# Open CASCADE 简介

整理人：金谊成

日期：2019.6.27

# 起源

Open CASCADE是一套开放原始码的CAD/CAM/CAE几何模型核心，源自于法国的Matra Datavision公司，这一套函式库系统原来是著名的CADCAM软体EUCLID的开发平台，但是在1998年， Matra Datavision改变了经营策略，从以销售软体为主改变为提供CAD/CAM/CAE软体服务为主的获利模式，并且在1999年公布Open CASCADE的程式原始码，任何人都可以在该公司的网站上下载并且使用这些程式码。自OpenCASCADE6.8开始授权协议改为LGPL协议，使得基于此技术的商业软件开发更加自由。

# 应用场景

Open CASCADE 技术是一个面向对象型的C++类库，他被设计用于开发复杂工程中的CAD/CAM/CAE应用程序。基于此项技术开发的应用程序可以同时满足广义或专用于CAD系统的二维或三维几何建模。

# 技术优势

## 免费证书

该项技术开源协议为LGPL，在开放源代码的同时不会对基于本技术开发的商业项目收取任何版权费用，可以大幅降低开发成本。

## 可靠性

该项技术来源自Matra Datavision公司的商业可视化引擎，自诞生起持续由庞大的技术社区维护，同时被许多商业公司作为产品开发的技术基础。其开放源代码的特性，使得开发者可以持续对代码库进行质量维护，保障其健壮性。

## 更新度高

较大的技术社区使得该项技术不断在演进和更新，保障技术质量与前沿性。

## 技术简介

该项技术中所提供的C++类和其他类型被分组到“包”中。“包”被组织成工具包(库)，开发者可以将应用程序链接到其中。最后，再将工具包分成七个模块。模块结构如图1所示。

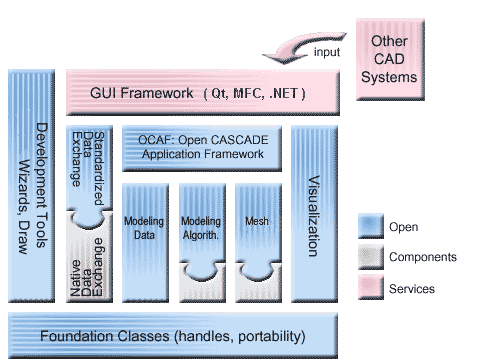


图1 OpenCASCADE模块图

## 基本类模块

基础类模块包含高级开放级联技术类使用的数据结构和服务，例如：

1）基本类型，如布尔、字符、整数或实数;

2）处理ASCII和Unicode字符串的字符串类;

3）处理静态或动态大小的数据聚合的集合类，如数组、列表、队列、集和哈希表(数据映射)。

4） 课提供常用的数值算法和基本的线性代数计算(加法、乘法、向量和矩阵的换位、解线性系统等)的C++类。

5）表示物理量和支持日期和时间信息的基本类型;

6） 提供定义和操作基本数据结构的基本几何和代数实体的实现的基本几何类型。

7）描述程序正常执行被放弃时的情况的异常类;

该模块还提供多种通用服务，如:

1）安全处理动态创建的对象，确保自动删除未引用的对象(智能指针);

2）可配置的优化内存管理器，提高了大量使用动态创建对象的应用程序的性能;

3）扩展的运行时类型信息(RTTI)机制，维护完整的类型层次结构，并提供在其上迭代的方法;

4）C++流的封装;

5）通过特定的分配器实现堆内存的自动管理;

6）表达式的基本解释器，便于创建定制脚本工具，表达式的通用定义等;

7）处理配置资源文件和可自定义消息文件的工具，方便应用程序中的多语言支持;

8）进度指示和用户中断接口，使得底层算法也有可能与用户进行通用、便捷的通信

C++与OCCT基本数据类型对照：

|  |  |
| --- | --- |
| **C++ Types** | **OCCT Types** |
| int | Standard\_Integer |
| double | Standard\_Real |
| float | Standard\_ShortReal |
| unsigned int | Standard\_Boolean |
| char | Standard\_Character |
| short | Standard\_ExtCharacter |
| char\* | Standard\_CString |
| void\* | Standard\_Address |
| short\* | Standard\_ExtString |

Handle类：一种基于OCCT的智能指针方法

## 4.2 建模数据模块

建模数据提供数据结构来实现三维对象的边界表示(BRep)。在BRep中，形状表示为拓扑中的几何集合。几何被理解为一个形状的数学描述，如曲线和曲面(简单或规范，贝塞尔，NURBS等)。拓扑是将几何对象绑定在一起的数据结构。

几何类型和实用程序提供几何数据结构和服务，例如:

1）点、向量、曲线、曲面描述:

