DM-Creater 系列

DM-Pillar 几何内核引擎

用户手册 V1.0

北京求解科技有限公司

目录

1　产品简介

3D 建模如今的应用非常广泛，国内外都有多个优秀产品，但是在机械制图尤其是曲面设计领域，国内的发展还比较薄弱，再此之前我们的工业界一直使用国外的同类产品，可是随着反全球化浪潮的盛行，在可预见的未来国外很有可能为了限制国内的相关行业发展限制甚至禁止国内使用，为了避免最坏的情况发生，也正是为了填补国内机械制图方面的空白，我们开发了 DM-Creater 系列产品，目前 DM-Creater 系列产品仅 DM-Pillar 几何内核引擎平台完成开发并发布，DM-Version 显示引擎、DM-Solver 约束求解器、DM-Editor 数字模型编辑器还在开发当中。

DM-Pillar 几何内核引擎---是一个完整的几何建模软件包，适用于开发需要 3D 几何建模功能的应用程序（包括但不限于 CAD/CAE/CAM 等软件系统），上层应用程序可利用求实建模引擎生成实体、曲面和线框等功能。本软件包提供一组面向对象的编程资源，该资源包括全套的高级几何基元、运算和查询功能。其计算精确度能够媲美全球一流水平，后续将根据实际的应用需求开发更好更全面的解决方案。

2　环境依赖

目前产品版本为 V1.0，动态库的编译环境为 Visual Studio 2019，使用时需要在 windows 下 C++17 编译，无其他依赖库。

3　使用入门

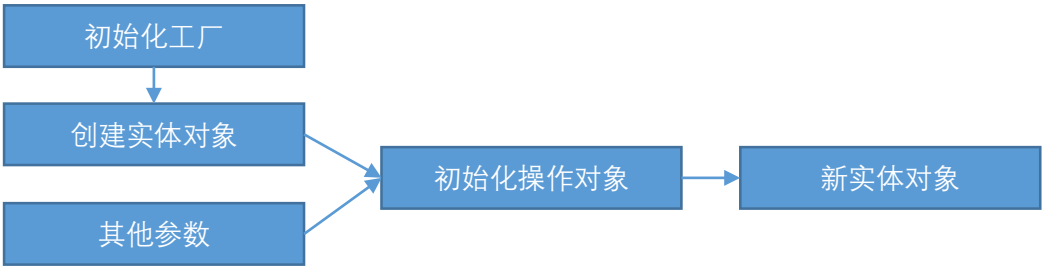

图 1 拓扑操作流程

本产品包含核心 dll 和对应头文件，使用时，直接包含所需头文件，将核心 dll 目录添加到程序即可使用，由于整体架构是以工厂类为核心，可以参看 5.1 创建与删除工厂开始使用。

4 引擎架构

引擎由四个基本模块构成，数学模块，几何模块，拓扑模块，文件导入/导出模块。其中几何模块又分为几何对象模块和几何操作模块，拓扑模块又分为拓扑结构模块和拓扑操作模块。

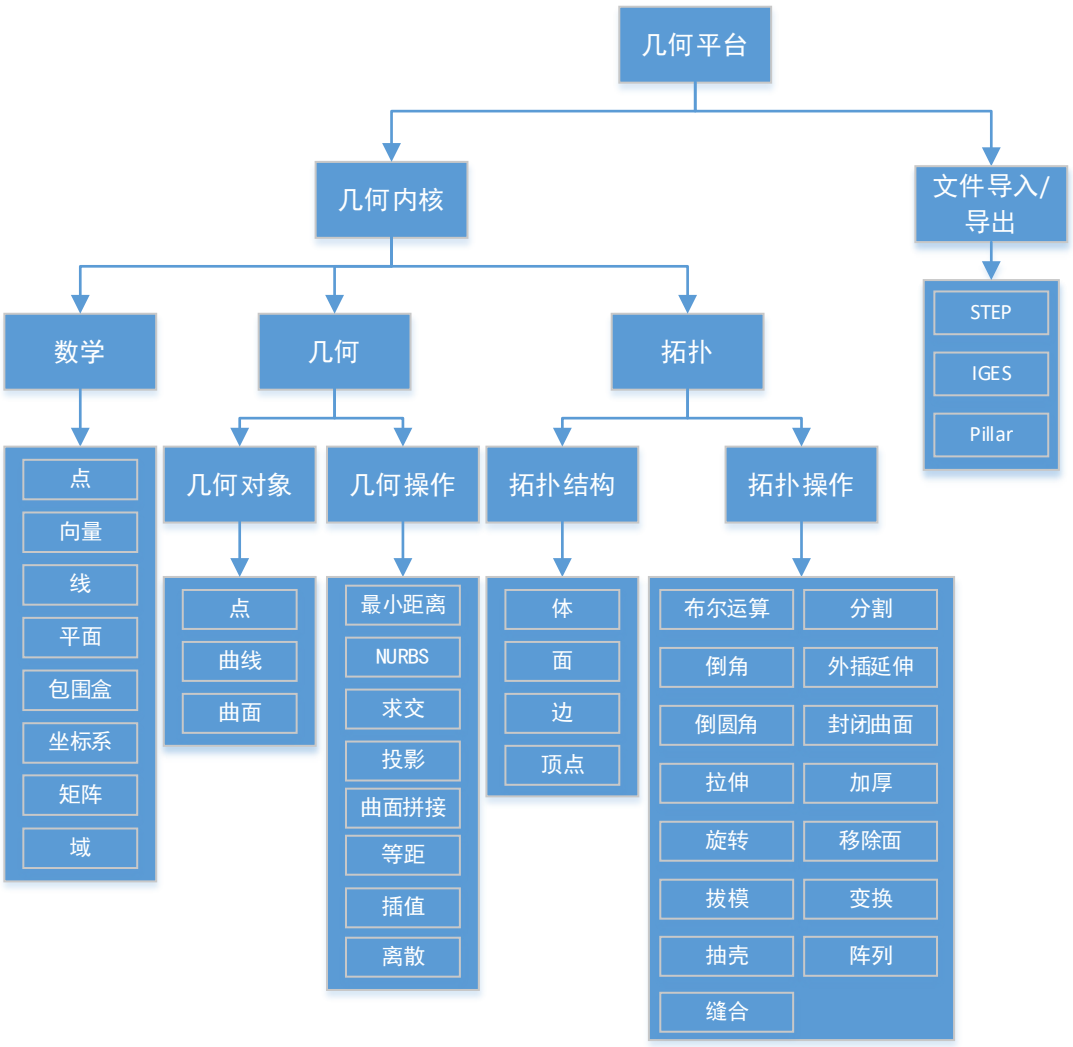

图 2 引擎产品架构

其中几何对象和拓扑结构都继承自 PLRGeometry 类，而 PLRGeometry 的子类对象都由 PLRGeoFactory 工厂类来管理，而数学、

几何操作、拓扑操作、文件导入\导出为单独的模块，由用户自己管理。

5 功能实现

详细描述如何实现各个功能，可适当穿插实际代码来举例。每个大功能分类可根据实际情况拓展小分类。

5.1 创建与释放工厂

为了方便管理和同步修改实体数据，本引擎的所有实体对象都储存在工厂类中进行统一管理，这大大降低了管理难度和发生错误的可能并提高了以后开发新功能的效率。

工厂类由两部分组成，PLRGeoFactory 工厂类，PLRFactory 工厂管理类，PLRFactory 工厂管理类采用单例模式，将保存一个 PLRGeoFactory 工厂类的指针，由于所有几何对象和拓扑结构都由工厂类来管理，所以每次创建实体对象之前必须要初始化工厂类。

调用下面代码将初始化创建工厂：

```
PLRGeoFactory* p_geofactory = PLRFactory::GetFactory();
if (nullptr == p_geofactory) return;
```

初始化工厂后就可以调用工厂中创建实体的方法创建实体，创建的实体对象只能通过工厂的 Remove 方法来移除，当释放工厂的时候，未移除的实体对象将由工厂统一释放。

调用下面代码之后将释放工厂：

```
PLRFactory::closeFactory();
```

5.2 创建简单物体

任何复杂物体其实都可以由简单物体不断拼接而成，但是当物体过于复杂时再使用简单物体拼接反而会使得操作过程更加复杂，

但对于组成产品的部分零件来说可能只是几个简单物体的组合，这个时候有接口可以直接生成简单物体将能加快建模进程。

本引擎提供了几个直接生成简单物体的全局接口，包括立方体、圆柱体、球体、圆环体、棱锥体。这些接口都在 PLRGeoFactory.h 文件中涉及，可以直接生成，要注意的是使用这些接口之前必须要初始化工厂对象，并传入生成参数。

5.2.1 代码示例

以创建立方体为例，可以使用如下代码进行生成：

```
PLRGeoFactory* p_geofactory = PLRFactory::GetFactory();
if (nullptr == p_geofactory) return;
//生成立方体的四个点
PLRMathPoint  pt1(0, 0, 0);
PLRMathPoint  pt2(100, 0, 0);
PLRMathPoint  pt3(0, 100, 0);
PLRMathPoint  pt4(0, 0, 100);
//生成立方体
PLRBody* p_cuboid_body = PLRCreateSolidCube(p_geofactory,pt1,pt2,pt3,pt4);
```

其中 p_geofactory 为工厂对象，pt1-pt2 点的连线是立方体沿 x 轴方向的边，pt1-pt3 点的连线是立方体沿 y 轴方向的边，pt1-pt4 点的连线是立方体沿 z 轴的方向的边。生成的 PLRBody 对象就是立方体对象。

除此之外还有生成圆柱体的接口 PLRCreateSolidCylinder，生成球体的接口 PLRCreateSolidSphere，生成圆环体的接口 PLRCreateSolidTorus，生成棱锥体的接口 PLRCreateSolidPyramid。

全局接口只是为了简化建模流程，通过类的成员函数同样可以生成简单物体。

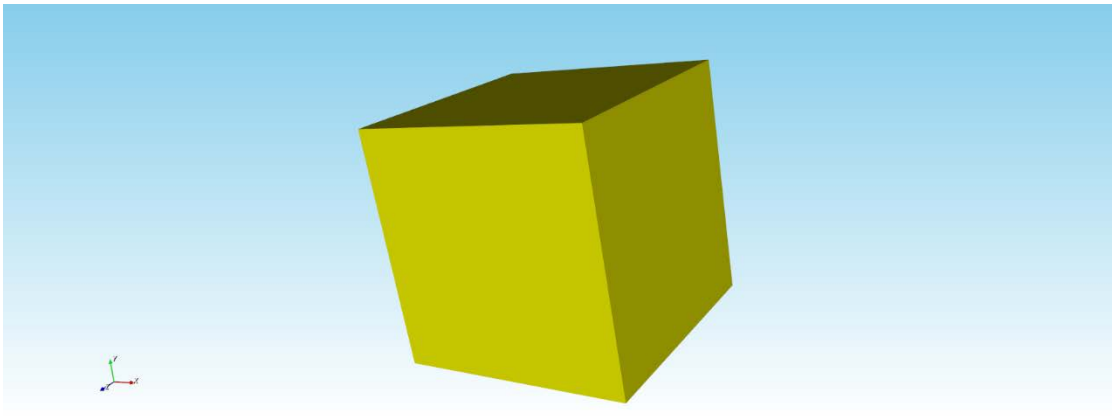

图 3 正方体

5.3 拔模

拔模是一种常用的建模技术，它是通过从一个二维轮廓延伸出一个新的三维体积或者表面来创建三维几何形状。在拔模过程中可以通过改变延伸距离、方向和轮廓形状等参数来控制生成的几何形状，由于拔模较为复杂，所以本引擎中由 PLRTopOpDraft、PLRDraftAngle、PLRDraftVariableAngleData、PLRDraftParams 几个类来共同实现拔模技术。

5.3.1 相关类

PLRDraftAngle 是一个存储了对多个面进行相同拔模操作信息的类，PLRDraftVariableAngleData 是一个存储了对多个面进行多个拔模操作信息的类，PLRDraftParams 是一个存储了拔模方向，中性面，多个面进行多个拔模操作的信息的类，而 PLRTopOpDraft 是实现拔模操作的类，通过传入 PLRDraftParams 定义的拔模操作信息以及添加限制面来达到拔模的效果。

当在执行拔模操作的时候，各个类的生成顺序是 PLRDraftAngle-> PLRDraftVariableAngleData ->PLRDraftParams->PLRTopOpDraft。

5.3.2 步骤

a) 定义拔模面（可定义多个）和拔模角度，生成PLRDraftAngle对象。(可生成多个PLRDraftAngle对象)

b) 将多个PLRDraftAngle对象赋值给PLRDraftVariableAngleData确定拔模的基本设置。

c) 定义拔模方向，中性面，和创建的PLRDraftVariableAngleData一起传入,生成PLRDraftParams对象。

d) 通过将PLRDraftParams对象传入PLRTopOpDraft从而实现拔模操作。

5.3.3 代码示例

可以使用如下代码拔模立方体：

```
PLRGeoFactory* i_factory = PLRFactory::GetFactory();
if (nullptr == i_factory) return;
//创建立方体
PLRMathPoint p1(0, 0, 0);
PLRMathPoint p2(0, 20, 0);
PLRMathPoint p3(40, 0, 0);
PLRMathPoint p4(0, 0, 40);
PLRBody* rectangle_ptr = PLRCreateSolidCube (i_factory, p1, p2, p3, p4);
// 设置拔模面
std::vector<PLRFace*> allfacelist;
rectangle_ptr->GetAllFaces(allfacelist);
PLRFace* face_to_be_drafted = allfacelist[0];
std::vector<PLRFace*> facelist;
```

```
facelist.push_back(face_to_be_drafted);
//设置中性面
PLRFace* neutral_face = allfacelist[1];
//设置拔模角
PLRAngle angle = 40.0;
//创建拔模角度 PLRDraftAngle
PLRDraftAngle * pdraft_angle = NULL;
PLRDraftVariableAngleData * pdraft_ribbon = NULL;
pdraft_angle = new  PLRDraftAngle (facelist, angle);
//创建拔模信息类 PLRDraftVariableAngleData
std::vector<PLRDraftAngle*> faces_and_angles;
faces_and_angles.push_back(pdraft_angle);
pdraft_ribbon = new PLRDraftVariableAngleData (faces_and_angles);
std::vector<PLRDraftVariableAngleData *> ribbons;
ribbons.push_back(pdraft_ribbon);
//设置拔模方向
PLRMathDirection draft_direction(0, 1, 0);
//创建拔模信息类 PLRDraftParams
PLRDraftParams * pdraft_domain = new  PLRDraftParams (draft_direction,
neutral_face,ribbons);
//创建拔模操作
PLRTopOpDraft * draft_ptr = PLRCreateTopOpDraft (i_factory, rectangle_ptr);
draft_ptr->AddDraftParams(pdraft_domain);
draft_ptr->Run();
//获取拔模实体
```

```
PLRBody* body_op2 = draft_ptr->GetResult();
```

