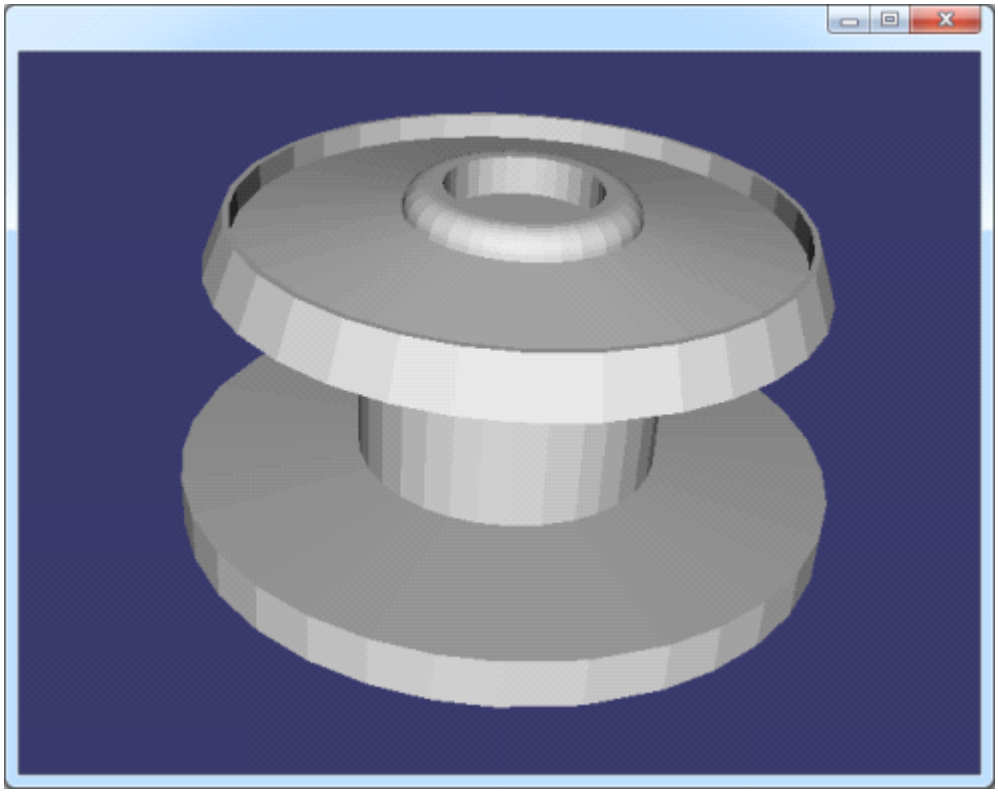

Figure 3.7 spool.off

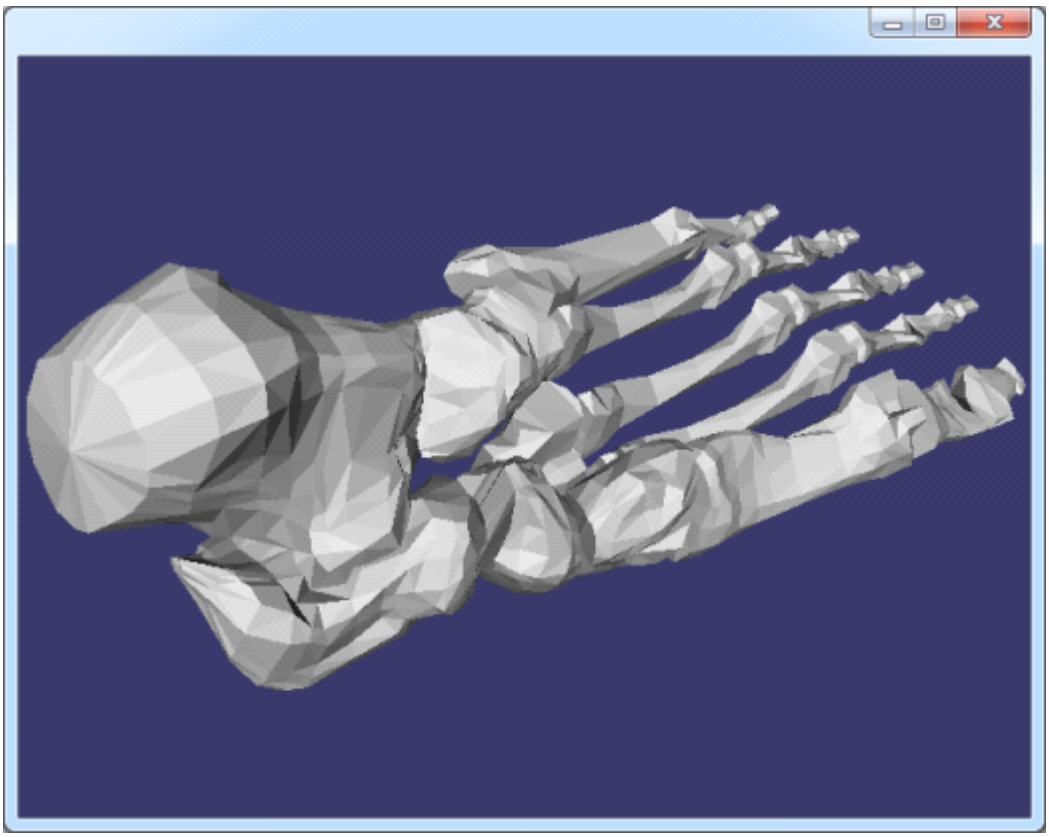

Figure 3.8 bones.off

