Geoshape 几何建模引擎软件
V1.1.030

使用说明

文档版本 01
发布日期 2024-9-2

深圳泊松软件技术有限公司

深圳泊松软件技术有限公司

地址：广东省深圳市龙岗区坂田街道岗头社区天安云谷产业园二期 4 栋 2301

邮编：518129

网址：https://www.poissonsoft.com

目录

1 概述

1.1 内容简介

本章主要介绍 Geoshape 几何建模引擎软件（Geoshape Geometric Modeler）的基本概念信息，如类结构体、标签和标识符，以及如何使用本软件的接口。包含如下内容：

- 1.2 产品简介：几何建模引擎软件概述。
- 1.3 基本概念：几何建模引擎软件的基本概念。
- 1.4 章节介绍：本手册的章节介绍。

1.2 产品简介

1.2.1 什么是几何建模引擎软件

Geoshape 几何建模引擎软件是一个精准边界表示（Boundary Representation，以下简称 Brep）的建模引擎软件，它通过实体的边界来表示实体。支持以下功能：

- 构建和操作实体对象，如布尔、切割和偏移；
- 计算质量和转动惯量，并执行碰撞检测；
- 支持以多种方式输出模型对象；
- 将模型对象存储在文件中，再进行检索。

本软件是 3D 建模工具的核心或“内核”，以下简称为“内核”或“建模引擎软件”。本文档假定使用本软件的系统是正在开发的应用程序，以下称为“应用程序”。

1.2.2 软件接口

建模引擎软件和应用程序之间存在如下接口：

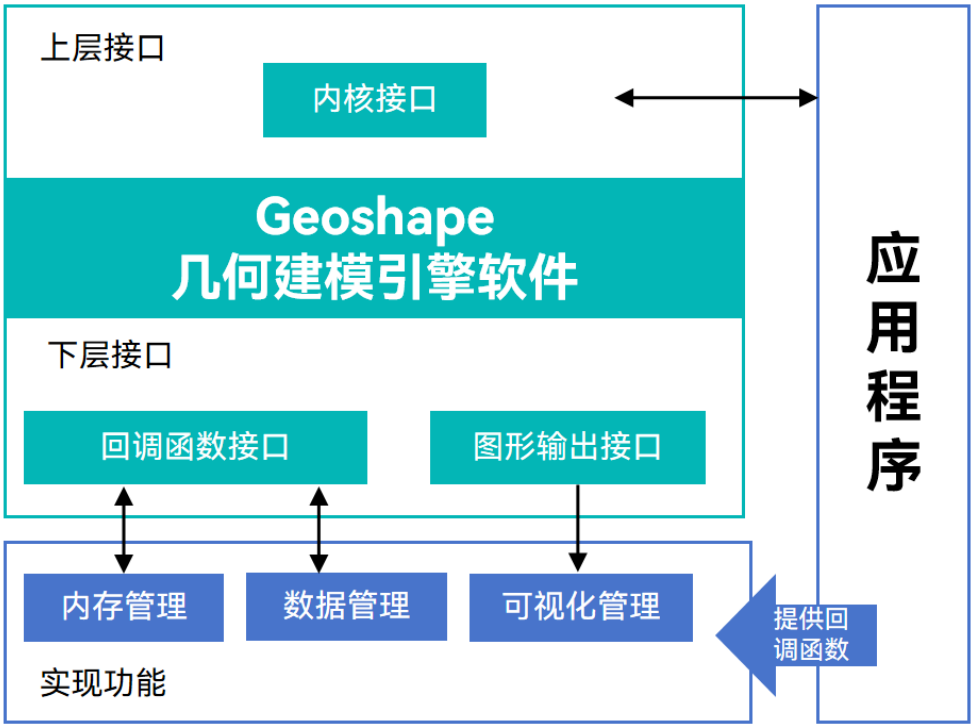
- 内核接口是本软件的上层接口，应用程序通过此类接口进行建模、操作对象和控制建模引擎的功能。
- 当建模引擎需要执行数据存储或系统类型的操作时调用下层接口，包括回调接口、GO（Graphical Output：图形输出）接口。

下层接口是建模引擎软件的组件，同时也可作为应用程序代码的一部分进行使用：

— 回调接口：内存管理和数据管理，所有应用程序都需要提供回调接口的代码。
— GO 接口：可视化管理（屏幕显示、打印等），大多数应用程序都需要提供 GO 接口的代码。

建模引擎软件接口的关系图请参见图 1-1 接口关系。

图 1-1 接口关系



说明
- 除 GO 接口外，大部分接口与建模引擎软件之间是双向通信的。
- 由于内核在运行过程中需要调用回调接口写入和读取文件，因此通过该接口从建模引擎软件传输信息是由内核控制的，而不是应用程序。

内核接口

应用程序通过调用内核接口访问建模引擎软件，包括如下四类功能：

- **模型构造**：构建、修改和组合模型。
- **查询**：查询有关模型特性、计算质量特性、几何信息或渲染信息的信息的函数。
- **模型管理**：加载和保存模型文件以及管理属性。
- **内核管理**：管理内核特性的功能，如回滚信息、回调函数、会话参数等。

回调函数接口

回调函数是一组必须作为应用程序代码提供的函数。当需要保存或检索数据时，内核都会读取回调接口中的函数。在应用程序代码调用内函数之前，需向建模引擎软件注册这些回调函数。

应用程序需要在回调接口中提供如下函数：

- 启动和停止会话的函数；
- 打开、关闭和写入文件的函数；
- 分配和释放内存的函数。

建模引擎软件提供了一个简单的回调接口示例（goCallbackExample.cpp）作为参考，您可在软件安装包中或接口文档中获取。

关于提供回调函数的介绍，请参见 2.3 提供回调函数。关于建模引擎软件所支持的回调函数功能的详细信息，请参见*《Geoshape 几何建模引擎软件 接口开发文档》*。

图形输出（GO）接口

当应用程序需要显示模型数据（在屏幕上显示或打印）时，需在提供回调函数的同时提供一组 GO 函数。通常情况下，GO 函数会将指令输出到应用程序提供的图形系统（如 OpenGL），以绘制内核请求的图形。与回调函数一样，GO 函数由内核调用。

建模引擎软件在使用内核的渲染函数创建图形数据时调用 GO 函数。内核逐段提供图形数据，因此调用一次内核渲染函数通常会生成多次 GO 函数的调用。

GO 接口的更多信息请参见*《Geoshape 几何建模引擎软件 接口开发文档》*。

1.2.3 基础使用场景

实体建模

实体建模包含创建和修改主体。主体也可以使用装配体组合：一个装配体包含多个主体及其排列信息，主体和装配体也可被称为模型。

建模引擎软件具有一系列建模操作，在创建和修改主体时可以使用不同的方法：

- 可从最小体（minimum body，即空间中的一个点）开始，然后逐次将其延伸（例如通过扫掠和旋转的方式），直至创建复杂实体。
- 可以从选定的基本实体开始，使用布尔运算进行组合，使用局部运算进行修改。

通过不同操作的组合，任何类型的主体都可以变成几乎任何形状。本软件支持多种类型的主体，除实心体外其他类型的主体通常用于实体建模的过渡阶段。

主体的结构、形状和属性由其拓扑结构和几何形状决定，这些实体类结合起来可以完全定义主体。

模型中使用结构的更多信息，请参见 3.2 模型结构。

获取模型信息

从建模引擎软件中获取模型信息的方法包括渲染、输出、查询，如表 1-1 获取模型信息方法所示：

表 1-1 获取模型信息方法

功能	说明
渲染	可使用渲染功能生成模型的图形数据，并输出数据到图形设备，如显示器、绘图仪或图像写入器。
输出	应用程序在保存模型数据时使用的保存功能由内核提供，模型的存储方式由回调函数控制。模型可以通过多种方式保存，例如作为文件系统上的一个或多个文件，或作为数据库中的条目。 当内核保存一个模型时，它会通过回调函数接口将数据写入传输文件。应用程序将提供传输文件的名称作为模型的键（key），以便于后续从文件中检索模型并将其加载到本软件。检索模型和保存模型均通过回调函数接口将模型数据读取到本软件。 若应用程序使用分区对会话中的模型进行分组，则可以保存和检索整个分区。
查询	查询功能支持查询任何主体的信息，包括如下使用场景： ● 计算实心体的质量特性。 ● 检测主体之间的碰撞（干涉）。 ● 识别实体中主体之间的关系或与其他主体的关系。 查询功能还可提取主体的几何信息（点、曲线和曲面），以应用于如下场景中： ● 分析模型，例如分析主体有多少种曲面。 ● 将几何数据发送到其他系统。 有关建模引擎软件查询功能的更多信息，请参见 5.2 查询功能。

附加数据到模型

除建模人员维护的内部结构之外，建模引擎软件模型还可以附加其他信息。

- **属性（Attribute）**：附加到实体的数据结构。可用于表示颜色、密度，或应用程序所需的其他信息，也可用于跟踪数据。多个字段类型（实数、字符、轴等）可以组合在一个属性中，更多信息请参见 13.3 属性。
- **组（Group）**：将主体中的相关实体（如构成凸台或孔的面）集合在一起，更多信息请参见 13.6 组。
- **用户字段（User field）**：建模引擎软件支持应用程序存储实体的单个信息字段，例如指向应用程序数据结构的指针。用户字段将附加到模型的每个实体中，长度是固定的，更多信息请参见 15.2.4 用户字段。

辅助功能

建模引擎软件还支持以下功能以辅助用户更灵活的进行建模操作。

表 1-2 辅助功能

功能	说明
回滚（rollback）	回滚是内核提供的一种记录模型数据并返回到历史内核状态的方法，通过回滚功能可撤销已完成的一个或多个操作。 回滚主要在调用内核函数错误后用于恢复建模引擎软件和模型的状态。回滚会将建模引擎软件的内存重置为历史状态，将影响拓扑项、几何项及附加数据。 回滚是通过回滚标记（会话标记或分区标记）控制的，建模人员可在会话中以适当的间隔设置回滚标记，内核将记录回滚标记间所有的模型操作。执行回滚操作后，模型将返回到所选回滚标记处的状态，内核还可从标记处继续向前或向后滚动。 回滚功能有如下两种使用方式： ● **会话回滚**：建模会话中的所有主体都可以一起回滚（或前滚）。 ● **分区回滚**：可使用分区（partition）功能将建模会话中的实体分割为子会话。一个分区可以包含一个或多个主体，并作为一个独立的单元进行存储，以便在回滚时只回滚单个分区。每个分区都可以单独回滚（或前滚），仅撤消（或恢复）该分区中主体相关的建模操作。 🕮 说明 如果使用回滚功能，则需要向建模引擎软件注册一个 delta 回调函数，相关信息请参见*《Geoshape 几何建模引擎软件 接口开发文档》*。
日志 （journal）	会话中调用的内核函数及相关入参出参都将记录在日志文件中，以便于应用程序进行调试。例如内核函数调用失败后，日志显示向该函数传递的参数值错误。 日志以文本为基础，便于检查，帮助您快速定位问题的根源。
快照（snapshot）	快照是整个内核内存和建模器参数的记录，主要应用于故障处理场景。您可通过创建会话标记和分区标记以创建快照。如果您向产品运维支持上报问题，可能会要求您提供快照文件（会话文件或分区文件）。

1.3 基本概念

介绍使用 Geoshape 几何建模引擎软件时涉及的基本概念。

1.3.1 对象类型

对象是在建模引擎软件和应用程序之间传递的数据项，每个对象都属于一个类，该类定义了对象的通用类型。本软件中类的继承关系图请参见图 1-2 继承关系图。

图 1-3 拓扑类和图 1-4 几何类显示了建模中使用的两个重要类的子类：拓扑（Topology）和几何（Geometry），类的详细描述可参见模型结构。

图 1-2 继承关系图

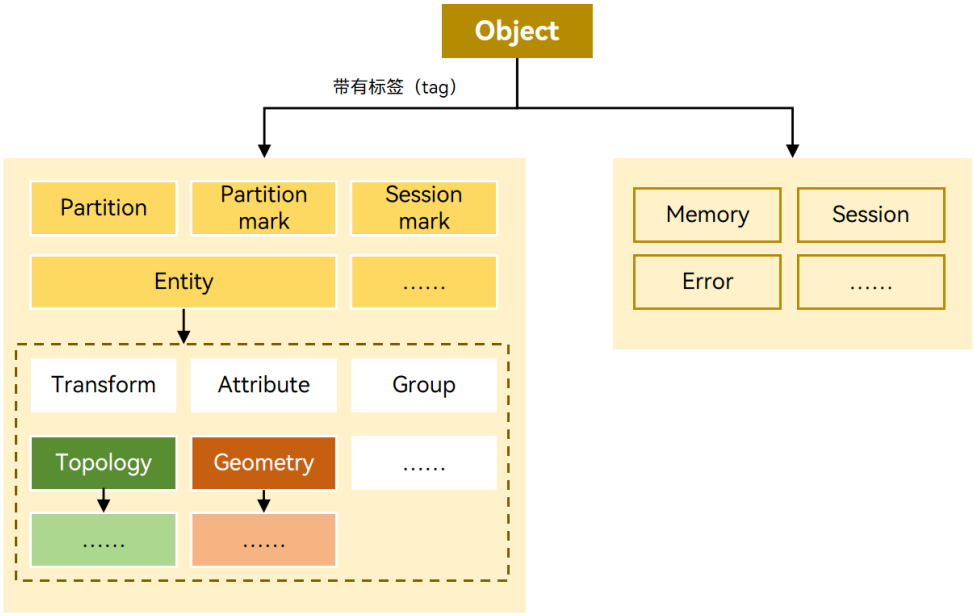


图 1-3 拓扑类

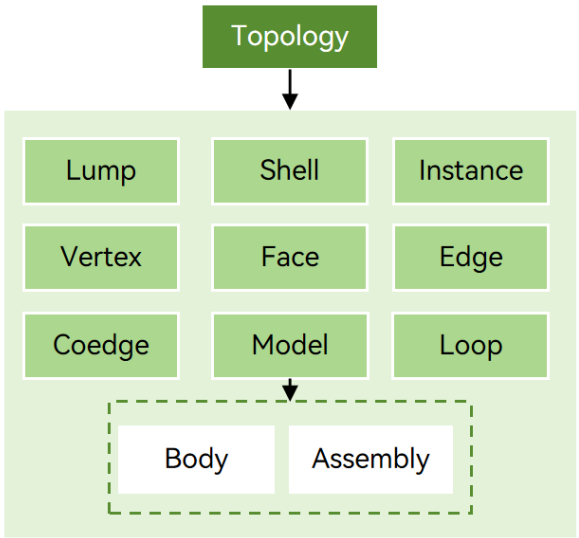
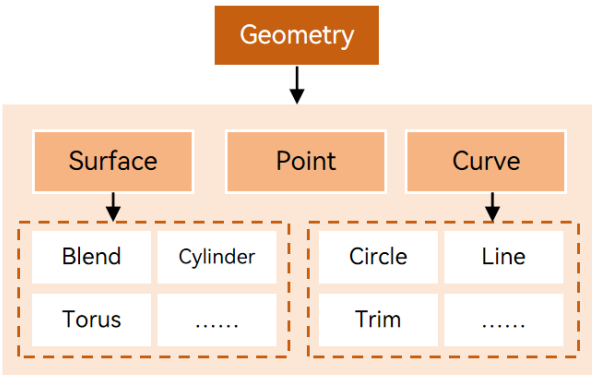


图 1-4 几何类



1.3.2 接口分类

内核接口函数都属于某一个类，在特定情况下也可在该类的子类下使用。类定义函数的用途，如下表所示：

类名称	函数用途
Session	会话管理，支持启动、维护和终止会话。
Topology	拓扑体管理，支持创建和修改几何体的拓扑结构，如顶点、边、面的添加、删除和修改，以及查询拓扑结构的信息。
Geometry	几何体管理，支持创建、修改和查询几何元素。
PrimitiveCreator	基本体构造器，支持创建基本体。
Visualization	可视化管理，支持设置视图参数、颜色、透明度等属性，控制渲染过程和效果，以及查询渲染状态和结果。
Entity	实体操作，支持创建、查询和删除实体。
Attribute	属性操作，支持获取、设置或修改实体的属性。
Euler	欧拉操作，支持优化、处理拓扑实体。
Algorithm	算法操作，支持执行算法函数。
System	系统管理，管理建模引擎软件整体的系统功能，如初始化、配置、资源管理和系统级别的交互。
Component	部件管理，支持创建、修改、查询装配体和实例。
Engine	引擎管理，支持创建/删除会话、创建 API 接口类、以及管理 License。

函数语法

内核接口函数的命名遵循小驼峰法则，是一串由大写字母和小写字母组成的字符串，通常由其所属类和操作描述的动词/名词组成。例如“deleteEdges”、“pmarkGetPartition”分别表示“删除指定边”和“查询分区标记所在分区”。

常见的操作描述如表 1-3 常见操作所示，常见缩略语如表 1-4 常见缩略语所示。

表 1-3 常见操作

动词	说明	示例
create	创建对象。	**createSheetCircle**
delete	删除对象。	**deleteGeometry**
set	存储/设置一个或多个字段。	**setUserField**
get	查询/获取字段的值。	**getType**
make	在实体上创建另一个实体。	**loopMakeEdge**
is	判断条件。	**isSurface**
compute/eval	计算。	**surfComputeDiscontinuity** **curveEvalHanded**

表 1-4 常见缩略语

缩略语	说明	示例
surf	surface，曲面 。	**bsurfAddUKnot**
attr	attribute，属性。	**attrGetDoubles**
attrDef	attribute define，属性定义。	**getAttrDefTag**

每个函数都包含输入参数和输出参数，用于提供数据和返回信息，输入参数不会因为内核接口调用而被修改。一个参数不能同时作为输入参数和输出参数。

部分函数的输出参数是可选的。当 NULL 指针作为输出参数的指针传递时，建模引擎软件将不计算或不返回此信息。

选项结构

可选参数可通过单独的选项结构（用“Option”后缀命名）来定义，结构中的每个选项都必须指定取值或使用默认值。结构体在定义的时候初始化，当需重复使用同一结构体时，可调用宏 **PSGM_API_STRUCT_RESET** 进行重置。

📖 说明
PSGM_API_STRUCT_RESET 并不具备释放内存的功能。

返回错误码

每个函数都会返回一个错误码作为返回值，表示该调用是否成功。在发生故障的情况下，错误码描述了可能的原因；也可通过函数的输出参数返回故障的更多详细信息，此时若返回错误码为零，需检查相应的输出参数。

应用程序可注册错误处理回调函数，以便于处理操作失败情况。当返回失败的错误码时，建模引擎软件将自动调用错误处理函数以执行故障处理所需要的操作。

1.3.3 实体类和标签

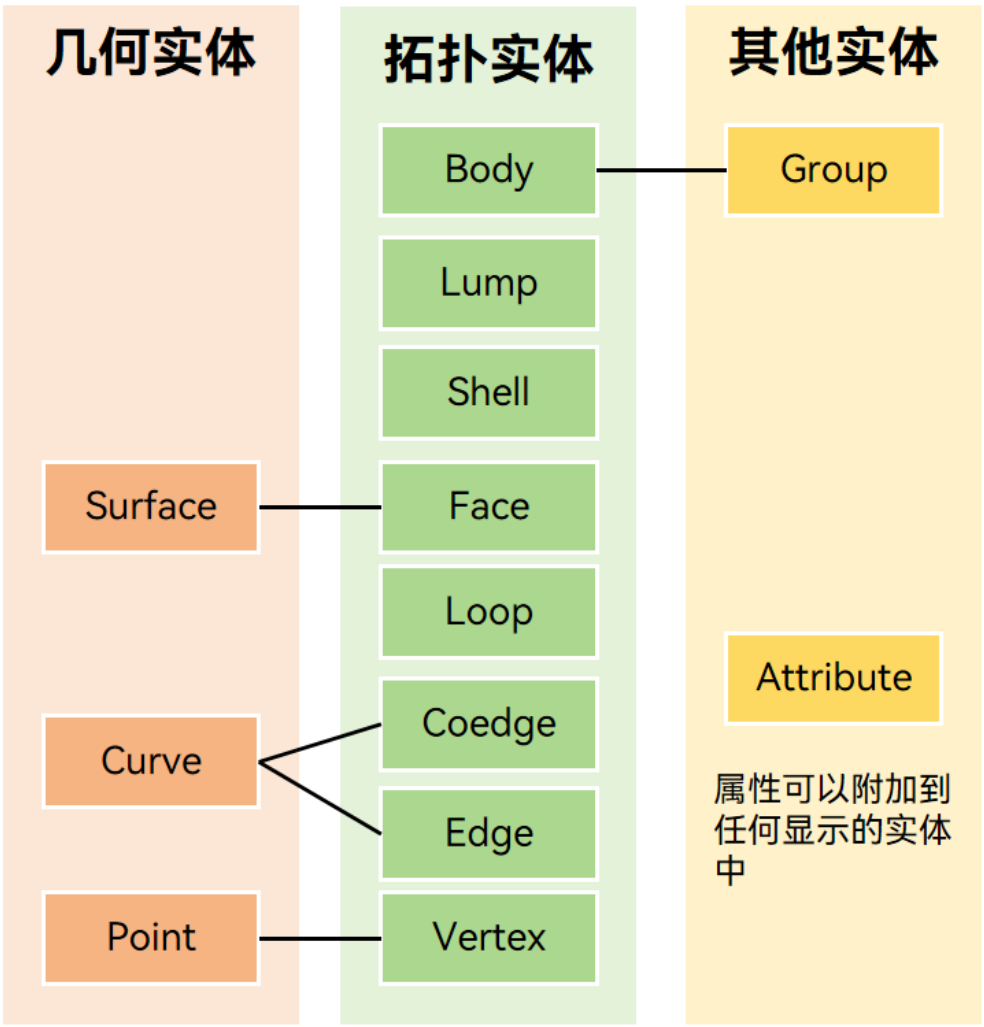
建模引擎软件模型由实体（entity）组成，实体是 Entity 类的对象。例如由单个圆柱体组成的模型（model）至少包含：

- 1 个**主体（body）**
- 1 个**块（lump）**
- 1 个**壳（shell）**
- 3 个**面（face）**
- 4 个**环（loop）**
- 4 个**半边（coedge）**
- 2 条**边（edge）**
- 3 个**曲面（surface）**
- 2 条**曲线（curve）**

它还可能包含一些属性或组，以及可以用于构造几何体的其他曲面、曲线和点（point）。

通常情况下，实体是模型的可识别部件，任何模型都可以表示为通过复杂关系链接的实体网络。图 1-5 模型实体显示了主体中可用的实体，以及它们之间的一些关系。图 1-3 拓扑类和图 1-4 几何类给出了与所示拓扑和几何实体相对应的类。

图 1-5 模型实体



本软件中实体及其之间关系的详细信息，请参见 3.2 模型结构。

标签

标签（tag）用于标识会话中的对象，包括实体、应用程序项（应用程序中的数据）、分区和其他有助于回滚的对象。

模型中的每个对象都有一个与之相关联的标签。在创建对象时，内核会创建一个对应的标签，内核通过该标签将实体从模型返回到应用程序。会话中的标签是唯一的，因此每个对象都可以被识别。但不同会话之间的标签并不一致：如果将相同的对象加载到不同的会话中，分配的标签不一定相同。

标签保存在整数变量中，会话中可使用标签的最大数量受使用平台的整数变量大小限制。要查询会话中剩余的标签数，可调用 **getRemainingTags**。

生命周期

标签只有当关联的对象仍然存在于内存中时才有意义（有效的），如果对象处于如下情况标签失效。

- 显式删除；
- 隐式删除（例如两个面之间多余的边被合并）；
- 回滚到创建对象之前的某个点来删除；
- 由于会话已停止而丢失。

应用程序可通过 **isEntity**、**isPartition**、**isMark** 和 **isPMark** 确定标签是否有效。如果失效标签被传递给内核，将返回错误“GenInvalidParameter”（如果启用了参数检查）。

null 标签

有一个名为 PSGM_API_NULL_TAG 的特殊常量标记，在文件 PSGMApiTypes.h 中定义。该标签表示不存在对应的实体，当没有合适的实体标签返回时，返回该标签。例如：

- 当请求附着到面的曲面标签时，如果面没有对应的曲面，则返回 PSGM_API_NULL_TAG。
- 执行错误操作后，若原接口需返回结果标签，则除了返回对应的错误码，还会返回一个 PSGM_API_NULL_TAG 作为结果标签。

未对实体标签初始化时，也可使用 PSGM_API_NULL_TAG 进行初始化。

持续性

许多建模操作都会导致实体被创建、删除或更改，此类更改将会影响这些实体标签的状态（创建或删除）。

面标签的持续性规则如下：

- 如果一个面收缩（截断），则其标签将持续存在。
- 如果将一个面拆分为多个面，则其中一个面继承原始面的标签，其他面的标签是新建的。

这些规则同样适用于边标签的持续性。

📖 说明
如果注册并使用了普通属性回调，则标签持续性将被禁用，更多信息请参见 13.2.2 属性回调。

相关操作

只有部分操作对标签和标签变量有意义：

- 标签仅在内核中创建，应用程序只能通过调用内核接口来获取。
- 应用程序将有效标签传递给内核，以指定需操作的实体对象。
- 标签作为整数变量可在应用程序中自由复制和保存。例如，可用于应用程序实体和内核实体之间的映射。
- 可使用运算符==和!=比较标签的相等性。
  - 若两个标签数值相等，则它们与模型中的同一实体相关联。
  - 两个不同的实体，即使在模型空间中精确重合，它们也具有不同的标签。
  - 在会话中标签编号越大，表示实体越新（最近创建的）。
- 对标签进行算术运算，或使用从内核获取标签之外的任何方式进行标签构建，都是没有意义的。

标识符

标识符（identifiers）是指自动附加到模型中所有实体（半边和模型本身除外）的整数值。标识符在指定模型内是唯一的（与标签不同，标签在整个会话内是唯一的）。当保存模型时会保存标识符（不会保存标签），因此标识符可以与模型键值一起存储在外部数据库中，以保持对指定实体的跟踪。

标识符可以为负数，但不为零。它们是在以下情况下创建的：

- 在模型中创建实体；
- 实体从一个模型移动到另一个模型（旧标识符失效）；
- 不属于模型的实体附着到一个模型上（例如在构造几何图形时）。

要从实体的标签中查找实体的标识符，可以使用函数 **getIdentifier**。

实体复制

可使用 **copyEntity** 复制会话中的任何实体。该函数接收一个实体标签和选项结构，包含用于控制函数行为的选项。它返回实体的副本（entityCopyTag）和一个包含跟踪信息的数据结构，可将原始实体中的部件映射至实体副本中的部件。

可用选项如下：

选项	描述
m_destinationTag	设置 entityCopyTag 目标的标签。可将实体的副本保存至指定的分区或模型。
m_wantUserFields	是否复制实体的用户字段。更多信息请参见 15.2.4 用户字段。
m_wantAttributes	是否复制实体的属性。更多信息请参见 13.3 属性。
m_wantTracking	是否创建跟踪信息。跟踪信息将原始实体中的部件与复制实体中的部件进行映射。例如，复制一个模型时，同时复制了该模型包含的所有实体（例如该模型中的面或边）。如果“m_wantTrackin”设置为“true”，则 **copyEntity** 将返回复制实体和原始实体的映射信息。实体结构的更多信息请参见 3.2 模型结构。

选项	描述
m_trackTypeArray	选择记录跟踪信息的实体类。若“m_wantTracking”设置为“true”，此选项控制应为哪些实体类创建跟踪信息；若未设置，将返回所有实体的跟踪信息。可只记录指定类的跟踪信息，以提高 **copyEntity** 的性能。

m_want*选项的默认值均为 false，因此在复制实体时不会复制或记录任何额外信息。将任意 m_want*选项设置为 true，以复制或记录与该选项相关的信息。

说明
copyEntity 返回的跟踪信息是建模引擎软件唯一支持的用于映射复制实体和原始实体的方法。例如，不应使用原始实体和复制实体中面的顺序或标识符等信息来推断映射信息。

如果需复制几何图形，则应使用函数 **copyGeometry**。可以复制几何图形的集合以及相关属性，并保留实体之间的关系，请参见复制几何。

1.4 章节介绍

本使用说明手册是 Geoshape 几何建模引擎软件重要的用户指南，它介绍了如何使用本软件的接口将 3D 建模和设计功能添加到应用程序。可搭配 *《Geoshape 几何建模引擎软件 接口开发文档》* 一起使用。

使用说明手册包含多个不同主题的章节，在进入后面几章中更详细的主题之前，建议您熟悉前三章的内容。本手册内容如下：

章节	描述
产品概述	介绍建模引擎软件、内核接口以及使用本软件所需了解的基本概念，请参见产品概述。
应用程序开发要素	介绍如何根据建模引擎软开发应用程序，包括应用程序的总体架构，需要提供的函数，内核接口的设计，建模操作时的跟踪更改和接收反馈等主题。更多信息请参见应用程序开发要素简介。
几何建模基本概念	介绍建模引擎软件的拓扑结构和几何结构、支持的不同主体类型以及容差建模的原理。更多信息请参见基本概念简介。
快速入门	应用程序调用 建模引擎软件的基础场景示例。更多信息请参见 4 快速入门
模型分析	介绍如何从建模引擎软件的主体中检索信息，例如进行拓扑和几何查询，计算质量特性信息，测量主体的距离并进行碰撞测试，以及检查模型的有效性。更多信息请参见模型分析简介。

章节	描述
模型编辑	介绍建模引擎软件支持的模型编辑操作，如填充孔、锥形面和删除实体。本章还描述了建模引擎软件的面编辑功能，该功能允许您在单个函数调用中执行各种不同的操作，同时保持模型的设计完整性。更多信息请参见模型编辑简介。
轮廓和曲面建模	介绍如何创建轮廓，然后通过拉伸、扫掠或放样来操纵轮廓以创建 3D 形状。更多信息请参见轮廓与曲面建模简介。
片状体和线框体	介绍了如何创建片状体和线框体，如何组合片状体（例如导入数据时），以及如何使用中间曲面生成技术从现有实体中创建片状体。更多信息请参见线框体与片状体简介。
拓扑操作	介绍拓扑操作的基本方法，包括欧拉操作、删除冗余拓扑和拆分拓扑实体。更多信息请参见拓扑操作简介。
压印和布尔	介绍压印和布尔的功能，包括将实体压印到其他主体上，在多个主体间进行布尔操作（合并、相减、相交），创建阵列、截面体以及检查主体如何相交。更多信息请参见压印与布尔简介。
偏移操作	介绍建模引擎软件的偏移操作。更多信息请参见偏移操作简介。
倒角	在模型中添加倒角是 3D 设计的基础能力，本章介绍建模引擎软件如何向边和面之间添加倒角。更多信息请参见倒角操作简介。
应用程序支持	介绍建模引擎软件提供的除 3D 建模以外的功能，可在应用程序中使用这些功能，例如使用属性来记录与应用程序的相关信息。更多信息请参见应用程序支持简介。
图形支持	介绍建模引擎软件提供的图形支持功能。本软件支持在显示设备上展示模型图像。还支持使用鼠标或触摸屏设备等方式从显示的图像中选择实体。更多信息请参见图形支持简介。
会话支持	介绍建模引擎软件提供的会话支持，包括多线程调用以及多处理器并行的能力。更多信息请参见会话支持简介。
错误处理	介绍如何在应用程序中处理各种类型的错误。更多信息请参见错误处理简介。

2 应用程序开发要素

2.1 内容简介

本章介绍如何将建模引擎软件集成到应用程序中，为 Windows 构建和运行示例应用程序，从应用程序调用建模引擎软件函数，跟踪建模操作以及获取有关操作的其他信息的信息，让您更简单高效地使用建模引擎软件。

本章仅为集成建模引擎软件到应用程序的入门介绍，更多详细信息请参见完整的文档集。

本章包含以下内容：

- 2.1 内容简介：介绍本软件与应用程序集成的注意事项。
- 2.2 设计和体系结构：介绍与应用程序集成所需的代码。
- 2.3 提供回调函数：介绍本软件使用的回调函数。
- 2.4API 接口编程概念：介绍 API 接口编程概念。
- 2.5 跟踪和标记：介绍了如何在本软件中跟踪建模操作。