选取立方体的一个面做被拔模的面，并设定拔模角度与拔模方向，设置中性面，就可以进行拔模操作了。

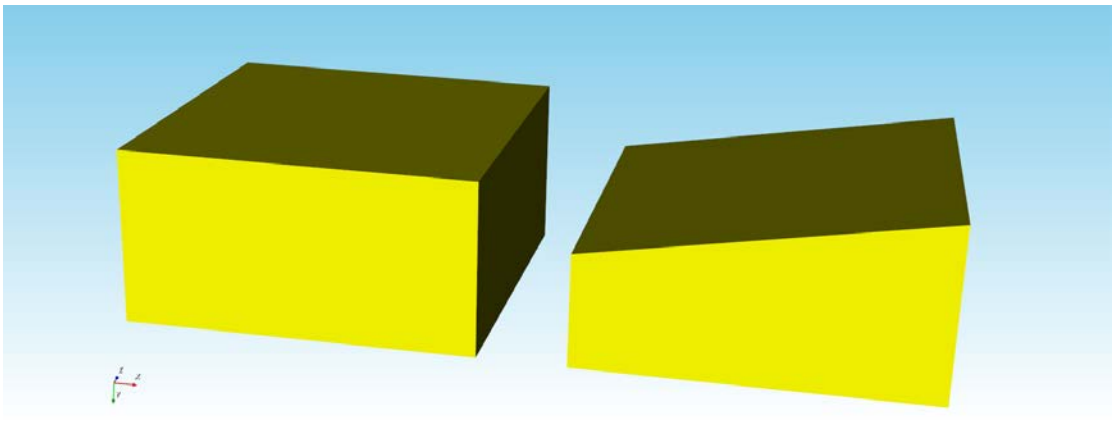

图 4 立方体拔模前后对比

5.4 拉伸

拉伸是建模中一项基本功能，它可以通过将点、曲线、曲面延伸一段线性距离来创建实体或片体或线体，它的优势是简单快速，可以较为直接的得到实体对象，拉伸只需要定义拉伸的方向和起止位置即可成功，同时可以设置限制元素来控制拉伸结果。

5.4.1 代码示例

拉伸使用的类是 PLRTopOpExtrude，可以使用如下代码为例拉伸一个点为一条线：

```
PLRGeoFactory* factory_ptr = PLRFactory::GetFactory();
if (NULL == factory_ptr) return;
//定义拉伸方向
PLRMathDirection Px(1, 0, 0);
//创建点
PLRBody* body_ptr = CreatePointBody(factory_ptr, 0, 0, 0);
//创建拉伸操作
```

```
PLRTopOpExtrude* op_ptr = PLRCreateTopOpExtrude(factory_ptr, body_ptr, &Px, 0,
50);
op_ptr->Run();
//获取拉伸结果
PLRBody* res_body_ptr = op_ptr->GetResult();
```

创建拉伸方向为（1,0,0）方向，创建拉伸点（0,0,0），然后创建拉伸操作从拉伸点沿拉伸方向的 0 点位置起拉伸到 50 位置。

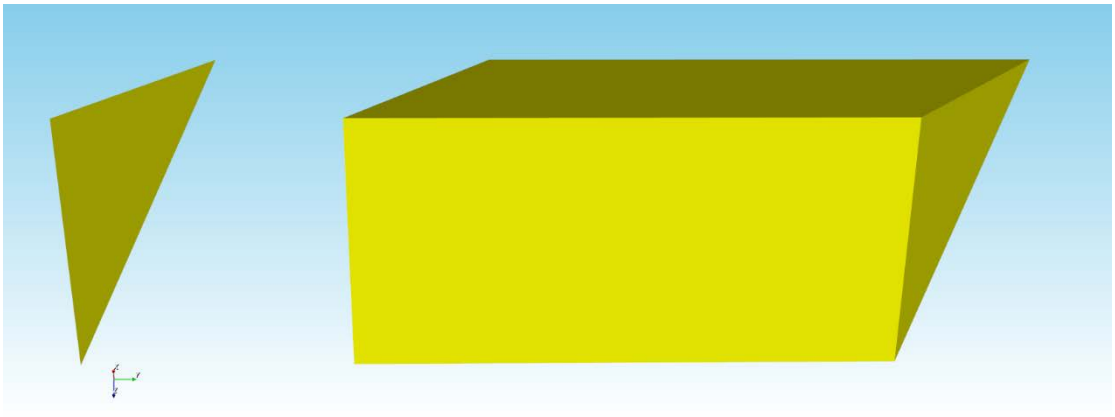

图 5 三角形拉伸前后对比

5.5 旋转

旋转是建模中一项基本功能，它可以通过将曲线、曲面绕旋转轴旋转一定角度来创建实体或片体，旋转只需要定义旋转轴和起始旋转位置，旋转角度即可成功。

5.5.1 代码示例

旋转使用的类是 PLRTopOpRevol，可以使用如下代码为例旋转一个曲面为一个实体：

```
PLRGeoFactory* pi_geofactory = PLRFactory::GetFactory();
if (pi_geofactory == NULL) return;
//定义要旋转的轮廓
PLRMathPoint v0O1(0, 0, 0), v0I1(50, 0, 0), v0J1(0, 30, 0);
PLRLine* piline[3] = {};
```

```
piline[0] = pi_geofactory->CreateLine(v001, v0I1);
piline[1] = pi_geofactory->CreateLine(v0I1, v0J1);
piline[2] = pi_geofactory->CreateLine(v0J1, v001);
PLRLine** pii_line = new PLRLine * [3];
pii_line[0] = piline[0];
pii_line[1] = piline[1];
pii_line[2] = piline[2];
PLRCurve** pi_crvline = (PLRCurve**)pii_line;
//生成旋转轮廓面
PLRBody* pi_boxbody = PLRCreateSkinBody(pi_geofactory, 3, pi_crvline);
//设置旋转轴和旋转角度
PLRMathAxis revoaxis;
revoaxis.Set(PLRMathPoint(0, 0, 0), PLRMathVector(0, 0, 1), PLRMathVector(0, 1,
0),PLRMathVector(-1, 0, 0));
//创建旋转操作
PLRTopOpRevol * pitopre = PLRCreateTopOpRevol (pi_geofactory, pi_boxbody,
&revoaxis, 0, 270);
pitopre->Run();
//获得旋转结果
PLRBody* pi_ret = pitopre->GetResult();
```

创建轮廓为（0,0,0），（50,0,0），（0,30,0）的连线形成的三角形，设置旋转轴为以（0,0,0）为起点，方向指向（-1，0，0）的旋转轴，起始方向为（0,0,1）方向计算旋转 270 度，然后执行旋转操作。

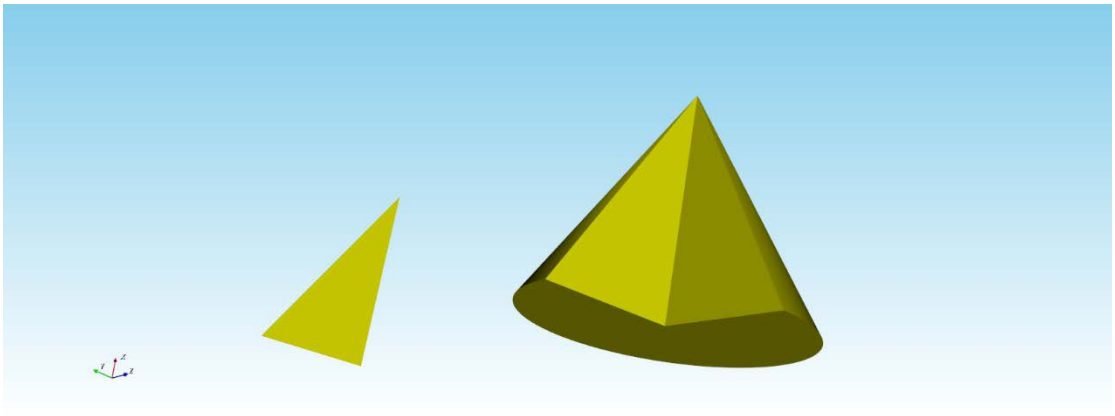

图 6 三角形旋转前后对比

5.6 抽壳

抽壳是在三维实体中创建具有指定厚度的薄壁，通过指定实体的面向内、向外的厚度来实现。如图 5.6-1 所示

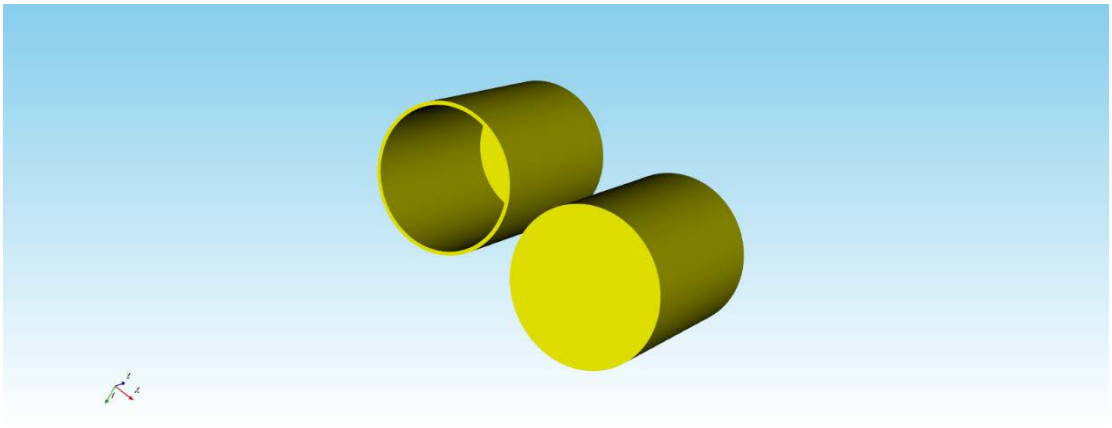

图 7 抽壳前后对比

5.6.1 相关类

a) PLRCreateTopOpSolidShell，创建 PLRTopOpSolidShell，入参要抽壳的体、向内厚度、向外厚度。

b) PLRTopOpSolidShell，抽壳实现类，由 PLRCreateTopOpSolidShell 函数创建，创建后，在设置完相关参数后，调用 Run 接口进行运算，调用 GetResult 获取抽壳后的后果。