2）利用坐标轴或坐标系统在三维空间进行定位

3）它们的几何变换，通过平移，旋转，对称，缩放变换和它们的组合。

4）通过插值和近似建立参数曲线和曲面;

5）将曲线和曲面转换为NURBS形式;

6）二维和三维曲线上点坐标的计算

7）几何对象间极值的计算。

拓扑定义了简单几何实体之间的关系。形状是一个基本的拓扑实体，可以分为组件(子形状):

1）顶点：与点对应的零维形状;

2）边：与曲线相对应的形状，在每条边上都有一个顶点;

### 3）Wire：由顶点连接的边的序列;

4）面：平面(二维)或曲面(三维)的一部分，由导线包围;

5）外壳：由wire的边界相连接得到的的面集合;

6）实体：三维空间有限的封闭部分，以壳为界;

7）复合固体：由外壳边界的面连接的固体的集合。

## 4.3建模算法

### 4.3.1 几何工具

提供如下算法：

1）计算2D、3D中的曲线、表面的交点

2）点到3D、2D曲线的投影

点到表面的投影

3D曲线到面的投影

3）在给定条件下构建曲线和圆

4）通过插值构建曲线和圆

示例：

**曲线相交示例：（C1C2为两曲线）**

Geom2dAPI\_InterCurveCurve Intersector(C1,C2,tolerance);

曲线C3的自交：

Geom2dAPI\_InterCurveCurve Intersector(C3,tolerance);

获得交点个数：

gp\_Pnt2d P = Intersector.Point(Index);

通过索引获得某交点：

gp\_Pnt2d P = Intersector.Point(Index);

获得相交所得的切分线段个数：

Standard\_Integer M = Intersector.NbSegments();

按照索引获得某一切分片段：

Handle(Geom2d\_Curve) Seg1, Seg2;

Intersector.Segment(Index,Seg1,Seg2);

Intersector.Segment(Index,Seg1); //曲线自交情况

获得相交计算器本身：

Geom2dInt\_GInter& TheIntersector = Intersector.Intersector();

**曲线与表面相交：**

GeomAPI\_IntCS Intersector(C, S);//C为曲线，S为表面

获取交点个数

Standard\_Integer nb = Intersector.NbPoints();

按照索引获得某交点：

gp\_Pnt& P = Intersector.Point(Index);

**表面间相交：**

计算器对象的实例化：

GeomAPI\_IntSS Intersector(S1, S2, Tolerance);

获得交曲线个数：

Standard\_Integer nb = Intersector. NbLines();

按照索引获得某交曲线：

Handle(Geom\_Curve) C = Intersector.Line(Index)

**插值计算：**

2D插值API示例：

Geom2dAPI\_Interpolate (const Handle\_TColgp\_HArray1OfPnt2d& Points,

const Standard\_Boolean PeriodicFlag,

const Standard\_Real Tolerance);

//第一个参数为已有的插值基点数组；第二个为是否按照周期插值标记；第三个为所允许的误差精度

获得锯齿线：

语法如上，除类名改为GeomAPI;

**在约束条件下构建线和圆：**

此处约束条件常指为相切问题如：求一个给定半径，且同时与另两个圆相切的圆

为了能对计算结果进行过滤，可利用如下参数：

1. Enclose（结果包含给定条件中的某图形）
2. Enclosed（结果被包含于给定条件中的几何图形）
3. Outside（结果在给定条件中的几何图形外）
4. Unqualified（相对位置不明确）

求直线示例：

GccAna\_Lin2d2Tan Solver(GccEnt::Outside(C1), GccEnt::Outside(C2), Tolerance);

// 该直线在圆C1、C2外部，且均相切

求圆示例：

GccAna\_Circ2d2TanRad Solver(GccEnt::Outside(C1), GccEnt::Outside(C2), Rad, Tolerance);

// 该圆被包含在圆C1,C2内，且半径为Rad

**投影计算：**

可计算的投影类型如下：

1）2D点在2D曲线上的投影

2）3D点在3D曲线或表面上的投影

3）3D曲线在表面上投影

在给定的法向量域中求2D点在2D曲线上的投影：

gp\_Pnt2d P;

Handle(Geom2d\_BezierCurve) C = new Geom2d\_BezierCurve(args);

Geom2dAPI\_ProjectPointOnCurve Projector (P, C,U1, U2));

//若不需加法向量限制域，U1，U2可缺省

获得投影点个数：

Standard\_Integer NumSolutions = Projector.NbPoints();

按照索引获取投影点位置：

Standard\_Real D = Projector.Distance(Index);

获得最近投影点：

gp\_Pnt2d P1 = Projector.NearestPoint();

获得最近投影点的参数：

Standard\_Real U = Projector.LowerDistanceParameter();

获得最近投影点到曲线的距离：

Standard\_Real D = Geom2dAPI\_ProjectPointOnCurve (P,C);