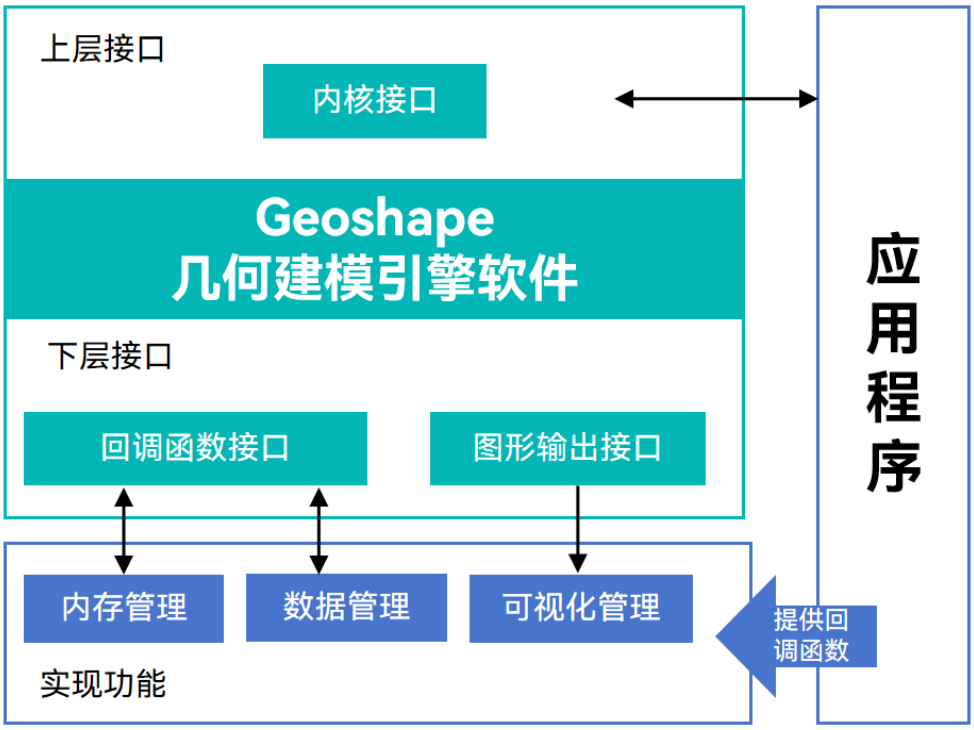
本章中介绍了在 Windows 系统上开发应用程序时最常见的问题，如果您的应用程序运行在不同的平台上，可在本软件的安装包中找到更多示例源代码，详细信息请参见 2.6 集成建模引擎软件。

2.2 设计和体系结构

为确保从应用程序代码中正确调用建模引擎软件接口，以执行实体建模的操作，您需提供对应代码以确保应用程序和建模引擎软件能够正确交互。本章介绍调用建模引擎软件接口前您需提供的代码。

建模引擎软件与应用程序的交互如图 2-1 建模引擎软件与应用程序交互所示。

图 2-1 建模引擎软件与应用程序交互



与本软件集成所需的代码分为两类：

- 回调函数代码。
- 调用内核接口函数的代码。

2.2.1 下层接口函数

应用程序通过下层接口控制建模引擎软件与操作系统进行交互，下层接口主要包含如下两个部分：

- 回调函数
- 图形输出（GO）函数

建模引擎软件会调用下层接口中的接口，因此在应用程序调用建模引擎软件函数之前，需向本软件注册这些接口。

📖 说明
GO 函数需要使用单独的图形库，当不需要处理模型显示时，不需要注册 GO 函数。

回调函数

回调函数主要包含如下功能：

- 文件处理：保存和检索建模引擎软件的模型文件和其他数据。
- 内存管理：为内部计算和数据结构存储分配内存。

GO 函数

当应用程序需要显示模型时，无论是在屏幕上渲染，还是将其打印到绘图仪或激光打印机，都需要为建模引擎软件提供 GO 功能。

每当应用程序调用建模引擎软件渲染功能以绘制一个或多个模型时，建模引擎软件都会调用 GO 函数封装输出的图形信息，并将其传递给图形库以渲染图像。

使用 GO 函数前需提供单独的图形库，可自己编写，也可使用 OpenGL 或 DirectX 等第三方库。

2.2.2 内核接口函数调用

建模引擎软件是作为一个函数库提供的，它被设计为一个组件软件工具包，您可以将其集成到应用程序中。建模引擎软件的接口称为内核接口，包含以下功能：

- 构建、修改和组合模型。
- 查找有关模型特性的信息（例如质量、几何信息或渲染信息）。
- 保存模型和将模型加载到建模引擎软件。

更多内核接口函数的介绍，请参见 2.4API 接口编程概念。

错误处理

应用程序需提供处理内核接口返回的错误的能力。更多详细信息请参见 2.3.6 错误处理。

2.3 提供回调函数

本章介绍了集成建模引擎软件时，应用程序中需要提供的回调函数，包含如下类型：

- 会话启动控制回调；
- 错误处理回调；
- 内存管理回调；
- 系统属性定义回调；
- 模型属性查询回调；
- Delta 回调；
- 文件处理回调。

2.3.1 需要提供的代码

部分回调功能是可选的，您可根据应用程序的需求提供对应的回调函数代码。本章节中回调函数名称为默认名称，您可在注册回调时自定义函数名称，该函数名称会映射到字段名称上。

必选功能

以下为应用程序必须提供的回调函数接口。

表 2-1 会话启动控制回调

回调函数	描述
start	此函数用于初始化回调。它是从内核函数会话开始调用的，是要调用的第一个回调函数。
stop	此函数关闭回调，并从内核函数会话停止调用。
abort	若内核操作因会话中止而停止，可调用此函数。它允许应用在中止后进行任何通用的清理，或者同时在应用代码中进行“恢复点”处长跳转。

表 2-2 文件处理回调

回调函数	描述
openWriteFile	此函数用于打开要写入的文件。
openReadFile	此函数用于打开要读取的文件。
write	此函数用于向文件写入数据。
read	此函数用于读取文件。
close	此函数用于关闭文件。

表 2-3 内存管理回调

回调函数	描述
alloc	此函数用于申请内存。
free	此函数用于释放内存。

可选功能

以下为可选功能，当应用程序需使用相关功能时，提供对应回调函数。

表 2-4 错误处理回调

回调函数	描述
onError	此函数用于错误处理。

表 2-5 图形输出回调

回调函数	描述
openSegment	此函数用于打开离散分层数据段。
segmentFacet	此函数用于输出类型为面片的离散单层数据段。
segmentLine	此函数用于输出类型为边的离散单层数据段。
closeSegment	此函数用于关闭离散分层数据段。

表 2-6 属性处理回调

回调函数	描述
attrDefSplit	此函数用于属性定义拆分。
attrDefMerge	此函数用于属性定义合并。
attrDefDelete	此函数用于属性定义删除。
attrDefCopy	此函数用于属性定义复制。
attrDefTransmit	此函数用于属性定义传递。
attrDefReceive	此函数用于属性定义接收。

表 2-7 Delta 回调

回调函数	描述
openForWrite	此函数用于打开一个新的指定分区标记（pmark）关联的 delta 文件，以写入数据。
openForRead	此函数用于打开一个已存在的 delta 文件，以读取数据。
close	此函数用于关闭一个已打开的 delta 文件。
write	此函数用于向指定 delta 文件写入数据。
read	此函数用于读取指定 delta 文件中的数据。
deleete	此函数用于删除指定 delta 文件。

2.3.2 注册回调函数

在启动建模程序并进行调用之前，使用 **registerApiCallbacks** 向建模引擎软件注册回调函数，该函数传入指向应用程序提供的回调函数的指针。

在启动建模会话及各回调功能前，应用程序需先向建模引擎软件注册回调函数，传入指向应用程序提供的回调函数的指针列表。注册回调函数涉及如下接口：

接口	注册回调
registerApiCallbacks	注册会话控制及文档处理回调。
memoryRegisterCallbacks	注册内存管理回调。
errorRegisterCallbacks	注册错误处理回调。
registerGraphicOutputCallbacks	注册图形输出回调。
deltaRegisterCallbacks	注册 delta 回调，打开分区回滚功能。
attrDefRegisterCallback	注册属性处理回调，以便于在指定属性的实体发生某些事件时调用。

2.3.3 文件处理

在建模引擎软件中建模的模型通过应用程序回调中的函数保存到外部存储。本软件创建的模型文件也适用于应用程序中的归档系统， 这可以采用主机上受控目录结构或某种数据库的形式。

本软件提供保存和检索大量数据的功能，以支持诸如保存和还原快照以及日志记录等操作。

文件扩展名

表 2-8 文件拓展名列表

文件格式	文件拓展名	描述
pmt	.pmt	Poisson 模型文本格式。
pmb	.pmb	Poisson 模型二进制格式。
stlText	.stl	STL 文本格式。
step	.step	STEP 格式。
journal	.json	日志记录。

在打开文件之前，回调应该测试文件是驻留在 DOS 风格的 FAT 设备上还是 NTFS 设备上。在不同系统之间传输文件时，可以简单地重命名文件。

文件格式

建模引擎软件将文件视为一系列字节流，按顺序进行写入或读取，文件必须以相同的格式写入和读取。通过回调写入某些文件时，应用程序需要指定它们是采用文本还是二进制格式。

- 在二进制（.pmb）文件中，字节流没有固有的含义或结构，它可以包含任何值的字节。
- 对于文本（.pmt）文件，字节流由打印字符组成，其中夹杂着换行字符（对应于C++中的“\n”）。

📖 说明
建议您将回调创建的文本文件作为流，即使用 LF 终止，而不是使用 DOS 默认值。

可移植性

本软件创建的文件在不同操作系统（具有不同的回调实现）之间是可移植的，但机器依赖二进制文件不能从一种类型的机器移植到另一种。因此文件可移植性如下：

- 机器依赖二进制文件应该在同一机器类型上的不同回调实现之间具有可移植性。
- 文本文件和中性二进制文件应该在不同机器类型上的不同回调实现之间具有可移植性。

要确保文本文件在不同机器类型之间可移植，回调的实现需要满足除换行符外，文本文件应仅包含 C++库函数 isprint 定义的打印字符。

如果回调的实现不遵循上述要求，则可能无法读取由其他系统创建的文件，也无法将其自己的文件转发给其他系统（例如在出现故障时进行报告）。此类问题可以通过运行回调验证测试来揭示，但请注意，这些测试无法验证文件格式是否与 C++运行时库一致，因此需要进行一些额外的检查以确认这一点。

2.3.4 内存管理

应用程序需提供两个内存管理回调函数，以允许建模引擎软件为其使用分配和释放内存，涉及如下场景的分配和释放：

- 内部计算；
- 数据结构存储。

这两个内存管理回调函数是 **alloc** 和 **free**，与 C 语言中的 malloc 和 free 函数非常相似，应用程序提供的函数定义需与 malloc 和 free 类型兼容。

为了与标准函数相比提高性能，您可考虑实现一些缓冲机制。例如，通过适当的定义：

- 小量内存（大约 0.1 MB）可以通过 **start** 回调函数设置的内存池来提供，并由应用程序管理。
- 大量内存（例如 1MB 或更多）可以从操作系统请求。

内存管理的更多信息，请参见 2.4.2 内存管理。

2.3.5 图形输出

当调用本软件的渲染函数时，生成的图形数据将通过一组由 **registerGraphicOutputCallbacks** 注册的图形输出回调函数（以下简称 GO 函数）进行输出，这些 GO 函数是在 **PSGMIApiGraphicOutputCallbacks** 中定义的。

🕮 说明
示例回调函数并不执行任何操作，如果您想使用渲染函数，需在 **PSGMIApiGraphicOutputCallbacks** 中定义 GO 函数，否则将无法获得图形数据。

图形输出接口

线数据（line data）是由建模引擎软件提供的接口 **renderGeometry** 和 **renderTopoFacet** 生成的，可以通过 GO 回调函数来输出。

GO 回调函数中的 iFail 参数表示数据的输出状态，可以取以下值之一：

- Continue：持续输出。
- Abort：用户中止输出。

线数据输出结构

通过 GO 函数输出的线数据被组织成数据段（segments），这些数据段对应于模型的可识别部分,数据段不一定是内核意义上的实体。数据段有下两种类型：

- 离散分层数据段（hierarchical segment）：通过调用 **openSegment** 打开数据段，并一致保持打开状态，直到调用 **closeSegment** 接口关闭。其他分层段或单层段可以在两者之间产生，这些可视为包含在该分层段中。分层段可以包含任意数量的段。
- 离散单层数据段（single level segment）：通过对 **segmentFacet** 或 **segmentLine** 的调用来打开和关闭。内核不创建单层段，单层段总是包含在分层段中。

🕮 说明
图形数据始终是分层输出的。

装配体图形数据

使用本软件接口创建的装配体必须被展平，并且装配体内的主体标签和变换矩阵必须被复制到实体数组中，然后才能由 **PSGMIApiGraphicOutputCallbacks** 函数渲染、输出。

与这些主体段相关联的标签一般是不同的。在两种情况下，您可能会收到具有相同标签的单独主体段：

- 实体数组中一个主体被多次引用，每个引用都在一个单独的段（或多个段）中输出。每种情况的都不同。
- 一个主体被零碎地输出，例如，一个主体的一部分被另一个主体遮挡，在这种情况下，构成主体的每个可见部分的线可能会被分别输出在不同的主体段中。

段输出函数

段输出函数包含 **openSegment**、**closeSegment**、**segmentFacet**、**segmentLine**。

openSegment 的作用是打开离散分层数据段，**closeSegment** 是关闭离散分层数据段。当前建模引擎软件支持的分层段类型包括：体数据段、面数据段和边数据段。

在输出离散单层数据段时，可调用如下两个回调函数：

- **segmentFacet**：输出类型为面片的单层段，其参数说明请参见表 2-9 segmentFacet 参数说明。
- **segmentLine**：输出类型为边的单层段，其参数说明请参见表 2-10 segmentLine 参数说明。

表 2-9 segmentFacet 参数说明

参数	说明
tagsCount、tags	与段关联的标签数量和标签列表。
geomsCount、geoms	段中的几何信息数量及具体的几何信息，与 facetInfo 相关，包括如下类型： ● 面片点（facet points） ● 面片点（facet points）+ 曲面法线（surface normals） ● 面片点（facet points）+ 参数（parameters） ● 面片点（facet points）+ 法线（normals）+ 参数（parameters）
occurenceCount	面片段数据出现的次数。
facetInfo	用于指示输出哪些内容，比如只输出点或者输出点+法向等。
iFail	图形输出状态，用于指示用户是否中止了图形输出。

表 2-10 segmentLine 参数说明

参数	说明
segmentType	用以指示当前输出的数据是什么类型。
tagsCount、tags	与段关联的标签数量和标签列表。
geomsCount、geoms	段中的折线数量与相关线数据，与 lineInfo 相关。
occurenceCount	线段数据出现的次数。
lineInfo	曲线的附加信息。
iFail	图形输出状态，用于指示用户是否中止了图形输出。

2.3.6 错误处理

建模引擎软件错误可能在多种情况下发生。例如，如果应用程序向一个内核函数传递了错误的数据，或者内核函数未能完成操作，本软件会向应用程序发送一个非零的错误码。在其他情况下，也可通过函数的输出参数之一的状态码返回更多关于失败的详细信息，此时错误码会返回零。

处理非零错误码失败

当建模引擎软件返回非零错误码时，应用程序可选择：

- 向建模引擎软件注册一个错误处理回调函数：应用程序可编写并向本软件注册一个错误处理回调函数。每当发生错误时，这个函数都会被调用，以执行某些恢复任务。建模引擎软件在从失败的内核函数返回之前自动调用此错误处理函数。
- 使用该异常：在遇到错误时抛出异常，这可以通过 C++中的 try/throw/catch 语句（或在 C 中使用等效的 setjmp/longjmp 函数）来实现。这允许程序执行避开某些代码段，并在处理完错误后在方便的地方恢复执行。如果您的应用程序结合了注册的错误处理回调和异常，那么在继续之前就不需要检查每个内核函数的返回状态。

无论你选择哪种策略，最终都需要采取相同的恢复操作。这个操作取决于错误的严重性，如表 2-11 错误等级所示。

表 2-11 错误等级

严重程度	错误状态	恢复操作
中度	操作失败，但涉及的模型没有改变。	应用程序可继续操作。
严重	操作中涉及的模型可能因被更改导致无效，会话中的其余模型正常。	应用程序可回滚至模型有效的状态。若未启动回滚，需重新启动会话并注册回调函数。
致命	会话异常，回滚功能也失效。	需重新启动会话并注册回调函数，如有必要，需重启应用程序。

注册错误处理回调函数

使用 **errorRegisterCallbacks** 注册一个错误处理回调函数。错误处理回调函数必须接收一个指向 **PSGMIApiErrorHandlerCallback** 结构的指针，**PSGMIApiErrorHandlerCallback** 结构包含了错误的详细信息，比如失败函数的名称、错误码和错误等级。

📖 说明
应用程序提供的错误处理回调函数不应该尝试修改当前错误的任何细节（比如通过修改 **PSGMIApiErrorHandlerCallback** 结构或返回不同的错误码）。建模引擎软件会将这些信息分开存储，任何这样的修改都不会有效果。

处理零错误码失败

可在*《Geoshape 几何建模引擎软件 接口开发文档》*中查看每个接口是否会通过参数字段以及通过哪个字段返回失败信息。

说明

对于通过参数返回故障信息的函数，请在应用程序中明确检查这些参数的内容，以确定操作的成功或失败。此种情况下，将不会调用错误处理回调函数。

2.3.7 启动和停止会话

使用 **registerApiCallbacks** 注册会话控制回调函数后，可调用 **start** 启动建模引擎软件会话，调用 **stop** 停止建模会话。

2.4 API 接口编程概念

建模引擎软件的 API 接口遵从 C 语言编写规范，是在头文件（.h 文件）中定义的标记、结构和函数的集合。您可以使用 C++、C 或转换后的 java、python 调用 API 接口。

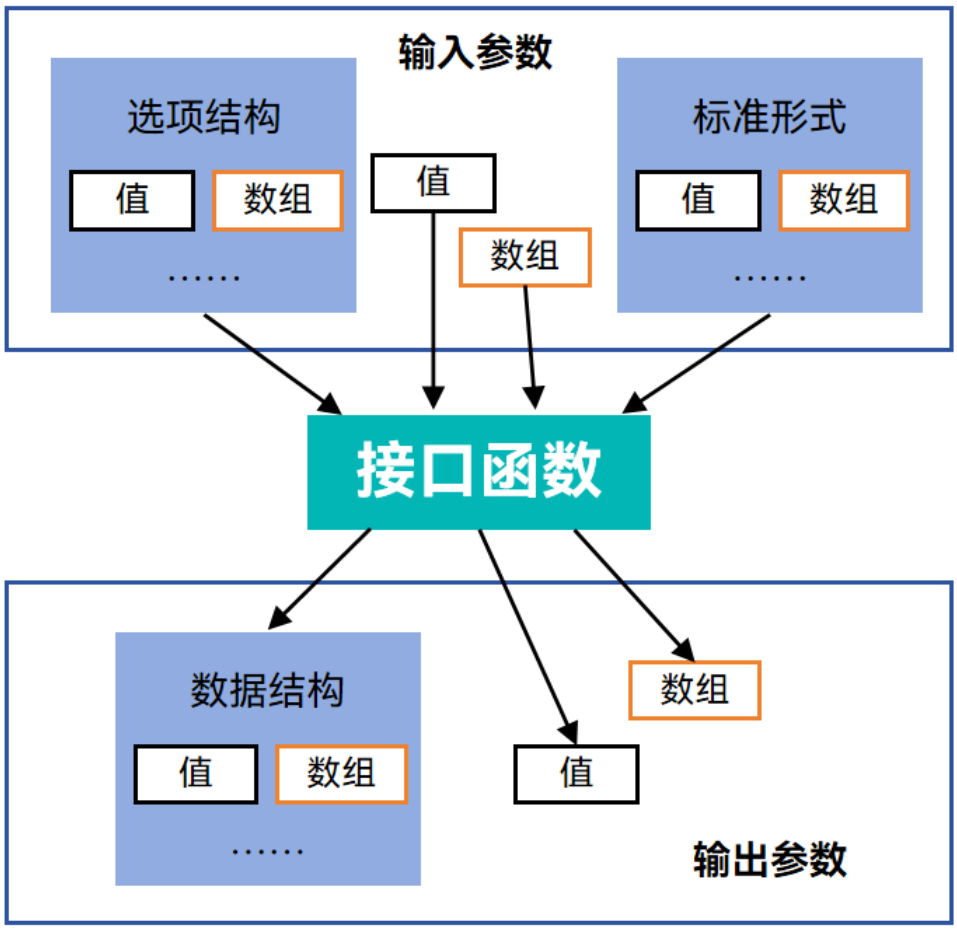
API 接口描述请参见*《Geoshape 几何建模引擎软件 接口开发文档》*，本软件还提供了代码示例（.cpp 文件），可在软件安装包或接口参考文档的“示例”中获取。

2.4.1 API 接口

内核接口的名称是一串由大写字母和小写字母组成的字符串，通常由其所属类和操作描述的动词/名词组成，以方便理解接口功能。例如“deleteEdges”、“pmarkGetPartition”分别表示“删除指定边”和“查询分区标记所在分区”。

每个接口都包含输入参数和输出参数，用于提供数据和返回信息，一个参数不能同时作为输入参数和输出参数。不同元素之间的关系如图 2-2 接口结构所示。

图 2-2 接口结构



内核接口和参数类型的更多信息，请参见 1.3.2 接口分类。

类

在建模引擎软件中，相似类型的对象属于同一个类，该类定义了这一类对象的通用类型。例如，所有圆柱面都属于 **PSGMApiCylinder** 类型。这些类按层次组合在一起，如图 1-2 继承关系图所示。

建模引擎软件的类具有以下特点：

- 类名称的形式为“PSGM<CLASS>”,其对象可以是无上下文的数据包（例如 Body），也可以是指向内核状态的句柄（例如 PartitionMark、SessionMark）。
- 某些类的对象具有标签，用于在会话中识别这些对象。标签的更多信息，请参见标签。
- 有些类具有一种称为标准形式的特殊结构，该结构用于封装该类的对象数据。例如在圆的标准形式（PSGMApiCircle）中，封装了圆心、法线轴、半径等数据，用来完整的表示一个圆。

选项结构体

在函数中使用的可选参数和开关通常会被封装到一个单独的结构体中，并作为一个选项结构体的参数传递，该结构体通常以“Option”结尾。结构体在定义的时候初始化，当需重复使用同一结构体时，可调用宏 **PSGM_API_STRUCT_RESET** 进行重置。

🕮 说明
PSGM_API_STRUCT_RESET 并不具备释放内存的功能。

可选返回参数

可选返回参数可在调用函数之前设置为 NULL，表示不返回此信息，也不为其分配空间。这些参数以“type ** const name”形式声明，备注为“可选”。以查询一个模型中的实例为例，其中查询到的实例标签列表（instanceTags）是可选返回参数，若将其设置为 NULL，将不会返回该数据。如下所示：

```
virtual PSGMApiErrorCode modelGetRefInstances(
const PSGMApiTag modelTag,                    //需要查询实例的模型标签
int* const instanceTagCount,                  //模型中实例的数量
PSGMApiTag **const instanceTags) = 0;         //实例的标签列表（可选）
```

🕮 说明
请在文档明确说明可使用 NULL 时采用此方式，否则可能导致传递错误参数。

通过引用传递变量

为保障代码的高效性、灵活性以及与新版本的兼容性，建模引擎软件通常在以下情况采用引用的方式接收变量：

- 传递的值过大；
- 传递的值大小是可变的；
- 数据结构定义在不同的建模引擎软件版本中可能发生变化。

常量的使用

Const 修饰符在建模引擎软件中用于保护应用程序的数据不被本软件以任何方式更改。

- 防止建模引擎软件更改传递给它的信息（输入参数），其编码形式为：const PSGMApiTag *pmarkTags。
- 防止建模引擎软件将返回信息（输出参数）放错位置，其编码形式为：int *const badPMarkTagCount。

当函数参数的类型使用 const 声明时，该参数是恒定向下传递的，任何深度的任何调用都不会修改该参数的参数值。

2.4.2 内存管理

建模引擎软件为函数执行过程中所需的数据分配内存空间，以保障函数的正常调用。当返回的参数无法在生成时确定其所需的内存时，需在函数执行时进行分配，该分配由应用程序在 **memoryRegisterCallbacks** 中注册的回调函数执行。此类参数不再使用时需释放其内存空间。

可变长度数据作为 C 数组从内核返回，根据其内存需求，可以分为以下三类。

- **在编译时声明内存空间**
  当应用程序在编译时声明空间，并通过一个常量指针将这块空间传递给建模引擎软件时，表示该空间的大小在编写代码时就已经确定。这种空间在栈上分配（对于局部变量）或在静态存储区分配（对于全局变量或静态变量）。由于这种空间的生命周期由编译器管理，因此不需要显式地释放它。在内核函数头中，这类参数可能表现为 type name[3]的形式，其中 name 是数组名，3 是数组大小。

  🕮 说明
  例外场景：数据的大小在编译时是已知的，但在返回结构体的字段中是通过指针声明的，由建模引擎软件决定是否为它分配空间，并设置相应的指针指向这块空间。

- **运行时分配内存空间**
  应用程序在运行时分配空间，并将指向该空间的常量指针传递给建模引擎软件。这些参数在函数头中显示为 type name[]或 type*const name，其中 name 是一个在编译时长度未知的数组。分配的空间在不再需要时由应用程序释放。
- **使用应用程序注册的内存分配函数分配内存空间**
  应用程序需声明一个指向返回类型的指针，并将一个指向该指针的常量指针传递给建模引擎软件。建模引擎软件将指针设置为指向返回的信息。这些参数通常在函数头中显示为 type **const name，也可显示为指针数组。分配的空间在不再需要时由应用程序释放。

内存管理函数

内存管理函数提供了一个接口，通过该接口应用程序可以实现从内核返回的可变长度参数的内存分配和释放：

- 应用程序可以清除由建模引擎软件分配的内存；
- 无论内存最初是在哪里分配的，都可以通过该接口释放空间。

🕮 说明
当返回“type **const name”形式的参数时，应用程序需调用 **freeMemory** 释放为其分配的所有内存。

使用 **allocMemory** 和 **freeMemory** 可为具有分层架构的应用程序分配和释放内存。如果内核函数执行成功，那么可以通过调用 **freeMemory** 来释放任何变长返回值的空间。如果内核函数执行失败，并且已经为任何变长返回值分配了空间，那么内核会通过调用已注册的释放函数来释放这些空间。

memoryRegisterCallbacks 函数可以选择性地注册 free 和 alloc 回调函数。这种类型的内存完全独立于内核会话，用于分配返回结构体、数组和内存块。有关内存管理的更多信息，请参见 2.3.4 内存管理。

包含指针的结构

如果函数返回一个包含指针的结构，其所指向的空间由内核分配，由应用程序释放。

标准形式是固定大小的结构，可指向可变长度的数组。例如 **getBCurve** 返回 **PSGMApiBCurve**，由于其大小固定，因此声明为：PSGMApiBCurve *const bcurve，**PSGMApiBCurve** 包含字段 double *m_knots 和内核分配给矢量节点的空间。

一般情况下，当返回的信息具有两层间接性时，空间由内核分配，通常通过 type **const name 的形式来声明。两层间接性也可能是分开的，其中第一层是一个指向结构的指针，第二层是结构内部的指针。

2.5 跟踪和标记

介绍建模引擎软件提供的一系列通用、可定制的工具，帮助应用程序精确跟踪实体创建和操作。例如，特征建模应用程序可以使用这些工具作为持久化标签策略的基本组件。

2.5.1 可用工具

本节介绍用于实现跟踪机制的工具和信息源。对于应用程序来说，只使用单一工具无法完整的实现跟踪机制，最佳的工具组合取决于应用程序中采用的总体跟踪策略以及使用的特定建模操作。

API 函数返回信息

大多数更改拓扑和几何实体的函数通过其返回的参数报告相关的变更信息。

跟踪信息可以以多种方式返回，具体取决于所使用的 API，包括：

- 使用数组或数组集合。
- 使用专用的跟踪返回结构。

跟踪信息始终返回由操作创建的新实体，并可能包含有关从中派生出新实体的原始或种子主体的信息，从而允许应用程序跟踪指定实体的来源。

错误诊断和状态返回码

所有的 API 接口函数在完成时都会返回操作成功或失败的相关信息。这些信息可以告诉您函数调用是否完全失败（即操作未完成），或者操作的状态（即函数已完成，但可能没有产生预期的结果）。

当函数失败并返回非零错误码（即不同于 NoError 的错误码）时，就会发生错误。例如，如果调用函数 **createSolidCylinder** 时使用负半径或高度，它将返回 GeomGenRadiusLessZero。如果返回了这样的错误，那么任何返回参数的值都是未定义的，不应用于跟踪（或其他任何）目的。

即使 API 函数返回零错误码（NoError），在其未能按预期执行任务且需返回更详细的问题诊断时，它也可能在函数的返回参数中返回一个失败状态码。

有关处理 API 函数返回的错误信息，请参见 16.2 错误处理。

标签持久性

当建模操作修改了实体时，标签的变更遵循以下规则：

- 当实体作为建模操作的一部分进行拆分时，其中一个生成的实体将保留原始实体的标签，另一个为新标签。
- 当两个或多个实体合并在一起形成单个实体时，将保留其中一个原始实体的标签，保留的标签取决于操作的性质。例如，当两个模型在一个简单的布尔运算中合并生成一个实体时，该结果实体与传递给布尔运算的目标实体具有相同的标签。
- 当实体作为建模操作的一部分收缩时，将保留其原始标签。

您可以在应用程序的跟踪机制中使用管理标记持久性的规则，因为它们允许您在处理建模操作导致的模型更改时在应用程序代码中做出某些假设。有关标签持久性的更多信息，请参见标签。

属性

属性是一种实体，为应用程序提供了一种将附加信息与模型关联起来的方式。当模型发生变化时，属性可以自动修改。您可以将属性附加到拓扑结构上，以及附加到模型中包含的任何几何体上。

属性在其包含的信息、可附加的实体以及建模操作下的行为都是可定制的。

当某个实体被修改时，与该实体关联的属性的行为取决于以下两个方面：

- 修改的性质（执行了什么操作）；
- 创建属性时使用的属性定义类别（描述属性包含内容的模板）。

例如，一个带有属性的面被分割成两个面，那么该属性可能会被删除、保留在一个结果面上或者传递到两个结果面上，这取决于属性的定义和建模操作的性质。

建模引擎软件预定义了七个属性类。每个属性类都有一套规则，用于管理该类属性在一组低标准建模操作下的修改方式。这些规则确保了属性的行为是可预测和一致的，

有助于维护模型的数据完整性和准确性。属性定义的更多信息，请参见 13.2 属性定义。

属性的更多信息，包括如何创建、修改和应用属性，请参见 13.3 属性。这些章节将提供详细的指导，帮助您充分利用属性来增强你的模型和应用程序的功能。

属性回调

当模型以特定方式修改时，属性的行为可通过属性回调来定制。您可使用属性回调来让应用程序观察（在只读回调的情况下）属性的行为，或者在常规回调的情况下拥有关于该属性行为的完全决策权。

由于属性的可定制性（在内容和行为方面），属性和属性回调可以成为应用程序跟踪和标记策略中的关键工具。根据应用程序的需求来定义和控制属性的行为，从而确保数据的一致性和准确性。

在建模引擎软件操作中，属性被广泛应用，用于标记和跟踪模型变化期间实体的变化。

> 说明
> 请避免修改已创建的属性定义（例如扩展该定义相关属性可附加的实体列表，或扩展定义中的字段集），这可能导致应用程序出现严重的模型兼容性问题。即使属性定义具有相同名称，但如果定义本身在其他方面有所不同，它们也不能在同一会话中共存。