c) PLRGeoFactory，工厂类，在调用 PLRCreateTopOpSolidShell 时入参。

d) PLRBody，实体对象类，面所在的实体，在 PLRCreateTopOpSolidShell 时入参。

e) PLRFace 开放面及指定厚度面。

5.6.2 步骤

a) 获取 PLRGeoFactory 实例。

b) 创建被抽壳的实体。

c) 调用 PLRCreateTopOpSolidShell，入参相应参数。

d) 调用 Append 设置开放面及指定厚度面。可选。

e) 调用 Run 运算。

f) 调用 GetResult 获取结果。

5.7 倒角

倒角是对 3D 实体或者面片的边进行斜面化的一种操作，通过设置与支持面和其它面的角度或长度定位其特征。如下图 5.7-1 所示

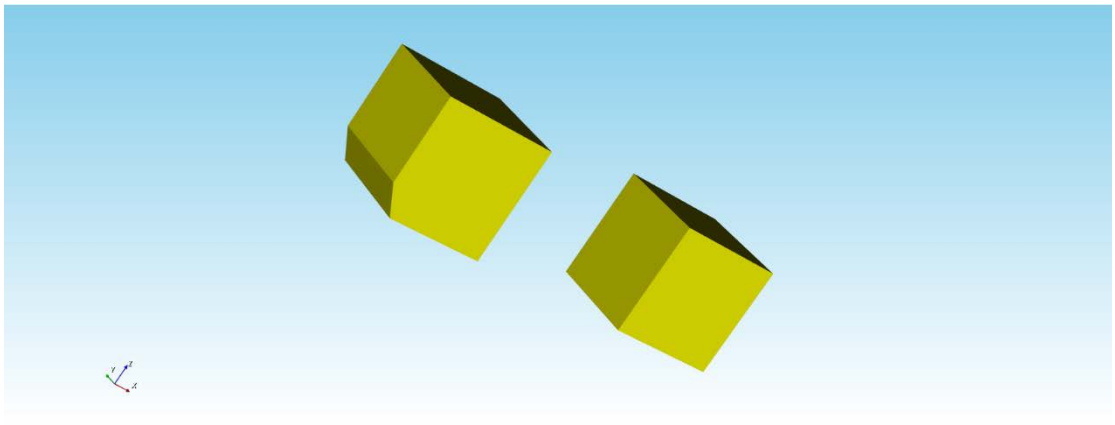

图 8 倒角前后对比

5.7.1 相关类

a) PLRCreateTopOpChamfer，创建倒角操作实例的 API，入参相关参数，创建 PLRTopOpChamfer 类。

b) PLRTopOpChamfer，倒角操作的参数设置、运算类。设置倒角区域，执行运算，获取结果。

c) PLRChamferParams，倒角区域类，定义边、面、倒角值、传播类型。

d) PLREdge ，被倒角的边。

e) PLRFace，被倒角边的支持面。

f) PLRTopChamferType，倒角参数类型，参数类型定义值参数的长度、角度。

g) PLREdgePropagationMode，倒角操作的传播类型。

5.7.2 操作步骤

a) 获取 PLRGeoFactory 实例

b) 创建实体

c) 创建倒模区域

d) 调用 PLRCreateTopOpChamfer，生成 PLRTopOpChamfer

e) 调用 Run 执行运算

f) 调用 GetResult 生成拔模结果

5.8 倒圆角

倒圆角是对 3D 实体的棱边进行圆角化处理的一种操作。可以通过对指定的边、面面区域、三切线倒角进行操作。如图 9，图 10，图 11。

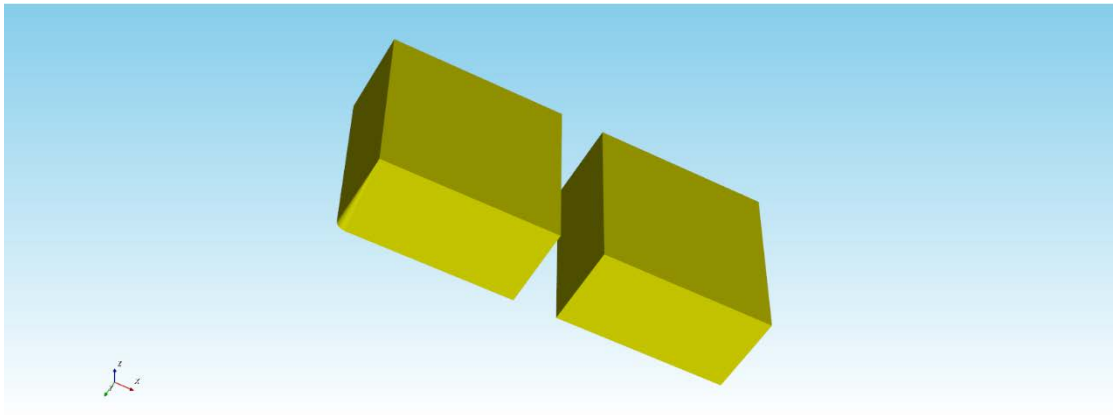

图 9 边倒圆角前后对比

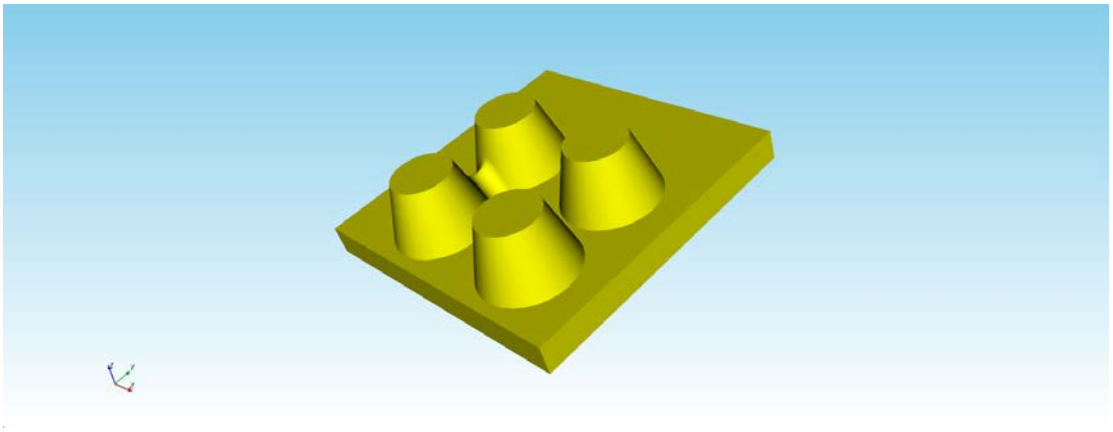

图 10 面面倒圆角前后对比

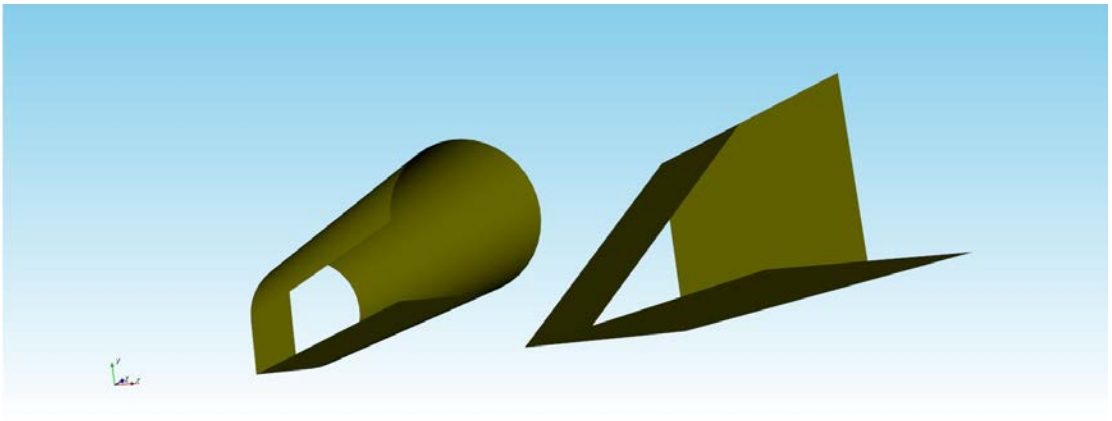

图 11 三切线倒圆角前后对比

5.8.1 相关类

a) PLRCreateTopOpFillet，创建 PLRTopOpFillet 实例的 API，入参相关参数，创建 PLRTopOpFillet 实例。

b) PLRTopOpFillet，倒圆角操作的参数设置、运算类。设置倒角区域，执行运算，获取结果。通过 Append 接口，设置基于边、基于面面、基于三切线的倒圆角操作。

c) PLREdgeFilletParams，定义了基于边的倒圆角操作，对边进行倒圆角，设置倒圆角的边、倒圆的参数、保留边、传播模式、倒圆角边类型。

d) PLRFilletVariableRadius，定义圆角半径、圆角边、倒

圆角位置、移除边等。

e） PLRFaceFaceFilletParams，定义了基于面面的倒圆角，在具有相同底座的凸台间创建一个圆弧的过渡。设置面、圆角半径、圆角类型、移除边等。

f） PLRFace ，定义面。

g） PLRFilletVariableRadius，定义弧半径、边。

h） PLREdge ，定义移除边。

i） PLRRemoveFaceFilletParams，定义了基于三切线倒圆角的操作，选择相对的两面和顶面，运算出一个与三个面都相切的圆弧面，并移除顶面。PLRFace 定义两相对面、顶面。

5.8.2 操作步骤

a）获取 PLRGeoFactory 实例。

b）创建实体。

c）创建 PLRTopOpFillet。

d）创建操作，PLREdgeFilletParams、PLRFaceFaceFilletParams、PLRRemoveFaceFilletParams 中的一种。

e） 调用 PLRTopOpFillet 的 Append 入参操作。

f） 调用 PLRTopOpFillet 的 Run 执行运算。

g） 调用 PLRTopOpFillet 的 GetResult 生成结果。

5.9 分割

分割是用一种实体分割另一种实体的操作，可以选择保留两部分中的一部分，也可两部分都保留。如图 5.9-1。

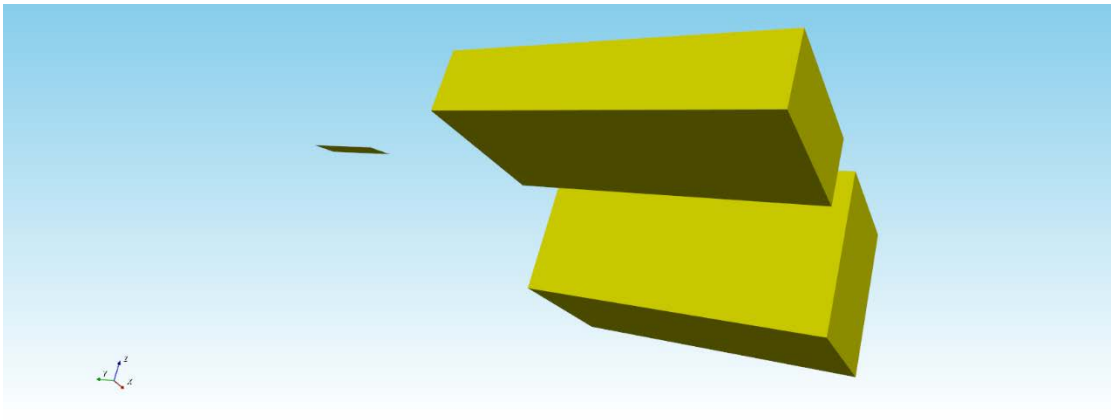

图 12 切割前后对比

5.9.1 相关类

a) PLRCreateTopOpSolidSplit，创建 PLRTopOpSolidSplit 的 API，入参被分割实体、保留的部分。

b) PLRTopOpSolidSplit，分割类，入参分割体及其它参数。

c) PLRBody，分割实体与被分割实体。

5.9.2 操作步骤

a) 获取 PLRGeoFactory 实例

b) 创建被分割实体、分割实体。

c) 创建 PLRTopOpSolidSplit，入参被分割实体、分割实体。

d) 调用 PLRTopOpSolidSplit 的 SetSplit 入参分割实体。

e) 调用 PLRTopOpSolidSplit 的 Run 执行运算。

f) 调用 PLRTopOpSolidSplit 的 GetResult 生成结果。

5.10 加厚

加厚是对实体在面上进行的延伸或者压缩，还可以指定面的厚度。如图 5.10-1。

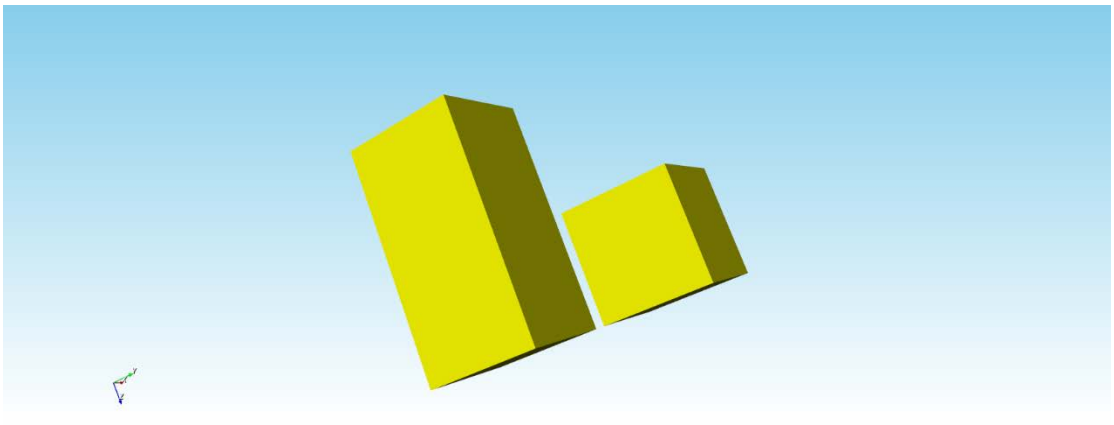

图 13 加厚前后对比

5.10.1 相关类

a) PLRCreateTopOpSolidThicken ，创建 PLRTopOpSolidThicken 的 API，入参相关参数。

b) PLRTopOpSolidThicken，加厚操作的参数设置、运算类。

c) PLRBody ，实体，加厚操作是在该体的面上进行。

d) PLRFace ，指定厚度的面。

5.10.2 操作步骤

a) 获取 PLRGeoFactory 实例

b) 创建被拉伸面的实体。

c) 创建 PLRTopOpSolidThicken。

d) 调用 PLRTopOpSolidThicken 的 Append 入参特殊加厚面。

e) 调用 PLRTopOpSolidThicken 的 Run 执行运算。

f) 调用 PLRTopOpSolidThicken 的 GetResult 生成结果。

5.11 厚曲面

厚曲面是把一个没有厚度的实体加厚成有厚度的实体，也就是把曲面转化为实体，需要加厚的曲面的偏移距离分为第一偏移和第二偏移，分别对应内偏移和外偏移，可同时有值，可仅一个有值。通过类 PLRTopOpThickenSkin 来实现。

可以通过全局函数 PLRCreateTopOpThickenSkin 创建一个加厚曲面的操作，示例代码：

```
/// 初始化工厂
PLRGeoFactory* factory_ptr = PLRFactory::GetFactory();
//获取工厂指针失败与否的判断
if (NULL == factory_ptr) return;
/// 创建一个封闭的矩形面
PLRMathPoint P1(0, 0, 0);
PLRMathPoint P2(0, 10, 0);
PLRMathPoint P3(0, 0, 15);
PLRMathPoint P4(0, 10, 15);
PLRCurve* line1 = (PLRCurve*)factory_ptr->CreateLine(P1, P2);
PLRCurve* line2 = (PLRCurve*)factory_ptr->CreateLine(P1, P3);
PLRCurve* line3 = (PLRCurve*)factory_ptr->CreateLine(P2, P4);
PLRCurve* line4 = (PLRCurve*)factory_ptr->CreateLine(P3, P4);
//创建线段数组
PLRCurve** i_curves = new PLRCurve * [4];
i_curves[0] = line1;
i_curves[1] = line2;
i_curves[2] = line3;
i_curves[3] = line4;
//通过封闭曲线创建一个面的片体
PLRBody* body_ptr = PLRCreateSkinBody(factory_ptr, 4, i_curves);
PLRTopOpThickenSkin* p_thick = PLRCreateTopOpThickenSkin(body_ptr, 8, 15);
p_thick->Run();
PLRBody* body_ret = p_thick->GetResult();
```

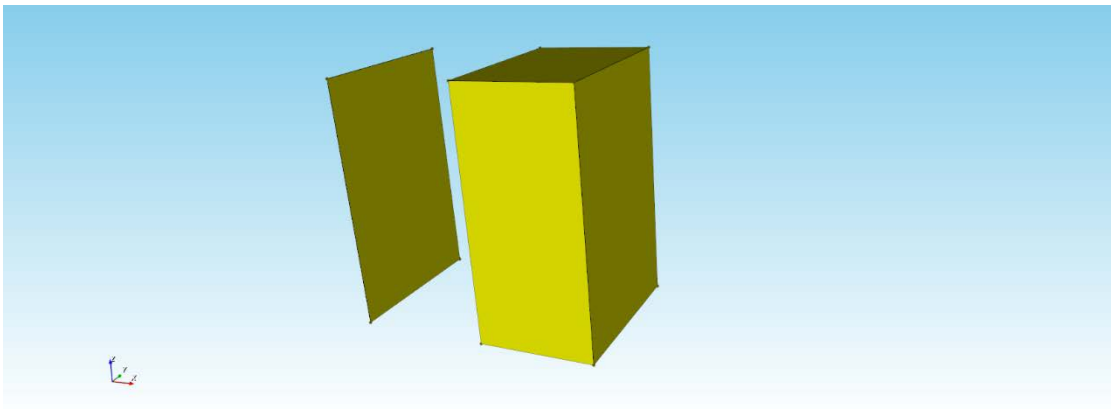

图 14 加厚平面前后对比

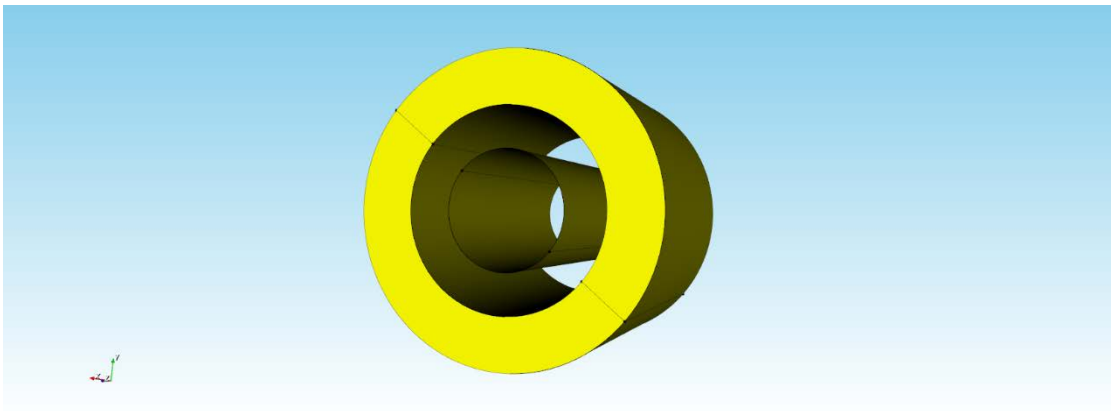

图 15 加厚圆柱面前后对比

5.12 封闭

封闭操作是将一个封闭的曲面生成一个实体，将不封闭的曲面以线性方式生成实体。该操作通过类 PLRTopOpClose 来实现。

可以通过全局函数 PLRCreateTopOpClose 来创建封闭实体，示例代码：

```
//获取工厂指针
PLRGeoFactory* p_geofactory = PLRFactory::GetFactory();
//获取工厂指针失败与否的判断
if (nullptr == p_geofactory) return;
//创建局部坐标系
PLRMathAxis cylinder;
double cylinder_radius = 10.;
double cylinder_axis_start = 0.;
```

```
double cylinder_axis_end = 50.;
double cylinder_angle_start = 0.;
double cylinder_angle_end = 2 * PLRPI;
//设置局部坐标系
cylinder.Set(PLRMathPoint(0, 0, 0), PLRMathVector(10, 10, 10), PLRMathVector(1,
10, 0), PLRMathVector(10, 10, 0));
//创建圆柱面
PLRCylinder* pi_cylinder = p_geofactory->CreateCylinder(cylinder,
cylinder_radius, cylinder_axis_start, cylinder_axis_end, cylinder_angle_start,
cylinder_angle_end);
PLRBody* cylinder_body = PLRCreateSkinBody(p_geofactory,
(PLRSurface*)pi_cylinder);
PLRTopOpClose* closed_ptr = PLRCreateTopOpClose(p_geofactory, cylinder_body);
closed_ptr->Run();
PLRBody* body_ret = closed_ptr->GetResult();
```