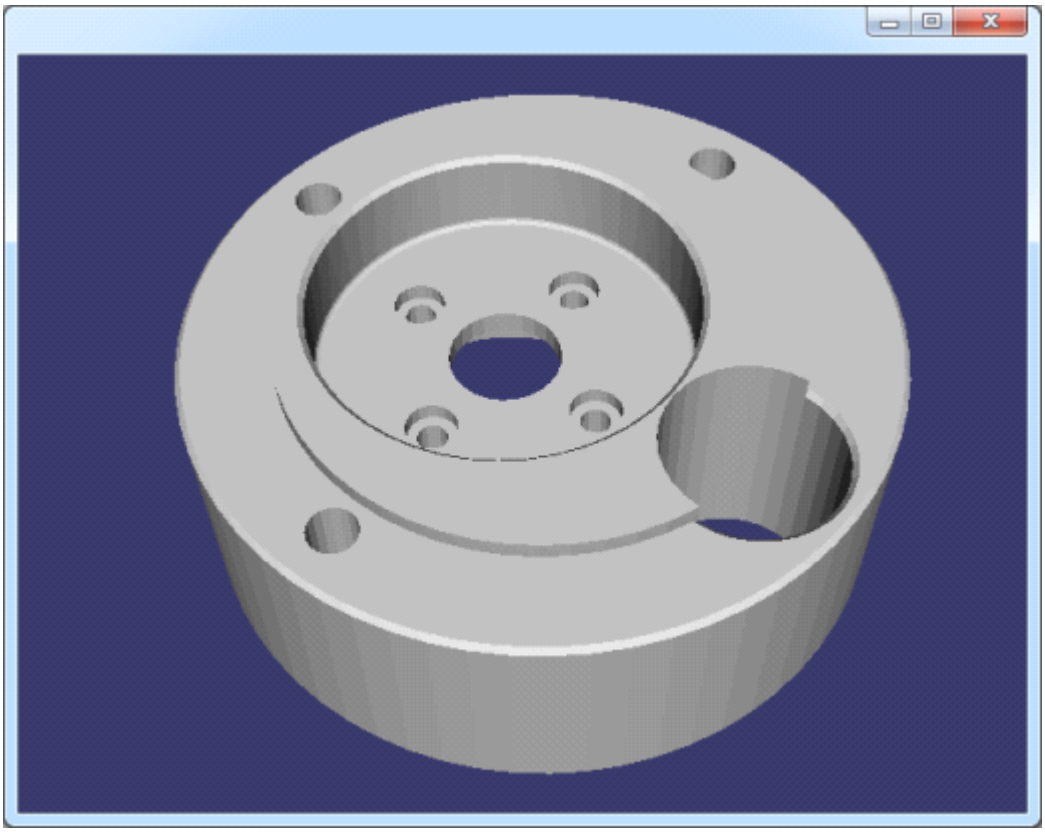

Figure 3.9 couplingdown.off

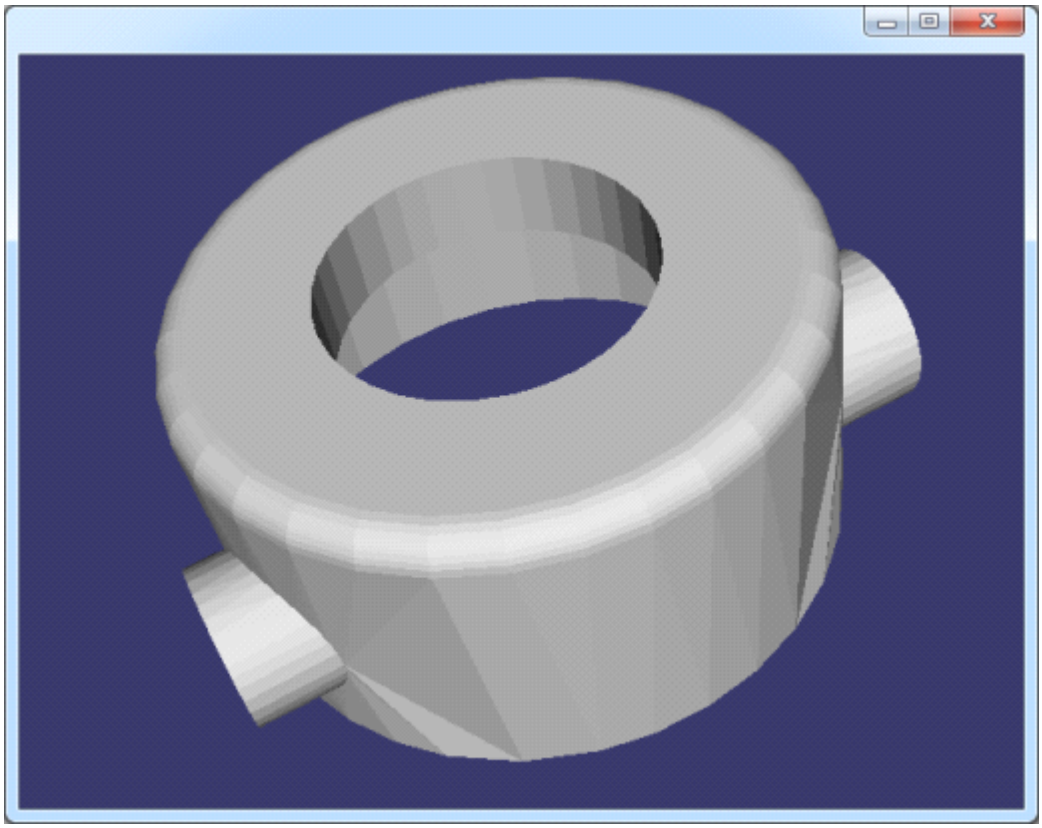