属性回调的更多信息，请参见 13.2.2 属性回调。

通过回滚跟踪实体变更

如果应用程序实现了分区回滚，通过回滚（或向前滚动）操作新增、更改或修改实体的信息也是一个有效的跟踪信息来源。通过跟踪这些变化，您可以更准确地了解模型在编辑过程中的状态，并对其进行更有效的管理。

更多信息，请参见滚动到分区标记。

通用跟踪信息

低级模型变更

标签持久化、属性行为和由回滚操作报告的变化均由低级模型变更（如拆分、合并、创建、删除和更改）产生，这些变化是指对模型的二进制和低级更改。通常，一系列低级建模操作会链接在一起，以形成在特定 API 调用下对模型进行更改。

实体序列

大部分的 API 接口返回的实体顺序是不固定的，因此应用程序不应依赖于从函数中返回实体的顺序，除非 API 文档明确说明顺序是有保证的。

其他跟踪实用程序

建模引擎软件还提供如下使用程序用于跟踪信息：

表 2-12 其他跟踪实用程序

实用程序	说明	参考文档
拓扑和几何连接性查询	在建模前后可通过查询函数检查拓扑和几何的连接性。这种策略通常与本章中描述的其他工具一起使用，是构成持久性标记策略的重要组成部分。	5.2 查询功能
几何定位信息	存储一个与面或边一起的三维空间点可助于唯一地标识一个实体，在建模引擎软件接口中，这些点被称为辅助点（help points）。为了创建或确定合适的辅助点，建模引擎软件提供了一系列查询函数，帮助确定边上的一个点或面内的一个点，或者查询指定的点是否位于面内或边上。	5.2 查询功能
用户字段	用户字段提供了一种将固定长度的字节数组与每个建模引擎软件实体相关联的方法。您可以在与指定定建模引擎软件实体相关联的用户字段中存储应用程序特定的信息。	15.2.4 用户字段

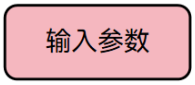
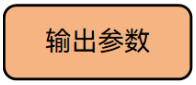
2.5.2 跟踪示例

本节将通过一个简单的示例来展示如何使用建模引擎软件 API 返回的信息来跟踪数据，以便在建模会话中的不同函数调用间传递实体。该示例将演示如何使用基本图形创建一个实心体，然后添加一个边缘圆角，最后添加一个圆柱形凸台。涉及的过程如下：

1. 创建一组曲线（curve），作为片状体轮廓（sheet profile）的基础。
2. 创建片状体轮廓：将曲线组合成线框体（wire body），创建面（face），并附加一个曲面（surface）。
3. 扫描轮廓以形成实心体（solid body）。
4. 对主体的一条边（edge）进行圆角处理。
5. 在主体上添加一个圆柱形（cylinder）凸台。

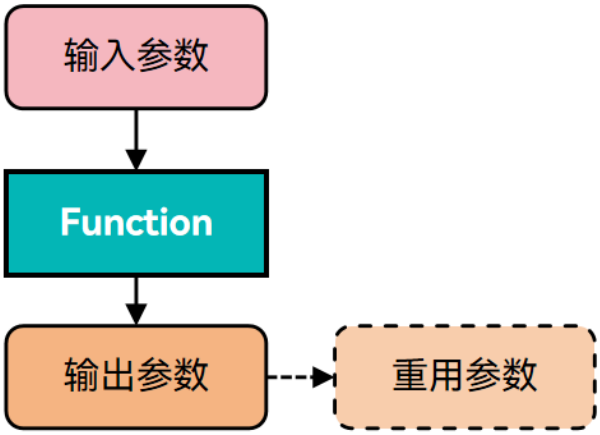
📖 说明
这个例子只是本章中描述的一些功能的简单演示。并不是执行跟踪的唯一方法，也不是最佳实践。

在整个示例中，函数调用图说明了需要提供给内核函数的参数以及返回的信息。这些图表说明了如何在整个建模过程中跟踪信息，以及如何在函数之间传递参数。图 2-3 函数调用图模板解释了这些图表的布局，每个图表都包含以下列出的组件：

组件	说明
函数名称 	函数名称以粗体显示在每个图的中央框中。
输入参数 	输入参数显示在图表的顶部，显示的参数名称与 API 接口编程参考手册相关条目中使用的字段名称相同。
输出参数 	输出参数显示在图表的底部，显示的参数名称与 API 接口编程参考手册相关条目中使用的字段名称相同。
重用参数 	在后续函数调用中重用的输出参数使用虚线箭头指示。此外，如果参数在具有不同名称的字段中重复使用，该名称将显示在图表中。

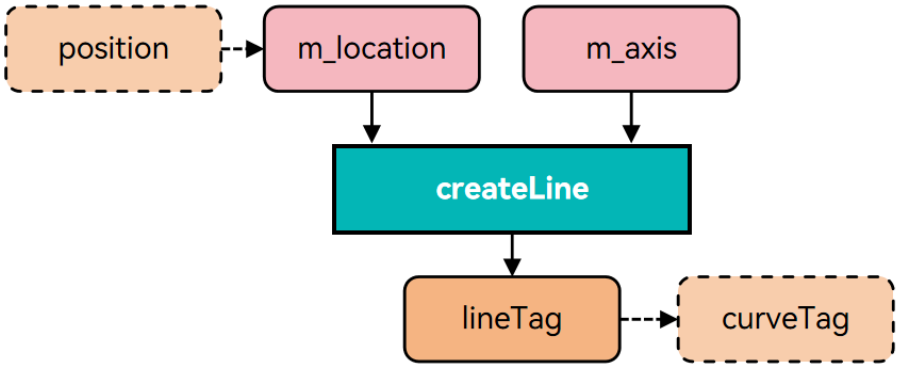
说明
函数调用图仅显示接收到的和返回的参数，这些参数对于示例过程中的信息跟踪非常重要。许多函数调用还将涉及其他参数（如选项结构），为了简单起见，此处未说明这些参数。

图 2-3 函数调用图模板



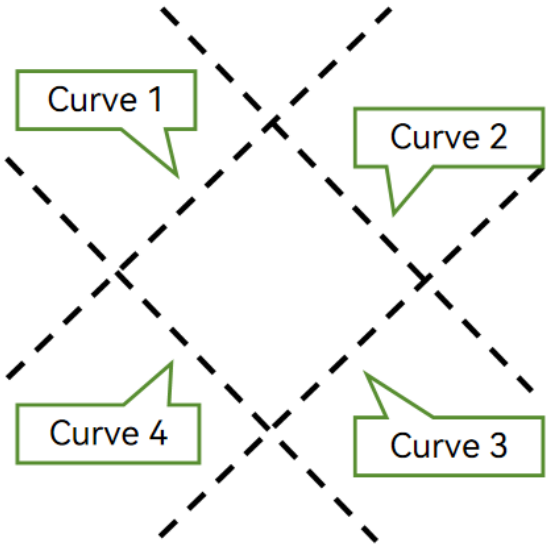
创建曲线以构建轮廓

图 2-4 调用函数创建曲线



首先，您需要创建一系列曲线，这些曲线将用于形成轮廓。简单的块的轮廓由四条线组成，通过对 **createLine** 的四个调用来创建这些线，通过 line 标准形式将 m_location（点）和 m_axis（方向）传递给每个调用，以创建四条在指定方向上穿过指定点的无边界线。

图 2-5 创建曲线以定义轮廓

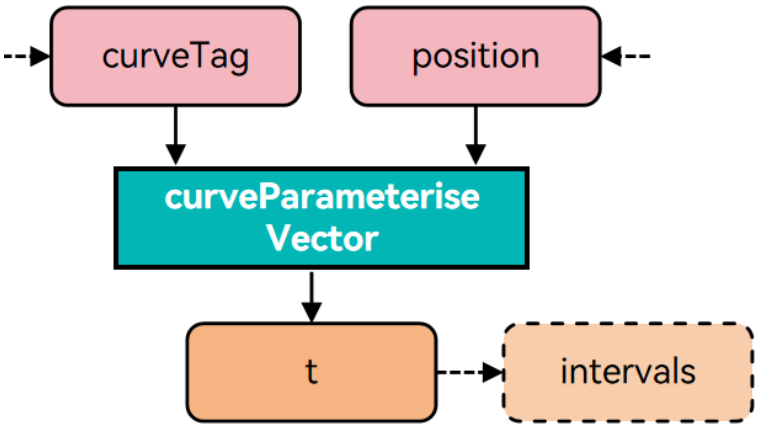


每次调用线返回参数中的 **createLine** 都会返回结果曲线的标签。

创建一个线框轮廓

使用 **createWireBodyByCurves** 来创建一个线框体，以定义一个片状体轮廓，使用 **curveParameteriseVector** 查找轮廓中每条线的间隔。

图 2-6 调用函数查找曲线上的参数

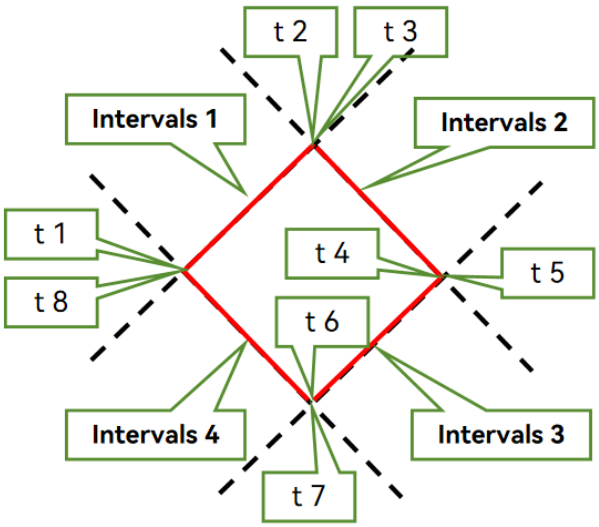


对于轮廓边界中的每一条线，调用 **curveParameteriseVector** 两次（共 8 次调用）：

- 第一次调用输入直线（curveTag）和直线经过的一个点（position）。
- 第二次调用输入同一条直线和该直线经过的另一个点。

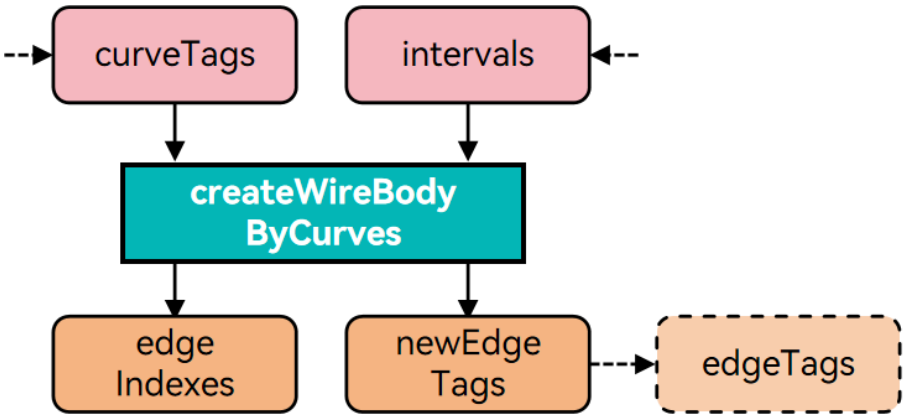
每次调用都会输出指定点在线上的曲线参数（t）。因此，该条线的间隔是包含这两个输出参数的数组。

图 2-7 创建线以定义轮廓



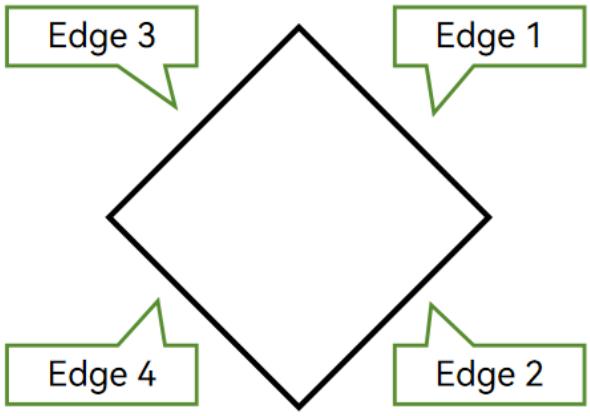
使用 **createWireBodyByCurves** 制作线框体，其中 curveTags 由调用 **createLine** 返回， Intervals 由调用 **curveParameteriseVector** 返回。

图 2-8 调用函数创建线框体



createWireBodyByCurves 为输入到的每条曲线创建一条边，并在 newEdgeTags 数组中返回这些边。因为该函数不能保证按输入曲线的相同顺序输出边，所以它还返回 edgeIndexes 数组，该数组将返回的 newEdgeTags 映射到它们的原始曲线。

图 2-9 创建一个线框体轮廓

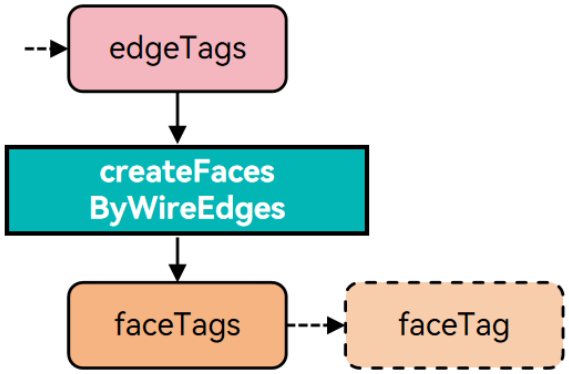


创建片状体轮廓

将现有的线框体作为扫掠操作的轮廓，可创建一个片状体。由于要创建实心块，首先需要将此线框体转换为一个薄片。要执行此操作，请先添加一个面，然后将曲面附着到该面。

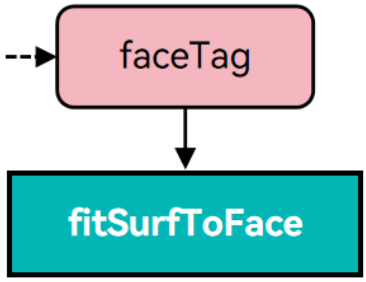
若要将面添加到线框轮廓，请使用 **createFacesByWireEdges**，将面附着到线框体的闭合环中。

图 2-10 调用函数将面附着到线框体



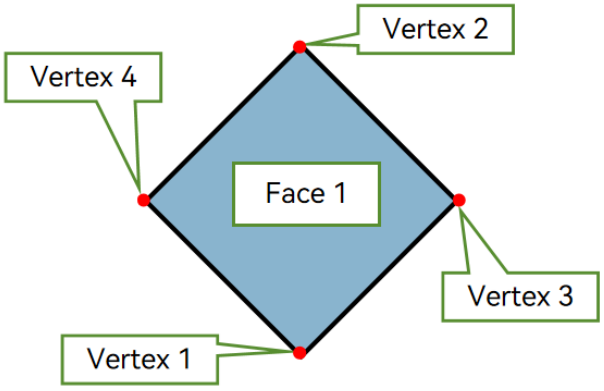
您需要将要创建的面的每个闭合环中的一条边传递给此函数，即调用 **createWireBodyByCurves** 返回的 newEdgeTags 中的一条边。通常情况下，应使用 edgeIndexes 数组将 newEdgeTags 与 **createLine** 返回的 lineTag 进行比较，以确保从每个闭环中选择一条边。

图 2-11 调用函数将曲面附着到轮廓的面



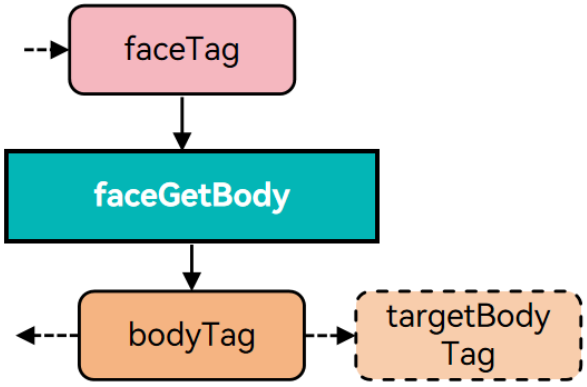
createFacesByWireEdges 输出为线框体创建的新面，将此面传递到 **fitSurfToFace** 中，以创建合适的曲面并将其附着到该面，从而创建所需的片状体轮廓。

图 2-12 通过线框体创建一个片状体轮廓



将面传递给 **faceGetBody** 以检索片状体轮廓主体，并将此主体传递到后续的几个函数调用中。

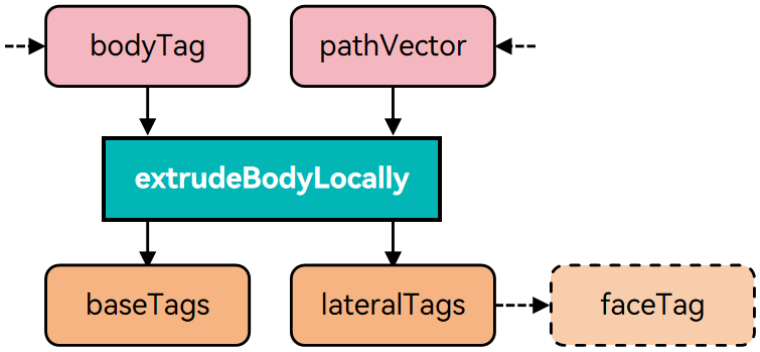
图 2-13 调用函数查找轮廓主体



拉伸轮廓以创建实心体

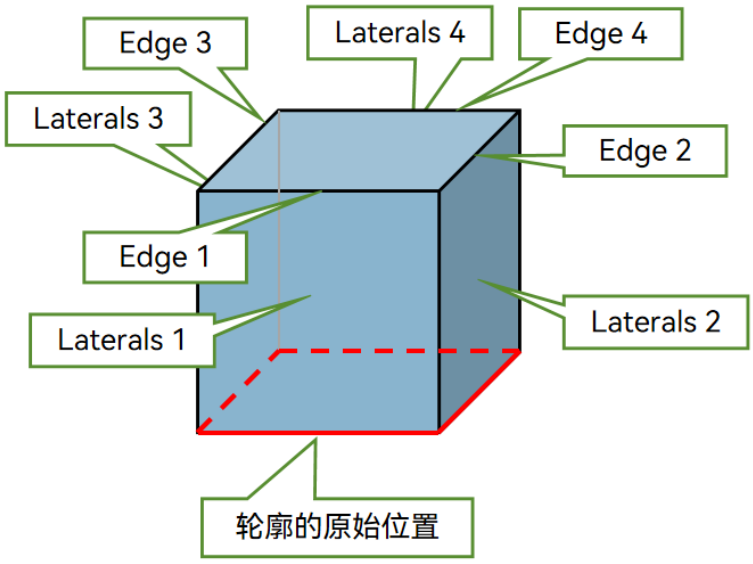
使用 **extrudeBodyLocally** 从已创建的片状体轮廓创建实心体。输入已创建的轮廓体以及定义拉伸方向的矢量路径。

图 2-14 调用函数将轮廓拉伸为实心块



extrudeBodyLocally 返回多个由拉伸操作创建的新边或面（lateralTags）以及原始轮廓中生成这些侧面的实体（baseTags）。

图 2-15 将轮廓拉伸为实心体

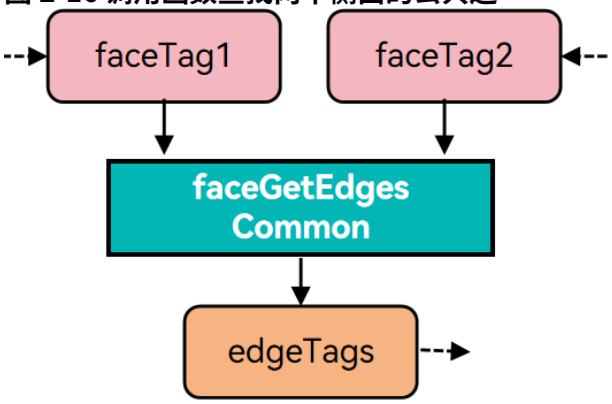


根据这些信息，可以确定拉伸体中的任何实体：

- 可以通过调用 **bodyGetFaces** 查找附加面（查找的面不能为原始轮廓和侧面中的面）。
- 可以通过调用 **faceGetEdgesCommon** 来查找面之间的边。
- 可以通过调用 **faceGetEdges** 来查找指定面周围的边。
- 根据需要，可以通过调用 **edgeGetVertices** 来查找边之间的顶点。

再使用 **faceGetEdgesCommon** 来识别第一个侧面和第二个侧面之间的公共边，用于后续的倒角操作。

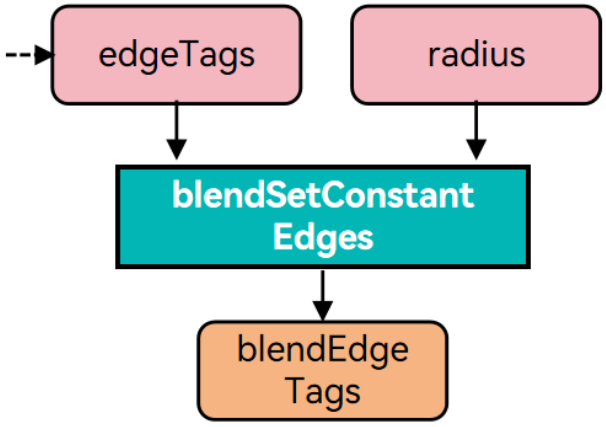
图 2-16 调用函数查找两个侧面的公共边



倒角边

将上个步骤中查找到的边和倒角半径传递给 **blendSetConstantEdges**，为这些边设置具有恒定半径的圆角倒角操作属性。

图 2-17 调用函数设置倒角边



通过 **blendApplyBody** 来应用倒角，该接口将在 blendTags 中返回新的倒角面，在 underFaceTagArrays 中返回与新倒角面关联的原始模型中的基础面。

图 2-18 调用函数应用倒角

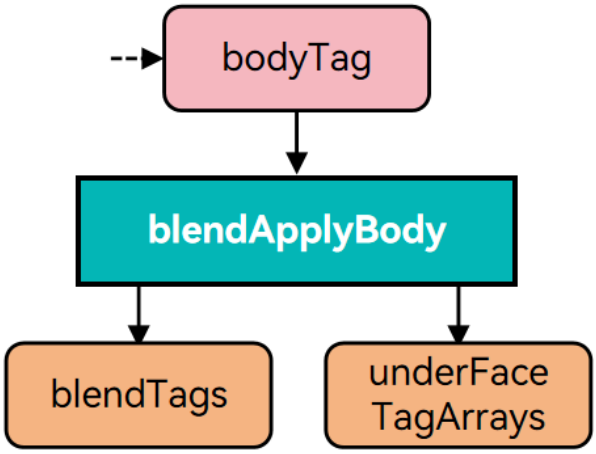
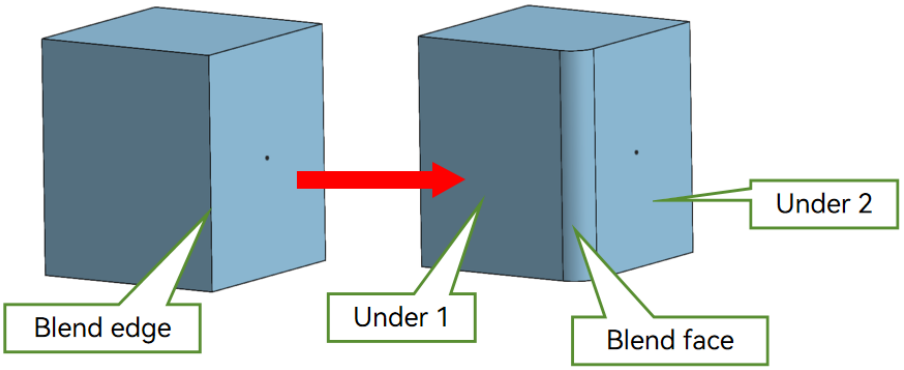


图 2-19 在一条边上添加倒角操作



说明

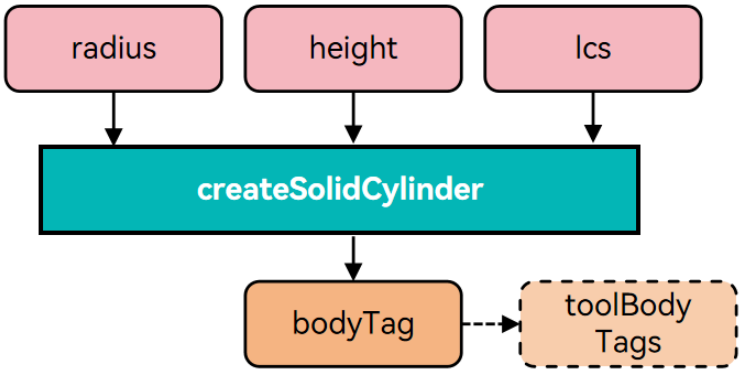
该示例仅倒角一条边，如果倒角了多条边，则每条倒角的边可能具有一组不同的基础面。已倒角的原始边的标签会在 topos 中返回，该标签会在应用倒角后失效，因为原始边已不存在。

添加圆柱形凸台

向已创建的倒角块中添加一个圆柱形凸台：创建一个圆柱体，在圆柱体和倒角块之间执行布尔合并。

使用 **createSolidCylinder** 创建具有指定半径、高度和位置的圆柱体。

图 2-20 调用函数创建圆柱体



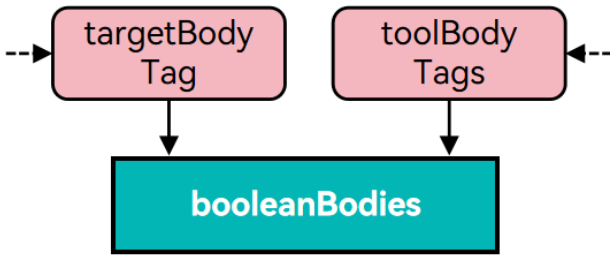
createSolidCylinder 是一个基本主体创建函数，在本示例中，您可使用 **bodyGetFaces** 查询该圆柱体的每个面，根据返回面的顺序来唯一标识每个面。

说明

一般情况下，不应依赖内核函数返回实体的顺序。

将创建的圆柱体作为 toolBodyTags，倒角块作为 targetBodyTag，调用 **booleanBodies** 进行布尔操作。

图 2-21 调用函数联合圆柱体和倒角块

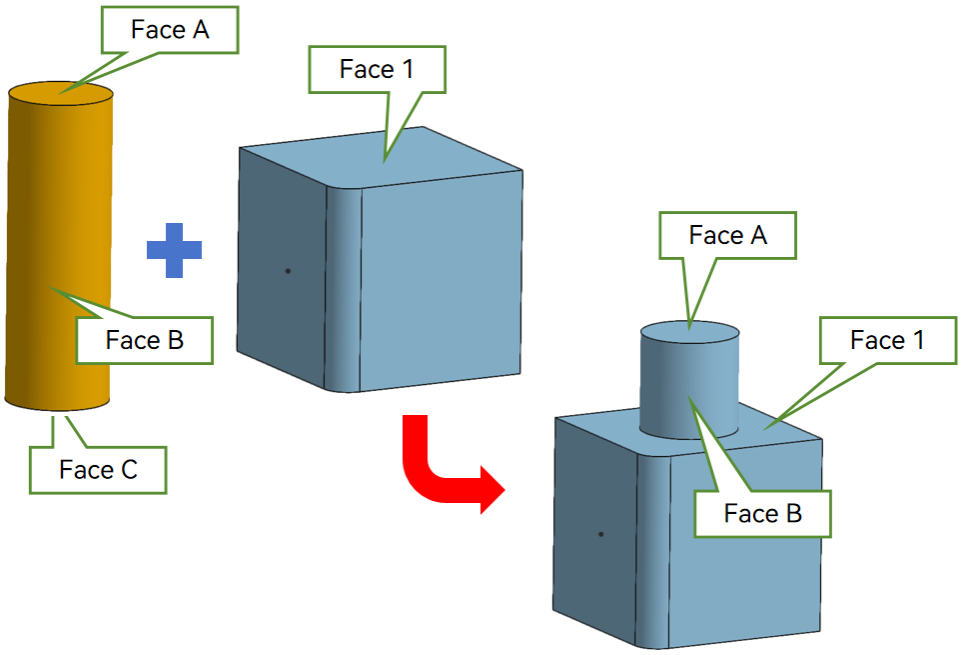


您可使用如下方法跟踪新生成的实体信息：

- 使用 **booleanBodies** 提供的跟踪返回值来追踪最终模型中的新边；

- 依靠标签持久性来发现凸台的面和修改后的面来源于哪些实体。在建模会话的这个阶段，模型中的所有面、边和顶点都可以被唯一识别。
- 采用属性标记的方法（如属性所述），能够准确地追踪所有拓扑类别中的拆分和合并事件。

图 2-22 合并倒角块和圆柱体以创建凸台



2.6 集成建模引擎软件

本小节介绍如何将建模引擎软件集成至应用程序中。

安装文件介绍

建模引擎软件安装文件包含如下文件夹：

- inc/api：包含内核所有接口、数据结构声明的头文件，您需将头文件添加至应用程序项目库中以使用内核的功能。
- lib：库文件，您需在应用程序项目库中链接库文件。
- example/src：示例代码文件，供您参考和学习如何调用本软件函数。

集成前准备

在集成建模引擎软件前，请确认您已完成如下准备。

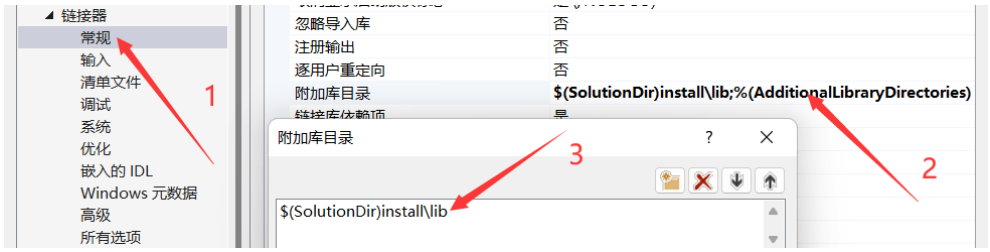
表 2-13 准备事项

准备项	说明
操作系统	本软件支持以下系统，本文以在 Windows 中集成为例： ● Windows — Windows 10（64-bit） — Windows 11（64-bit） ● Linux — OpenEuler：基于 OpenEuler 的商业发行版，如银河麒麟 V10 SP3 — Centos：CentOS-7-x86_64
建模引擎软件安装文件	在泊松服务网站或华为云商店中获取建模引擎软件安装文件。 说明 不同操作系统软件包不同，请注意选择与您系统匹配的软件包。
License 文件	购买 Geoshape 几何建模引擎软件并签订合同后，将合同号、申请 License 的数量及对应 MAC 地址以邮箱形式发送给泊松支持人员，以获取 License 文件（PoissonGM.lic）。
应用程序代码	您的应用程序需包含如下代码： ● 调用内核函数的代码； ● 提供给内核的回调函数代码，详细信息可参见 2.3 提供回调函数。
应用程序项目库	已创建应用程序项目库并安装 C++工具，本文以 Visual Studio 2019 为例。

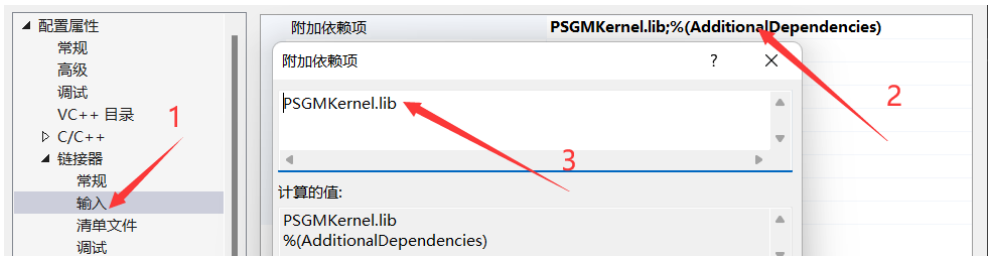
集成步骤

1. 打开 Visual Studio，创建或打开您的 C++项目。
2. 右键单击项目名称，选择“属性”，进入项目的属性页。
3. 在**配置属性→C/C++→常规**中，将建模引擎软件安装文件中的“inc”文件夹添加到“附加包含目录”中（实际目录以您存放安装文件路径为准），使编译器能够找到并包含头文件。