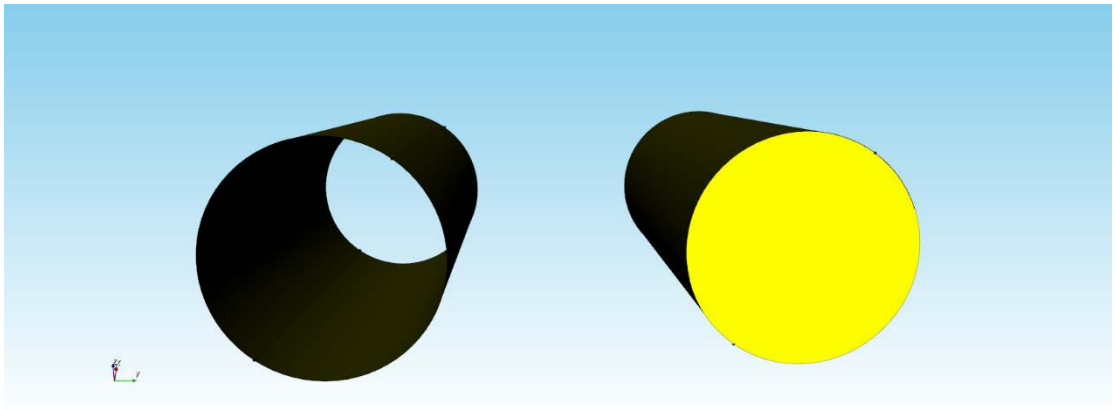

图 15 封闭前后对比

5.13 填充

填充操作是将一个封闭的轮廓填充成一个面，该操作通过类 PLRTopOpFill 来实现。

可以通过全局函数 PLRCreateTopOpFill 来实现。

示例代码：

```
PLRGeoFactory* p_geofactory = PLRFactory::GetFactory();
```

```
PLRInt number_of_wires = 4;
//定义点和方向
PLRMathPoint d1(25., 25., 0);
PLRMathPoint d2(-25., 25., 0);
PLRMathPoint d3(-25., -25., 0);
PLRMathPoint d4(25., -25., 0);
std::vector<PLRMathPoint> d;
d.push_back(d1);
d.push_back(d2);
d.push_back(d3);
d.push_back(d4);
d.push_back(d1);
PLRBody* p_array_of_body_wires[4];
int i = 0;
for (i = 0; i < number_of_wires; i++)
{
    PLRLine* p_line = p_geofactory->CreateLine(d[i], d[i + 1]);
    PLRCurve* p_curve = (PLRCurve*)p_line;
    short res[1] = { 1 };
    PLRBody* p_profile = PLRCreateWireBody(p_geofactory, p_curve, 1);
    p_array_of_body_wires[i] = p_profile;
}
PLRTopOpFill* ptr_topological =
PLRCreateTopOpFill(p_geofactory,number_of_wires,(const
PLRBody**)&p_array_of_body_wires, NULL);
```

```
ptr_topological->Run();

PLRBody* p_result_body_top_fill = ptr_topological->GetResult();
```

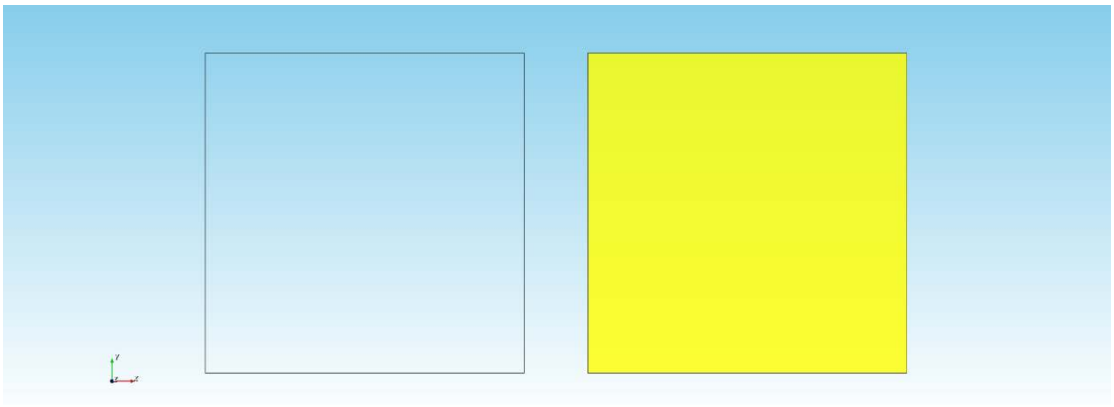

图 16 填充前后对比

5.14 求交

求交是求曲线或者曲面与另一个曲线或者曲面的交点，分为三种：

曲线与曲线求交(PLRGeoOpIntersectionCrvCrv 类)，用全局函数 PLRCreateIntersectionCrvCrv 创建；

曲线与曲面求交(PLRGeoOpIntersectionCrvSur 类)，用全局函数 PLRCreateIntersectionCrvSur 创建；

曲面与曲面求交(PLRGeoOpIntersectionSurSur 类)，用全局函数 PLRIntersectionSurSur 创建。

5.14.1 曲线与曲线求交

代码示例：

```
//获取工厂指针

PLRGeoFactory* geo_factory = PLRFactory::GetFactory();

//获取工厂指针失败与否的判断

if (nullptr == geo_factory) return;

//创建直线

PLRMathPoint p1(0, 0, 0);
```

```
PLRMathPoint p2(0, 10, 0);
PLRLine* line_ptr = geo_factory->CreateLine(p1, p2);
PLRBody* line_body = PLRCreateWireBody(geo_factory, line_ptr, 1);
PLRMathPlane plane;
PLRCircle* curve_ptr = geo_factory->CreateCircle(5, plane,0,PLRPI);
PLRBody* curve_body = PLRCreateWireBody(geo_factory, curve_ptr, 1);
PLRGeoOpIntersectionCrvCrv* inter = PLRCreateIntersectionCrvCrv(geo_factory,
line_ptr, curve_ptr);
int point_num = inter->GetNumberOfPoints();
int curve_num = inter->GetNumberOfCurves();
std::vector<PLRBody*> inter_points;
for (int i = 0; i < point_num; i++) {
    PLRCartesianPoint* inter_point = inter->GetCartesianPoint(i);
    PLRBody* inter_body = CreatePointBody(geo_factory, inter_point);
    inter_points.push_back(inter_body);
}
```

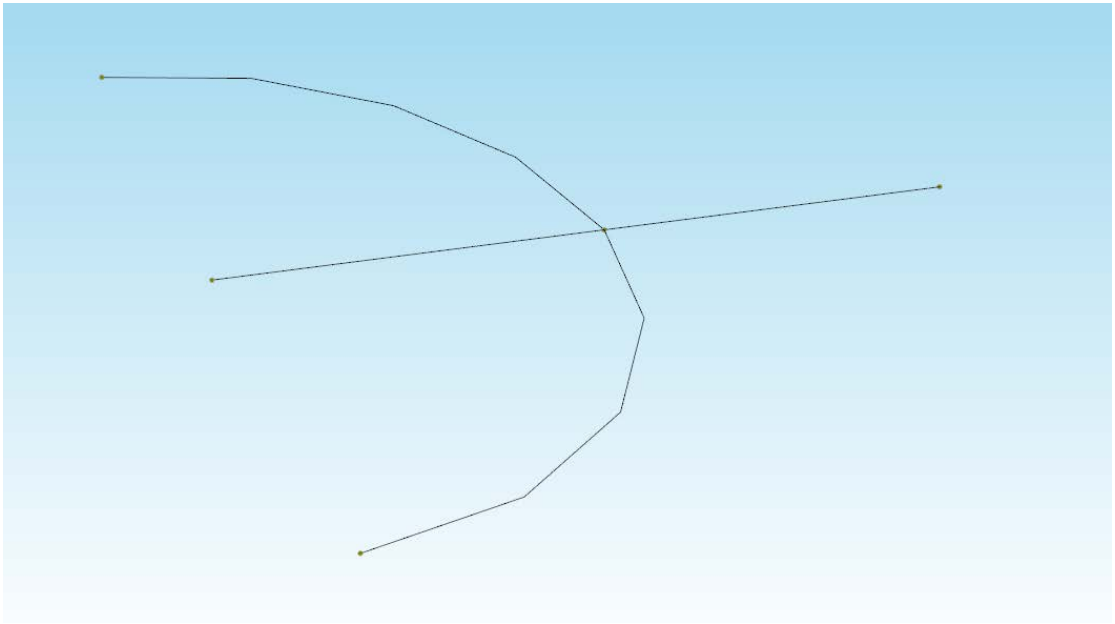

图 17 曲线与曲线求交结果为点

5.14.2 曲线与曲面求交

代码示例：

```
//获取工厂指针
PLRGeoFactory* geo_factory = PLRFactory::GetFactory();
if (nullptr == geo_factory) return;
//创建正方形
PLRMathPoint p1(-10, 5, 0);
PLRMathPoint p2(10, 5, 0);
PLRMathDirection py(0, 0, 1);
PLRLine* line_ptr = geo_factory->CreateLine(p1, p2);
PLRBody* line_body = PLRCreateWireBody(geo_factory, line_ptr, 1);
//拉伸操作
PLRTopOpExtrude* op_ptr = PLRCreateTopOpExtrude(geo_factory, line_body, &py,-
10,10);
op_ptr->Run();
PLRBody* square_body = op_ptr->GetResult();
delete op_ptr;
op_ptr = NULL;
std::vector<PLRFace*> square_sur;
square_body->GetAllFaces(square_sur);
//创建半圆线
PLRMathPlane plane;
PLRCircle* curve_ptr = geo_factory->CreateCircle(10, plane,0,PLRPI);
PLRBody* curve_body = PLRCreateWireBody(geo_factory, curve_ptr, 1);
//求交
```

```
PLRGeoOpIntersectionCrvSur* inter = PLRCreateIntersectionCrvSur(geo_factory,
curve_ptr, square_sur.at(0)->GetSurface());
int point_num = inter->GetNumberOfPoints();
int curve_num = inter->GetNumberOfCurves();
std::vector<PLRBody*> inter_points;
for (int i = 0; i < point_num; i++) {
    PLRCartesianPoint* inter_point = inter->GetCartesianPoint(i);
    PLRBody* inter_body = CreatePointBody(geo_factory, inter_point);
    inter_points.push_back(inter_body);
}
```

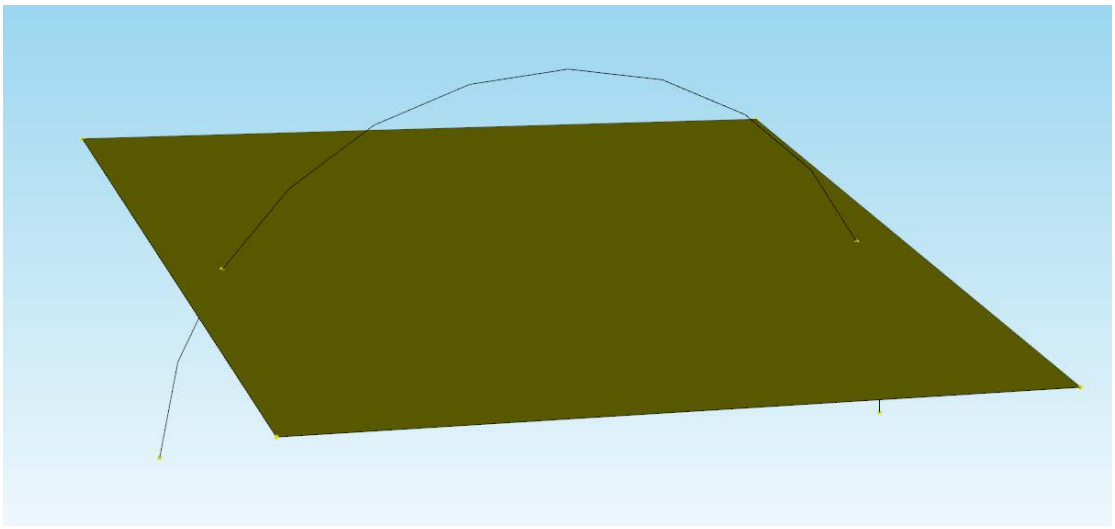

图 18 曲线与曲面求交结果为点

5.14.3 曲面与曲面求交

代码示例：

```
//获取工厂指针
PLRGeoFactory* geo_factory = PLRFactory::GetFactory();
if (nullptr == geo_factory) return;
//创建正方形
PLRMathPoint p1(0, 5, -10);
```

```
PLRMathPoint p2(0, 5, 10);
PLRMathDirection py(0, 1, 0);
PLRLine* line_ptr = geo_factory->CreateLine(p1, p2);
PLRBody* line_body = PLRCreateWireBody(geo_factory, line_ptr, 1);
//拉伸操作
PLRTopOpExtrude* op_ptr = PLRCreateTopOpExtrude(geo_factory, line_body, &py,-
10,10);
op_ptr->Run();
PLRBody* square_body = op_ptr->GetResult();
delete op_ptr;
op_ptr = NULL;
std::vector<PLRFace*> square_sur;
square_body->GetAllFaces(square_sur);
//创建半圆柱面
PLRMathAxis cylinder;
double cylinder_radius = 10.;
double cylinder_axis_start = -10.;
double cylinder_axis_end = 10.;
double cylinder_angle_start = 0.;
double cylinder_angle_end = PLRPI;
cylinder.Set(QiuJiePillar::PLRMathPoint(0, 0, 0), PLRMathVector(10, 10, 10),
PLRMathVector(1, 10, 0), PLRMathVector(10, 10, 0));
PLRCylinder* pi_cylinder =
geo_factory->CreateCylinder(cylinder,cylinder_radius,cylinder_axis_start,cylind
er_axis_end,cylinder_angle_start,cylinder_angle_end);
```

```
PLRBody* pi_body = PLRCreateSkinBody(geo_factory, (PLRSurface*)pi_cylinder);
//求交
PLRGeoOpIntersectionSurSur* inter = PLRCreateIntersectionSurSur(geo_factory,
pi_cylinder, square_sur.at(0)->GetSurface());
int point_num = inter->GetNumberOfPoints();
int curve_num = inter->GetNumberOfCurves();
std::vector<PLRBody*> inter_curves;
for (int i = 0; i < curve_num; i++) {
    PLRCurve* inter_curve = inter->GetCurve(i);
    PLRBody* inter_body = PLRCreateWireBody(geo_factory, inter_curve,1);
    inter_curves.push_back(inter_body);
}
```