Figure 3.10 rotor.off

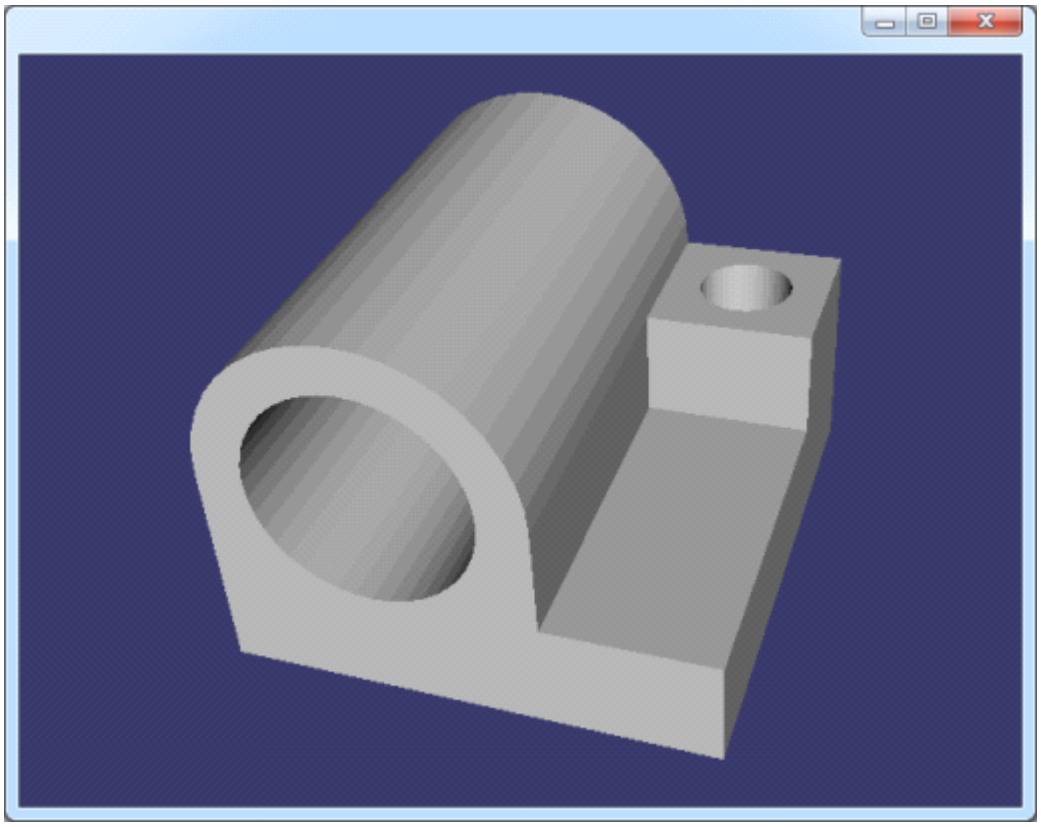

Figure 3.11 joint.off

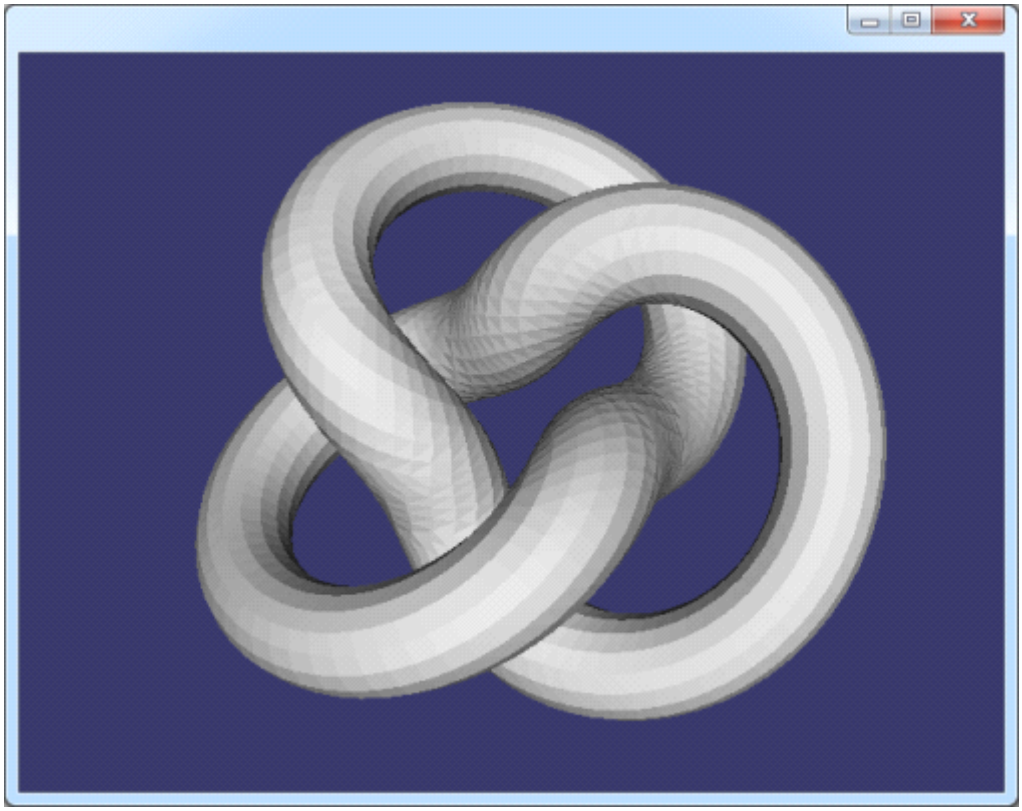