4. 在**配置属性→链接器→常规**中，将“lib”文件夹添加至“附加目录库”（实际目录以您存放安装文件路径为准）。



5. 在**配置属性→链接器→输入**中，将库文件（PSGMKernel.lib）添加至“附加依赖项”中，完成库文件链接设置。



6. 在 Visual Studio 上方的菜单栏中，选择**生成→生成解决方案**，编译并生成可执行文件。
7. 将“lib”文件夹中的 dll 文件和 License 文件添加至可执行文件的目录中。
8. 在 Visual Studio 上方的菜单栏中，选择**调试→开始执行**，运行代码。

使用内核函数

您的代码通过包含相应的头文件来使用内核的功能。例如：

```
#include "PSGMApi.h"
```

然后，您可以调用内核提供的接口来实现您的需求。具体的接口描述及使用方法请参见*《Geoshape 几何建模引擎软件 接口开发文档》*。

使用示例代码文件

“example/src”文件夹中包含了多个示例代码，展示了如何使用内核的不同功能。您可在 Visual Studio 项目中添加示例代码文件，进行编译和运行。

示例代码的详细说明可参见*《Geoshape 几何建模引擎软件 接口开发文档》*的“示例代码”章节，通过代码分析可帮助您理解如何使用内核功能。

3 几何建模基础概念

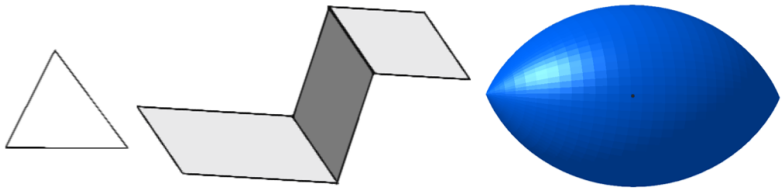
3.1 内容简介

本章介绍使用建模引擎软件创建、编辑和处理模型时所涉及的基本概念。同时也会介绍用于创建模型的部件、主体的类型和建模引擎软件模型空间使用的精度。此外，本章还介绍了建模引擎软件的几何实体和数据传输能力。

本章包含以下章节:

- 3.2 模型结构：介绍组成建模引擎软件模型的实体或用于管理建模引擎软件会话的实体。
- 3.3 主体类型：介绍建模引擎软件中的主体类型。
- 3.4 会话精度和局部精度：介绍建模引擎软件中的精度。
- 3.5B 曲线和 B 曲面：介绍 B 几何在建模引擎软件中的应用。
- 3.6 变换：介绍建模引擎软件支持的齐次坐标变换（homogenous coordinate transformations）。
- 3.7 装配体和实例：介绍装配体和实例，并说明它们在传输文件中的用途。

图 3-1 建模引擎软件中使用的主体类型



3.2 模型结构

介绍组成建模引擎软件模型的主要实体和用于管理建模引擎软件会话的实体。这些实体分为四大类型：

- 拓扑实体
- 几何实体
- 会话管理对象

- 其他数据实体

可以使用建模引擎软件中的查询和输出接口找到这些实体之间的连接信息。更多内容请参见 5.2 查询功能。

说明
相关内容:

- 3.3 主体类型
- 3.7 装配体和实例

3.2.1 拓扑实体

拓扑实体(topological entity)是用于构成模型骨架的实体。本节将详细介绍不同类型的拓扑实体。

主体

主体（body）是建模引擎软件用来建模的基础拓扑实体。

一个主体由一个或多个部件（component）组成，以下是可用来组成主体的部件的类型：

表 3-1 部件类型

部件类型	描述
空心体（empty body）	一个无界空心区域。
橡子体（acorn body）	两个或两个以上孤立顶点。
线框体（wire body）	一个在任何顶点处不超过 2 条边的相连的边集合。
片状体（sheet body）	在任意边上相交不超过两个面的一组连接面。片状体可以是开放的，也可以是闭合的。
实心体（solid body）	一个实心区域。
一般实体（general body）	以下任何实体的连接集合： ● 块（lump） ● 面（face） ● 边（edge） ● 顶点（vertex）

这些部件可以组合成以下类型的主体：

表 3-2 可由部件组合成的主体类型

类型	维度	描述
空体（empty body）	0	零个部件。空体具有不包含任何部件的无限空区域。
橡子体（acorn body）	0	两个或两个以上孤立顶点。 最小体（ minimum body）只有一个孤立的顶点。可以使用 **createMinimumBodyByPoint** 创建一个最小主体。
线框体（wire body）	1	一个或多个线框体部件。线框体有一个或多个块、一条或多条边、零个或多个顶点。
片状体（sheet body）	2	一个或多个片状体部件。片状体有一个或多个块、一个或多个壳、一个或多个面和曲面。
实心体（solid body）	3	一个或多个实心体部件。每个实心体部件都有一个连续的边界体积。实心体有一个或多个块、一个或多个壳、一个或多个面和曲面。
一般实体（general body）	0，1，2，3，or mixed	可包含以下主体： ● 一般部件（general components） ● 不同类型的部件（components of different types） 说明 一般部件可以是非流形的。有关流形和非流形主体的信息，请参见 3.3.1 流形主体。

您可以使用 **bodyGetType** 找到指定主体的类型。

部件相关内容请参见部件。

如果情况允许，可以使用 **bodySetType** 来转换主体的类型。例如，可以将实心立方体转换为封闭的片状体。有关更多细节，请参见 3.3.4 更改主体类型。

包含一个以上部件的主体称为不相交主体（disjoint body）。例如，如果一个不相交的片状体由几个片状体部件组成，每个片状体部件都由一个或多个可以开放或封闭的面组成。

图 3-2 有效线框体和无效线框体

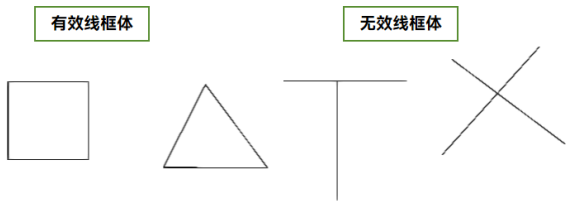
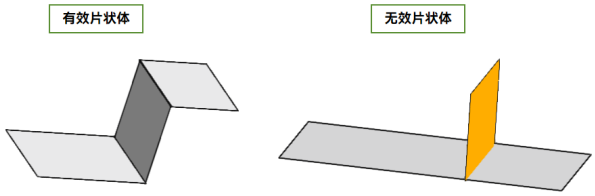


图 3-3 有效片状体和无效片状体



主体可以包含以下拓扑实体：

- 块
- 壳
- 面
- 环
- 边
- 半边
- 顶点

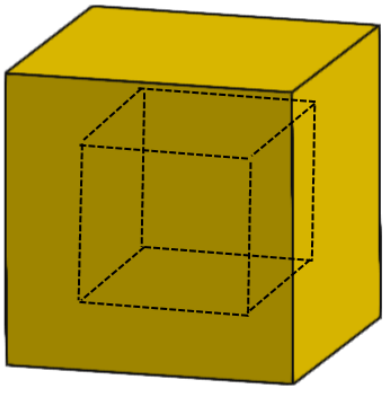
此外，主体中的每个面、边和顶点都需要附加一个几何实体（一个曲面、曲线或点），才能作为一个完全定义的、有效的模型。

块

块（lump）是三维空间的连通子集，其边界是一组壳（如果块的类型为线框体，则其边界为一组边）。 可以使用 **lumpGetType** 来查询一个块的类型。块的类型有：实心体、片状体和线框体。

下图所示的中空立方体有一个块，即实心体材料所占据的部分为一个块。

图 3-4 中空立方体



壳

壳（shell）是一组面的集合。

面

一个面（face）是一个曲面（surface）的有界子集，其边界是多个或零个环的集合。

拥有零个环的面可以形成一个闭合的实体，如一个完整的球面。

环

一个环（loop）是一个面边界的连接部件。一个环可以是：

- 由不同半边组成的一个有序环形结构。
- 一组顶点。

半边的顺序表示其所属边在环中公共顶点上连接的顺序，在连接边的同时需要考虑每个半边的方向。

环在每个方向上不能包含重复的边。

当从面上方向下看并沿着环的方向看时，面在环的左侧。

如果环有半边，则环中的顶点必须连接半边所属的边。没有半边的环只有一个顶点。

环有以下三种构成方式：

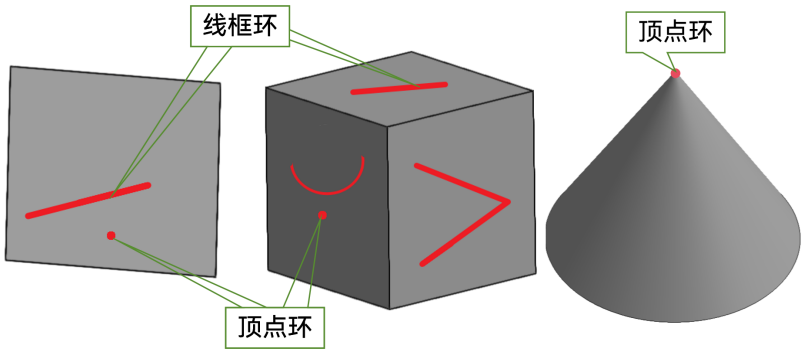
- 由一个半边构成，该半边所属的边上没有顶点。
- 至少由一个半边和一个顶点构成。
- 由一个顶点构成，也被称为孤立环。

建模引擎软件中有几种不同类型的环，环的类型取决于环所属面的底层曲面。

顶点环与线框环

顶点环和线框环是通用的输出参数，在实心体和片状体上这些环类型取决于面内的边和顶点。如图所示：

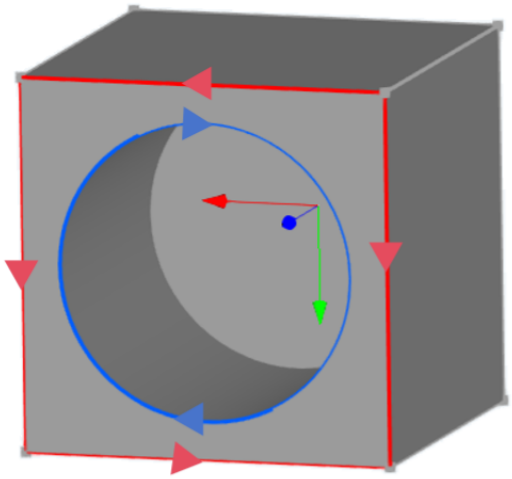
图 3-5 顶点环与线框环



内环与外环

非周期曲面上的每个面都包含一个外环（outer loop），和任意数量的内环（inner loop）。外环围绕面的边界，内环为面上的孔洞，如下图所示：

图 3-6 一个面上的外环与内环



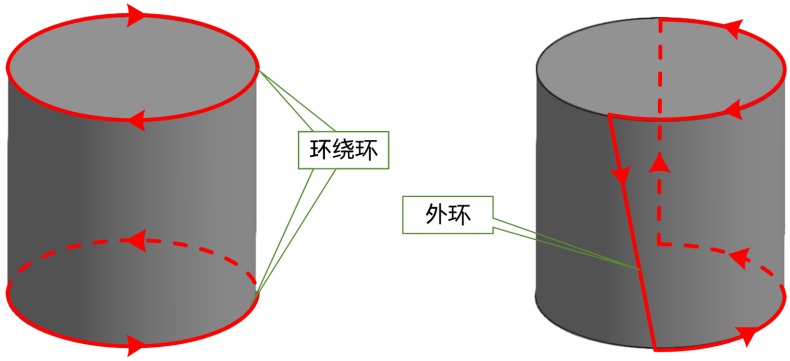
如果您需要找到面的参数外环（parametric outer loop），请使用**faceComputeOuterLoop**。

环绕环

在圆柱拓扑的周期参数空间中围绕自身一圈的封闭环被称为环绕环（winding loops）。要在圆柱面上定义面的边界，您需要：

- 一组匹配的环绕环
- 一个外环

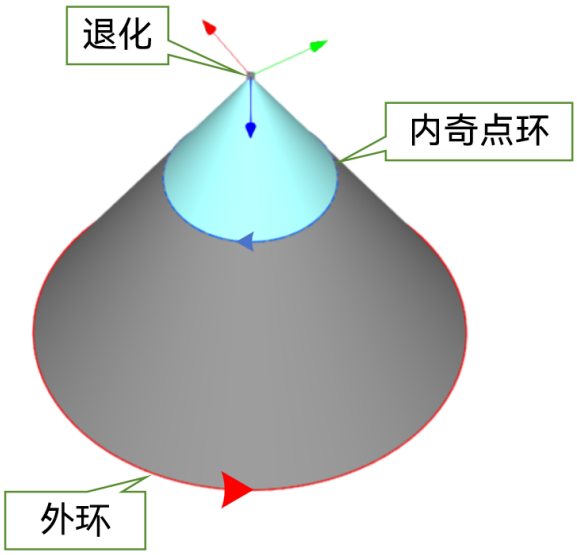
图 3-7 使用环绕环和外环定义圆柱面的边界



奇点环

与非周期性面类似，圆柱拓扑可以包含任意数量的内环。如下所示的圆锥面，在 U 参数上是周期性的，在 V 参数上退化成点。只包含一个外环和任意数量的内环。如果内环围绕退化点，则它们也是环绕（winding）的，下图中蓝色环为灰色面的内奇点环。

图 3-8 圆锥拓扑中的环

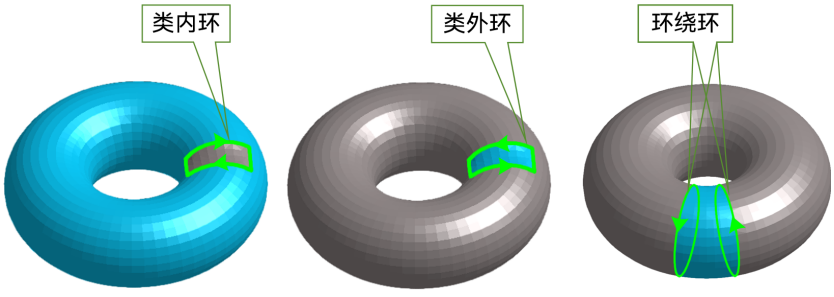


类内环与类外环（likely inner/outer loops）

对于在两个参数上都呈现周期性的曲面，如圆环面，可能存在环绕环。这些环在任一参数方向上可能环绕一次或多次。除此之外的任何其他环，要么构成了开放区域（open regions）的外边界（即外环），要么形成了整个封闭曲面的孔洞（即内环）。包含环绕环的往往是内环，不完全包含在参数空间中的环可能是外环。

双周期曲面上看起来像外环的环为类外环（likely outer），双周期曲面上看起来像内环的环为类内环（likely inner）。如图 3-9 圆环面中的环所示。

图 3-9 圆环面中的环

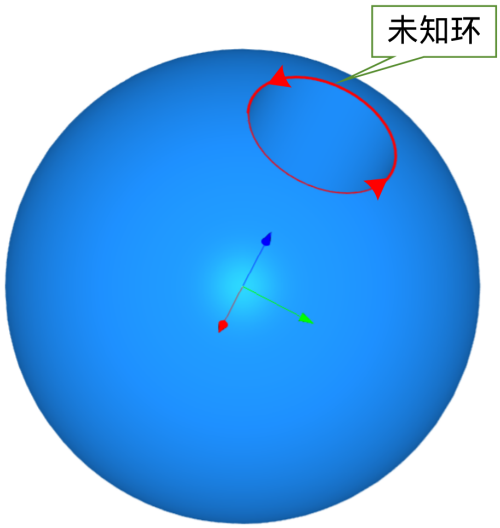


未知环

球面拓扑中的封闭环（在一个参数中封闭，在另一个参数中退化）将整个曲面分为两部分。球面上有三种类型的封闭环：

- 不环绕极点的类外环。
- 环绕两个极点的类内环。
- 未知环为仅环绕一个极点的闭合环，将曲面分为两部分。如图 3-10 未知环示例所示。

图 3-10 未知环示例



您可以使用 **loopGetType** 查询环的类型。

半边

半边(coedge)用于指明边在不同面上的方向。简单来说，它告诉我们在环的路径中，边如何被有方向地使用。半边具有以下属性:

- 方向（sense）：指示半边的方向以及其拥有环的方向是否与其拥有边的方向相同。
- 曲线：半边所属的边必须设置局部精度，否则该曲线为空。具有局部精度的边不会附着几何，只在其半边上附着几何。

边

边（edge）被用来表示单个曲线的一段有界部分。边的边界是由零个、一个或两个顶点组成的集合。一条边拥有以下内容：

- 一个起始顶点，该点可能为空。
- 一个结束顶点，该点可能为空或与起始顶点相同。
- 零个或多个半边。
- 一条曲线，该曲线可能为空。

半边的顺序代表其所属的面关于边的空间顺序（spatial ordering），当应用右手螺旋法则时（即朝着边的方向看），半边的顺序是顺时针的。

边可以有零个或任意数量的半边：

- 线框边没有半边。
- 层流边有一个半边。
- 如果一个普通边（normal edge）不是线框边和层流边，则该普通边具有两个相反方向的半边。
- 一般边（general edge）有两个具有相同方向的半边。

您可以使用 **edgeGetType** 来返回指定边的所有信息。如果边中的一个顶点为空，则另一个顶点也必须为空。在这种情况下，边表示整个周期曲线。

顶点

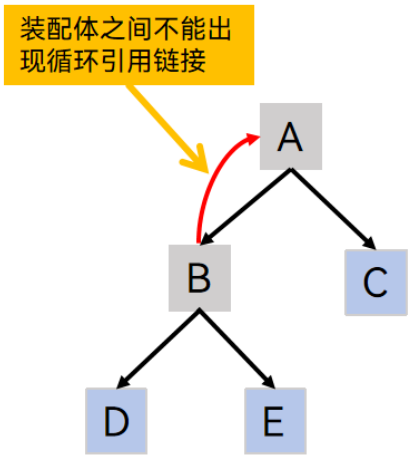
顶点（vertex）表示空间中的一个点。顶点有一个点，该点可能为空。

装配体

装配体（assembly）是一组实例的集合，其中每个实例引用一个模型（一个主体或另一个装配体）和一个变换，该变换用于将模型固定在装配体的坐标系中。

装配体不能直接或间接地引用自身。因此，装配体和模型的树状结构不能是循环的，如下图所示。

图 3-11 装配体和模型的循环树和无循环树



装配体还可以具有构造几何，包括直接附加到装配体的曲面、曲线和点。

使用 **assemblyCreateEmpty** 可以创建一个新的空装配体。

使用 **assemblyGetModelsAndTransforms** 可以查询指定装配体中所有的模型和变换。

变换必须是刚体运动或：不能使用缩放、剪切或透视变换。

实例

一个实例（instance）可以引用其所属的装配体、它所实例化的模型以及一个变换（transform）。如果变换为空，则该变换被视为恒等变换（identity transform）。

使用 **createInstance** 可以创建一个新的实例。该接口假定实例所引用的模型和变换已经被创建。

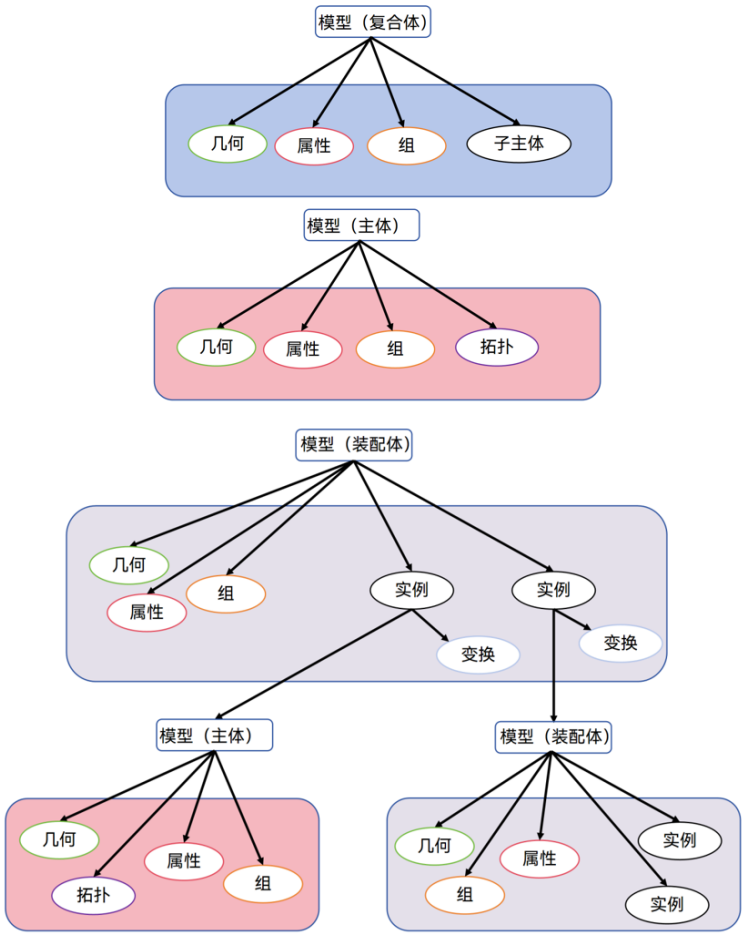
模型

模型（model）表示一个主体或一个装配体。

要将模型存档到外部存储，请使用 **saveModel**。要将模型从外部存储加载到内存中，请使用 **loadModel**。有关更多信息，请参见 13.7 存储 。

模型中的每个实体都有一个标签和一个标识符（半边没有标识符）。标签在所有建模引擎软件会话中是唯一的，标识符只在指定的主体或装配体所在的会话中是唯一的。

图 3-12 模型内标识符的唯一性



上图说明了模型内标识符是唯一的。图中每个彩色框中的实体都具有唯一的标识符，但跨彩色框的实体则没有。因此，主体中的实体有可能与引用该主体的装配体中的实体具有相同的标识符。

部件

主体由一个或多个部件（component）组成，其中每个部件都是由一组拓扑实体构成的最大连接集。一个单独的部件是从主体中任一拓扑结构开始，按以下任何一种连接方式形成的拓扑实体集：

表 3-3 部件的连接方式

连接方式	描述
块（lump）连接	所有包围实心块的壳都与实心块相连，因此这些壳也是相互连接的。
面（face）连接	面两侧的壳都与该面相连，因此这些壳也是相互连接的。
边（edge）连接	在该边相交的所有面都与该边相连，因此这些面也是相互连接的。
顶点（vertex）连接	在该顶点相交的所有面和边都与该顶点相连，因此这些面和边也是相互连接的。

由于部件是一组实体，而不是单个实体，在建模引擎软件中，部件由包裹部件的壳来表示。

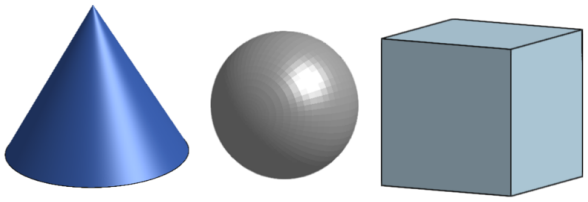
3.2.2 几何实体

几何实体(geometric entity)有三类：

- 曲面（surface）
- 曲线（curve）
- 点（point）

几何实体的主要功能是规定主体的几何形状。这些几何实体被称为结构几何（principal geometry）。在一个完全定义的模型中，几何实体根据图 3-14 几何实体和拓扑实体关系图所示的关系附加到拓扑实体上。

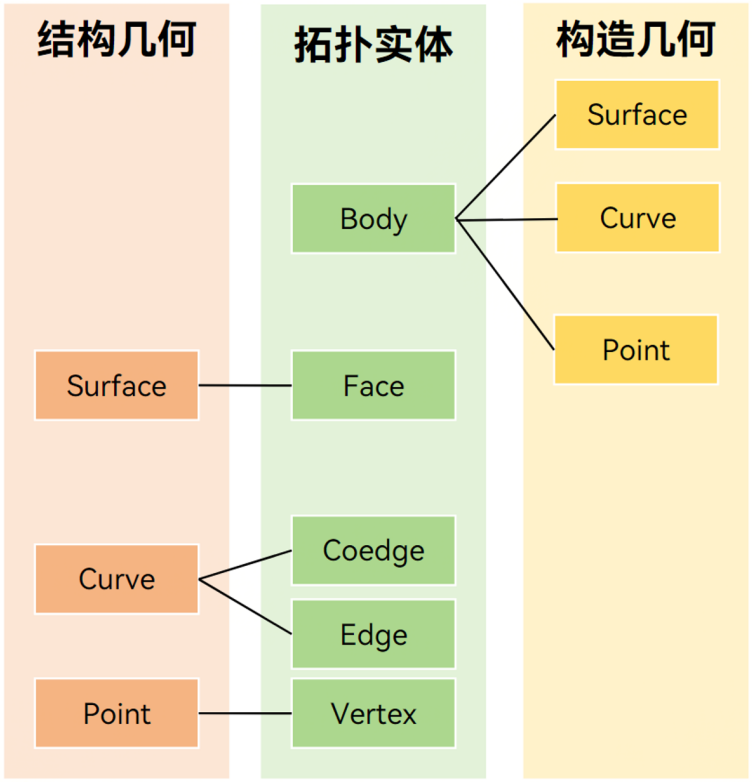
图 3-13 三维几何示例



几何实体也可以直接附加到主体上，以形成构造几何，从而将相关实体保存在主体中。例如，您可能希望将一个点附加到主体上，以表示其重心。这在图 3-14 几何实体和拓扑实体关系图有所说明。

孤立几何体是指未附着到任何拓扑实体（甚至是主体）上的几何体。孤立几何存在于当前分区中，这样就可以在使用几何之前创建几何，例如创建曲面然后再将其附着到主体上。

图 3-14 几何实体和拓扑实体关系图



曲面

曲面（surface）主要附着在面上，但也可以作为构造几何附着在主体上。通常，每个面都附着了一个曲面，但随着模型的创建或修改，它可能会暂时和面分离。曲面有以下类型：

- 平面（plane）
- 圆柱面（cylinder）
- 圆锥面（cone）
- 圆环面（torus）
- 球面（sphere）
- B 曲面（B-surface）
- 偏移曲面（offset surface）
- 扫掠曲面（sweep surface）
- 旋转曲面（spin surface）
- 混合曲面（blend surface）

曲线

曲线主要附着到模型的边或半边上，也可以作为构造几何附加到主体上。曲线有以下几种类型：

- 直线（straight line）
- 圆（circle）
- 椭圆（ellipse）

- 相交曲线（intersection curve）
- B 曲线（B-curve）
- SP 曲线（SP-curve）
- 折线（polyline）

点

点（point）主要附着在顶点上，也可以作为构造点附着在主体上。所有的点都是笛卡尔点（Cartesian points）。

几何共享

可以在单个模型中使用几何共享来减少模型的整体大小，适用于以下情况：

- 具有相同方向的边或半边可以共享同一条曲线。
- 面可以共享同一个曲面。

🕮 说明
离散几何不能以这种方式共享。

几何依赖

如果一个实体的定义引用了另一个实体，则这两个几何实体之间存在依赖关系。例如，偏移曲面（offset surface）的标准形式定义引用了基础曲面（base surface），所以偏移曲面与其基础曲面之间存在依赖关系。

您可以使用函数 **geomGetDependents** 和 **geomGetGeomOwners** 来查找任何指定几何实体的依赖关系。

复制几何

您可以使用 **copyGeometry** 来复制一组几何实体，同时保留它们之间的几何依赖关系。此接口接收结构几何、构造几何、孤立几何或它们的组合，并创建这些几何的副本，同时创建这些几何实体定义中引用到的几何实体。此外，它还能确保将输入几何体中的任何重复内容只复制一次，并在副本中引用而非重复复制。

copyGeometry 按照实体被提供的顺序返回这些实体的副本。如果输入的几何实体包含重复的实体，这些实体只会被复制一次，但在返回的结果中会出现多次。此外，该接口还返回跟踪信息，将原始几何体与它们的副本关联起来。

默认情况下，孤立几何将作为原始几何所在分区中的孤立几何进行复制，而构造几何和结构几何将作为原始几何模型中的构造几何进行复制。

copyGeometry 接收一个几何实体数组，以及一个选项结构，该结构支持以下选项：

选项	描述
m_destinationTag	该几何需要复制到哪个实体中，可能是： ● 建模引擎软件：默认情况下，孤立几何将作为原始几何所在分区中的孤立几何体进行复制，而构造几何和结构几何将作为原始几何模型中的构造几何进行复制。 ● 一个分区：所有复制的几何都将成为所提供分区中的孤立几何。 ● 一个模型：所有复制的几何都将成为所提供模型中的构造几何。
m_copyDependents	对几何依赖的复制级别控制，可以是以下情况： ● Auto（仅在下列一个或多个为真时，才会复制相关的几何图形，Auto 为默认值）： — 该几何将被复制到一个不同的分区中。 — 原始几何和复制几何不能共享几何依赖。 — 依赖几何被显式地指定为输入几何。 ● Always：所有在所提供几何上的几何依赖都会被复制。 ● Never：永远不会复制几何。
m_copyAttributes	是否复制附加到几何上的属性： ● Auto：自动复制所有附加到几何上的属性。 ● Always：所有在所提供几何上的元素都会被复制。 ● Never：永远不会复制属性。

不良几何属性

不良几何属性可能会导致几何出现视觉效果和计算上的问题，因此应尽可能避免。以下是几何具有不良几何属性的一些情况：

- 曲面的 U 和 V 导数近乎平行。
- 几乎但不是完全退化的曲面。
- 具有混合物理退化的曲面。
- 非 C1 连续的曲线和曲面。
- 曲线和曲面 C1 连续，但其导数大小变化很大。
- 曲率较大或曲率变化较快的曲线和曲面。
- 一阶导数大于 1000 或小于 0.001 的曲线和曲面。
- B 样条几何上的节点非常接近。
- B 几何具有不必要的高重数内部节点。
- 非建模引擎软件创建的 B 曲面螺旋线。

3.2.3 会话管理对象

以下类型的会话对象用于回滚管理。

分区

分区（partition）中包含一组模型和其他数据，这些数据的状态可以在会话内独立于其他分区进行回滚和前进，以协助您的应用程序进行特征建模。

会话可能只有一个分区，在这种情况下，会话中所有实体都在该分区中。有关分区的更多信息，请参见 13.4 分区 。

回滚操作会使用到以下对象。

分区标记

分区标记（partition mark，以下简称 pmark）是在分区中设置的回滚标记，用于记录分区在某一时刻的状态。分区可以向前或向后回滚到任何已设置的 pmark。设置 pmark 会创建一个增量（delta）。有关更多信息，请参见 13.5.1 分区回滚 。

增量

增量（delta）记录从一个 pmark 回滚到相邻 pmark 所需的更改。它们通过增量回调函数进行输出。有关更多信息，请参见 13.5.2 增量 。

会话标记

会话标记（session mark，以下简称 mark） 记录了在指定时刻整个建模会话的状态。通过在每个分区中创建 pmark 来设置 mark。

说明
分区也可以作为完整条目进行保存和检索。分区文件中包含分区的 pmark、可选增量、以及在该分区中建模的实体。

3.2.4 其他数据实体

还有其他可用的建模引擎软件实体，这些实体可以操纵模型并为模型附加额外的数据，或者定义额外的结构。

属性

属性（attribute）是可以附加到任何单个拓扑、单个几何实体或任何组（group）上的数据结构。它们包含特定类型的字段，这些字段在属性定义（attribute define）中指定。

属性的存在与否，以及存储在属性中的数据，会在其所属的实体参与建模操作时自动更新，或者可以通过用户定义的回调函数进行控制。有关更多信息，请参见 13.3 属性 。

组

组（group）允许用户将多个实体组合在一起，以便有效地管理和操作它们。组可以包含各种类型的实体，如面、边、顶点、曲面、曲线和点，这对创建复杂而灵活的模型结构很有帮助。

组在特征建模中非常有用：例如，您可以创建一个特征，如凸台，并创建一个包含构成该特征的所有实体的组。然后，您可以通过将单个属性附加到组来标识该特征，而无需将属性附加到组中每个单独的实体上。