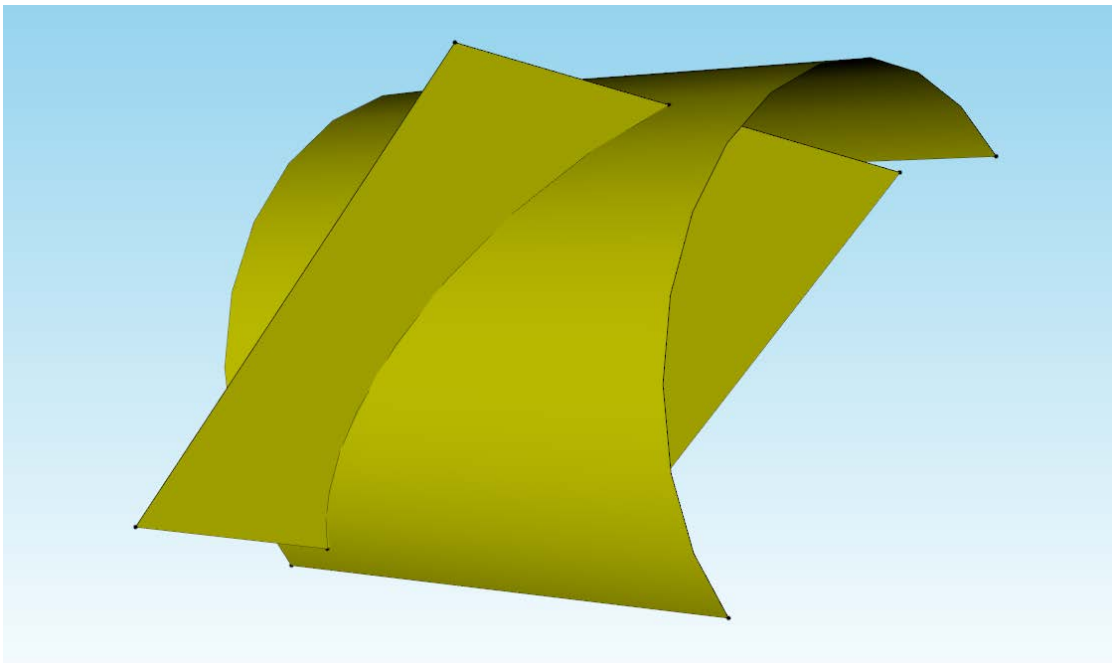

图 19 曲面与曲面求交结果为点

5.15 扫掠

扫掠是一条(或多条)引导线移动一条轮廓线而成的曲面。引导线可以是开放曲线，也可以是闭合曲线。创建扫掠曲面包括：显式

扫掠、直线扫掠、圆式扫掠、二次曲线扫掠、适应性扫掠 5 种扫掠方式。

5.15.1 显式扫掠

显式扫掠定义的一条轮廓线沿着一条或两条引导线运动形成的曲面。应用显示扫掠创建扫掠曲面的方式有两种：使用一条引导线、使用两条引导线。

代码示例：

```
//获取工厂指针
PLRGeoFactory* factory_ptr = PLRFactory::GetFactory();
if (nullptr == factory_ptr) return;
double Parry1[3] = { 0, 0, 0 };
double Parry2[3] = { 10, 0, 0 };
double Parry3[3] = { 10, 20, 0 };
PLRMathSetOfPointsND pointND(3, 3);
pointND.AddPoint(Parry1);
pointND.AddPoint(Parry2);
pointND.AddPoint(Parry3);
PLRCurve* curve_ptr = (PLRCurve*)factory_ptr->CreateSplineCurve(&pointND, 0, 1,
2, NULL);
//创建包含三次样条曲线的线框
PLRBody* profile_ptr = PLRCreateWireBody(factory_ptr, curve_ptr, 1);
PLRMathPoint P1(0, 0, 0);
PLRMathPoint P2(0, 0, 50);
//创建线段
PLRCurve* curve_line_ptr = (PLRCurve*)factory_ptr->CreateLine(P1, P2);
```

```
PLRBody* guide_ptr = PLRCreateWireBody(factory_ptr, curve_line_ptr, 1);
//创建引导线集合和轮廓集合
std::vector<PLRBody*> guide_arry = { guide_ptr };
std::vector<PLRBody*> profile_arry = { profile_ptr };
//声明扫掠操作指针
PLRTopOpSweep* op_ptr = PLRCreateFrFTopologicalSweep(factory_ptr, &guide_arry,
&profile_arry, kPLRFrFTopologicalSweepType_Other);
PLRLimitedFunctionX* angle_ptr =
(PLRLimitedFunctionX*)factory_ptr->CreateConstLimitedFun(0, 360, 50);
op_ptr->SetAngularLawsInDegree(&angle_ptr, 1);
op_ptr->Run();
PLRBody* ret_ptr = op_ptr->GetResult();
delete op_ptr;
op_ptr = nullptr;
```

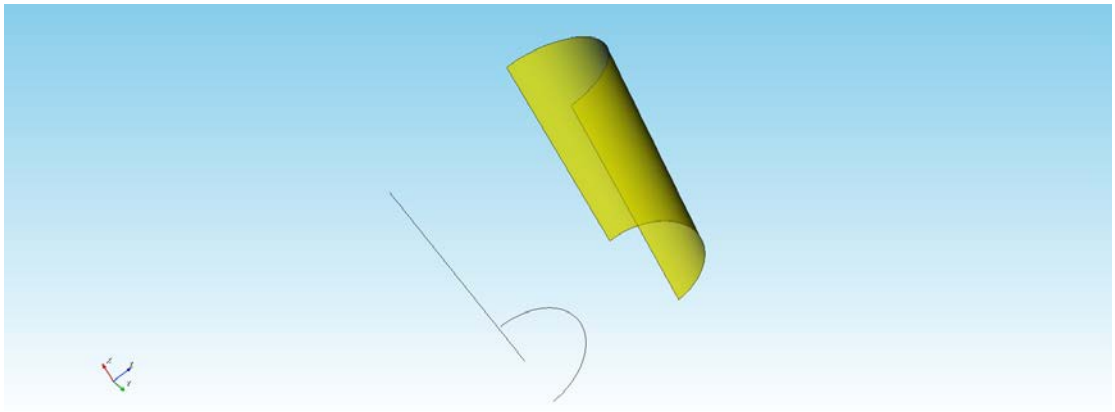

图 20 显式扫掠一条引导线前后对比

5.15.2 直线扫掠

通过直线扫掠方式创建扫掠曲面时，系统自动以直线为轮廓，所以只需要定义引导曲线即可。直线扫掠创建扫路曲面有 6 种方式：两极限、极限和中间、使用参考曲面、使用参考曲线、使用切面、使用双切面。

以两极限为例，该扫掠是以两条曲线作为引导线，两条曲线的端点作为极限，轮廓就是两条曲线的端点的连线

代码示例：

```
//获取工厂指针
PLRGeoFactory* factory_ptr = PLRFactory::GetFactory();
if (nullptr == factory_ptr) return;
double parry1[3] = { 0, 0, 0 };
double parry2[3] = { 0, 0, 20 };
double parry3[3] = { 40, 0, 0 };
PLRMathSetOfPointsND point_nd(3, 3);
point_nd.AddPoint(parry1);
point_nd.AddPoint(parry2);
point_nd.AddPoint(parry3);
//创建三次样条曲线
PLRCurve* curve_ptr = (PLRCurve*)factory_ptr->CreateSplineCurve(&point_nd, 0,
1, 2, NULL);
PLRBody* profile_ptr = PLRCreateWireBody(factory_ptr, curve_ptr, 1);
double parry4[3] = { 0, 180, 20 };
double parry5[3] = { 0, 300, 10 };
PLRMathSetOfPointsND point_nd1(3, 3);
point_nd1.AddPoint(parry1);
point_nd1.AddPoint(parry4);
point_nd1.AddPoint(parry5);
//创建三次样条曲线
```

```
PLRCurve* curve_ptr1 = (PLRCurve*)factory_ptr->CreateSplineCurve(&point_nd1, 0,
1, 2, NULL);
PLRBody* guide_ptr1 = PLRCreateWireBody(factory_ptr, curve_ptr1, 1);
PLRMathPoint p1(40, 0, 0);
PLRMathPoint p2(40, 350, 0);
PLRCurve* curve_ptr2 = (PLRCurve*)factory_ptr->CreateLine(p1, p2);
//创建包含直线的线框
PLRBody* guide_ptr2 = PLRCreateWireBody(factory_ptr, curve_ptr2, 1);
//创建引导线集合和轮廓集合
std::vector<PLRBody*> guide_arry = { guide_ptr1 , guide_ptr2 };
std::vector<PLRBody*> profile_arry = { profile_ptr };
std::vector<PLRGeometry*> limit_arry = { guide_ptr1 , guide_ptr2 };
//声明扫掠操作指针
PLRTopOpSweep* op_ptr = PLRCreateFrFTopologicalSegmentSweep(factory_ptr,
&limit_arry);
//两极限的设置
std::vector<PLRGeometry*> limit_guide1 = { guide_ptr1 };
std::vector<PLRGeometry*> limit_guide2 = { guide_ptr2 };
//设置引导曲线
op_ptr->SetLimitGuides(&limit_guide1);
//设置参考曲线
op_ptr->SetFunctionalGuides(&limit_guide2);
op_ptr->Run();
PLRBody* ret_ptr = op_ptr->GetResult();
//释放扫掠操作指针
```

```
delete op_ptr;

op_ptr = nullptr;
```

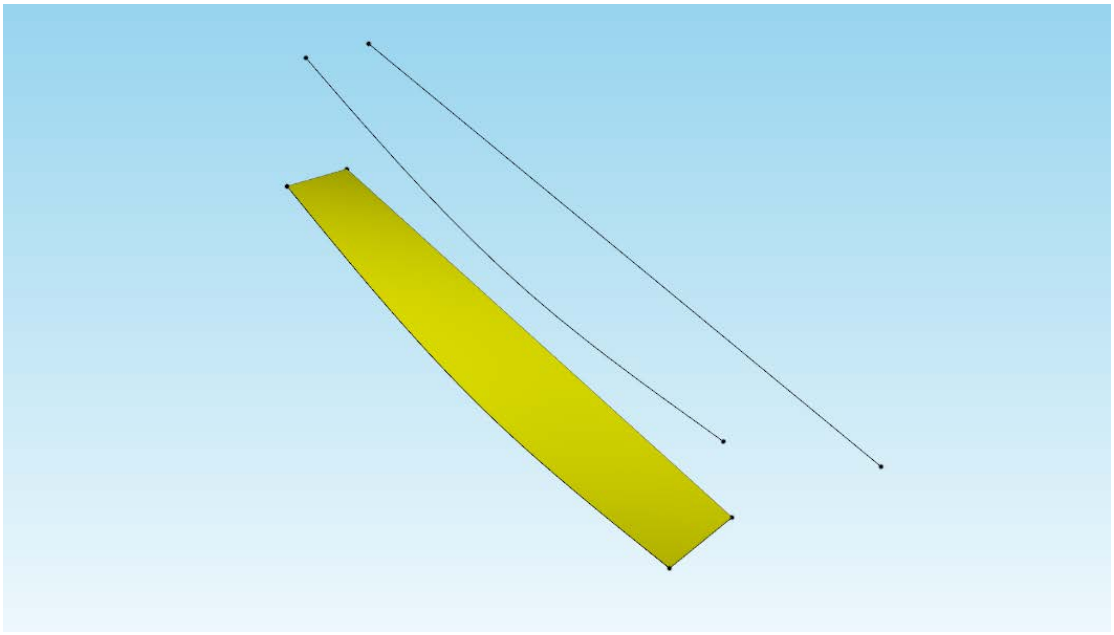

图 21 直线扫掠两极限前后对比

5.15.3 圆弧扫掠

使用圆弧扫掠曲面时，系统自动以圆弧作为轮廓曲线，所以只需指定引导曲线即可。包括：三条引导线、两个点和半径、中心和两个角度、圆心和半径、两条引导线和切面、一条引导线和切面、限制曲线和切面。

以三条引导线为例，三条引导线是通过三条引导线，以圆弧作为轮廓曲线创建出来的扫掠。

代码示例：

```
//获取工厂指针

PLRGeoFactory* factory_ptr = PLRFactory::GetFactory();

if (nullptr == factory_ptr) return;

double parry1[3] = { 0, 0, 0 };

double parry2[3] = { 0, 180, 20 };
```

```
double parry3[3] = { 0, 300, 10 };
double parry4[4] = { 10, 0, 20 };
double parry5[5] = { 20, 100, 40 };
double parry6[5] = { 30, 290, 40 };
PLRMathSetOfPointsND point_nd(3, 3);
point_nd.AddPoint(parry1);
point_nd.AddPoint(parry2);
point_nd.AddPoint(parry3);
PLRMathSetOfPointsND point_nd1(3, 3);
point_nd1.AddPoint(parry4);
point_nd1.AddPoint(parry5);
point_nd1.AddPoint(parry6);
PLRCurve* curve1_ptr = (PLRCurve*)factory_ptr->CreateSplineCurve(&point_nd, 0,
1, 2, NULL);
PLRBody* guide_curve1_ptr = PLRCreateWireBody(factory_ptr, curve1_ptr, 1);
PLRCurve* curve2_ptr = (PLRCurve*)factory_ptr->CreateSplineCurve(&point_nd1, 0,
1, 2, NULL);
PLRBody* guide_curve2_ptr = PLRCreateWireBody(factory_ptr, curve2_ptr, 1);
PLRMathPoint p1(40, 0, 0);
PLRMathPoint p2(40, 350, 0);
PLRCurve* curve_line_ptr = (PLRCurve*)factory_ptr->CreateLine(p1, p2);
PLRBody* guide_line_ptr = PLRCreateWireBody(factory_ptr, curve_line_ptr, 1);
std::vector<PLRGeometry*> limit_arry = { guide_curve1_ptr , guide_curve2_ptr,
guide_line_ptr };
//圆弧扫掠
```

```
PLRTopOpSweep* op_ptr = PLRCreateFrFTopologicalCircleSweep(factory_ptr,
&limit_arry);
op_ptr->Run();
PLRBody* ret_ptr = op_ptr->GetResult();
delete op_ptr;
op_ptr = nullptr;
```