Figure 3.12 knot1.off

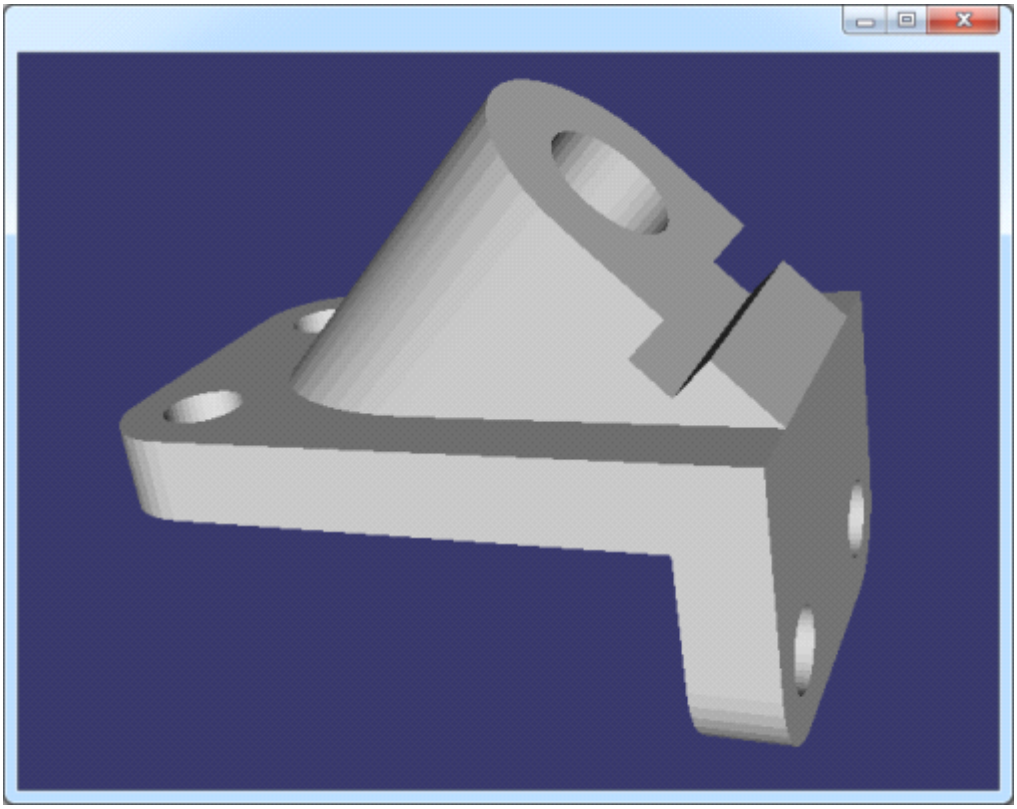

Figure 3.13 anchor.off