如果组中有实体在建模操作中被修改，该组将会由内核自动进行更新。有关更多信息，请参见 13.6 组。

变换

变换（transform）表示几何操作，如平移、镜像等。它们可以直接用于某些建模和渲染操作中，如创建装配体。

应用程序条目

应用程序条目（appitem）在您的应用程序中充当某些数据片段的标识符。它包含标签，这些标签不是用来在建模引擎软件会话中标识实体的，而是用来在您的应用程序中标识某些内容。

3.2.5 流形主体中拓扑与几何的关系

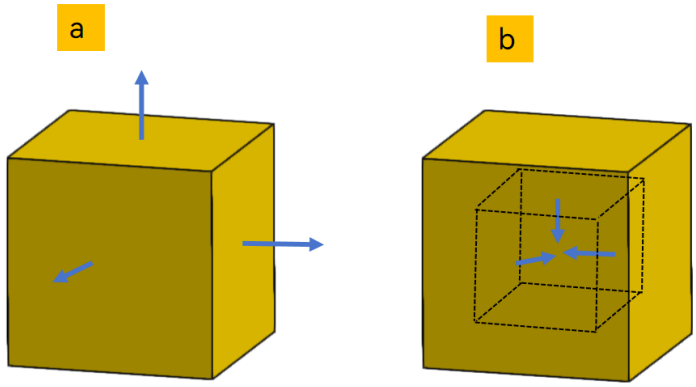
本节更详细地讨论了拓扑实体与几何实体之间的关系。

面法线

实心流形主体（manifold body）中各个面的法线必须指向实心体区域之外。

外部壳的面法线指向外部。内部壳（即实心体内的空洞）的面法线指向内部。

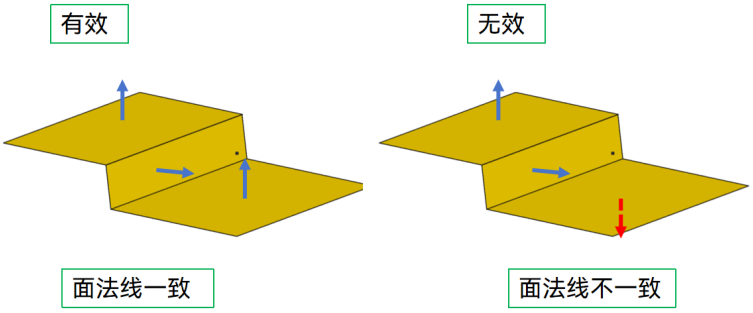
图 3-15 一个外部壳和一个内部壳的面法线



反向体（negative body）是“内外翻转”的实体，其面的法线指向实心体内部。唯一可以对反向体执行的操作是对其进行反向（negate）。

对于流形片状体（manifold sheet bodies），面的法线也必须指向远离片状体的方向。片状体中的面法线必须保持一致，如下所示。

图 3-16 片状体面的法线



自然曲面方向

曲面的自然方向取决于曲面的类型。每个曲面都有一个自然的方向。如下表所示：

表 3-4 不同类型曲面的自然方向

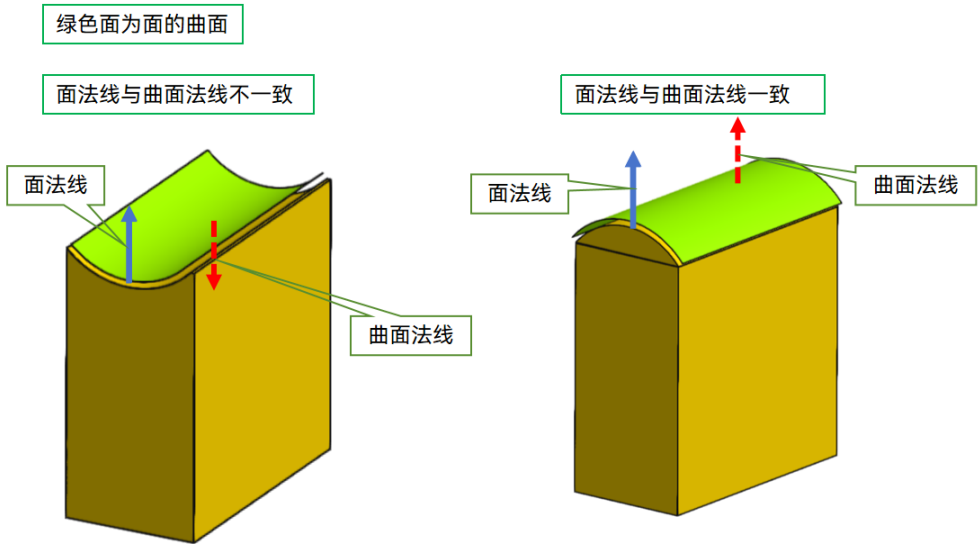
曲面	自然方向
平面（plane）	平面的自然方向沿着其法线方向。
圆柱面（cylinder）	圆柱面的自然方向为远离其轴线的方向。
圆锥面（cone）	圆锥面的自然方向为远离其轴线的方向。
球面（sphere）	球面的自然方向为远离其中心的方向。
圆环面（torus）	圆环面的自然方向为远离其主轴线所在圆的方向。

在参数形式中，曲面的自然方向为 dU 与 dV 的叉积方向。您可以使用 **bodyReverseOrientation** 来反转曲面的方向，这也会导致曲面法线方向的反转。

面方向标志

面与其曲面之间的关系由面方向标志（face orientation flag）确定，该标志指示面法线相对于曲面法线的方向。如果此标志为 true，则面法线与曲面法线平行，否则，面法线将与曲面法线反平行，如下图所示：

图 3-17 面方向标志



faceGetOrientedSurface 返回一个面的曲面以及面方向标志。当应用程序需要获取指定点处的面法线时，它需要同时调用 **faceGetOrientedSurface** 和 **surfEvalWithNormal**。

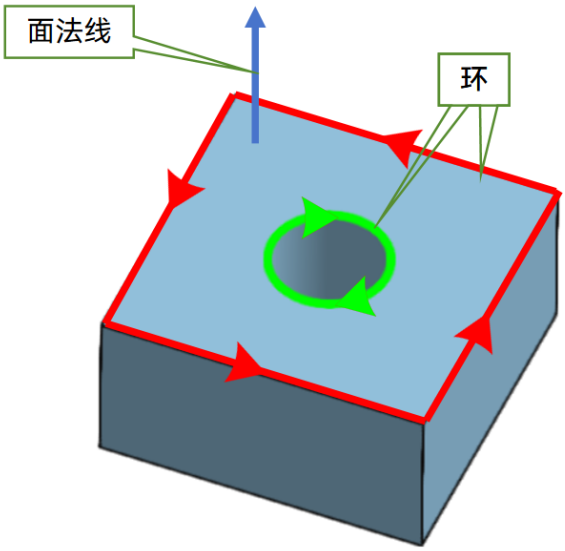
环、半边和边的方向

环（loops）、半边（coegde）和边（edges）都有方向，并且有规定来定义它们与面法线以及彼此之间的关系。这些规定如下：

- 在沿着面法线方向向下查看面时，面位于环的左侧，如图 3-18 一个带有圆柱形凹槽的立方体顶面的环方向所示。
- 环代表曲面的边界，环由一组闭合的半边组成，因此半边的方向与其所在的环的方向相同。

可以通过调用 **coedgeIsPositive** 来确定边相对于半边的方向。

图 3-18 一个带有圆柱形凹槽的立方体顶面的环方向



曲线的自然方向

曲线的方向由曲线切线决定，切线总是指向参数值增加的方向。您可以使用 **curveMakeCurveReversed** 来反转曲线的方向。

边和半边的方向标志

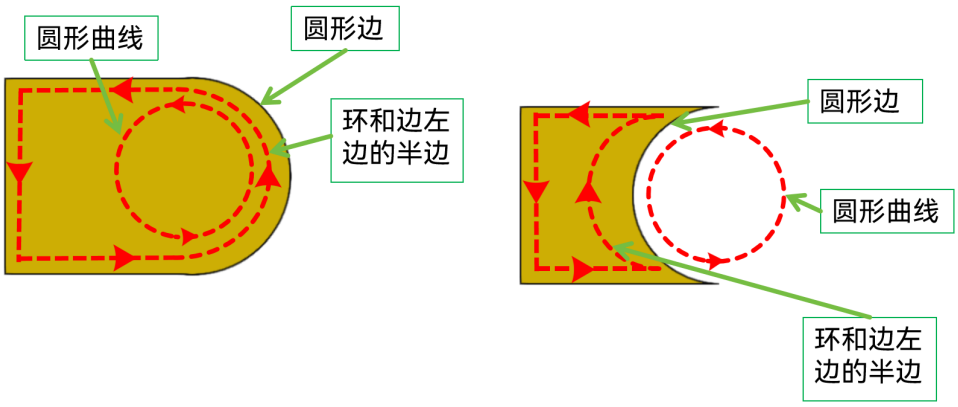
边（或半边）的方向标志是一个关键的参数，它决定了边（或半边）和附着在边（或半边）上的基础曲线切线之间的方向关系。如果这个标志是 true，那么边（或半边）的方向与基础曲线的切线方向平行；如果这个标志是 false，那么边（或半边）的方向与基础曲线的切线方向是反向平行的。

edgeGetOrientedCurve 和 **coedgeGetOrientedCurve** 都返回以下值:

- 边（或半边）上附着的曲线。
- 一个标志，用于指示边（或半边）相对于附着在该边（或半边）上的曲线的切线方向。

当您的应用程序需要获取边在指定点处的方向时，需要调用 **curveEvalWithTangent** 和 **edgeGetGeometry** 。获取半边在指定点处的方向时需要调用 **coedgeGetGeometry** 和 **curveEvalWithTangent**。

图 3-19 曲线、边、半边之间方向的关系



3.2.6 精确边和容差边

精确边

精确边是在建模引擎软件中创建的边，精确边上有一条精确的曲线。

容差边

容差边中没有曲线，其半边中有 SP 曲线。

3.3 主体类型

本建模引擎软件支持创建流形主体。

📖 说明
相关章节:3.2 模型结构。

3.3.1 流形主体

流形主体按拓扑复杂程度逐渐递增可以分为四种类型：橡子体（acorn body）、线框体（wire body）、片状体（sheet body）和实心体（solid body）。本节中给出的定义是每种类型完全指定的流形主体的定义。在几何上，也有可能创建不完整的流形主体，这样的流形主体被称为具有缺失几何的流形主体。

实心体的创建过程中通常会经历不同类型的流形主体阶段。从一个最小体（minimum body）或橡子体开始，在其上添加曲线并定义曲线的位置会将其转变为线框体。封闭线框体轮廓并为其添加一个曲面，可以将线框体转化为片状体。片状体可以通过扫掠或旋转来形成一个实心体。这些过程的细节将在本章后面部分进行说明。

以下各节将详细描述构成流形主体各个类型，并说明在使用不同类型的流形主体时可能存在的限制或约束条件。

橡子体

这是最简单的一种主体：一个零维主体，由空间中的两个或多个点组成。

橡子体只包含一种几何，即附着在橡子体顶点上的点。

线框体

线框体是从最小体升级而来，是一维拓扑主体。线框体中的每个部件都是一组相连的边，在线框体中：

- 一个开放部件（open component）有两个端点。
- 一个闭合部件（closed component）没有端点。

限制：

- 线框体由一个或多个块组成，每个块有一条或多条线框边（wire-frame edges）、零个或多个顶点。
- 线框体中的每个顶点必须被一到两条边使用，即没有橡子体顶点。

由于连接性的约束，主体中的每个线框体部件都要满足以下条件之一：

- 闭合线框：每个顶点恰好有两条边。
- 开放线框：除了两个端点处的顶点各自只连接一条边，每个顶点都恰好连接两条边。

片状体

片状体是二维拓扑主体。

限制：

- 片状体中的每个部件必须至少包含一个面。片状体中的每个部件可以是开放的也可以是闭合的。
- 片状体不得包含任何线框边（wire-frame edges）和橡子体顶点（acorn vertices）。
- 片状体中没有实心块。
- 仅包含开放部件的片状体有一个包含任意数量壳体的无界空心块，每个壳体都拥有一个连接的面集。
- 在片状体中，每个闭合的连接面集都有一个有界空心块，代表闭合连接面集的内部。

片状体中的每条边必须是以下之一：

- 普通边（normal edge）或流形边（manifold edge）：有两条半边，并且半边方向相反。
- 层流边（laminar edge）：只有一条半边，且该半边只能出现在开放的片状体中。

片状体中的每个顶点必须属于一个孤立环（isolated loop），或属于一个或多个边。在后一种情况下，顶点的所有边都必须是普通边，使用这些边的面必须形成一个单一的边连接的集合参见图 3-20 开放片状体（open sheet body）中的顶点（一个面集，所有边都是普通边），或者可以是一个或多个边连接的集合，每个集合必须包含恰好两个层流边和任意数量的普通边。

图 3-20 开放片状体（open sheet body）中的顶点（一个面集，所有边都是普通边）

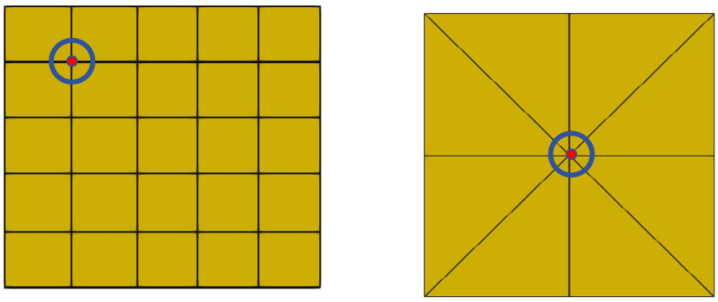


图 3-21 开放片状体中的顶点（一个面集，一条普通边和两个边界边）

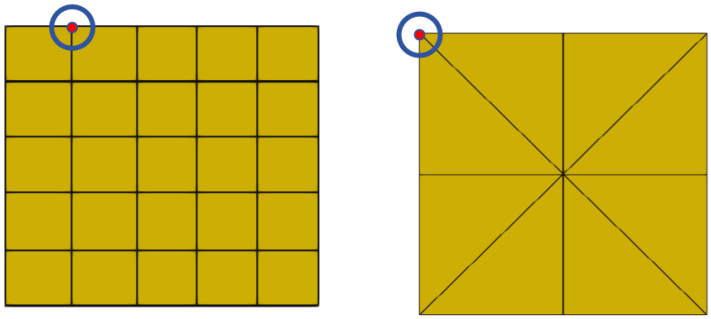
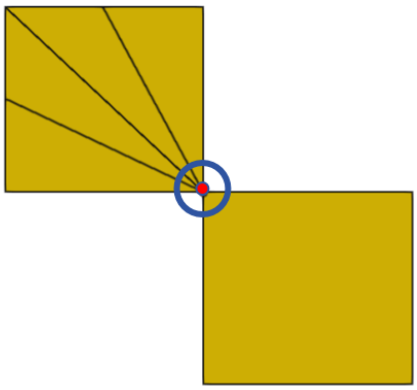


图 3-22 在开放的片状体中的非流形有效顶点（两个面集）



说明
尽管片状体中边和顶点的约束与实心体中的非常相似，但它们并不保证片状体是流形的，因为开放片状体中的顶点规则允许非流形主体。

实心体

实心体是三维拓扑主体，占据有限的体积。实心体中每个部件的体积是连续的。

限制：

- 每个实心体必须包含至少一个面，且不得包含任何线框边或橡子体顶点。
- 实心体中的每条边必须具有两个方向相反的半边。实心体中的每个顶点必须属于一个孤立环，或属于一个或多个边；在后一种情况下，使用这些边的面必须形成一个边的连接集合（仅考虑在顶点相交的边之间的连接）。

说明
这些约束条件可以确保所创建的实心体是流形的。

3.3.2 创建流形主体

有四种方式创建流形主体：

- 通过原始数据（raw data）创建基本体（primitive body）：使用描述基本体几何属性的原始数据（如顶点坐标、面信息等）来创建基本体。
- 通过几何创建流形主体：这种方法涉及使用现有的几何元素（如点、线、面等）来构建主体。
- 通过拓扑创建流形主体：拓扑描述了主体内部的结构和连接关系，而不关注其具体的几何位置。从拓扑创建主体也就是定义主体的边、面和顶点之间的连接关系。
- 通过现有体创建流形主体：使用已有的流形主体作为基础，通过修改、变换或组合来创建新的流形主体。

在创建主体时，所有必需的实体（如面、边等）也会随之自动生成。以 **createMinimumBodyByPoint** 接口为例，它会构建一个最小体，这个最小体只有一个顶点，并将指定的点与该顶点关联起来。

从原始数据创建基本体

创建基本体的接口会自动生成这些基本体所需的所有拓扑和几何。用于创建基本体的接口包括：

- 创建片状体的接口：
  - **createSheetCircle**
  - **createSheetPolygon**
  - **createSheetRectangle**
- 创建实心体的接口：
  - **createSolidBlock**
  - **createSolidCone**
  - **createSolidCylinder**
  - **createSolidPrism**
  - **createSolidSphere**
  - **createSolidTorus**

新物体的位置可以根据需要以局部坐标系（local coordinate system）或世界坐标系（world coordinate system）为基准。

世界坐标系是用于定义尺寸盒的基准坐标系，其原点位于尺寸盒的中心，坐标轴与尺寸盒的各个边平行。

局部坐标系用于在一个便利坐标系（convenient coordinate system）中定义一个实体，然后分别展示该实体在世界坐标系中的位置和方向。

getCircle 返回圆在其局部坐标系中的定义，并返回一个 m_lcs，m_lcs 给出了局部坐标系在世界坐标系中的位置和方向。新实体的局部坐标系是通过 PSGMApiLCS 传递给创建函数来定义的。该参数是可选的，如果提供为 NULL，则默认为世界坐标系。

PSGMApiLCS 有三个字段：

- m_location：局部坐标系原点在世界坐标中的位置。
- m_axis：局部坐标系的 Z 轴在世界坐标中的方向。
- m_refDirection：局部坐标系的 X 轴在世界坐标中的方向。

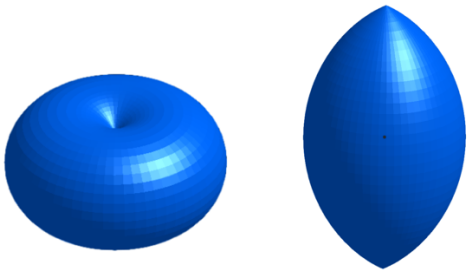
局部坐标系的 Y 轴是通过 X 轴来确定的。

请注意，m_axis 和 m_refDirection 必须是正交的单位向量。

所有旋转几何（spun geometry）如：圆、圆锥、圆柱、球、圆环，它们的周期性参数化接缝在局部坐标系的 X 轴上。

您可以使用 **createSolidTorus** 创建苹果形环面和柠檬形环面，如下图所示。

图 3-23 苹果形环面和柠檬形环面



从几何中创建主体

可以从现有几何图形中创建主体的接口有：

- **createMinimumBodyByPoint**
- **createWireBodyByCurves**
- **createSheetBodyBySurface**

- **createSolidConeBySurface**
- **createSolidCylinderBySurface**
- **createSolidSphereBySurface**
- **createSolidTorusBySurface**

如果现有的几何体是孤立几何体，它将被合并到新的主体中。

如果现有的几何体不是孤立几何体（即它附着在另一个主体上），则会制作一个该几何体的副本并合并到新的实体中。

除了上述的接口之外，还可以用 **createSheetPlanar** 来创建一个平面的片状体，其边界为一个不规则的多边形，片状体的边界由其顶点指定。

孤立几何

以下接口可用于创建孤立几何体：

- **createPoint**
- **createBCurve**
- **createCircle**
- **createEllipse**
- **createLine**
- **createBCurve**
- **createCircle**
- **createBSurface**
- **createOffsetSurface**
- **createPlane**
- **createCone**
- **createCylinder**
- **createSphere**
- **createTorus**

从拓扑中创建主体

createSheetBodiesByFaces 可以通过现有流形或非流形主体中的面创建片状体。

从现有主体中创造主体

复制流形主体：可以使用 **copyEntity** 复制一个流形主体来创建一个新的主体。

复制体主体的拓扑结构：可以使用 **bodyCopyTopology** 来复制一个主体的拓扑结构。

3.3.3 将流形主体转变为更复杂的主体

以下操作通过对拓扑结构较为简单的实体进行操作来生成特定类型的主体。

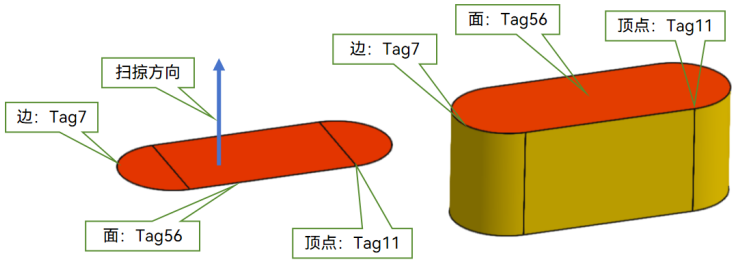
扫掠和旋转

扫掠（sweep）和旋转（spin）可以改变主体的类型：

- 通过扫掠最小体，将最小体转变为线框体。
- 通过扫掠线框体，将线框体转变为片状体。
- 通过扫掠片状体，将片状体转变为实心体。

当一个片状体被扫掠或部分旋转时，整个片状体都会被变换，因此，原始的面、边、顶点和片状体的几何体最终位于变换后的位置。如下图所示。

图 3-24 扫掠或部分旋转片状体后实体的位置



说明
在旋转物体时，需要注意轴的定位。如果某个内部顶点恰好位于旋转轴上，则不能使其完成一整圈（即 2π）的旋转，否则会产生非流形曲面，导致片状体畸形。通过拉伸或旋转很容易创建自交的物体，可以通过对生成的主体使用 **checkBody** 来检查自交。

压印

imprintCurveOnBody 、**imprintCurvesOnFace** 具有两个主要用途：

- 在适当的实体上创建新的面，然后拉伸（或旋转）该面以延伸实体。
- 通过在面、块或线框体上压印曲线来创建轮廓。

当轮廓闭合时，轮廓的类型取决于一般拓扑会话参数：

- 如果该参数处于开启状态，轮廓将保持为线框体。
- 如果该参数处于关闭状态，轮廓将变为片状体。

压印与几何共享

当压印一个面的曲线产生多条边时，曲线被这些边共享。同样，将一个面分割为两个或更多部分会导致原始面的曲面被所有新的面共享。

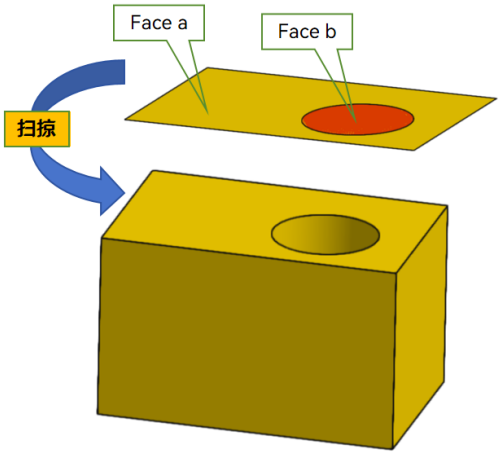
添加橡胶面

createFacesByWireEdges 可以用于将橡胶面（rubber face,即没有几何附着的面）附加到线框体中的线框边的环，从而创建一个片状体。在 8.2 线框体建模的中有更详细的解释。

穿孔

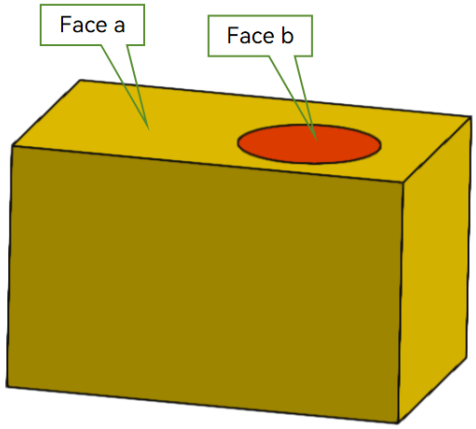
faceDeleteFromSheet 可以移除压印在片状体上的面，从而使片状体具有一个穿孔。具有穿孔的片状体扫掠后的结果为带有穿孔的实心体，如下图所示，在扫掠之前“Face a”穿透了面“Face b”，所以扫掠结果是一个带有穿孔的实心体。

图 3-25 扫掠前穿孔一个面



下图中的主体没有穿孔，扫掠结果为一个没有穿孔的实心体：

图 3-26 扫掠一个没有穿孔的片状体



上图中，顶部和底部的两个的圆是压印在面的边，但没有移除边内的面，面没有被穿孔。

3.3.4 更改主体类型

在主体拓扑结构允许的前提下，您可以使用 **bodySetType** 来更改主体的类型。该接口接收一个主体和一个主体类型，然后尝试将此主体转换为该类型。

仅支持以下转换：

表 3-5 支持的主体类型转换

原始主体类型	目标主体类型
General	Solid/Sheet/Wire/Acorn/Minimum
Solid/Sheet/Wire/Acorn/Minimum	General
Solid	Sheet
Sheet	Solid

根据接收到的主体的拓扑结构，可能不总是可以根据要求更改主体的类型。例如，当从片状体转换为实心体时，以下情况可能会导致 **bodySetType** 执行失败：

- 片状体没有闭合。
- 片状体可能是由一个或多个闭合部件组成的不连贯主体，不同部件的面方向不一致。

📖 说明
bodySetType 在某些情况下（如，将实心体转换为片状体时）会修改实体的块类型，但不会更改接收到的实体的拓扑结构。例如，把一般体转换为线框体时，无法删除面以创建线框体。

3.4 会话精度和局部精度

本章将向您介绍建模引擎软件在建模操作中使用的精度，包含以下内容：

- 建模引擎软件使用的默认会话精度以及相关的限制。
- 为非本建模引擎软件创建的模型设置边和顶点的局部精度。

📖 说明
相关章节：3.2.6 精确边和容差边。

3.4.1 会话精度

本建模引擎软件执行所有计算时都遵循固定的精度标准，这些标准被称为会话精度和会话角度精度：

- 会话线性精度：建模引擎软件的线性精度。小于此值的长度被视为零，相差不超过此值的两个长度被视为相等。会话精度由 **getLengthPrecision** 返回。
- 会话角度精度：建模引擎软件的角度精度。小于此值的角度被视为零，相差不超过此值的两个角度被视为相等。会话角度精度由 **getAnglePrecision** 返回。

本建模引擎软件所创建的模型非常精确。通常，只有当两点之间的距离小于 $1.0e^{-8}$ 个单位时，它们才会被视为重合。同样地，只有当两个方向之间的角度小于 $1.0e^{-11}$ 弧度时，它们才会被认为平行。因此，传递给本建模引擎软件的数据至少应达到此精度要求。

为确保精度得到正确处理，一个实体的所有部分必须位于一个尺寸为 1000*1000*1000 的包围盒内。

请避免使用几何尺寸接近线性精度和角度精度的模型。需要确保非重合顶点间距大于线性精度 100 倍，非平行边夹角大于角度精度 100 倍，以减少精度问题，提高模型准确性及稳定性。

3.4.2 局部精度

本建模引擎软件允许用户导入其他建模软件创建的模型，但这些模型可能由于较低的创建精度而导致拓扑数据和几何数据之间的不一致。用户可以通过调整导入模型中边和顶点的局部精度来处理这类不一致问题，这种设置局部精度的方式被称为容差建模。

在容差建模中，通过将边视为管、顶点视为球，并根据需要增大它们的尺寸，使原本因精度问题而不相交的几何能够相交。为了有效应用这一功能，需要仔细检查每个模型的拓扑结构，并在数据出现不一致时相应地调整局部精度。通过使用 **repairEdge** 或 5.6 检查实体中提到的相关接口，用户可以在本建模引擎软件中成功建模，同时确保数据的准确性和一致性。

设置边上的精度

设置边精度的结果取决于该边是否之前已经更改过精度。以下是可能的场景：

- 将边上的普通曲线替换为具有局部精度的 SP 曲线。
- 在已经具有局部精度的边上设置新的局部精度。
- 移除边的局部精度。

可以使用 **repairEdge** 来设置边的局部精度。**repairEdge** 会分析边的几何数据，并根据需要调整其精度，以确保数据的正确相交和模型的一致性。

您也可以使用 **edgeSetPrecision** 来设置边的局部精度。

普通曲线替换为具有局部精度的 SP 曲线

当在几何建模软件中处理具有曲面参数空间 G1 不连续性的边时，可能无法直接从原始曲线和相关面的曲面计算出整条 SP 曲线。在这种情况下，需要将原始曲线在参数奇点或不连续处拆分，并在边上引入额外的顶点。对于拆分后新边上的每个相关半边，会分别计算它们的 SP 曲线，并为它们设置适当的局部精度。这样做可以确保在建模过程中正确处理曲面的复杂几何特性，保持模型的一致性和准确性。

repairEdge 在设置局部精度时，如果发现 G1 不连续点，就会该点处自动分割边。

在已经具有局部精度的边上设置新的局部精度

可以使用 **optimiseEdge** 来重置边的局部精度。该接口在尝试降低当前精度的同时，确保精度在降低后仍能满足附着在边上的 SP 曲线的需求。

为已有局部精度的边设置小于原精度的精度。

为已有局部精度的边设置大于原精度的精度。

移除边的局部精度

您可以使用 **edgeResetPrecision** 来移除边的局部精度，移除局部精度的边将再次由无界的三维 SP 曲线几何表示。为了重新创建这条曲线，建模引擎软件会移除附着在边上的 SP 曲线，并通过找到相关面的曲面的交线来计算新曲线的位置。这一过程如下图所示。

设置点的局部精度

当局部精度应用于顶点时，该精度值表示顶点周围不确定性球体的半径。精度值越低，意味着精度越高，不确定性球体的半径越小。

顶点精度通常是设置边精度的副产品，通过调用 **edgeSetPrecision** 或其他建模操作来设置。

在调用 **optimiseVertex** 时，顶点精度可能会被修改。您的应用程序通常不需要为单个顶点设置精度。如果确实需要，可以使用 **vertexSetPrecision** 来设置顶点精度，但前提是：

- 新的精度不小于现有精度；
- 新的精度不小于汇聚于此的边的最大精度；
- 与该顶点连接的任何边都不完全位于不确定性球体内。

可以使用 **vertexGetPrecision** 查询顶点的当前精度。

3.5 B 曲线和 B 曲面

本建模引擎软件集成了与 B 曲线（B-splines）和 B 曲面（B-surfaces）相关的功能。这意味着您可以将曲线和曲面附加到边和面上，然后通过本建模引擎软件接口对它们应用任何操作，如从布尔运算和附加属性。

B 几何是通过参数化定义的分段多项式或有理多项式几何。在本建模引擎软件中表示为 B 样条（NURBS）曲线和曲面。

3.5.1 创建 B 曲线和 B 曲面

本建模引擎软件中 B 曲面和 B 曲线的创建方式分为显示创建和自动创建两大类。

显式创建是指用户直接向本建模引擎软件的接口提供创建 B 曲线或 B 曲面所需的数据，自动创建是指在建模操作过程中自动完成的创建，例如，当用户在本建模引擎软件中对一条 B 曲线进行扫掠操作时，系统会自动生成一个 B 曲面作为扫掠结果。

B 曲线和 B 曲面有以下创建方法：

- 通过使用 B 样条控制点来创建。
- 通过使用分段数据来创建。
- 通过扫掠或旋转 B 曲线来创建。
- 通过将线、圆或椭圆转换为 B 曲线来创建。
- 通过放样一组 B 曲线来创建 B 曲面来创建。

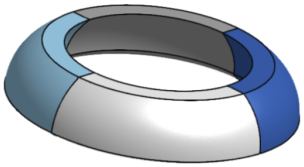
通过 B 样条控制点创建 B 曲线和 B 曲面

可以使用 **createBCurve** 和 **createBSurface** 分别根据 B 样条控制点和节点向量创建曲线或曲面。通过这种方式，既可以创建有理的 B 样条曲线和 B 曲面，也可以创建非有理的 B 样条曲线和 B 曲面。下图展示了带有控制多边形的 B 样条曲线。