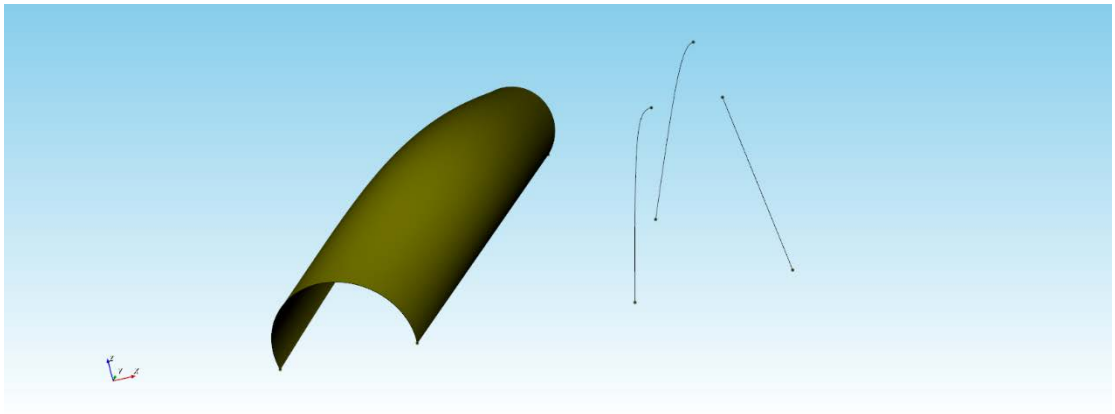

图 22 圆弧扫掠三条引导线前后对比

5.15.4 二次曲线扫掠

使用二次曲线扫掠创建扫掠曲面时，系统自动以二次曲线作为轮廓曲线，所以只需要指定引导曲线即可。子类型包括：两条引导曲线、三条引导曲线。

以两条引导线为例，两条引导曲线是通过两条定义的引导曲线以二次曲线为轮廓创建出扫掠曲面。

代码示例：

```
//获取工厂指针
PLRGeoFactory* factory_ptr = PLRFactory::GetFactory();
if (nullptr == factory_ptr) return;
//创建一个平面
PLRMathPoint p1(10, 0, 0);
PLRMathPoint p2(10, 0, 20);
```

```
PLRMathDirection px(1, 0, 0);
PLRCurve* curve_line_ptr = (PLRCurve*)factory_ptr->CreateLine(p1, p2);
PLRBody* line_body_ptr = PLRCreateWireBody(factory_ptr, curve_line_ptr, 1);
PLRBody* line_face_body_ptr = CreateExtrudesOperator(factory_ptr,
line_body_ptr, &px, 0, 30);
ManageTool::AddGeometry(line_face_body_ptr);
std::vector<PLRGeometry*> body_arry;
PLRBody* curve_face_body_ptr = nullptr;
PLRBody* line_body_ptr2 = nullptr;
//创建一个曲面
double parry1[3] = { 5, 0, 0 };
double parry2[3] = { 3, 5, 5 };
double parry3[3] = { 4, 7, 15 };
PLRMathSetOfPointsND point_nd(3, 3);
point_nd.AddPoint(parry1);
point_nd.AddPoint(parry2);
point_nd.AddPoint(parry3);
PLRCurve* curve_ptr = (PLRCurve*)factory_ptr->CreateSplineCurve(&point_nd, 0,
1, 2, NULL);
PLRBody* curve_body_ptr = PLRCreateWireBody(factory_ptr, curve_ptr, 1);
PLRMathDirection paxis(-15, 20, 0);
curve_face_body_ptr = CreateExtrudesOperator(factory_ptr, curve_body_ptr,
&paxis, 0, 30);
body_arry = { line_body_ptr, curve_body_ptr };
//创建扫掠
```

```
PLRTopOpSweep* op_ptr = PLRCreateFrFTopologicalConicSweep(factory_ptr,
&body_arry);
//设置切面和角度
PLRLimitedFunctionX* angle_ptr =
(PLRLimitedFunctionX*)factory_ptr->CreateConstLimitedFun(0, 360, 0);
op_ptr->SetLimitGuideSlopeCondition(1, line_face_body_ptr, angle_ptr);
op_ptr->SetLimitGuideSlopeCondition(2, curve_face_body_ptr, angle_ptr);
//设置张度
PLRLimitedFunctionX* tens_ptr =
(PLRLimitedFunctionX*)factory_ptr->CreateConstLimitedFun(0, 1, 0.5);
op_ptr->SetTensLaw(tens_ptr);
op_ptr->Run();
PLRBody* ret_ptr = op_ptr->GetResult();
delete op_ptr;
op_ptr = nullptr;
```

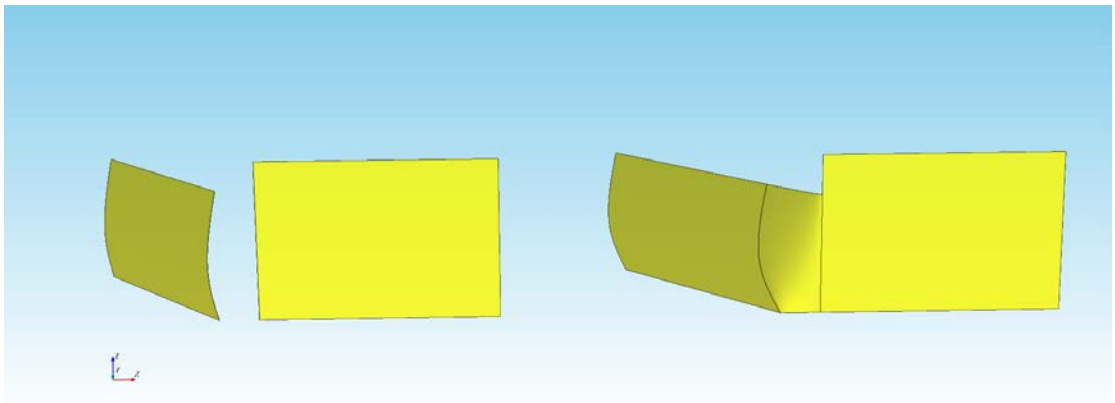

图 23 二次曲线扫掠两条引导线前后对比

5.16 投影

投影是根据空间某一物体对某面的投影图来研究该物体在空间的形状, 大小和位置

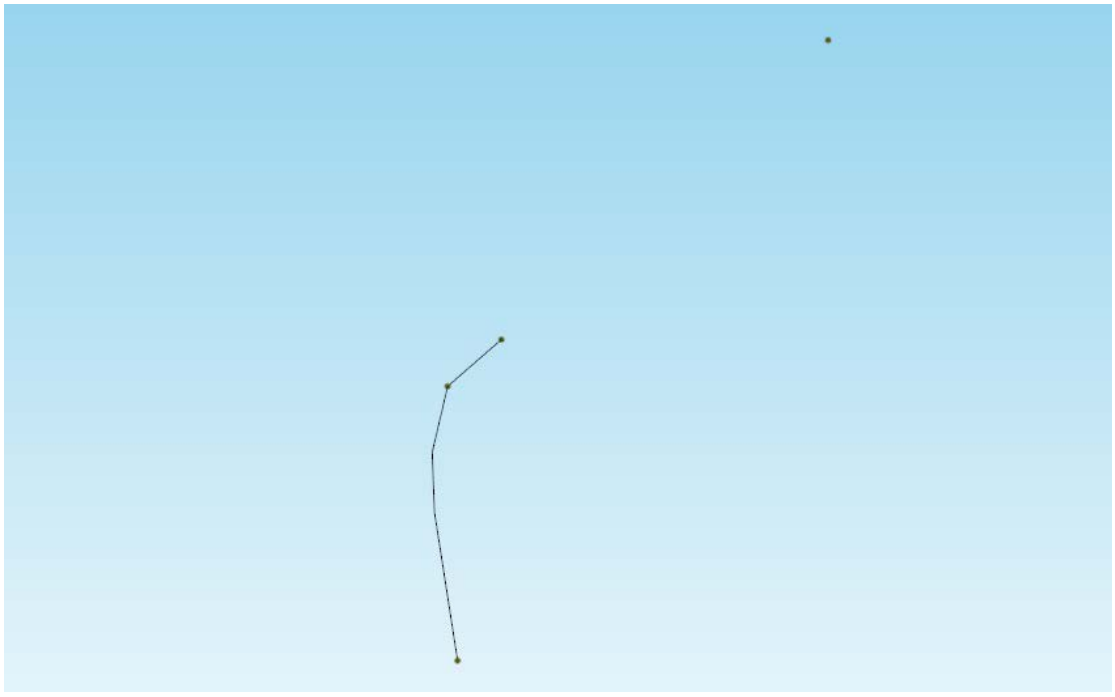

图 24 点投影到线 交叉点为投影点

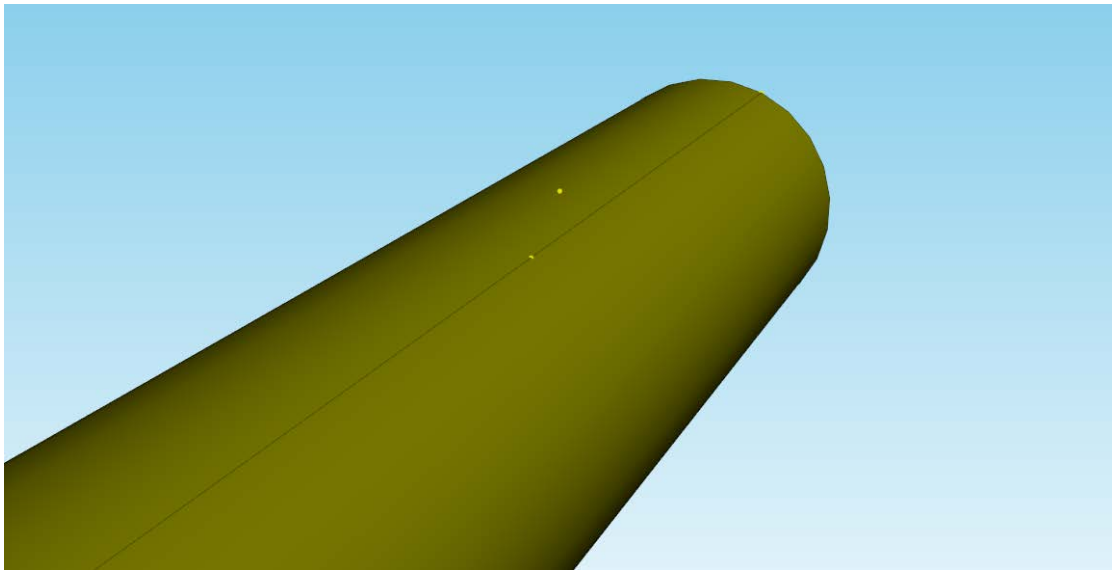

图 25 点投影到曲面

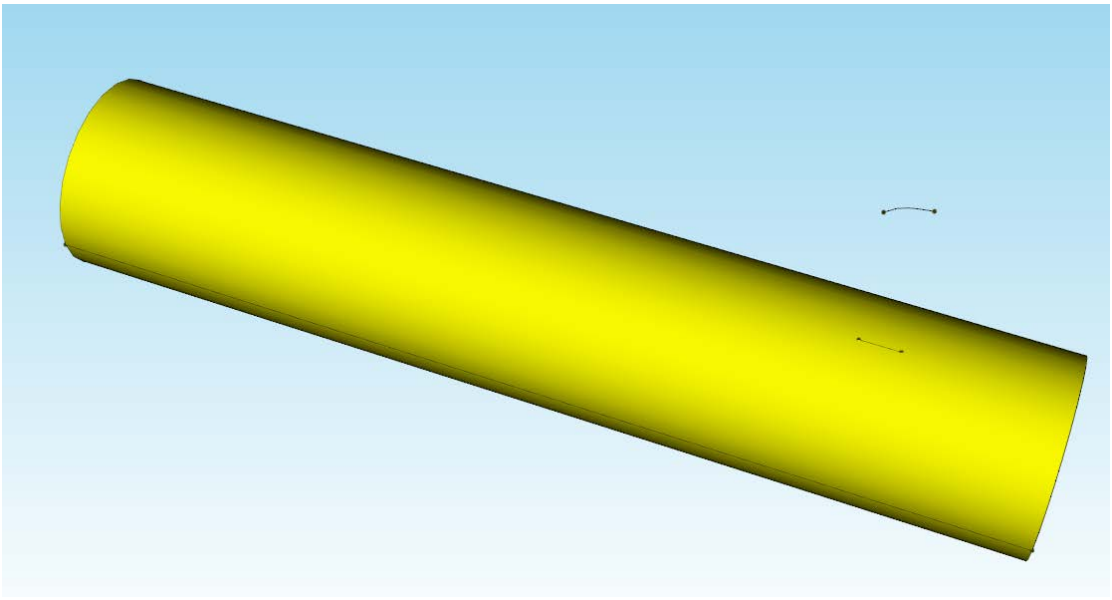

图 26 曲线投影到曲面

5.16.1 点到线的投影相关类

a) PLRCreateProjection 创建 PLRGeoOpProjectionPtCrv，创建倒角操作实例的 API，入参相关参数，创建 PLRGeoOpProjectionPtCrv 实例

b) PLRGeoOpProjectionPtCrv 点投影到线上的操作，获取投影点，设置点到曲线的最大距离，获取点集的数量

c) PLRPoint 要投影的点

d) PLRCurve 点要投影的曲线

5.16.2 操作步骤

a) 创建投影点

b) 创建要被投影的曲线

c) 创建 PLRGeoOpProjectionPtCrv 传入点跟曲线跟据参数设置是否调用 Run 函数

d) 调用 GetCartesianPoint 获取曲线上的投影点

5.16.3 点在面上的投影的相关类

a) PLRCreateProjection 创建 PLRGeoOpProjectionPtSur

实例的 API,入参相关参数，创建 PLRGeoOpProjectionPtSur 实例

b) PLRGeoOpProjectionPtCrv 点投影到线上的操作，获取投影点，设置点到曲线的最大距离，获取点集的数量等

c) PLRPoint 要投影的点

d) PLRCurve 点要投影的曲线

5.16.4 点再面上投影操作步骤

a) 创建投影点

b) 创建要被投影的曲线

c) 创建 PLRGeoOpProjectionPtCrv 传入点跟曲线

d) 调用 GetCartesianPoint 获取曲线上的投影点

5.16.5 线在面上投影相关类

e) PLRCreateProjection 创建 PLRGeoOpProjectionPtCrv，创建倒角操作实例的 API，入参相关参数，创建 PLRGeoOpProjectionCrvSur 实例