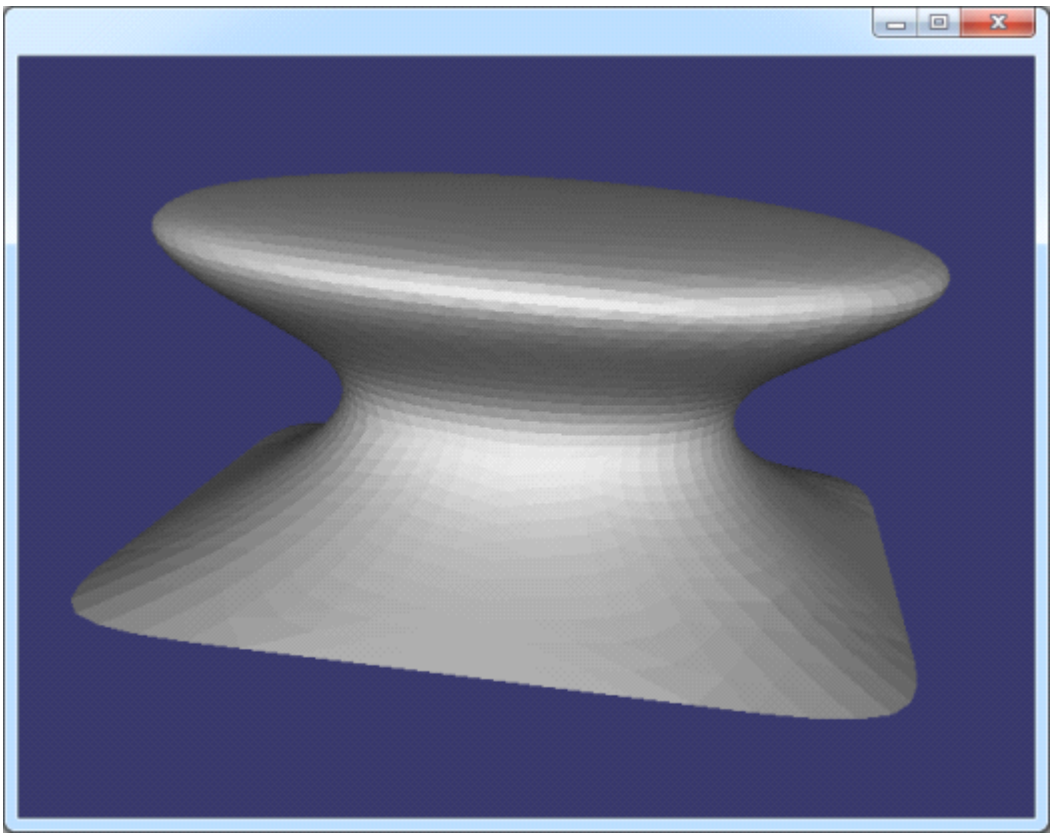

Figure 3.14 mushroom.off

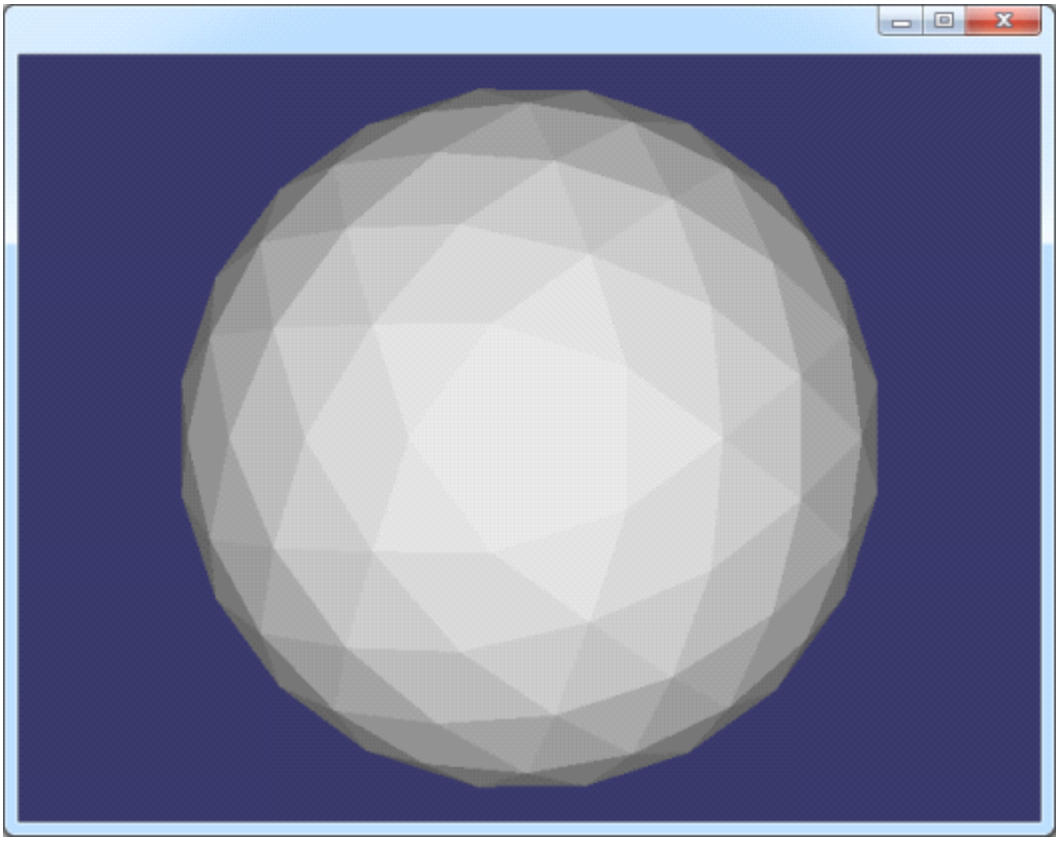

Figure 3.15 sphere.off