图 3-27 带有控制多边形的 B 样条曲线

可以强制曲线或曲面为周期性曲线或曲面。周期曲线必须是闭合的并平滑地自交。曲面可以在 U 或 V 方向上是周期性的，从而得到管状曲面，或者在这两个方向上都是周期性的，从而得到环形曲面。下图展示了一个周期曲面。

图 3-28 一个周期曲面



通过分段数据创建 B 曲线

可以使用 **createBCurveByPiecewise** 根据分段数据创建 B 曲线。分段数据可以是：贝塞尔曲线（Bezier）、埃尔米特曲线（Hermite）、多项式曲线（polynomial）和泰勒级数曲线（Taylor series）。这里所说的“分段”是指曲线或曲面的每个段与其他段是独立的，因为曲线和曲面在相邻段之间是位置（G0）连续的。曲线或曲面可以是有理的也可以是非有理的。

通过曲线创建 B 曲线

bcurveJoin 通过组合一系列首尾相接的曲线来创建 B 曲线。通常，生成的曲线会根据参数化进行调整，以便在可能的情况下，将原本 G1 连续的连接点变为 C1 连续。您可以通过提供连接点处的节点值来覆盖这一属性，这些值可以通过选项结构进行指定，从而使建模引擎软件用这些值来建立连续性条件而不使用计算出来的值。

通过已有的曲线和曲面创建 B 几何

以下接口可以通过已有的 B 几何创建新的 B 几何：

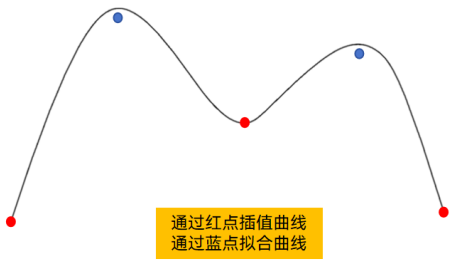
- **createBCurveByCurve**：根据已有曲线的某个参数区间生成一个精确的 B 曲线表示，通过设定该 B 曲线的连续性使其更加平滑。
- **createBSurfBySurface**：根据已有曲面的某个参数区间生成一个精确的 B 曲面表示。

通过插值创建 B 曲线

您可以使用 **createBCurveBySpline** 接口在参数空间中插值一组位置来创建 B 曲线。建模引擎软件拟合一条分段曲线，并通过一组合适的节点和控制点来支持生成的曲线，该曲线在指定位置处的斜率和曲率都是连续的。

通过插值和拟合一组位置创建的 B 曲线。

图 3-29 通过插值和拟合一组位置创建 B 曲线



重建 B 曲线和 B 曲面

建模引擎软件提供了 **createBCurveByFitCurve** 和 **createBSurfByFitSurface** 允许您通过从现有曲线或曲面上采样的一组点构建 B 曲线和 B 曲面：这个过程通常被称为曲线拟合或曲面拟合。这两个接口主要用于以下场景：

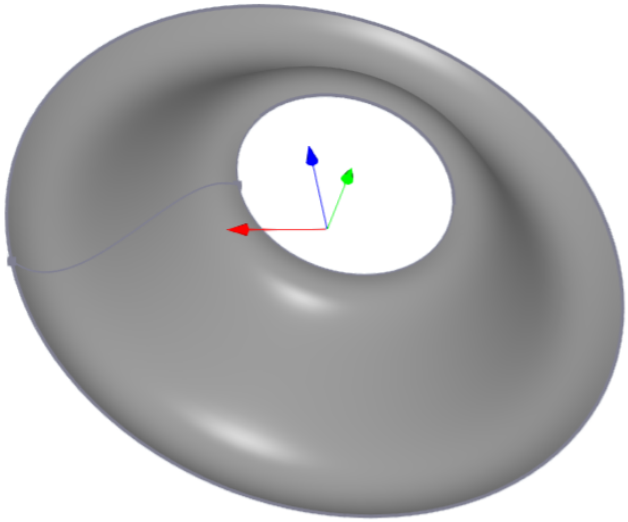
- 改善现有几何实体的质量。例如，将曲线或曲面数据导入建模引擎软件之后，可以使用这两个接口重新构建非 C2 连续的数据。
- 根据曲线或曲面的定义创建几何实体，而不是基于实际的几何数据。例如，您可以基于正弦曲线的定义创建一个 B 曲线。

建模引擎软件通过采样点将 B 曲线或 B 曲面拟合到所提供的曲线或曲面数据上来保证返回的 B 几何体具有 C2 连续性。

扫掠和旋转 B 曲线

可以使用 **extrudeBCurve** 和 **spinBCurve** 将 B 曲线拉伸或旋转为 B 曲面。当旋转 B 曲线时，自旋角不得大于 2π 或小于-2π。

图 3-30 通过旋转 B 曲线创建 B 曲面



3.5.2 修改 B 曲线和 B 曲面

本建模引擎软件提供了多种方式来修改模型中的 B 曲线和 B 曲面，下面将为您详细介绍这些功能。

添加或删除节点

您可以通过向曲线和曲面添加节点（knot）来为曲线和曲面添加参数或参数线。

可以使用 **bcurveAddKnot** 向 B 曲线添加节点，或者使用 **bsurfAddUKnot** 和 **bsurfAddVKnot** 向 B 曲面添加 U 向或 V 向的节点。这会增加曲线上的线段数量或曲面上的行或列的数量。

也可以通过 **bcurveRemoveKnots** 或 **bsurfRemoveKnots** 移除曲线或曲面中的节点来删除曲线或曲面上的参数或参数线。您可以通过 **bsurfRemoveKnots** 中的 dirType 指定应该在 B 曲面中删除哪个参数方向上的结点。

紧致 B 曲线和 B 曲面

本建模引擎软件提供了 **bcurveClampKnots** 和 **bsurfClampKnots** 这两个接口，它们的主要作用是确保 B 曲线或 B 曲面满足贝塞尔曲线或贝塞尔曲面特征。这两个接口在执行成功后并不会返回任何特定的信息或结果。如果输入的 B 曲线或 B 曲面已经满足了贝塞尔特征，那么这两个接口将会保持其原样，不做任何修改。

升高或降低度数

您可以使用 **bcurveRaiseDegree** 来提高 B 曲线的度数，使用 **bsurfRaiseDegree** 来提高 B 曲面的度数。

相应地，您也可以使用 **bcurveLowerDegree** 和 **bsurfLowerDegree** 来撤销 **bcurveRaiseDegree** 和 **bsurfRaiseDegree** 效果。例如，在提高度数后导致曲线或曲面出现轻微扰动时就可以使用 **bcurveLowerDegree** 和 **bsurfLowerDegree** 来降低度数。

延伸 B 曲线

您可以使用 **extendBCurve** 来延伸 B 曲线，此接口接收一个 B 曲线以及一个选项结构，该结构包含了关于如何延伸 B 曲线的详细信息，有以下四个字段：

- **m_version**：选项结构的版本，用来指定使用那一版本的选项结构。
- **m_low**：控制 B 曲线在较低参数端如何进行延伸。
- **m_high**：控制 B 曲线在较高参数端如何进行延伸。
- **m_closedType**：是否延伸闭合或周期曲线。

3.5.3 用 B 曲线和 B 曲面建模

本节介绍了一些基于 B 曲线和 B 曲面构建模型的方法。

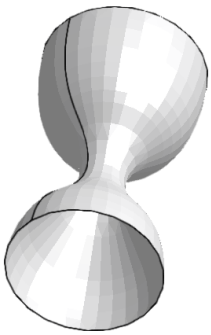
扫掠和旋转

可以通过在面上绘制 B 曲线来创建轮廓，然后再对这个轮廓（必须是闭合的）进行扫掠或旋转，从而生成一个实心体。

所有线框体和片状体都可以用 **extrudeBodyLocally** 和 **spinBody** 进行扫掠和旋转。

可以通过旋转复合 B 曲线制作片状体。

图 3-31 通过旋转复合 B 曲线制作的片状体

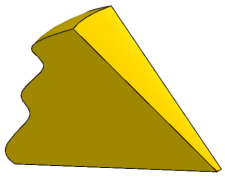


可以通过扫掠一个由 B 曲线和直线组成的平面轮廓来创建实心体。

图 3-32 通过扫掠轮廓创建的实心体

可以通过旋转一个 B 曲面创建实心体。

图 3-33 通过旋转 B 曲面创建的实心体



使用限制

B 曲面是有界的，B 曲面的边界定义了其形状和范围的极限，超出这个边界的部分并不存在。当您尝试在一个曲面片上的面进行某些变换或编辑时，如果这些操作涉及到将曲面扩展到曲面的边界之外，那么这些操作可能无法执行，因为 B 曲面的定义和属性不允许其超出其原始的边界范围。

3.6 变换

本建模引擎软件支持齐次坐标变换，并提供了相应的接口对几何体进行各种空间变换，如：平移、缩放（等比例）、旋转和镜像。

变换具有类别和有效性，可以将变换应用于单个几何实体也可以应用于整个主体。

创建变换

许多基本的变换都可以通过以下接口直接创建：

- **createEqualScaleTransform**
- **createMirrorTransform**
- **createRotationTransform**
- **createTranslationTransform**
- **createViewTransform**

除了上述接口外，您还可以通过 **createTransform** 直接创建任何变换。例如，可以使用 **createTransform** 通过以下矩阵一个在 X 轴上移动 1、在 Y 轴上移动 2、在 Z 轴上移动 3 的平移变换。$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

本建模引擎软件支持大型变换（large transformation），即实体位于尺寸框内，但可以通过大型变换定位在尺寸框外。大型变换不能直接应用于几何实体或拓扑实体，但可以将它们传递给将变换作为输入参数的接口，如：

- **pickBodies**
- **rangeEntities**
- **clashOfTopologies**
- **massPropsOfTopologies**
- **boxOfTopology**

为了在处理大型变换时保持准确性，建议使用特定的变换接口来创建所需的变换，例如：使用 **createEqualScaleTransform** 创建等比例缩放变换、使用 **createMirrorTransform** 创建镜像变换、使用 **createRotationTransform** 创建旋转变换，并使用 **transformTransform** 来组合它们。

变换的应用范围

变换可以应用于几何实体、主体、装配体、实例、向量和面。

变换几何实体

transformGeometry 允许用户对一组几何实体进行变换。此接口既可以直接修改原始的几何实体，也可以创建并修改它们的副本。在使用时，您需要向此接口提供三个主要的输入参数：一个几何实体数组、一个变换矩阵以及一组选项。

选项中包含：

- m_tol：如果几何实体无法被精确变换，则用此容差表示新几何体与原始几何体之间的匹配程度。
- m_modify：是创建新的几何实体，还是修改原始几何实体。
- m_wantOutGeom：是否在 outGeomTags 数组中返回变换后的几何实体。
- m_wantExact：是否返回每个几何实体被精确变换的信息，还是在指定的容差范围内被变换的信息。

变换主体

transformBody 用于对指定的主体进行变换，并在必要时对主体进行近似处理以匹配变换。此接口接收一个片状体或实心体、一个变换矩阵以及一个容差值作为输入，在调用结束后输出变换结果。

主体被变换后，为了获得期望的结果，可能需要对面上的曲面和边上的曲线进行近似处理，这会导致变换后的曲面之间需要重新计算交线。如果曲面和曲线能够被精确变换，则不用重新计算曲面之间的交线。

变换其他实体

除了主体和几何实体外，您还可以使用以下接口对其他实体应用变换：

- **transformAssembly**
- **transformFace**
- **transformInstance**
- **transformTransform**
- **transformVector**
- **transformDirection**

变换检查和变化分类

可以使用 **checkTransform** 对指定的变换进行简单的有效性检查。

classifyTransform 是一个更全面的接口，它不仅能执行 **checkTransform** 所执行的简单检查，还可以告诉您变换的基本类型（平移、旋转、缩放或组合变换）以及检测变换过程中出现的偏差。例如，一个纯旋转变换矩阵，由于某些原因（如数值误差或计算过程中的近似），它并不是完全正交的，**classifyTransform** 会检测出这种偏差，并告诉您偏离正交状态的程度。

3.7 装配体和实例

本建模引擎软件中的装配和实例主要是为了使应用程序能够通过本建模引擎软件保存模型文件，而不是为了提供复杂的装配体建模功能。

关于保存文件的更多内容，请参见 13.7 存储。

装配体

装配体是模型的实例集合：

- 装配体可以包含构造几何（曲面、曲线和点）。
- 装配体可以附加属性。
- 由装配体拥有的组只能包含该装配体的实例和构造几何。
- 装配体的引用必须是无环的，即一个装配体不能直接或间接地实例化自己。
- 装配体不能实例化另一个分区中的模型。

实例

实例是指向装配体中包含的模型的指针：

- 一个实例会引用一个装配体、一个模型和一个变换。
- 实例具有实体 ID，装配体直接引用的实例都有一个不同的实体 ID。

- 一个装配体可以包含同一模型的任意数量的实例。
- 实例不能存在于装配体之外，并且它们所实例化的模型不能为空。

说明
一个模型根据变换的不同可以生成多个实例，多个实例可以放到一个装配体中。

创建装配体或实例

以下接口用于创建装配体或实例：

- **assemblyCreateEmpty**：创建一个空装配体。
- **createInstance**：按标准形式创建一个实例，即该实例包含装配体、变换和模型。
- **assemblyCreateLevelAssembly**：用于创建一个新的一级装配体，它是指定装配体的一个展开版本。

修改装配体或实例

以下接口用于修改装配体或实例：

- **instanceChangeModel**：更改实例以实例化不同的模型。
- **instanceReplaceTransform**：替换一个实例的变换。
- **transformAssembly**：通过变换一个装配体的所有实例来变换它。
- **transformInstance**：通过变换实例的变换矩阵来变换实例。

查询装配体或实例

以下接口用于查询装配体或实例的相关信息：

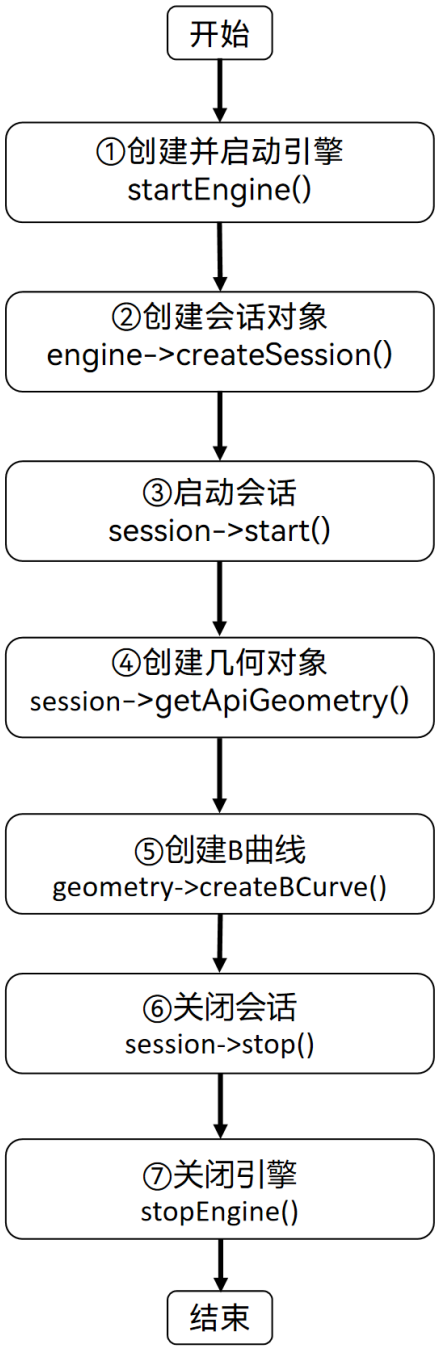
- **assemblyGetModels**：返回由该装配体直接实例化的模型。
- **assemblyGetInstances**：返回被该装配体直接引用的实例。
- **assemblyGetModelsAndTransforms**：返回直接被装配体引用的实例的模型和变换。
- **getInstance**：以实例的标准形式返回实例的信息这包括实例所属的装配体、所引用的模型以及变换。

4 快速入门

本章以“创建 B 曲线”为例，帮助您快速掌握接口调用流程，包括创建几何建模引擎、启动建模引擎、创建 会话、开启会话、创建 B 曲线、停止会话、关闭引擎释放引擎所占用的资源。建模引擎软件支持创建单引擎、多会话，本示例中仅实现创建单个引擎、单个会话。

“创建 B 曲线”的功能实现与接口调用流程如下图所示：

图 4-1 接口调用流程图



说明

本例中涉及的引擎、会话类对象、几何类对象在创建时进行初始化，后续可直接使用。

①调用 PSGMApi.h 头文件中的 startEngine 接口创建并启动一个建模引擎 engine，创建时需要传入创建选项，也可以使用默认选项。本例中使用的是默认选项。

```
PSGMApiEngineOption engineOption;
PSGMIEngine* engine = startEngine(engineOption);
```

②调用 engine 对象的 createSession 接口创建一个会话类对象，被创建的会话对象有一个默认分区，后续创建的实体都会保存在该默认分区中。

```
PSGMIApiSession* session = engine->createSession(nullptr);
```

③调用 session 对象的 start 接口启动会话。启动会话时需要给 start 传入启动会话选项。本例中使用的是默认选项。

```
PSGMApiSessionStartOption sessionStartOption;
auto errCode = session->start(sessionStartOption);
```

④调用 session 对象的 getApiGeometry 接口创建几何对象。

```
PSGMIApiGeometry* geometry = session->getApiGeometry();
```

⑤调用几何类对象的 createBCurve 创建 B 曲线。接口文档中可以找到 createBCurve 接口输入数据的详细说明。

```
PSGMApiTag curveTag;
int degree = 1;
int knotCount = 9;
int vertexCount = 7;
int vertexDim = 3;
double knot[9] = {0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1};
int knot_mult[9] = {1, 1, 1, 1, 1, 1, 1, 1, 1};
double vertex[21] = {-4, -4, 0, -5.31, -0.483, 0.685, 0, 4.788, -0.332, 1.567,
-4.577,
0.846, -1.985, 2.635, -2.301, 5.851, 0.583, 1.296, 6.7, -1.126, 1.685};
PSGMApiBCurve BCurve;
BCurve.m_degree = degree;
BCurve.m_knotCount = knotCount;
BCurve.m_vertexCount = vertexCount;
BCurve.m_vertexDim = vertexDim;
BCurve.m_knots = knot;
BCurve.m_knotMults = knot_mult;
BCurve.m_vertices = vertex;
errCode = geometry->createBCurve(BCurve, &curveTag);
```

⑥调用 session 对象的 stop 接口停止会话。

```
session->stop();
```

⑦调用 engine 对象的 stopEngine 接口释放 engine 所占用的资源。

```
stopEngine();
```


完整示例代码：

```
#include <string.h>
#include <cstring>
#include <iostream>
#include <string>


#include "PSGMApi.h"


int main()
{
    PSGMApiEngineOption engineOption;
    PSGMIEngine* engine = startEngine(engineOption);
    PSGMIApiSession* session = engine->createSession(nullptr);

    std::cout << "Create  bcurve" << std::endl;


    PSGMApiSessionStartOption sessionStartOption;
    auto errCode = session->start(sessionStartOption);
    PSGMIApiGeometry* geometry = session->getApiGeometry();


    PSGMApiTag curveTag = 0;


    int degree = 1;
    int knotCount = 9;
    int vertexCount = 7;
    int vertexDim = 3;
    double knot[9] = {0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1};
    int knot_mult[9] = {1, 1, 1, 1, 1, 1, 1, 1, 1};
    double vertex[21] = {-4,    -4,     0,     -5.31,  -0.483, 0.685, 0,     4.788, -0.332,
1.567, -4.577,
                         0.846, -1.985, 2.635, -2.301, 5.851,  0.583, 1.296, 6.7,   -1.126, 1.685};
    PSGMApiBCurve BCurve;
    BCurve.m_degree = degree;
    BCurve.m_knotCount = knotCount;
    BCurve.m_vertexCount = vertexCount;
    BCurve.m_vertexDim = vertexDim;
    BCurve.m_knots = knot;
    BCurve.m_knotMults = knot_mult;
    BCurve.m_vertices = vertex;


    errCode = geometry->createBCurve(BCurve, &curveTag);


    if (PSGMApiErrorCode::NoError != errCode)
    {
        std::cout << "Create bcurve failed, errCode is " << static_cast<int>(errCode) <<
std::endl;
    }
    else
    {
        std::cout << "closed bcurve's tag" << curveTag << std::endl;
    }


    session->stop();
```

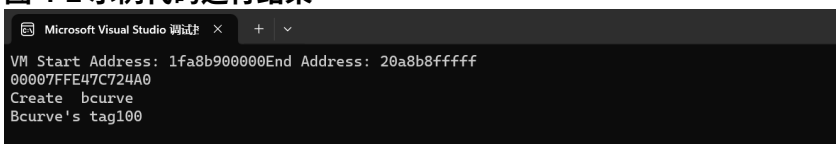


```
    stopEngine();


    return 0;
}
```

示例代码运行结果如下：

图 4-2 示例代码运行结果



5 模型分析

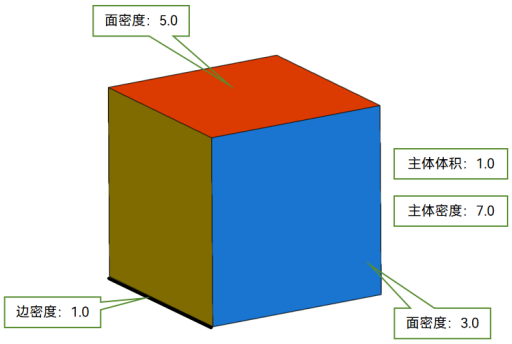
5.1 内容简介

在整个模型的创建过程中，模型以准确的 3D 形式表达，因此本软件提供了一套全面的接口来查询模型的详细信息。本章主要向您介绍如何使用这些接口来查询模型中的拓扑实体与几何实体的详细信息，并计算模型的各种质量属性，如面积、体积、长度、重心以及转动惯量（moment of inertia）。此外，本章还会告诉您如何检测有碰撞的实体，以及如何测试一个主体的有效性，有助于您及时发现并修复潜在的问题。

本章包含以下内容：

- 5.2 查询功能：介绍本建模引擎软件中的数据查询和输出接口。
- 5.3 查询质量属性：介绍如何计算一个实体的质量属性。
- 5.4 计算最大最小距离：介绍如何计算实体之间的距离。
- 5.5 检查碰撞：介绍如何检测实体之间的碰撞，以及检测碰撞的类型。
- 5.6 检查实体：介绍检查功能，以及如何使用检查功能来检查实体。

图 5-1 一个立方体的质量属性



5.2 查询功能

本章主要向您介绍每个查询和输出接口的作用，以便您根据具体情况调用最合适的接口。

5.2.1 查询拓扑实体

本节主要向您介绍查询拓扑实体类型的接口，拓扑实体类型与相关联的几何实体无关。

接口名称	描述
bodyGetType	查询拓扑实体的类型。
edgeGetType	
coedgeGetType	
shellGetType	
vertexGetType	
loopGetType	
lumpGetType	
coedgeIsPositive	查询指定的半边与其包含的边是否同向。
loopIsIsolated	查询指定的环是否为孤立环。

5.2.2 查询几何实体

本节主要向您介绍查询几何实体信息的接口。

接口名称	描述
getBCurve	这些接口通过给定的几何实体的标签来查询它们的标准形式。 应用程序在添加这些几何实体时，需要按照几何实体的标准形式进行添加。添加时，应用程序不用初始化其中的任意字段。
bcurveGetKnots	
bsurfGetKnots	
getBSurface	
getCircle	
getCone	
getCylinder	
getEllipse	
getLine	
getOffsetSurface	
getPlane	
getPoint	
getSPCurve	
getSphere	
getSpinSurface	
getExtrudeSurface	
getTorus	
getTransform	
topoCategoriseGeometry	查询附着到给定拓扑实体上的几何实体的种类。
edgeGetGeometry	查询给定边包含的几何实体信息。
coedgeGetGeometry	查询给定半边包含的几何实体信息。
geomGetDependents	查询给定几何实体依赖的其他几何实体的集合。
geomGetGeomOwners	查询给定几何实体的几何所有者集合。
geomGetCategory	查询一个几何实体的类别。

5.2.3 判断实体类型

以下接口用于判断实体类型：

接口	描述
isEntity	判定输入的内容是否为实体。
isCurve	判定输入的实体是否为曲线。

接口	描述
isGeometry	判定输入的实体是否为几何。
isModel	判定输入的实体是否为模型。
isSurface	判定输入的实体是否为曲面。
isTopology	判定输入的实体是否为拓扑实体。

5.2.4 查询从属关系

如果您需要使用单个接口获取到主体的所有拓扑实体，请使用 **bodyGetTopology** 接口。如果要检索特定的拓扑实体，您需要使用下表中介绍的相应接口。

接口名称	描述
bodyGetEdges	查询指定主体包含的拓扑实体。
bodyGetFaces	
bodyGetCoedges	
bodyGetFirstEdge	
bodyGetFirstFace	
bodyGetLoops	
bodyGetLumps	
bodyGetShells	
bodyGetVertices	
bodyGetLaminarEdges	查询指定主体中的层流边。层流边只有单个面，所以层流边只有一条半边。
curveGetEdges	查询指定曲线所附着的实体。
curveGetCoedge	
curveGetModel	
curveGetCommonSurfaces	查询将指定曲线作为公共曲线的一对曲面。

接口名称	描述
edgeGetBody	查询与指定边相连的实体。
edgeGetCurve	
edgeGetFaces	
edgeGetCoedges	
edgeGetFirstCoedge	
edgeGetNextInBody	
edgeGetOrientedCurve	
edgeGetShells	
edgeGetVertices	
getType	查询实体所属的类。
getIdentifier	查询实体的标识符。
getPartition	查询指定实体所在的分区。
faceGetBody	查询包含指定面的实体。
faceGetEdges	查询指定面的所有边。
faceGetFacesAdjacent	查询与指定面相邻的所有面。
faceGetFirstLoop	查询与指定面相连的实体。
faceGetLoops	
faceGetNextInBody	
faceGetOrientedSurface	
faceGetShell	
faceGetSurface	
faceGetVertices	
faceGetEdgesCommon	查询两个面共有的边。
faceComputeInteriorVector	返回指定面内部的任意向量，以及该向量的曲面 U 参数和 V 参数。返回的向量位于指定面的内部，不会返回位于其边界边或顶点上的向量。
faceComputeOuterLoop	返回指定面的外环。其上下文是曲面参数空间中的环，而不是模型空间中的外环，即使 **faceComputeOuterLoop** 返回类型为 Outer 的环，该接口也可能查询失败。

接口名称	描述
coedgeGetBody	查询与指定半边关联的实体。
coedgeGetCurve	
coedgeGetEdge	
coedgeGetFace	
coedgeGetNextInLoop	
coedgeGetNextOfEdge	
coedgeGetOrientedCurve	
coedgeGetPreviousInLoop	
coedgeGetPreviousOfEdge	
loopGetBody	查询与指定环相关的实体。
loopGetEdges	
loopGetCoedges	
loopGetFirstCoedge	
loopGetNextInFace	
loopGetVertices	
modelGetAllAttrDefinitions	查询与指定模型相关的实体。
modelGetConstructionPoints	
modelGetConstructionSurfaces	
modelGetGroups	
partitionGetAssemblies	查询与指定分区相关的实体。
partitionGetBodies	
partitionGetGeometries	
partitionGetTransforms	
pmarkGetEntities	查询在分区滚动到指定 pmark 时将创建、删除或修改的实体。
pointGetModel	查询与指定点相关的实体。
pointGetVertex	
lumpGetBody	查询与指定块相关的实体。
lumpGetShells	
lumpGetWireEdges	
getModels	查询会话中加载的实体。

接口名称	描述
getPartitions	查询会话中的分区。
getAttrDefinitions	查询会话中用户定义的属性定义的详细信息。
bodyGetIsolatedVertices	查询与指定壳相关的实体。
shellGetBody	
shellGetOrientedFaces	
shellGetLump	
surfGetFaces	查询与指定曲面相关的实体。
surfGetModel	
surfGetCommonCurves	
vertexGetBody	查询与指定顶点相关的实体。
vertexGetFaces	
vertexGetIsolatedLoops	
vertexGetOrientedEdges	
vertexGetPoint	
vertexGetShells	

5.2.5 查询参数几何

5.2.5.1 评估参数几何

以下接口用于评估参数几何：

接口	描述
curveEvaluate	在指定的曲线上，根据指定的参数评估一个点，并计算其导数。
curveEvalCurvature	计算曲线的法线、主方向和曲率。
curveEvalWithTangent	在指定的曲线上，根据指定的参数评估一个点、该点的导数以及通过该点的切线。
surfEval	在指定的参数对处评估曲面上的一个点及其导数。
surfEvalCurvature	计算曲面的法线、主方向以及曲率。
surfEvalWithNormal	在指定的曲面上，根据参数对来评估一个点、该点的导数以及该点的法线。

当使用参数几何评估接口时，给定的参数可以位于指定的范围之外。对于这种情况，通常有两种处理方式：

- 当给定的参数超出范围，并且接口指示指定的曲线不是周期曲线时，返回的点和导数通常是在曲线的延伸部分上评估的。
- 对于 B 曲面和偏移 B 曲面，当给定的参数超出范围时，可能会评估计算的结果也可能不会返回。

5.2.5.2 参数几何的手性评估

以下接口是评估参数几何中所描述接口的手性评估版本。这些接口使应用程序可以控制评估的方向。

接口	对应的非手性接口
curveEvalHanded	**curveEvaluate**
curveEvalWithTanHanded	**curveEvalWithTangent**

这些接口与其非手性（non-handed）的等价接口在行为上是一致的，除了在间断点或周期性接缝（seam）处，它们会根据定向参数 PSGMApiHandType 所指示的方向来接近评估点。定向参数可以取以下值：

- Left：左手方向求值是从参数值以下逼近的。
- Right：右手方向求值是从参数值以上开始逼近的。

5.2.5.3 查询几何属性

以下接口可用于查询几何属性：

接口	描述
curveComputeVectorInterval	查询由位于曲线上的两个位置向量所界定的曲线的参数区间。
edgeIsPlanar	边是否在平面上。
coincidentGeometries	查询两个几何是否重合。
bcurveComputeG1Discontinuity	查找 B 曲线或 B 曲面上的 G1 不连续性。
bsurfComputeG1Discontinuity	
coedgeComputeCurveParameter	计算与指定曲面或指定曲线参数相对应的半边曲线参数或曲面参数。
coedgeComputeSurfParameter	
faceIsUVBox	检测特定参数空间中的面是否是为矩形。更多信息请参见参数空间包围盒。
faceIsPeriodic	检测特定参数空间中的面是否是周期性的。更多信息请参见确定面的周期性。

5.2.6 一般查询

5.2.6.1 查询边界

本建模引擎软件提供了一系列查找对象边界的接口，这些接口会用到以下内容：

- 轴对齐包围盒与非轴对齐包围盒
- 参数空间包围盒
- 区间
- 极值

本节简要介绍可用的接口。有关详细信息，请参见*《Geoshape 几何建模引擎软件 接口开发文档》*中每个接口的文档。

5.2.6.1.1 轴对齐与非轴对齐包围盒

包围盒用于描述一个围绕实体的虚拟盒子，一个包围盒有六个坐标。

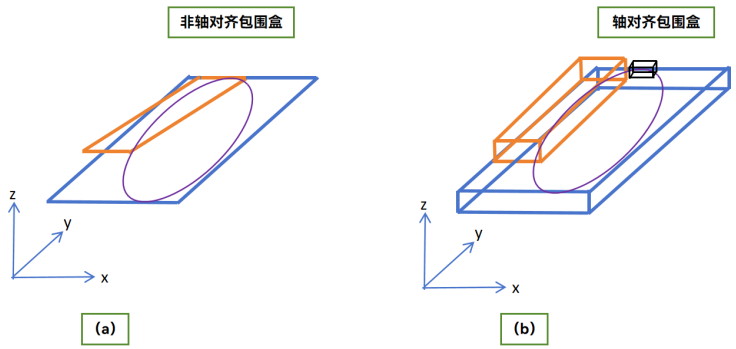
轴对齐包围盒的边总是与本建模引擎软件的坐标系对齐。非轴对齐包围盒则不需要以这种方式对齐其边，它们的轴也是相互正交的。