a) PLRGeoOpProjectionCrvSur 点投影到线上的操作,获取投影点，设置点到曲线的最大距离，获取点集的数量等

b) PLRCurve 投影的曲线

c) PLRSurface 投影曲面

d) PLRCrvLimits 曲线参数范围

e) PLRSurLimits 曲面参数范围

5.16.6 线在面上投影操作步骤

a) 创建投影点

b) 创建要被投影的曲线

c) 创建 PLRGeoOpProjectionPtCrv 传入线跟曲面

d) 调用 GetPCurve 获取曲面上的投影线

5.17 等距

可以进行曲面，球面，曲面等距。

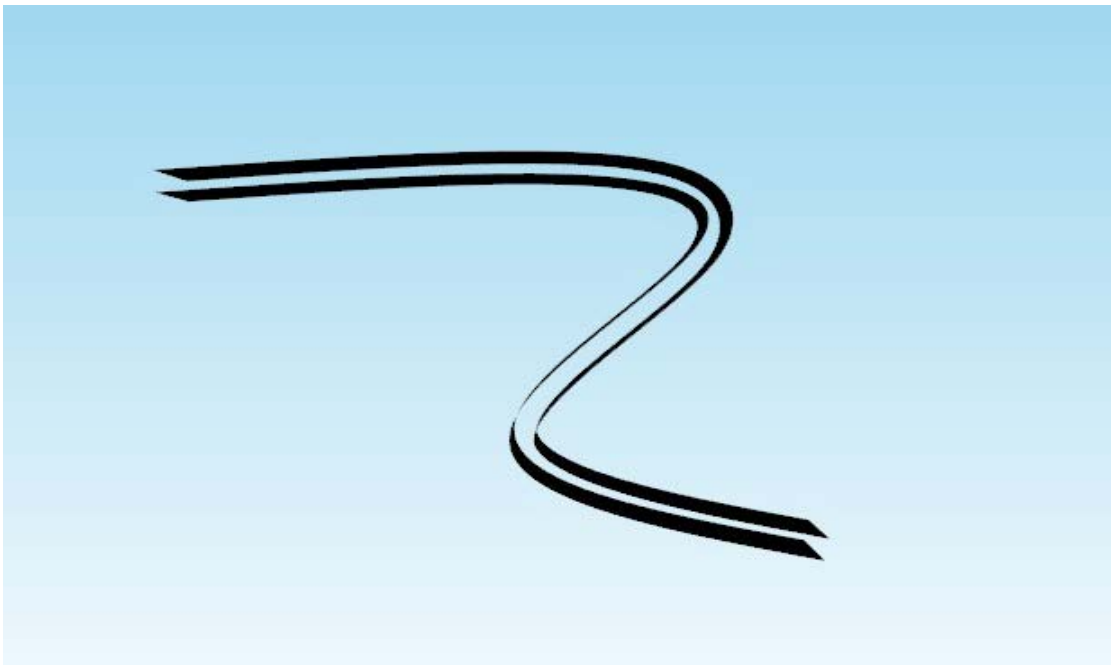

图 27 等距曲面

5.17.1 相关类

a) PLRGeoOpOffsetSur 等距曲面的操作，获取结果曲面，设置参数

b) PLRSurface 曲面

c) PLRSurLimits 曲面参数

5.17.2 步骤

a) 获取 PLRGeoFactory 实例

b) 调用 CreateOffsetSurface 函数得到等距曲面

5.18 放样

给定的轮廓沿着给定的导线形成相应的复杂模型

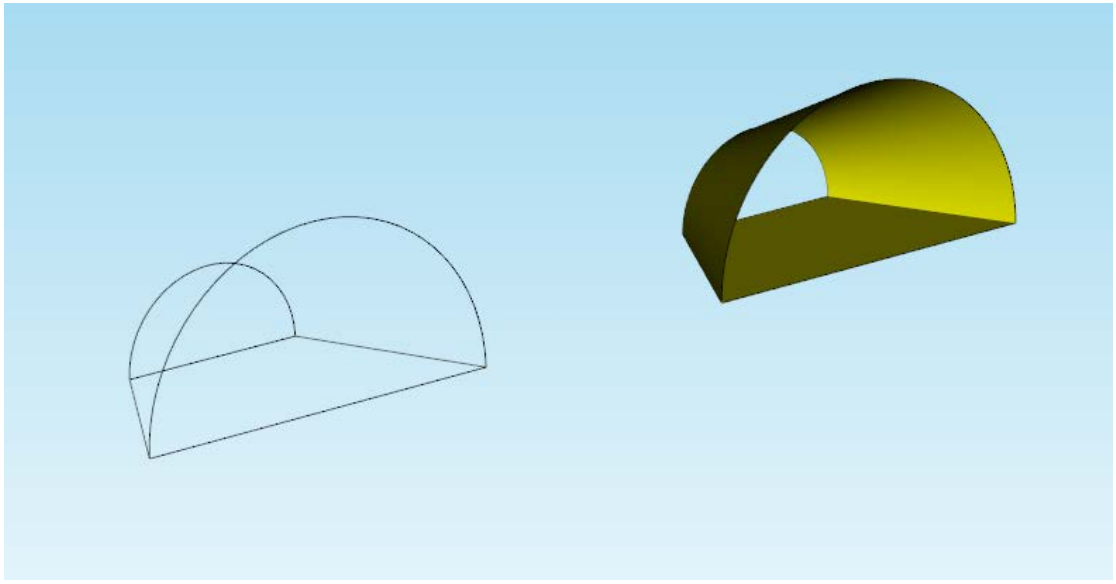

图 28 放样前后效果对比

5.18.1 相关类

a) PLRTopOpSweep 放样的操作类

b) PLRBody 导线和剖面 作为参数

5.18.2 步骤

a) 通过 PLRCreateFrFTopologicalSweep 传入导线和面的得到 PLRTopOpSweep *操作指针

b) 调用 Run 执行运算 如需设置参数可在 Run 之前调用相应的 API 进行设置

c) 调用的 GetResult 获取结果。

5.19 求最小距离

用去获取两个对象之间的最小距离。

5.19.1 点到曲线的最小距离相关类

a) PLRGeoOpDistanceMinPtCrv 点到曲线的最小距离的操作，获取距离，设置参数。

b) PLRCircle 曲线

c) PLRMathPoint 点

5.19.2 点到曲线的最小距离操作步骤

a) 通过 PLRCreateGeoOpDistanceMinPtCrv 传入相应的点跟曲线获取 PLRGeoOpDistanceMinPtSur* 操作指针

b) 调用 GetDistance 获取最小距离

5.19.3 点到曲面的最小距离相关类

a) PLRGeoOpDistanceMinPtSur 点到曲线的最小距离的操作，获取距离，设置参数

b) PLRMathPoint 点

c) PLRSurface 曲面

d) PLRSurLimits 曲面参数范围

5.19.4 点到曲面的操作步骤

a) 通过 PLRCreateGeoOpDistanceMinPtSur 传入相应的点跟曲面获取 PLRGeoOpDistanceMinPtSur*操作指针

b) 调用 GetDistance 获取最小距离

5.19.5 曲线到曲线的最小距离相关类

a) PLRGeoOpDistanceMinCrvCrv 曲线到曲线的最小距离的操作，获取距设置参数

b) PLRCircle 曲线

5.19.6 曲线到曲线的最小距离步骤

a) 通过 PLRCreateGeoOpDistanceMinCrvCrv 传入相应的曲线获取 PLRGeoOpDistanceMinCrvCrv *操作指针

b) 调用 GetDistance 获取最小距离

5.19.7 曲线到曲面的最小距离相关类

a) PLRTopOpDistanceBodyBodyMin 曲线跟曲面最小距离的操作，

获取距离和设置参数

b) PLRCurve 曲线

c) PLRSurface 曲面

5.19.8 曲线到曲面的最小距离的步骤

a) 通过 PLRCreateTopOpDistanceBodyBodyMin 传入相应的曲线获取 PLRTopOpDistanceBodyBodyMin *操作指针