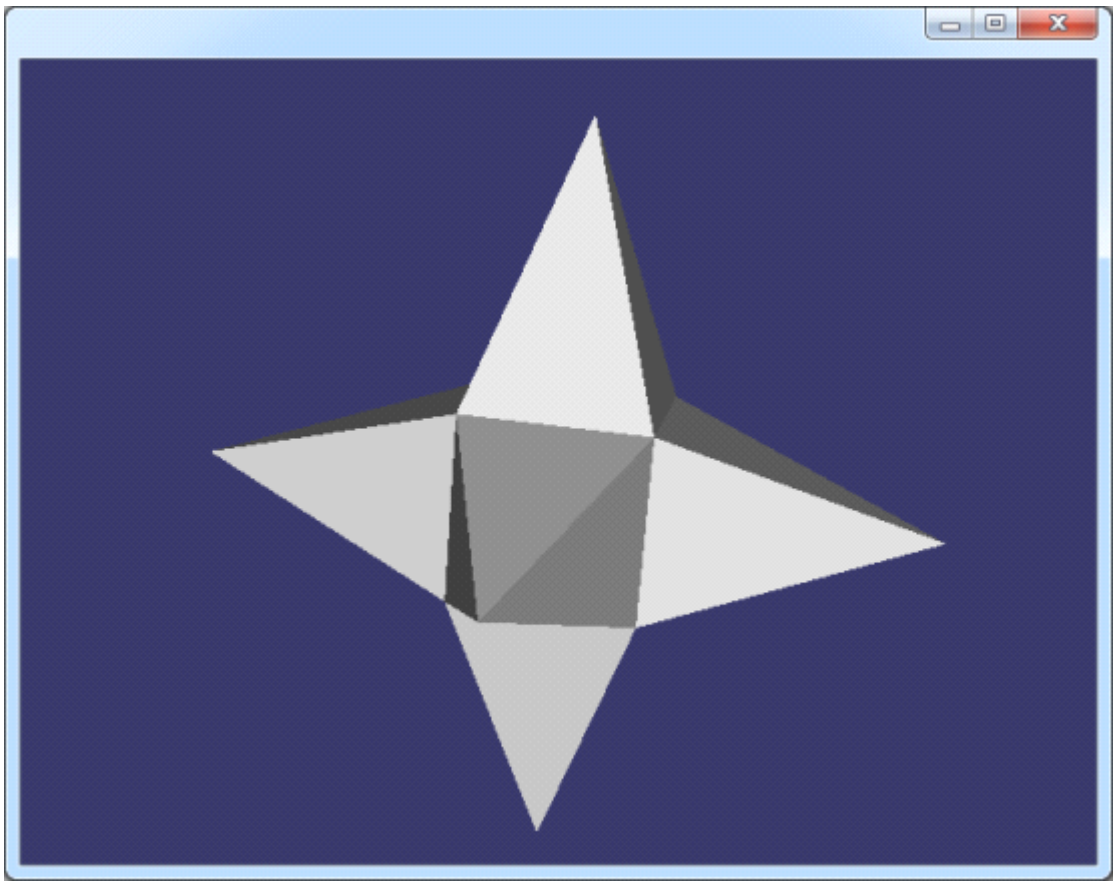

Figure 3.16 star.off

看到这些三维模型，很有感觉！在有关计算机图形学的期刊上有可能也会看到上面的模型。

四、结论 Conclusion

三角网格在计算中用来近似表示三维模型。存储三角网格的标准方式是使用索引三角网格方式。结合 OpenCascade 中的数据结构，将 CGAL 示例中的 off 文件在 OpenSceneGraph 中显示出来，感觉很棒！