所有盒子都有维度和大小：

- 三维包围盒，大小为体积。
- 二维包围盒，大小为面积。
- 一维包围盒，大小为长度。

下图展示了围绕 B 曲线（红色）的轴对齐和非轴对齐包围盒。图（a）展示了三个非轴对齐包围盒，图（b）展示了三个轴对齐包围盒；整个 B 曲线的包围盒用蓝色线表示；三分之一曲线的包围盒用橙色线表示；九分之一曲线的包围盒用黑色线表示。无论是轴对齐还是非轴对齐的包围盒都会随着包围对象的大小而变化。

图 5-2 轴对齐和非轴对齐包围盒



以下接口用于查找包围盒：

接口	描述
boxOfCurve	返回指定曲线的轴对齐包围盒。

接口	描述
boxOfSurface	返回指定曲面的轴对齐包围盒。
uvBoxOfCoedge	返回的包围盒将给定的半边限定在半边所属的面的参数空间中。
boxOfTopology	该接口返回一个轴对齐包围盒，该包围盒可以包围接收到的拓扑本身或拓扑的变换结果。变换可以通过 transfTags 参数提供。如果该参数被设置为 null，则拓扑不会被变换。 可以通过 m_wantTopoBoxes 选项选择是否返回每个拓扑的包围盒。 有关此接口的更多信息，请参见 *《Geoshape 几何建模引擎软件 接口开发文档》* 中 **boxOfTopology** 的详细描述。

5.2.6.1.2 参数空间包围盒

以下接口用于查找参数空间包围盒：

接口	描述
faceIsUVBox	检查面是否在参数空间中为矩形，如果是，则返回一个紧密的参数空间包围盒。
uvBoxOfFace	查找面的封闭参数空间包围盒。通常，这是一个对精确参数空间包围盒的合理紧密近似。

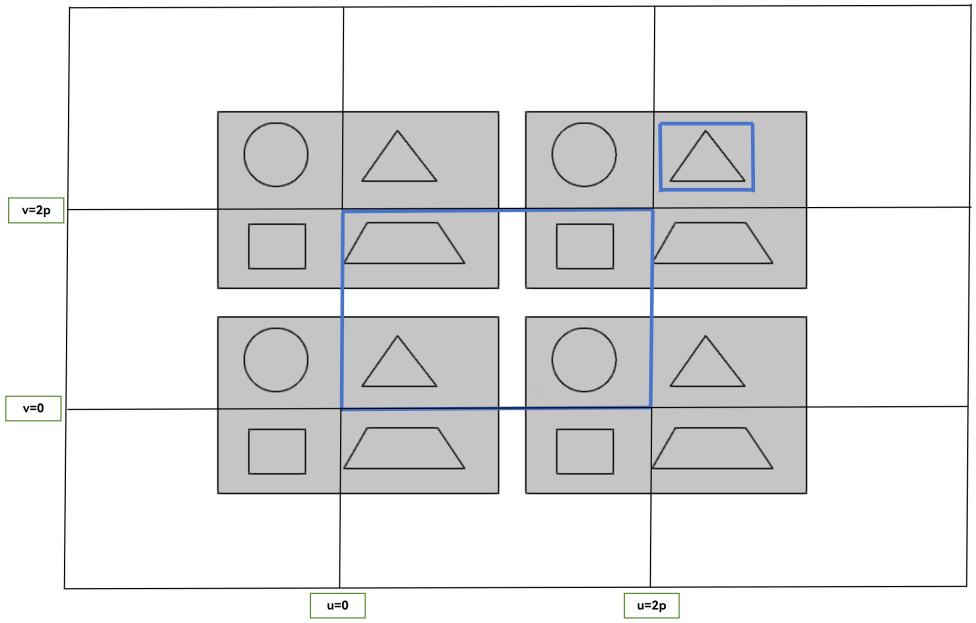
如果需要在同一个面上多次调用 **uvBoxOfFace** 并获得一致的结果，建议在每次调用 **uvBoxOfFace** 之前先调用 **faceIsUVBox**。

faceIsUVBox 用于确定指定的面在参数空间中是否为矩形。如果面是矩形，那么返回的 UV 参数将位于由 **surfGetUVBox** 给出的曲面的参数范围内，除非对应的参数是周期性的。当参数是非周期性的时，返回的 u 和 v 参数直接对应曲面的参数范围。当参数是周期性的时，面的第一个参数位于该范围内，而两个参数之间的差异不超过周期，即面跨越了周期性参数的边界。

uvBoxOfFace 用于在给指定面的曲面的参数空间中定位一个包围该面的参数空间包围盒。**surfGetUVBox** 会返回 UV 参数的取值范围。

下图中的蓝色线框表示两个的参数空间包围盒。较大的包围盒包含由 **surfParameteriseVector** 返回。较小的包围盒由 **uvBoxOfCoedge** 返回。

图 5-3 参数空间包围盒



5.2.6.1.3 区间

以下接口用于查询区间：

接口	描述
edgeGetInterval	计算附着在边上的曲线的参数区间。
coedgeGetInterval	计算附着在半边上的曲线的参数区间。

5.2.6.2 查询空间关系

以下接口用于查询实体与向量的空间关系：

接口	描述
positionVectorInBody	查询由给定的向量或参数表示的点是在指定主体的内部、外部还是边界上。
positionVectorsInFace	查询由给定的向量或参数表示的点是在指定面的内部、外部还是边界上。

这些接口还会返回与点重合或包含点的实体的最低维度的唯一拓扑实体。

5.2.6.3 向量比较

以下接口用于比较向量：

接口	描述
vectorIsEqual	检查两个向量在当前会话精度内是否相等。

接口	描述
vectorIsZero	检查向量在当前会话精度内是否为零。
vectorIsParallel	检查两个向量在当前会话角度精度范围内是否平行。

有时，一个有效的导数向量（derivative vector），即结合了方向和大小的导数向量，例如一个奇异点（singularity）附近的一阶导数可能会被 **vectorIsZero** 误判为零向量。在这种情况下，您应该使用 **vectorNormalize** 从导数向量中创建一个单位向量。

5.2.6.4 变换比较

transformIsEqual 用于检测两个变换是否相等。

📖 说明
只有当变换不包含非均匀缩放或透视项时，才能比较它们。

5.2.6.5 查询面的周期性

可以使用 **faceIsPeriodic** 来确定一个面在特定的参数方向上是否具有周期性。此接口接收一个面作为输入，并返回两个值，分别描述该面在 U 参数方向和 V 参数方向上的周期性。

5.2.6.6 查找不连续

如果一个曲线或曲面存在不连续，可以使用 **curveComputeDiscontinuity** 和 **surfComputeDiscontinuity** 来查找这些不连续的详细信息。这些接口可以识别 1 到 3 级的解析不连续（analytic discontinuities）或几何不连续（geometric discontinuities）。可用选项：

- m_level：要查找的最大不连续程度。默认情况下，返回所有解析不连续(层次 1 到 3)。
- m_interval：要检查的曲线区间或曲面区间。默认情况下，检查整个曲线或曲面。

5.2.6.7 查找自交几何

如果一个曲线或曲面发生自相交，可以用 **curveComputeSelfIntersections** 和 **surfComputeSelfIntersections** 来查找每个自相交点的详细信息。这些接口分别接收一个曲线或曲面作为输入，并返回一个结构体，其中包含找到的自交点数量以及每个自交点的详细信息。

可以将这些信息直接传递给 **curveFixSelfIntersections** 或 **surfFixSelfIntersections** 进行修复。

本建模引擎软件能够识别三种不同类型的自相交：

- 一般自交（general）：这种自交指的是两个或多个参数区间在空间中的相同位置发生重叠。
- 奇点自交（singularity）：这种自交是由某种奇异性引起的。
- 混合相交（mixed）：这种自交包含了一般自交和奇点自交。

5.2.6.8 查找退化曲面和退化曲线

如果一个曲面或曲线包含退化（degeneracy），可以用 **surfComputeDegenerates** 和 **curveComputeDegenerates** 来查找每个退化的详细信息。它们分别接收一个曲面或曲线作为输入，并返回一个结构体，其中包含找到的退化的数量以及每个退化的详细信息。

🕮 说明
此功能不支持离散几何。

每个曲面退化会返回以下信息：

- 退化的类型。
- 退化出现的位置：如果退化元素占据空间中的一个单独点，则返回该点在三维空间中的位置坐标。
- 退化处的曲面法线：如果可以，会计算并返回退化处曲面的法线方向。
- 退化的参数空间包围盒：该包围盒用于描述退化在参数空间中的范围，该包围盒可能在 U 方向或 V 方向具有零宽度，或者在这个两个方向上同时具有零宽度。

每个曲线退化会返回以下信息：

- 退化的类型。
- 退化出现的位置：如果退化元素占据空间中的一个单独点，则返回该点在三维空间中的位置坐标。
- 退化的参数空间包围盒：该包围盒用于描述退化在参数空间中的范围，该包围盒可能具有零宽度。

这些信息随后可以直接传递给 **surfFixDegenerates** 和 **curveFixDegenerates** 用于修复退化。

本建模引擎软件能够识别五种类型的曲面退化：

- 参数退化（Parametric）：参数空间中的退化。
- 物理凹面退化（PhysicalConcave）：物理空间中的凹面退化。
- 物理凸面退化（PhysicalConvex）：物理空间中的凸面退化。
- 物理混合退化（PhysicalMixed）：物理空间中的混合退化，它同时包含凹面退化和凸面退化。
- 未定义退化（PhysicalUndefine）：物理空间中的退化，其凸凹性无法确定。

本建模引擎软件能够识别三种类型的曲线退化：

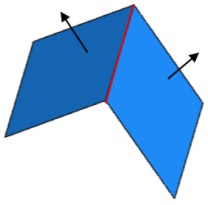
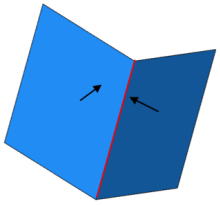
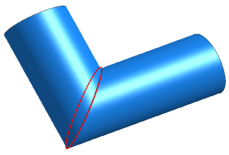
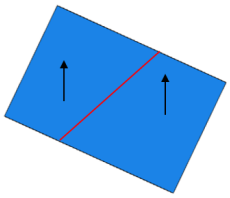
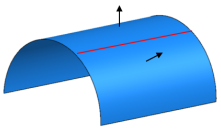
- 参数退化（Parametric）：参数空间中的退化。

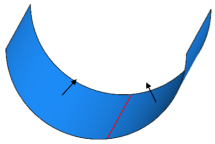
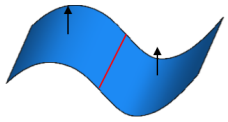
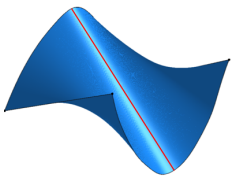
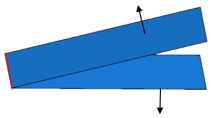
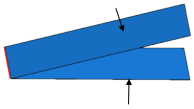
- 物理退化（Physical）：物理空间中的退化。
- 曲面退化（OnSurface）：仅针对 SP 曲线（即依赖于基础曲面的曲线）的退化。

5.2.6.9 查询边的凹凸性

可以使用 **edgeGetConvexity** 返回边的凹凸性。该接口接收一个边和一个选项结构，并返回一个凹凸性值即 PSGMApiEdgeConvexityType 的值。

可以识别以下凹凸性(在每个示例中，返回凹凸性的边用红色显示，面法线用蓝色显示)：

凹凸性	**PSGMApiEdgeConvexityType 的值**	**示例**
凸	Convex	
凹	Concave	
可变凹凸性 凹凸性会沿着边变化。	Variable	
光顺平整 两个面的法线平行，并且两个曲面的曲率均为零	SmoothFlat	
光顺突起 两个曲面的法线是平行的，可能两个曲率都为正，也可能一个曲率为正，一个曲率为零。	SmoothConvex	

凹凸性	PSGMApiEdgeConvexityType 的值	示例
光顺凹陷 两个曲面法线是平行的，可能两个曲率都是为负，也可能一个曲率为负，一个曲率为零。	SmoothConcave	
光顺弯曲 两个曲面法线平行，也可能一个曲率为正，一个曲率为负。	SmoothInflection	
可变光顺凹凸性 曲面法线平行，曲率的正负沿边发生变化。	SmoothVariable	
刃凸 曲面法线反向平行的，曲率之和是非负的。	KnifeConvex	
刃凹 曲面法线反向平行的，曲率之和是非正的。	KnifeConcave	

在检测凹凸性时，可以通过选项结构中的 m_hasAngularTol 和 m_angularTol 选项来设置角度容差。

5.3 查询质量属性

本章介绍了如何使用 **massPropsOfTopologies** 来评估质量以及其他属性（如面积、体积、长度、重心和转动惯量）。**massPropsOfTopologies** 可以计算单个实体或一组实体的质量属性，在计算一组实体的质量属性时，需要确保这组实体属于同一类型。此外，它还会使用附加到实体或实体集合的密度属性。更多关于属性的信息，请参见属性。

5.3.1 接口介绍

massPropsOfTopologies 接收以下参数：

输入参数	描述
topoTagCount	拓扑实体的标签的数量。
topoTags	一组拓扑实体的标签。

输入参数	描述
accuracy	accuracy 决定了计算所请求的质量属性时所需的工作量大小。它的取值范围在 0.0 到 1.0 之间。在实际应用中，取值范围在 0.99 到 0.999999 之间。这些值并不是百分比，而是计算中迭代次数的指示器。
option	这是一组用于评估指定拓扑实体质量属性的选项。

massPropsOfTopologies 输出以下参数：

输出参数	描述
amount	所选拓扑结构中的物质数量。这取决于实体的类型。
mass	密度和数量值的乘积的积分。
gravityCenter	在考虑的整个数量范围内，密度接口在给定位置处与位置向量本身的乘积的积分。
inertiaMoment	围绕一个点的惯性张量，其中该点是重心。
periphery	所选拓扑结构的边界数量，取决于实体的类型。

下表为 **massPropsOfTopologies** 的选项摘要：

选项	描述
m_mass	需要计算的质量属性。
m_bound	要返回的精度界限（bound on accuracy）的类型。
m_single	是将输入的实体作为单个实体使用还是作为多个实体使用。 请参见处理局部密度属性以获取更多信息。

5.3.2 计算质量

可以使用质量选项来查找实体的质量属性。此选项可以采用以下值：

- No：未找到任何数据。
- Mass：找到实体的质量和数量。
- CentreOfGravity：找到实体的质量、数量和重心。
- MomentOfInertia：找到实体的质量、数量、重心和转动惯量。这是默认值。

实体的质量用于找到重心，而重心又是计算转动惯量所必需的。

说明
可以在没有密度属性的拓扑实体上计算质量属性。在计算时这些实体将被视为密度属性为 1 的实体。

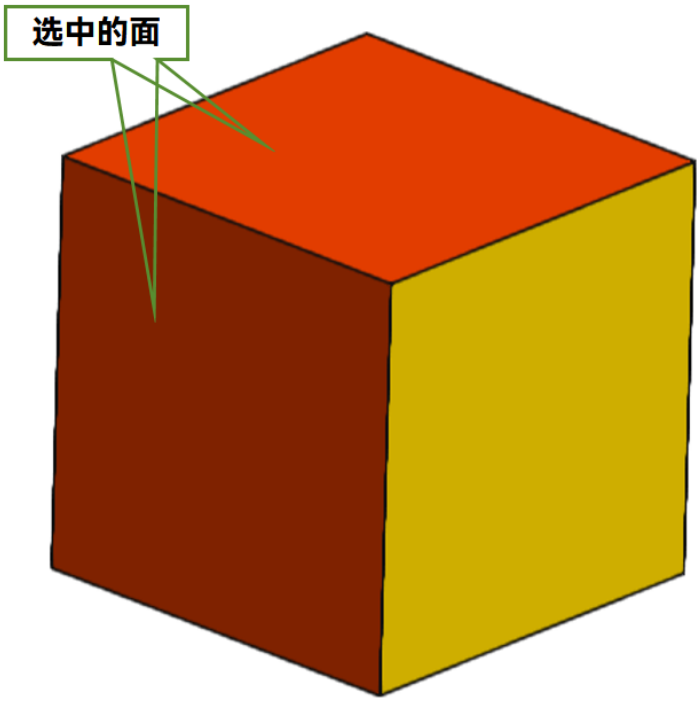
5.3.3 计算外围

m_periphery 用于控制是否计算指定拓扑的外围的总量，即 amount，可以采用以下值：

- No：不计算实体的外围。
- Yes：计算实体的外围。这是默认值。

对于一般体，只能计算具有面或实心体区域（即维度为二和三）的实体的外围。块、面和边的外围为单个拓扑的外围之和。下图为单位立方体的两个相邻面的外围。这两个面的外围被计算为这两个面中边的长度的 8 倍。

图 5-4 计算选中的两个面的外围



5.3.4 误差限制

您可以使用 m_bound 选项来决定结果的误差范围。此选项可以采用以下值：

值	描述
None	未给定误差。这是默认值。

值	描述
Modulus	误差范围是在给定数值的基础上，加上或减去一个模数来表示。
Interval	误差范围由给定的一个数值和一个区间表示。

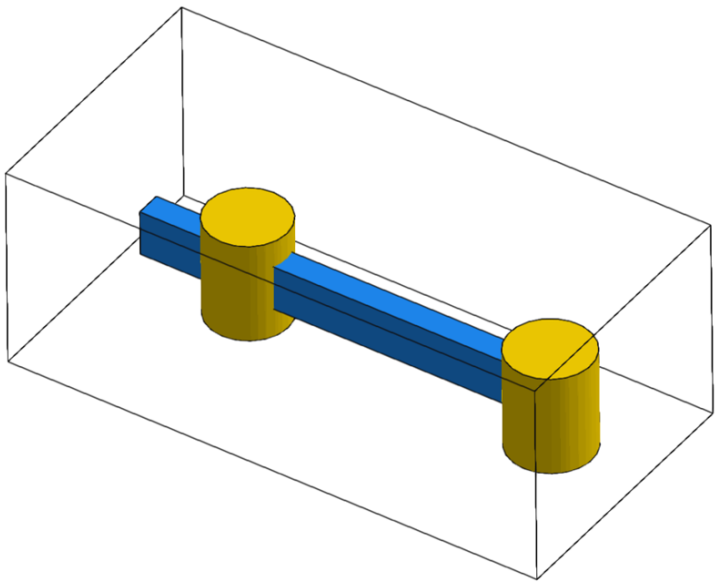
强烈建议在计算质量特性时指定误差范围。

5.3.5 计算方式

m_single 选项能够将一组面或片状体体视为单个实体，然后可以使用单个实体来计算所需的质量特性，但不包括其质量。忽略附加到单个面或边的密度属性，并且假定实体密度为 1 来计算转动惯量。此选项还允许您计算一组未缝合（unsewn）的片状体的质量特性。缝合后的片状体将形成封闭体积，类似于单个实体。

下图显示了一个带有复杂空洞的立方体。计算圆柱体空洞（橘红色）和立方体空洞（蓝色）的负体积（negative volume）时，需要提供立方体的四个面和圆柱体的六个面，以及 m_single 选项。在使用这种方式计算里立方体内部的圆柱体空洞和立方体空洞的体积时，将会把它们视为单个实心体。

图 5-5 带有复杂空洞的立方体



5.3.6 处理局部密度属性

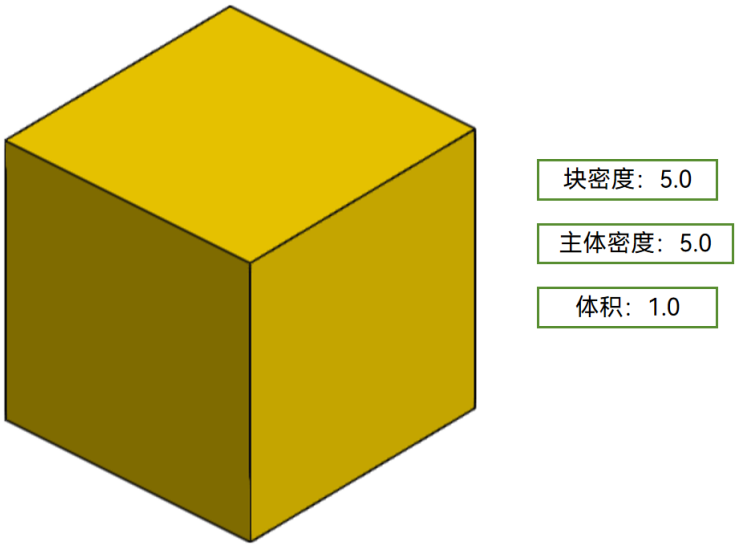
在建模引擎软件中，附加到块、面、边或顶点的密度属性称为局部密度。有关如何使用属性的更多信息，请参见 13.3 属性。局部密度可以用于特定的局部拓扑，以选择性地改变该区域全局体密度的解释。局部密度可以让模型中的面具有不同的密度，或者

没有密度。通过局部密度可以为模型添加主体密度，为面添加面密度。局部密度对质量特性计算的影响由选项 m_sameDimDensity 和 m_lowerDimDensity 控制，这两个选项都可以取以下值:

值	描述
Override	如果局部密度存在则使用局部密度，否则将使用主体密度。仅由 m_sameDimDensity 使用。
Additive	将局部密度属性添加到主体密度中，得到拓扑实体的总密度。
Ignore	忽略局部密度。
Unset	未设置局部密度。

下图及其附表显示了 m_sameDimDensity 选项在取不同选项值时，一个边长为单位长度的立方体的预期质量。该立方体的主体密度为 7.0，立方体内部实心体区域附加的密度为 2.0。该立方体的体积为 1.0。

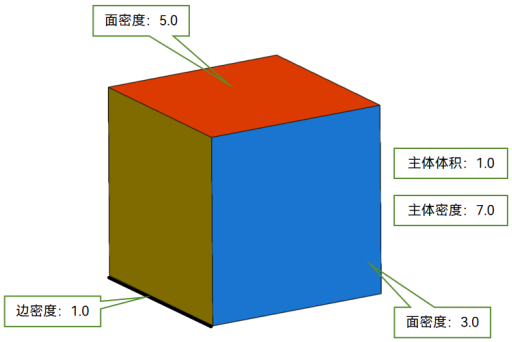
图 5-6 使用 m_sameDimDensity 选项生成的 Mass 值



值	mass
Override/Unset	2.0 x 1.0 =2.0
Ignore	7.0 x 1.0=7.0
Additive	(7.0+2.0) x1.0 =9.0

下图及其附表显示了 m_lowerDimDensity 选项在取不同选项值时，一个边长为单位长度的立方体的预期质量。该立方体的主体密度为 7.0、边密度为 1.0、两个面密度分别为 3.0 和 5.0。该立方体的体积为 1.0。

图 5-7 使用 m_lowerDimDensity 选项生成的 Mass 值



值	mass
Additive/Unset	(5.0 + 3.0 + 7.0 + 1.0) x 1.0 =16.0
Ignore	7.0 x 1.0 = 7.0

5.3.7 计算带有变换的拓扑的质量属性

可以使用 m_transfTagCount 和 m_transfTags 选项来计算具有刚性变换和镜像变换的拓扑实体的质量属性。

5.3.8 计算一般体和装配体的质量属性

massPropsOfTopologies 还可以计算一般体和装配体的质量属性；

类型	Amount	质量	重心	转动惯量	外围
装配体	总量（total amount）	质量	部件的重心	部件的转动惯量	总外围（total periphery）
三维一般体	体积	质量	重心	转动惯量	面区域（face area）
二维一般体	曲面面积	质量	重心	转动惯量	边长度（edge length）
一维一般体	长度	质量	重心	转动惯量	none
零维一般体	none	质量	重心	转动惯量	none

massPropsOfTopologies 会对装配体和子装配体中的所有实体出进行计算。这些实体必须是相同的实体类型。有关装配体的信息，请参见 3.7 装配体和实例。

5.3.9 识别零质量

您可以使用 m_massEqualZero 选项来控制当至少一个提供的拓扑具有零质量和数量时的行为。该选项采用以下值：

- Fail：当找到质量和数量为零的拓扑时，停止计算并返回错误。这是默认值。
- Report：计算所有输入拓扑的质量属性，并返回所有质量大于零的拓扑的总质量。

5.3.10 质量属性定义

建模引擎软件使用特定的定义来计算质量属性。质量被定义为物体密度与其量（当密度恒定时）的乘积的积分。这里的“量”代表物体中的物质数量，它取决于拓扑实体的类型。例如，对于实心体，这个量就是实心体的体积；对于片状体，这个量是面积；对于线框体，这个量是长度。密度从附加的密度属性中获取。如果没有附加密度属性，则密度默认为 1。关于属性的更多信息请参见 13.3 属性。

质量的数学方程如下：

$$Mass = \int \rho(r) dV$$

重心的定义是密度在给定位置与位置向量本身在整个体积上的乘积。它有如下的数学方程式：

$$G = \frac{1}{M} \int r dm = \frac{1}{M} \int \rho(r) r dV$$

转动惯量的数学方程如下：

$$I = \int_V \rho(r)((r \cdot r)E_3 - r \otimes r) dV$$

r 是从质心到体积 V 中某点的向量，$r \otimes r$是它们的外积，E_3 是 3x3 单位矩阵，V 是完全包含对象的空间区域。在建模引擎软件中，惯性张量矩阵（inertia tensor matrix）可以通过计算得到的转动惯量来确定，其定义为：$r \otimes r = \begin{bmatrix} xx & xy & xz \\ yx & yy & yz \\ zx & zy & zz \end{bmatrix}$。

在建模引擎软件中，返回的张量是关于一个点的转动惯量，这个点是在标准笛卡尔坐标系中定义的重心。

5.4 计算最大最小距离

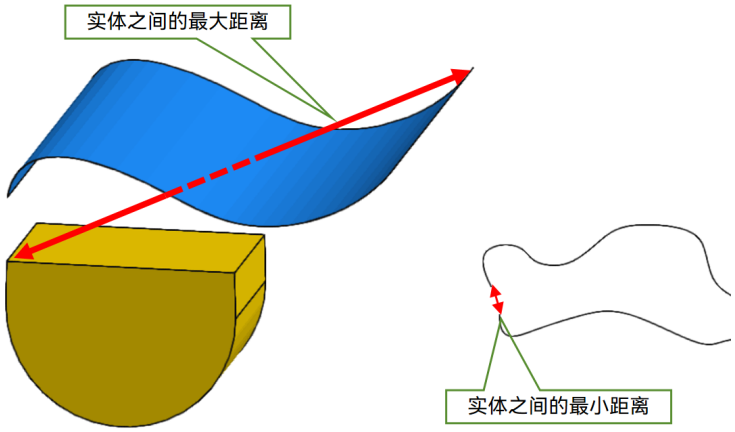
建模引擎软件提供了一系列用于计算实体间最大最小距离的接口。这些接口可用于以下对象：

- 两个几何实体或两组几何实体。
- 两个拓扑实体或两组拓扑实体。

- 一个拓扑实体和一个几何实体。
- 一组拓扑实体和一组几何实体。

此外，还可以计算一个实体或一组实体组与某个点或一组点之间的最小距离。

图 5-8 计算实体间的最大最小距离



5.4.1 可用接口

本节介绍用于计算实体间最大最小距离的接口。

下表中的接口用于计算两个实体之间的最大距离和最小距离，为了方便，这些接口在后续章节中被称为标准接口。

接口	描述	支持的实体
rangeGeometry	计算两个几何实体之间的全局最小或最大距离。	点、曲线、曲面
rangeTopology	计算两个拓扑实体之间的全局最小或最大距离。	顶点、边、面、主体
rangeTopoToGeometry	计算一个拓扑实体与一个几何实体之间的全局最小或最大距离。	点、曲线、曲面、顶点、边、面、主体

下表中的接口用于计算两组实体之间的最小距离和最大距离。为了方便，这些接口在后续章节中被称为数组接口。

接口	描述	支持的实体
rangeGeometries	两组几何实体之间的全局最小或最大距离。	点、曲线、曲面
rangeTopologies	两组拓扑实体之间的全局最小或最大距离。	顶点、边、面、主体

接口	描述	支持的实体
rangeToposToGeometries	一组拓扑实体与一组几何实体之间的全局最小或最大距离。	点、曲线、曲面、顶点、边、面、主体
rangeEntities	两组经过变换的实体之间的全局最小或最大距离。	点、曲线、曲面、顶点、边、面、主体

下表中的接口用于计算位置或位置数组与实体或实体数组之间的最小距离。为了方便，这些接口在后续章节中被成为向量接口(最后两个接口也属于前面提到的数组接口)。

接口	描述	支持的实体
rangeGeomToVector	一个几何实体与一个点之间的全局最小距离。	点、曲线、曲面
rangeGeomToVectors	一个几何实体和一组点之间的全局最小距离。	点、曲线、曲面
rangeTopoToVector	一个拓扑实体与一个点之间的全局最小距离。	顶点、半边、边、面、主体
rangeGeomsToVector	一组几何实体与一个点之间的全局最小距离。	点、曲线、曲面
rangeToposToVector	一组拓扑实体与一个点之间的全局最小距离。	顶点、半边、边、面、主体
rangeEntitiesToVectors	一组变换后的实体和一组点之间的全局最小距离。	点、曲线、曲面、顶点、半边、边、面、主体

您可以使用 **rangeEntities** 和 **rangeEntitiesToVectors** 来计算经过变换的实体数组之间的距离。

实体数组的变换是通过 transfTags 参数提供的。仅支持刚性变换（即镜像、平移、旋转或这些变换的组合）。

如果将这些参数设置为 PSGM_API_NULL_TAG，则数组中的实体不会进行变换。

5.4.2 可用选项

下表描述了用于计算全局最小或最大距离的可用选项。

说明
并非所有计算最大最小距离的接口都提供以下所有选项。选项使用范围见选项描述。

选项	描述
m_hasTol、m_tol	默认情况下，建模引擎软件使用线性精度计算最大和最小距离。可以通过这些选项指定计算精度的容差，从而更宽松地计算最小或最大距离。 使用这些选项不会影响以下方面的准确性： ● 正在计算距离的实体。也就是说，增加计算的容差并不意味着接近彼此的实体会被接口解释为接触或相交。 ● 返回数据的精度。尽管返回的端点可能不是距离计算的最优值，但它们符合建模引擎软件的线性精度。 适用范围：除 **rangeEntities** 和 **rangeEntitiesToVectors** 之外的所有接口。
m_bound	此选项允许指定返回距离的最大或最小限制。它是一个结构体，包含以下字段： ● m_hasUpperBound：是否需要提供距离上限。 ● m_upperBound：距离上限，如果提供了这个值，接口只会在计算出的距离小于这个值时返回距离。 ● m_hasLowerBound：是否需要提供距离下限。 ● m_lowerBound：距离下限，如果提供了这个值，接口只会在计算出的距离大于这个值时返回距离。 适用范围：所有接口。 如果同时提供下限和上限，则下限必须小于上限。

选项	描述
m_optLevel	此选项允许指定用于解析紧密相邻的局部最大距离或最小距离的分析级别。可以选择以下两个值之一： ● Performance：返回针对性能优化的结果。使用此值可确保结果尽快返回，但在复杂情况下，可能会返回局部最小距离或最大距离，而不是全局最大最小距离。这是默认值。 ● Accuracy：返回针对精度优化的结果。在复杂情况下，使用此值可能比使用 Performance 返回更精确的结果，但会计算降低性能。 适用范围：除 **rangeEntities** 和 **rangeEntitiesToVectors** 之外的所有接口。
m_guessCount1、m_guesses1、m_guessCount2、m_guesses2	在计算最小或最大距离时，可以通过估算（estimate）所提供实体上可能找到距离的端点位置来改善建模引擎软件的性能。可以使用这些选项提供您认为接近最大或最小距离的端点位置。更多内容请参见 5.4.3 通过评估提高性能。 适用范围：除 **rangeGeometries**、**rangeTopologies** 和 **rangeToposToGeometries** 之外的所有接口。
m_rangeType	指定是计算给定实体之间的全局最小距离还是全局最大距离。可以选择以下两个值之一： ● Minimum：查找最小距离。这是默认值。 ● Maximum：查找最大距离。 适用范围：除向量接口之外的所有接口。