b) 调用 GetDistance 获取最小距离

5.20 几何变换

几何变换对实体进行平移、旋转、缩放、镜像、仿射等操作，

a) 平移：实体在某个方向进行移动，不改变实体本身的形状和大小。

b) 旋转：实体绕旋转轴旋转一定的角度，不改变实体本身的形状和大小。

c) 缩放：对实体放大或缩小。

d) 镜像：对实体进行镜像。

e) 仿射：对实体进行仿射变换。

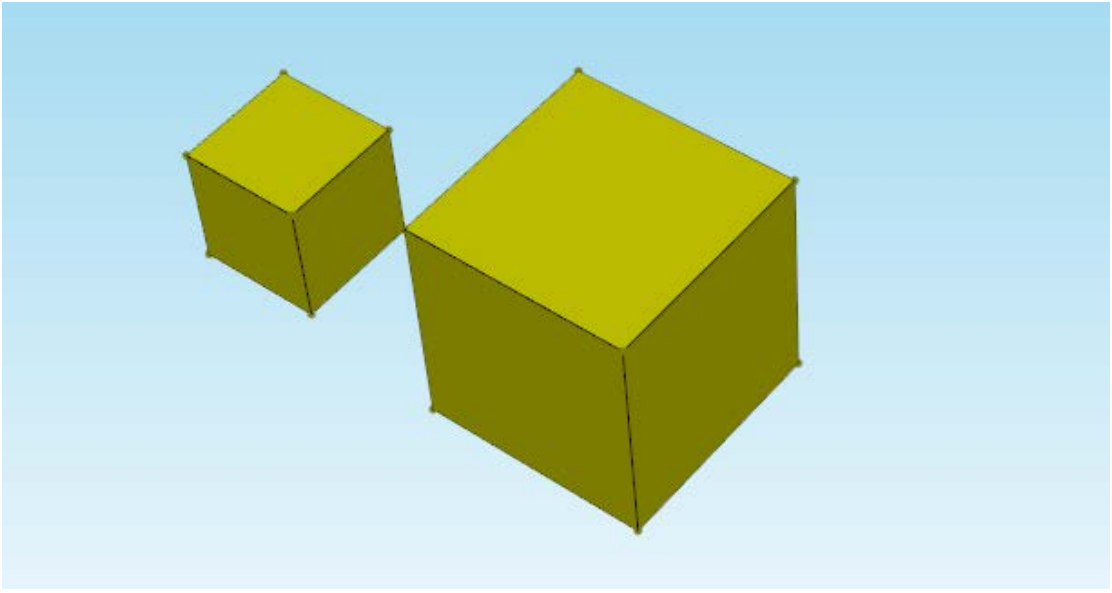

图 29 缩放对比

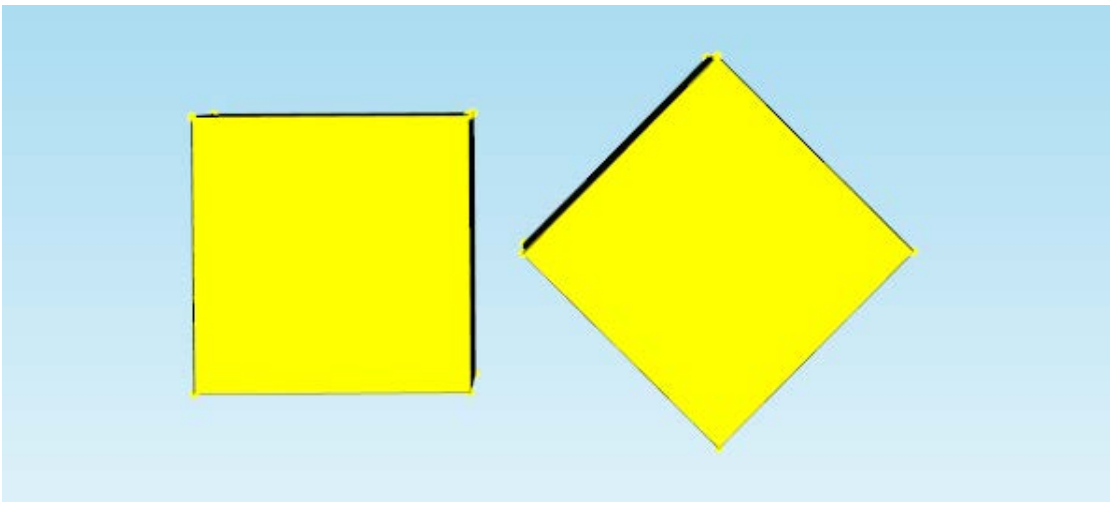

图 30 旋转对比

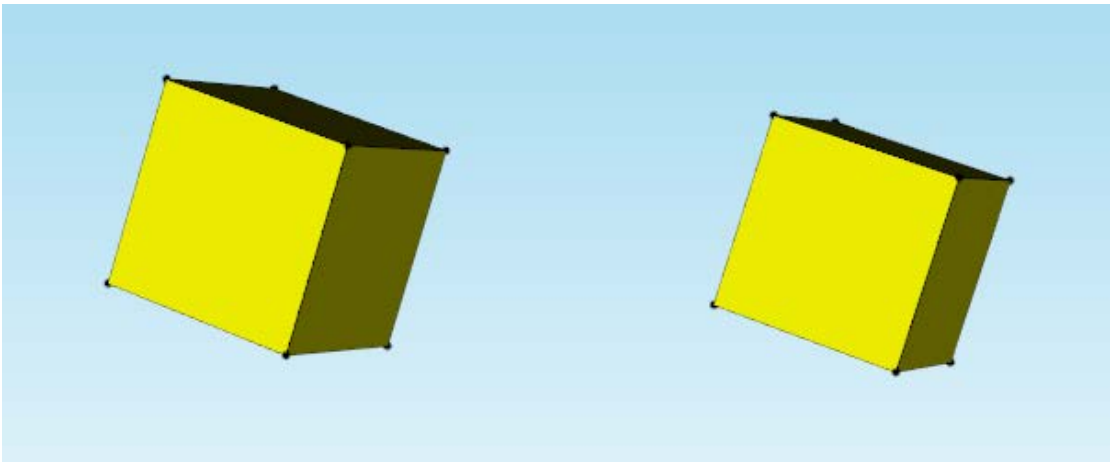

图 31 平移对比

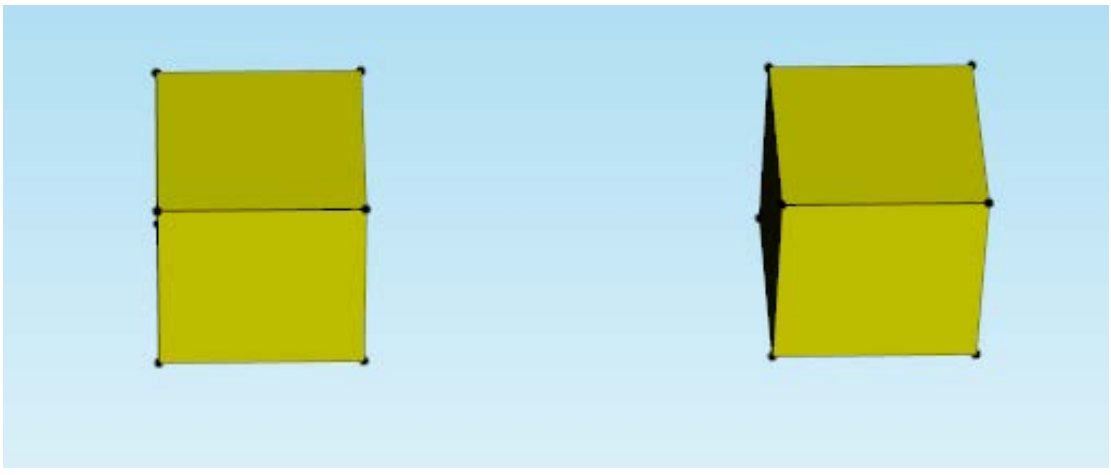

图 32 镜像

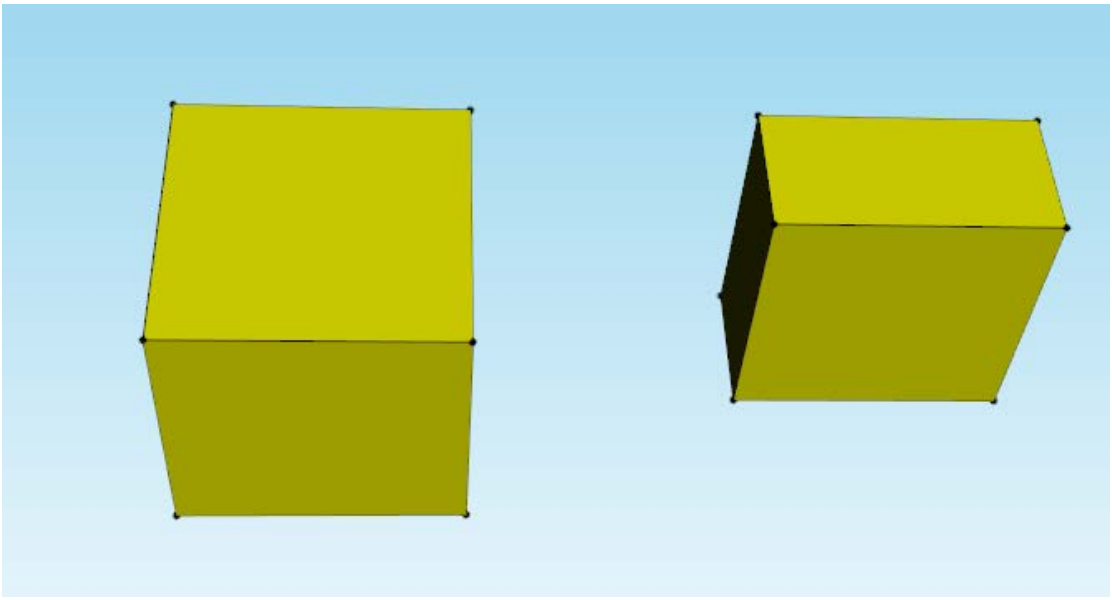

图 33 反射变换对比

5.20.1 几何变换相关类

a) PLRTopOpTransform 进行几何变换操作的类。

b) PLRMathVector 三维向量，用于设置平移时的方向和距离。

c) PLRMathLine 三维线，用于设置旋转时的旋转轴。

d) PLRMathPlane 三维平面，用于设置镜像和仿射变换时的参数

e) PLRMathTransformation 三维坐标变换，用于设置缩放，平移，旋转，仿射时的变换矩阵。

5.20.2 几何变换步骤

a) 创建几何变换操作 PLRCreateTopOpTransform，传入需要进行平移的实体

b) 创建平移向量 PLRMathVector

c) 通过 SetTranslation 设置平移的距离和方向，并以实体所在的位置为平移原点

d) 调用 Run

e) 最后调用 GetResult，获取到平移之后的实体

f） 旋转，缩放，镜像，仿射的操作和平移类似，只是设置的参数不同，可以参考平移的步骤进行创建。

5.21 外插延伸

外插延伸对线、二维的平面或曲面边界进行延伸的操作，通过设置延伸类型和长度，完成对边界的延伸，延伸的类型可以根据需要自行选择。

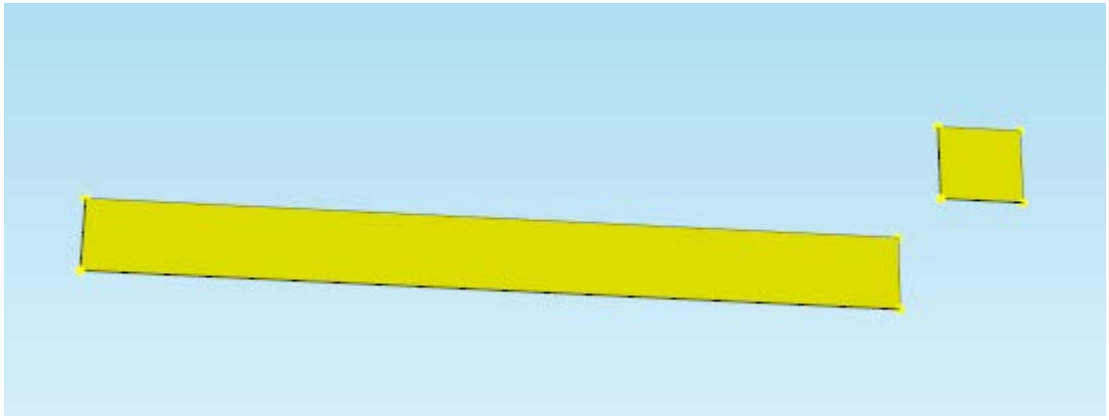

图 34 平面延伸

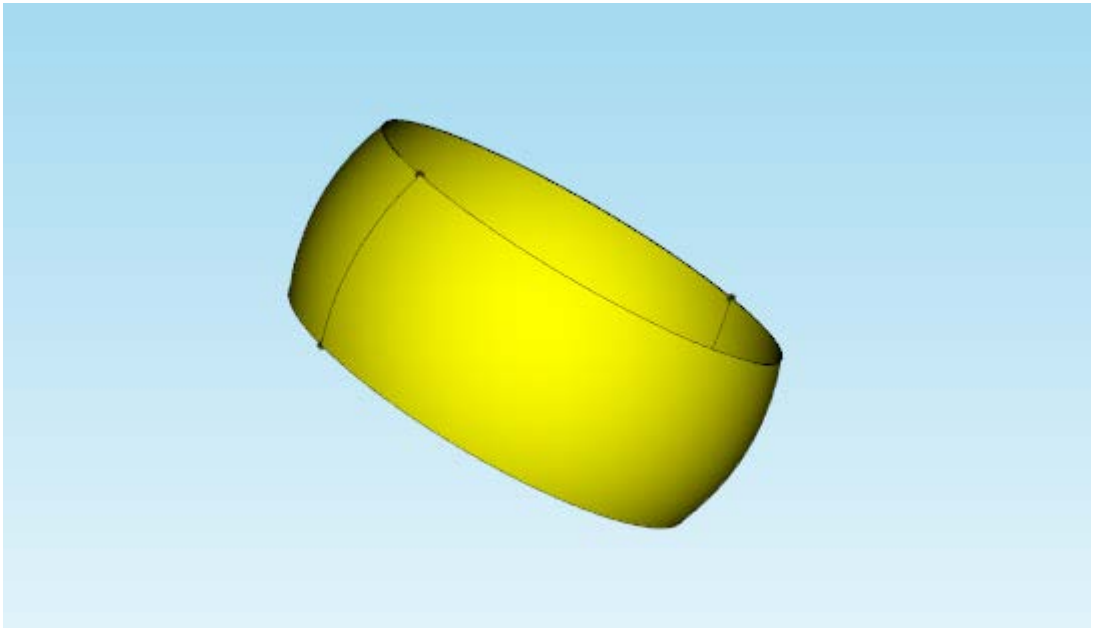

图 35 曲面延伸之前的曲面

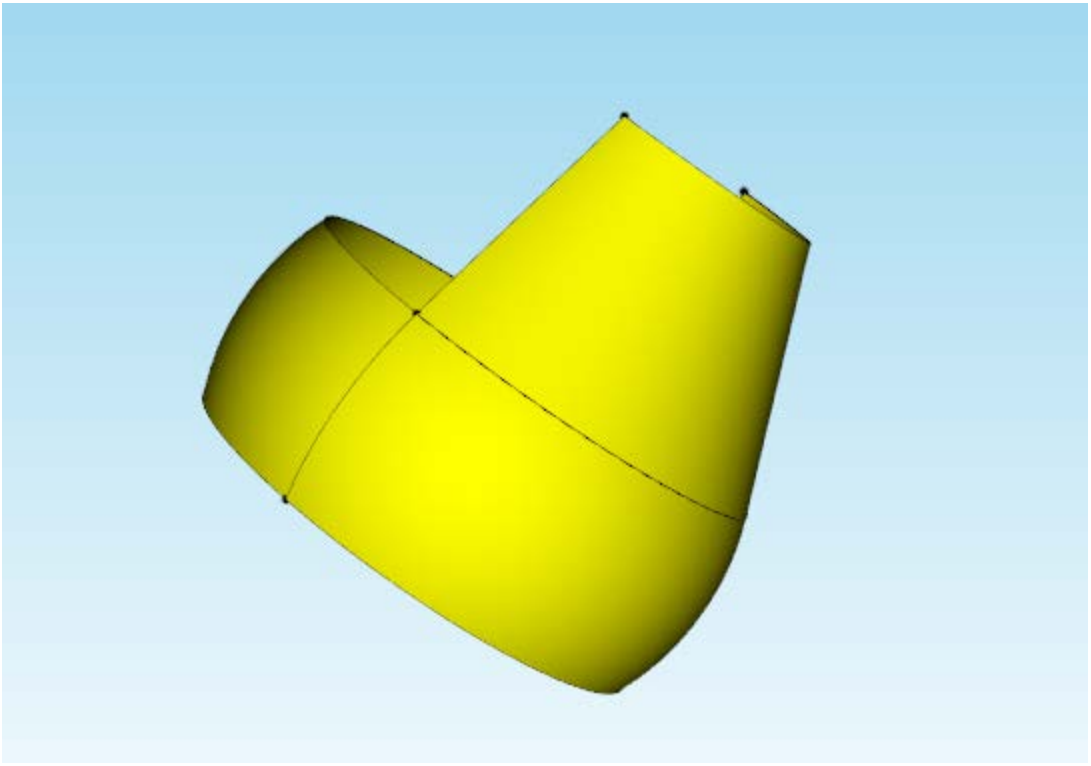

图 36 曲面延伸之后的曲面

5.21.1 外插延伸相关类

a) PLRTopOpExtrapolateSkin 进行延伸操作的类，对外插延伸进行创建和修改。

b) PLRBody 需要延伸的体和对应的边界。

5.21.2 外插延伸的步骤

a) 创建外插延伸操作 PLRCreateTopOpExtrapolateSkin,将需要延伸的体，和体对应的边界传入。

b) 调用 SetLength 设置延伸的长度。

c) 调用 Run。

d) 调用 GetResult 获得延伸后的体。

5.22 阵列

阵列可分为矩阵阵列和圆形整列两种，可以对点、线、面以及实体进行阵列变换

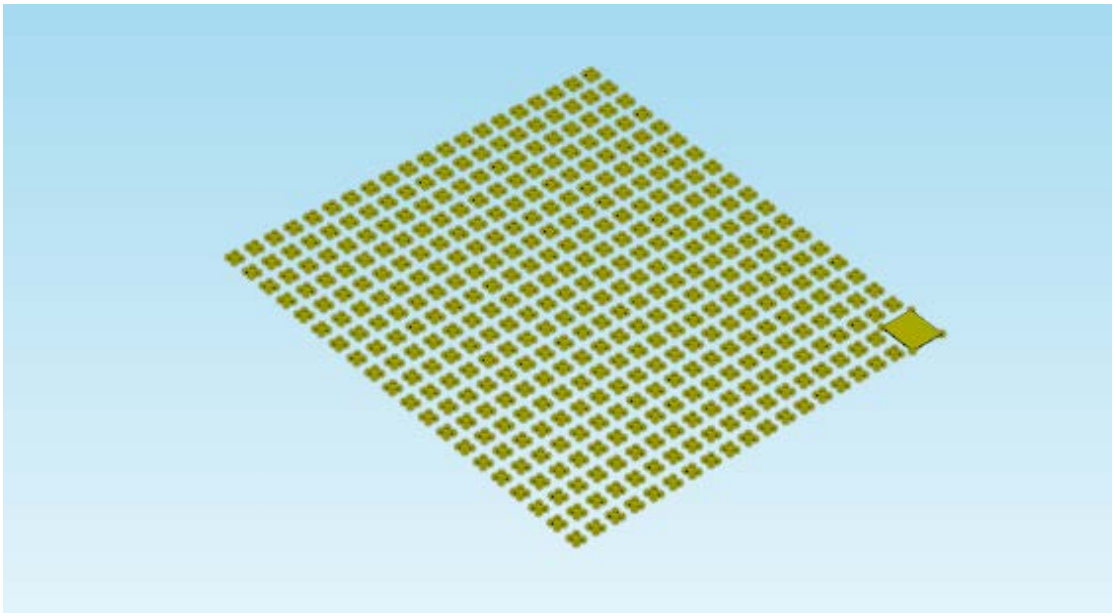

图 37 平面的矩阵阵列

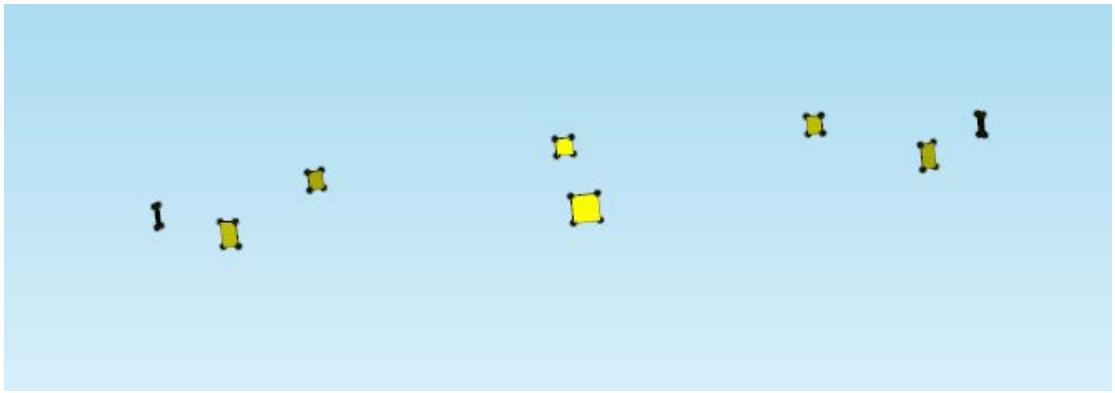

图 38 平面的圆形阵列

5.22.1 阵列涉及的类

a) PLRMathTransformation 三维坐标变换 用于设置阵列参数

b) PLRBody 体，用于设置被阵列的体或进行布尔运算的体

c) PLRPlane 平面，用于设置阵列参考平面

d) PLRMathLine 线，用于设置圆形阵列时的轴线

5.22.2 矩阵阵列的步骤

a) 平面的矩阵阵列

b) 创建需要被阵列的平面

c) 创建与阵列结果进行布尔运算的平面

d) PLRMathTransformation 创建阵列转换的列表

e) 设置阵列的参考平面

f) 通过全局函数 PLRCreateTopPattern，创建阵列并获取阵列的结果

5.22.3 圆形阵列的步骤

a) 平面的圆形阵列

b) 创建平面

c) 创建与阵列结果进行布尔运算的面

d) PLRMathLine 设置轴线

e) PLRMathTransformation 创建转换的列表

f) 创建阵列的参考平面

g) PLRCreateTopPattern，创建阵列并返回阵列结果

5.23 计算几何属性

计算实体的表面积、体积、长度、重心惯性矩阵等几何属性，

a) 表面积：三维实体只计算外表面，壳体计算外表面和内表面

b) 长度：只用于计算线

c) 重心：壳体重心为假设壳有一个相同厚度时的重心，线重心则是假设线有一个相同的粗细的横截面时的重心

d) 惯性矩阵：计算壳体和线的惯性矩阵与计算重心时相似

5.23.1 计算几何属性的相关类

a) PLRTopOpPhysicalProperties，计算几何属性的操作类，可对实体进行体积、表面积等计算，还可以获取到计算的相对误差。

b) PLRMathPoint，三维笛卡尔点，用于接收计算后的重心。

c) PLRMath3x3Matrix，三阶方阵，用于接收计算后的惯性矩阵。

5.23.2 计算几何属性的步骤

a) 通过全局函数 PLRCreateTopOpPhysicalProperties 创建计算几何属性的操作。

b) 调用 GetInertiaMatrix，获取惯性矩阵

c) 调用 GetCenterOfGravity，获取重心

d) 调用 GetWetArea，获取表面积

e) 调用 GetVolume,获取体积

f) 调用 GetLength，获取长度

5.24 Pillar 文件读写

本三维建模引擎的文件读取有自己特有的存取格式（*.pillar）,相关函数为 PLRFactory 类中的 SavePLRFile、LoadPLRFile 函数，可以选择保存显示对象的列表，保存形式是二进制类型，将保存实体对象的几何关系和拓扑关系。

5.25 IGES 文件读写

本三维建模引擎支持图形标准文件 IGES 格式，相关函数为 PLRFactory 类中的 SaveIGSPLRFile、LoadIGSPLRFile 函数，可以读取和保存 IGES 格式文件。

5.26 STEP 文件读写

本三维建模引擎支持图形标准文件 STEP 格式，相关函数为 PLRFactory 类中的 SaveSTPPLRFile、LoadSTPLRFile 函数，可以读取和保存 STEP 格式文件。

6 技术支持

北京求解科技是一家专注于三维建模方向的专业化公司，如果在

使用 DM-Pillar 几何内核引擎时遇到任何问题，可以访问北京求解公司官网 www.qiujiekeji.com 上查询，或发送邮件到 service@qiujiekeji.com。