选项	描述
m_paramBoundCount1、m_paramBounds1、m_paramBoundCount2、m_paramBounds2	指定几何实体（有界几何实体和无界几何实体）的边界。可以使用 PSGMApiRangeParamBound 结构提供边界： ● 如果无界几何是曲线，则提供一个区间。 ● 如果无界几何是曲面，则提供一个参数空间包围盒（uvBox）。 对于有界实体，这只是限制该实体的计算范围（range of interest）。 对于无界实体，这可以确保最大距离计算不会返回无限值。 当使用至少一个无界几何计算最大距离时，必须使用此选项；否则会导致接口错误。 适用范围：涉及至少一个几何实体的所有标准接口和数组接口。
m_paramEntity	指定在发现拓扑实体与向量之间的最小分离点位于子主体上时，是否在返回的 PSGMApiRangeEntitiesResult 中返回向量和参数。此选项仅在传递面、边或半边时影响接口的行为。默认值为 Topology。 适用范围：涉及至少一个拓扑实体的向量接口和 **rangeToposToGeometries**。

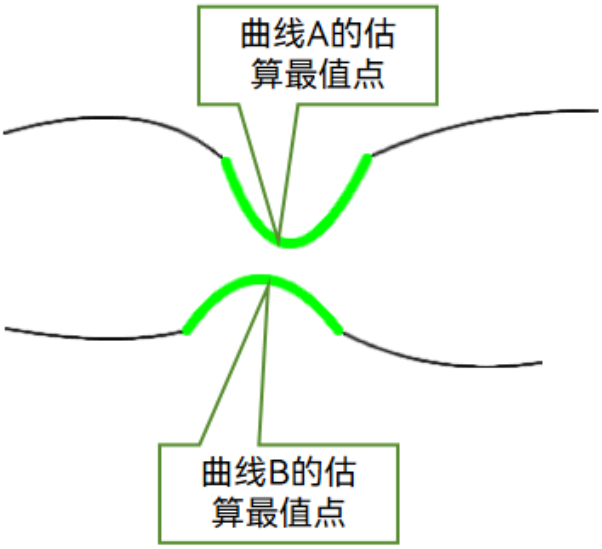
5.4.3 通过评估提高性能

m_guesses、m_guesses1 和 m_guesses2 选项（根据调用的接口不同，名称也有所不同）允许您提供一个参数或向量，表示您认为在该实体上距离的最小或最大端点位置附近的位置。提供估算最值点，可以提高这些接口在计算距离时的性能。

在已知参数信息的情况下，建议使用参数空间的估算，而不是向量估算。在内部，向量估算会通过将最值估算点投影到曲面上转换为参数估算。这需要额外的计算时间，而曲面复杂区域附近的向量位置（例如高曲率）可能会导致转换出次优解的 UV 参数估算。

下图展示了如何使用 **rangeGeometry** 计算两条曲线之间的最小距离。对于每条曲线，突出显示的参数被作为评估传递给接口。然后，**rangeGeometry** 根据提供的参数开始寻找解决方案，从而提高性能。

图 5-9 在计算两条曲线之间的最小距离时提供估算最值点



使用此选项时，需要为每个被计算距离的实体提供单独的评估，具体如下：

- 对于标准接口，可以为每个实体提供一个或两个评估。如果能提供两个评估，性能会得到最大改善。
- 对于数组接口 **rangeEntities**，您只能为单个实体提供一个评估。
- 对于向量接口（除了 **rangeGeomToVectors**），您只需要为涉及到的实体提供一个评估，为已知向量提供评估是没有意义的。
- 对于 **rangeGeomToVectors**，您需要通过 m_guesses 为传递给接口的每个位置提供一个评估。

不能为以下数组接口提供评估：

- **rangeGeometries**
- **rangeTopologies**
- **rangeToposToGeometries**

可以使用 PSGMApiRangeGuess 结构来提供评估。该结构包含以下字段：

- m_type：评估数据的类型。以下是可能的类型：
  - None：没有评估这是默认值。
  - Param：一个参数
  - Vector：一个位置。
- m_params：如果评估类型为 Param，则该字段包含相应的参数信息。它是一个包含最多两个参数的数组：
  - 为曲线或边提供数据时，请在 m_params[0]中提供曲线参数。
  - 为曲面或面提供数据时，请使用 m_params[0]和 m_params[1]同时提供曲面参数。
- m_vector：如果类型为 Vector，则该字段包含相应的位置信息。如果解决方案位于面的边上，则可以提供一个向量。

说明
不能在单个 PSGMApiRangeGuess 结构中同时提供参数评估和位置评估。

说明
不允许为涉及距离计算的主体提供评估。例如，查找一个面和一个主体之间的最小距离时，不能为主体提供评估。

5.4.4 返回结果

下面将介绍距离计算接口的返回数据。

除 **rangeEntities** 和 **rangeEntitiesToVectors** 之外所有计算距离的接口，都具有以下结果结构：

字段	描述
rangeResultType	一个用于指示是否找到解决方案的标记。可取以下值之一： ● Found：成功找到最小或最大距离。 ● Lower：未找到大于 m_lowerBound 的距离。 ● Upper：未找到小于 m_upperBound 的距离。 ● NotFound：未能确定分隔。（failed to identify the separation）
result	一个包含计算出的距离以及距离位置的细节的结构。 结构的具体内容取决于调用的接口： ● 对于所有向量接口，将返回一个 PSGMApiRange1Result，其中包含有关向量接口接收到的单个实体的端点位置的详细信息。 ● 对于所有其他接口，将返回一个 PSGMApiRange2Result，其中包含接口接收到的两个实体的端点的详细信息。 每个端点位置的详细信息包含在一个 PSGMApiRangeEnd 结构中。每个接口返回信息的描述，请参见接口文档。

rangeEntities 的结果结构包含以下字段:

- m_resultCount：距离的数量。
- m_results：一个用于指示是否找到解决方案的标记。可取以下值之一：

  - Found：成功找到最小或最大距离。
  - Lower：未找到大于 m_lowerBound 的距离。
  - Upper：未找到小于 m_upperBound 的距离。
  - NotFound：未能确定距离。
- m_dists：一个双精度数组，一个数组元素就是一对实体之间的距离。
- m_ends1：端点位置数组（End positions array）。
- m_ends2：端点位置数组。

rangeEntitiesToVectors 的结果结构包含以下字段：

- m_resultCount：距离的数量。
- m_results：一个用于指示是否找到解决方案的标记。可取以下值之一：
  - Found：成功找到最小或最大距离。
  - Lower：未找到大于 m_lowerBound 的距离。
  - Upper：未找到小于 m_upperBound 的距离。
  - NotFound：未能确定距离。
- m_dists：一个双精度数组，一个数组元素就是一对实体之间的距离。
- m_ends：端点位置数组。

5.5 检查碰撞

clashOfTopologies 用于检测任何组合的实心体、片状体和线框体之间的碰撞。此接口接收两组拓扑作为输入，并返回发生碰撞的实体，可以选择是否输出有关碰撞的信息，如：碰撞类型、发生碰撞的目标体和工具体。

检查碰撞的控制选项如下：

选项	描述
m_pairTagCount	需要忽略的实体对的数量。
m_firstPairTags	需要忽略的实体对的前半部分。
m_secondPairTags	需要忽略的实体对的后半部分。
m_findAll{false}	如果 m_findAll 为 true，则会查找两组拓扑之间所有的碰撞，如果 m_findAll 为 false，则会在找到第一个碰撞后停止查找。
m_findInt	检查碰撞是否存在真正的干涉（true interference），如果 m_findInt 为 false，则只会检查存在的碰撞，如果 m_findInt 为 true，则会通过 m_clashType 给出的每个碰撞的类型。

说明
相关章节：5.4 计算最大最小距离。

5.5.1 碰撞类型

当 m_findInt 参数设置为 true 时，**clashOfTopologies** 会返回检查到的碰撞类型。碰撞类型被包含在结构体 PSGMApiClash 的 PSGMApiClashType 结构中。 任何碰撞都可以被归类为以下类型之一：

- 干涉（interference）：实体的边界拓扑相互交叉。
- 接触（abutment）：实体的边界拓扑相互接触。
- 包含（containment）：一个实体完全在另一个实体内部，且它们的边界拓扑没有接触。

说明
当检查到涉及顶点的碰撞时，**clashOfTopologies** 不会返回碰撞的类型。

5.5.1.1 同维度主体间的碰撞

当检测到相同维度的实体之间的碰撞时，碰撞类型根据实体的边界拓扑行为（behaviour of the bounding topology of the entities）进行分类：

- 主体的边界拓扑是它的面。
- 面的边界拓扑是它的边。
- 边的边界拓扑是它的顶点。

PSGMApiClashType 可取以下值之一：

值	描述
None	没有碰撞。
Interfere	如果两个实心体共享一个公共体积（common volume），它们就会相互干涉。 如果两个面共享一个公共区域（common area），或者相交，它们就会相互干涉。 如果两条边共享一个公共长度（common length），或者相交，它们就会相互干涉。
AbutNoClass	两个实心体相互接触，但不共用一个体积时，两个实心体邻接（abut）。 当两个面在空间中共享一条共同的曲线或点，且该曲线或该点位于至少一个面的边界拓扑上时，两个面邻接（abut）。 当两条边共享一个点，且该点至少位于其中一条边的拓扑边界上时，两条边邻接。

值	描述
Exists	返回是否存在碰撞。
AInB、BInA	当一个实体完全位于另一个实体内部，且它们的边界拓扑不接触时，两个实体之间的碰撞类型为包含。

5.5.1.2 不同维度主体间的碰撞

与同一维度的实体一样，在检测不同维度实体之间的碰撞时，会根据实体的边界拓扑行为给碰撞分类，PSGMApiClashType 可取以下值之一：

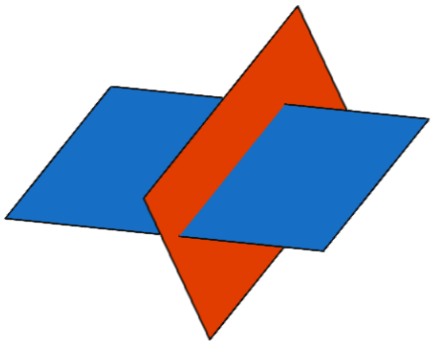
值	描述
None	没有碰撞。
AInB、BInA	如果一个物体完全位于另一个物体的边界拓扑中，而两个主体的边界拓扑没有接触，那么这两个主体的碰撞类型为包含。
Exists	存在碰撞，只用于说明存在碰撞，不区分碰撞类型。
AbutNoClass	如果两个主体的边界拓扑接触，且没有公共顶点，那么这两个主体的碰撞类型为邻接。
Interfere	如果在空间中有一个公共点位于两个主体的边界拓扑中，那么这两个主体的碰撞类型为干涉。

5.5.1.3 片状体间的碰撞

片状体之间的碰撞有些特殊，因为它不仅仅取决于边界拓扑行为。

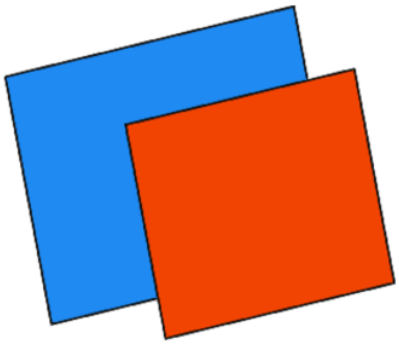
两个片状体只有在它们实际相交时才会发生干涉，如下图所示：

干涉
（interference）



两个位于同一平面内的片状体将会贴合在一起，如下图所示：

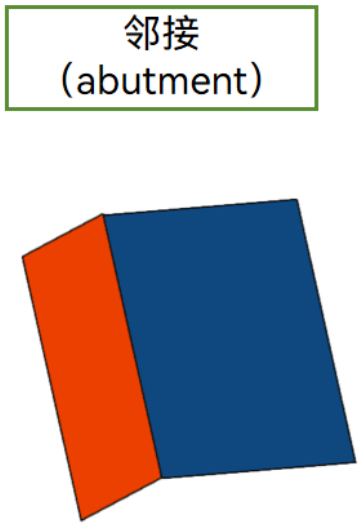
邻接
（abutment）



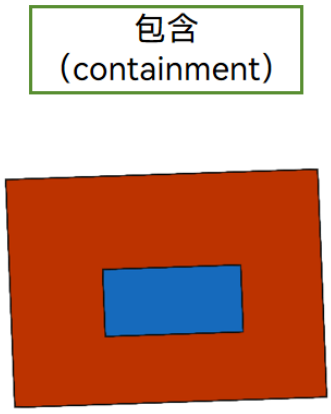
说明

对于面来说，上图中的碰撞将被归类为干涉。

一个片状体的边与另一个片状体的面邻接，如下图所示：



一个片状体包含另一个片状体，如下图所示：



5.5.2 返回信息

clashOfTopologies 会返回在两个拓扑集合中找到的相互碰撞的任何实体。这些实体的类型取决于两个因素：

- 检测碰撞的主体的类型。
- 碰撞的性质。

下表显示了如果存在碰撞，将返回哪些实体：

发生碰撞的主体	干涉碰撞和邻接碰撞返回的实体	包含碰撞返回的实体
实心体与实心体	面	主体

发生碰撞的主体	干涉碰撞和邻接碰撞返回的实体	包含碰撞返回的实体
实心体与片状体	面	主体
实心体与线框体	边或者面	主体
片状体与片状体	面	面
片状体与线框体	边	主体
线框体与线框体	边	边

第二列显示了检测到干涉或邻接时返回的实体。第三列显示了一组拓扑完全包含在另一组拓扑内部时返回的实体。

在查找不同维度的拓扑实体之间的碰撞时，首先返回具有最低维度的实体。在查找实心体和线框体之间的碰撞时，可能会返回两种类型的实体：

- 如果线框体与实心体上的边相交或邻接，则返回这些边。
- 如果线框体与实心体上的面相交或邻接，则返回这些面。
- 如果以上情况都不符合，则判定线框体与实心体之间的碰撞类型为包含。

5.5.3 面碰撞被忽略的情况

面碰撞（face clashes）可能在以下情况中被忽略：

- 因为只有当一个实体的面与另一个实体的面发生碰撞时才会被检测到，所以当一个实体完全包含在另一个实体内部时，可能检测不到碰撞。这是因为当一个实体完全位于另一个实体的内部时，即使存在体积上的重叠，它们之间不会有面的直接碰撞。
- 一般体中的线框边（wire-frame edge）与另一个实体中的面或线框边之间的碰撞不会被检测到。

5.6 检查实体

一般情况下，大多数接口不会检查接收到的实体是否有效，如果传入无效实体，则会导致这些接口不能正常工作。所以在给接口传入实体时，须要检查主体的有效性。

实体检查功能有以下作用：

- 协助应用程序的调试工作。
- 验证建模操作的结果。
- 协助从导入的数据中构建有效的模型。
- 指导用户修复无效的模型。

5.6.1 相关接口

默认情况下，检查接口会对提供的实体执行所有能执行的检查，调用者可以通过选项来控制执行哪些检查。以下为可用的检查接口：

接口	描述
checkAssembly	检查一个装配体。
checkBody	检查一个主体。
checkEdge	检查一条边。
checkFace	检查一个面。
checkGeometry	检查一个点、一条曲线或一个曲面。
checkTransform	检查一个变换。

不同类型的检查按照特定的顺序分组进行，每组检查只有在前面的检查全部通过时才会执行。因此，如果前面的检查失败，依赖其结果的后续检查将不会执行。这意味着检查功能不能保证在存在多个缺陷的情况下能够发现所有缺陷。

每个检查接口都有一个特定的选项结构，用于设置要执行哪些检查以及返回多少个错误。以下是这些选项的简要描述：

选项	描述
m_maxFaults	为发现的缺陷数量设置上限。有关更多信息，请参见 5.6.1.1 返回的缺陷数量。
m_geom	选择要执行的几何检查的类型。
m_corrupt	是否检查损坏的数据结构和标识符。
m_attribute	是否检查系统属性的有效性。
m_bgeom	是否执行 B 几何有效性检查。
m_topoGeom	选择几何与拓扑之间一致性检查的级别。
m_sizeBox	是否执行尺寸包围盒（size box）违规检查。
m_face	是否执行面自交检查。
m_loop	是否执行面的环一致性检查。
m_shell	检查壳的方向和壳的一致性。

下表总结了在检查不同类型的主体时可使用的检查选项：

选项	装配体	主体	边	面	面对（face pair）	几何	变换
m_maxFaults	√	√	√	√	√	√	√

选项	装配体	主体	边	面	面对（face pair）	几何	变换
m_geom	√	√	√	√	-	√	-
m_corrupt	√	√	-	-	-	-	-
m_attribute	√	√	√	√	-	√	-
m_bgeom	-	√	√	√	-	-	-
m_topoGeom	-	√	√	√	-	-	-
m_sizeBox	-	√	√	√	-	-	-
m_face	-	√	-	√	-	-	-
m_loop	-	√	-	√	-	-	-
m_shell	-	√	-	-	-	-	-

关于选项的详细信息，请参见*《Geoshape 几何建模引擎软件 接口开发文档》*。

5.6.1.1 返回的缺陷数量

这些检查接口返回的缺陷数量不会超过 m_maxFaults。如果在 m_maxFaults 设置为零时发现错误，则接口将返回错误码 AlgCheckError 。如果这样，应用程序强制所有检查出来的缺陷都导致错误。缺陷的元素通过缺陷检查的标准形式输出。

如果将 m_maxFaults 设置为大于零的值，则最多可以返回 m_maxFaults 个检查缺陷。为了尽可能获取更多缺陷，应用程序应将 m_maxFaults 设置为一个较大的数，例如 1000。但即使 m_maxFaults 足够大，也不能保证找到所有缺陷。

通常情况下，每个几何实体只返回最严重的缺陷。

5.6.1.2 检查系统属性

可以通过以下方式之一进行系统属性有效性检查：

- 在调用通用检查接口时通过选项来选择检查系统属性有效性。
- 调用 **checkAttributes**。
- 在设置系统属性时进行有效性检查。

本节介绍不同的检查方式以及如何控制系统属性检查。

有关属性的更多信息，请参见属性定义和属性。

检查项

在检查系统属性时，会检查以下内容：

检查项	描述
字符串字段的有效性	检查系统属性的所有字符串字段是否有效。
数值的有效性	如果系统属性字段具有可定义的有效值范围，则检查数值是否有效。
数组长度的有效性	检查每个系统属性中的数组是否提供了正确的数据量。例如： ● SDL/TYSA_DENSITY 的实数字段应该有一个值。 ● SDL/TYSA_HATCHING ● SDL/TYSA_PLANAR_HATCHING

建模引擎软件根据被检查的信息自动执行相关检查。

在执行一般检查时检查系统属性

在使用 **checkBody**、**checkAssembly**、**checkEdge**、**checkFace** 或 **checkGeometry** 对一个或多个实体执行一般检查时，如果有 m_attribute 选项则可以使用 m_attribute 选项来检查附加到这些实体的系统属性的有效性。此选项可取以下值：

- No：不执行系统属性有效性检查。
- Yes：执行系统属性有效性检查。

执行系统属性有效性检查可能返回的故障，请参见检查系统属性缺陷。

只检查系统属性

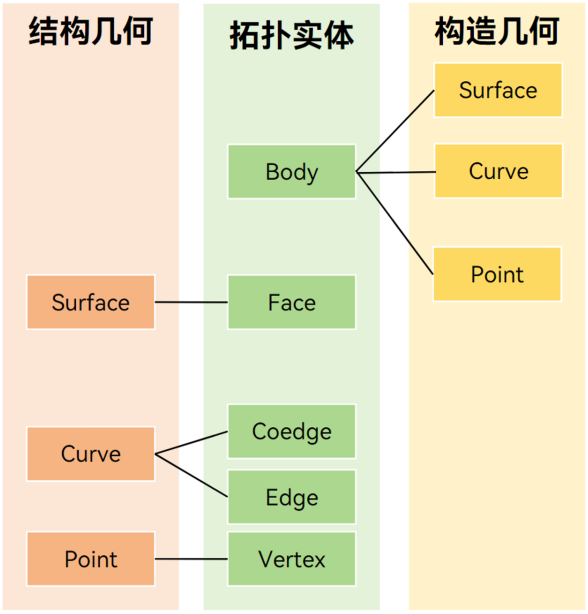
可以使用 **checkAttributes** 检查附加到一组实体的系统属性。该接口接收以下参数：

- entityTag：包含要检查的系统属性的实体。
- attrDefTag：使用此参数可以仅检查特定类型的系统属性：
  - 要检查附加到给定实体的所有系统属性，请将其设置为 PSGM_API_NULL_TAG。
  - 要检查特定类型的系统属性，请将其设置为要检查的系统属性的属性定义。

  可以使用 **getAttrDefTag** 来查找系统属性的属性定义。所有系统属性的名称都以“SDL/TY”开头。
- option：检查选项。

如果 m_checkSub 选项为 true，则会根据下图中的层次结构检查给定实体的任何子主体。例如，如果实体是一个环，那么当 m_checkSub 为 true 时，建模引擎软件将检查环、其半边、边和顶点，以及附着到半边或边的任何曲线和附着到顶点的任何点。

图 5-10 检查子主体上的系统属性时使用的实体层次结构



与其他检查接口一样，**checkAttributes** 返回一个包含返回令牌的结构，描述了发现的任何故障。请参见检查系统属性缺陷。

检查指定的系统属性

当您设置属性的值时，建模引擎软件会自动进行属性检查。使用以下接口设置系统属性值时，都会进行检查：

- **attrSetIntegers**
- **attrSetNamedIntegers**
- **attrSetDoubles**
- **attrSetNamedDoubles**
- **attrSetString**
- **attrSetNamedString**
- **attrSetVectors**
- **attrSetNamedVectors**
- **attrSetAxes**
- **attrSetNamedAxes**
- **attrSetPointers**
- **attrSetNamedPointers**

在这些情况下，只调用与所设置字段类型相关的检查。例如，对 **attrSetDoubles** 的调用不会执行字符串字段的有效性检查。

5.6.2 返回的缺陷

实体可能因多种缺陷而无法通过检查；并非每种缺陷都意味着实体完全损坏或不再适合建模。本节提供了有关每种缺陷类型（除了在使用选项时返回的缺陷类型）的详细信息，以及针对每种情况应采取的行动。

大多数建模操作会对检查出缺陷的实体（例如，具有一个无效的面、边或几何体）继续执行局部操作，只要有问题的区域不直接涉及到操作就行。通常使用局部操作或布尔操作将有缺陷的实体完全移除。

同样，许多建模操作可以在未通过面一致性检查的实体（例如，面与面相互穿插）上继续进行操作，尽管消隐线算法可能会失败；质量特性计算可能会给出误导性的答案。

下表为缺陷类型总结：

缺陷	描述
数据结构损坏：BodyCorrupt	返回的实体已损坏。这可能是由于调用者在建模操作执行失败（例如，布尔运算失败）后未回滚导致的；也可能是建模引擎软件中存在严重错误，应该报告；还有可能是建模引擎软件接口的调用序列无效所导致的。调用者必须回滚或重新启动建模引擎软件。
环形引用：AssemblyCyclicRef	该装配体存在自引用。这可能是由于该装配体至少具有一个实例，其子部件最终是一个父部件。 如果在建模引擎软件中创建的装配体中出现此缺陷，则应将其报告为建模引擎软件中的严重错误。

5.6.2.1 检查系统属性缺陷

如果将 **checkAssembly** 、**checkBody**、**checkEdge**、**checkFace** 或 **checkGeometry** 中的 m_attribute 选项设置为 Yes，则返回的缺陷结果中还会包含以下字段：

- m_entityTag1：被检查的属性。
- m_entityTag2：被检查的属性的所有者。
- m_position：有缺陷的几何体内部的某个位置。

使用可以 **checkAttributes** 进一步查找缺陷的相关信息，**checkAttributes** 可能返回以下类型的缺陷：

缺陷	描述
字段中的数据无效	系统属性的字段填充了无效的数据量。

缺陷	描述
无效的 Unicode 字符串	系统属性中的 Unicode 字符串无效。
字节字段超出范围	系统属性的字节字段超出（有效）范围。
空字段	系统属性中的必填字段未填充任何数据。
整数字段超出范围	系统属性的整数字段超出（有效）范围。
无效数据	系统属性具有无效数据。
非单位长度向量字段	所需为单位长度的向量字段不符合要求。
实数字段超出范围	系统属性的实数字段超出（有效）范围。
短整数字段超出范围	系统属性的短整数字段超出（有效）范围。
位置向量字段超出范围	位置向量字段超出范围（通常在尺寸包围盒之外）。

5.6.3 何时使用检查

检查是一个耗时的操作。在执行一系列局部操作时关闭局部检查可以节省时间，或者其中一个中间操作需要创建一个无效主体，可以调用 **checkBody** 来检查最终的结果体。如果检查表明结果体是无效的，可以回滚到局部操作之前的一个点，使用不同的方法来重新建模使结果体有效。

在执行可能导致主体无效的操作之前，需要尽量确保主体是有效的。这可以通过定期调用 **checkBody** 来确保，特别是在进行复杂或关键的建模步骤时。

如果在执行局部操作时启用局部检查，建模引擎软件会检查每个受影响的面以确保它们与自身和附近面的子集一致。

如果受影响的面与自身不一致，返回的参数是未定义的，并且会返回一个非零错误，以指示主体即使作为中间阶段也是无效的。

选择何时使用局部检查或全面检查取决于具体需求和工作流程。对于大型或复杂的模型，全面检查可能会更加耗时，但它可以提供对整个主体状态的全面概述。对于小型或简单的模型，或者只需要验证特定操作的结果时，局部检查可能会更快。

🕮 说明
在应用倒角时，局部检查作为 **blendApplyBody** 操作的一部分执行。

扫掠和旋转

在旋转体时，必须注意轴的定位。例如，如果一个线框体的内部顶点位于轴上，则该线框体的旋转角度不能等于或大于 2π ，因为这会导致旋转结果为非流形片状体。同

样，通过扫掠或旋转容易创建自交主体。请务必检查创建结果，可以通过启用局部检查或使用 **checkBody** 对结果主体进行检查。

有关这些限制的更多信息，请参见 **spinBody** 的描述。

倒角线框体和片状体上的顶点

如果已经启用局部检查，则会在倒角操作后进行主体检查。对于平面体（planar body），倒角后的主体是有效的，但对于非平面体（non-planar body），局部检查可能会检查出倒角后的主体无效。

创建曲线和曲面

在建模引擎软件中创建曲线和曲面时，对生成的曲线和曲面有一些限制。这些限制是为了确保曲线或曲面不会自交或具有尖点（cusp）。一些接口必须使用经过检查的曲面，曲面检查只执行一次，检查结果存储在数据结构中。

可以使用 **checkGeometry** 接口来检查曲面是否可用。

有关参数曲线和参数曲面的限制的详细内容，请参见 B 曲线和 B 曲面。

应用程序如何使用检查

应用程序应为用户提供对局部检查的控制，并应提供随时检查主体的选项：建议提供两种主体检查，一种包含面与面一致性检查，另一种不包含。除最简单模型外其他所有模型的面与面一致性检查都是最耗时的，因此只有在怀疑面存在自交时才使用。

通过经验积累，您将能够判断在建模过程中何时使用局部检查以及多久使用一次局部检查。刚开始时，您可能会经常使用局部检查，但随着经验的积累，您将能够在不需要检查时跳过检查，从而更快地工作。

6 模型编辑

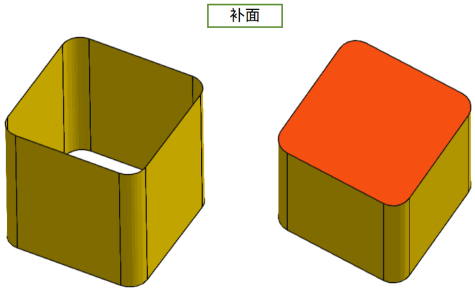
6.1 内容简介

本章主要介绍如何操作主体中的面。可以通过修改面来修改模型，也可以用面填充主体中的孔洞。还可以通过变换面、旋转面、扫掠面或偏移面来创建实体。本章还介绍了通过光顺曲线和曲面来修改模型。

本章包含以下内容：

- 6.3 面拔模：介绍如何通过修改面，让模型更易脱模。
- 6.4 移动面：介绍如何变换面。
- 6.5 光顺：介绍如何光顺曲线和曲面。

图 6-1 修改面的操作示例



6.2 简化几何

在几何建模引擎中，同样的形状，可以有不同的表达方式。在各种表达方式中，用户大概率是希望数据越简单越好。假设有一个由 BSpline Surface 构成的 Face，如果它的所有控制顶点都在同一平面上的话，那么用户可能会更希望用平面来表达几何。

比如 B 样条曲线的控制顶点在同一直线上就可以简化为一条直线，B 样条曲面同理。再比如一个 Rational 的 B 样条曲线如果可以无损转换为 Non-rational 的 B 曲线的话，这对用户来说也是能起到简化几何的作用的。

本建模引擎软件提供了两个接口用于在一个主体中简化几何：

- **simplifyBodyGeometry**：简化一个主体的几何。
- **simplifyFaceGeometries**：简化主体中面的几何。

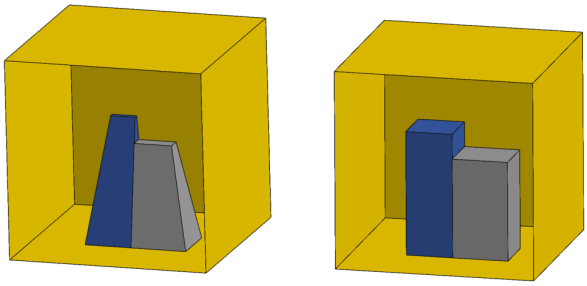
下表中说明了不同类型的几何的原始形式和简化形式。

几何类型	原始形式	简化形式
曲线	有理 B 曲线	无理 B 曲线
	B 曲线	线或圆
曲面	有理 B 曲面	无理 B 曲面
	B 曲面	平面、柱面、锥面、球面、圆环面

6.3 面拔模

本章介绍了在模具制作或模型铸造过程中，如何使用拔模来修改面。拔模可以使模型能够轻松地从模具中取出，如下图所示：

图 6-2 已拔模模型和未拔模模型



拔模通常用于建模注塑或铸造模型，通过改变主体侧面角度，实现模型的轻松脱模。对一组面进行拔模时会经历两个阶段：

1. 分析阶段：在这一阶段，主体中的面被分为需要拔模的面集和不需要拔模的面集。
2. 拔模阶段：在这一阶段，需要拔模的面的曲面会被替换为等斜曲面，曲面替换可以通过 **taperFaces** 实现。等斜曲面是指，在该曲面的点上，最陡的曲面切线与脱模方向之间的角度都等于拔模角度。

这两个阶段中，需要提供一个指向模具内部的拔模方向向量和一个拔模角度。

说明
在拔模过程中，等斜线曲线被定义为曲面的陡峭区域与非陡峭区域之间的边界。在等斜线曲线的所有点上，面法线与拔模方向之间的角度等于 π/2 减去拔模角度。

此功能为离散几何提供部分支持。

6.3.1 面拔模接口

面拔模接口 **taperFaces** 拥有以下参数：

参数	描述
faceTagCount	拔模操作用到的面的数量。
faceTags	拔模操作用到的面的标签列表。
referenceTags	拔模操作用到的参考实体。
dir	面拔模的方向。
angle	面拔模的角度。

6.3.1.1 参考实体

taperFaces 在为每个面进行拔模处理时需要通过 referenceTags 提供参考实体。参考实体用于定义拔模曲面。参考实体可以是以下实体之一：

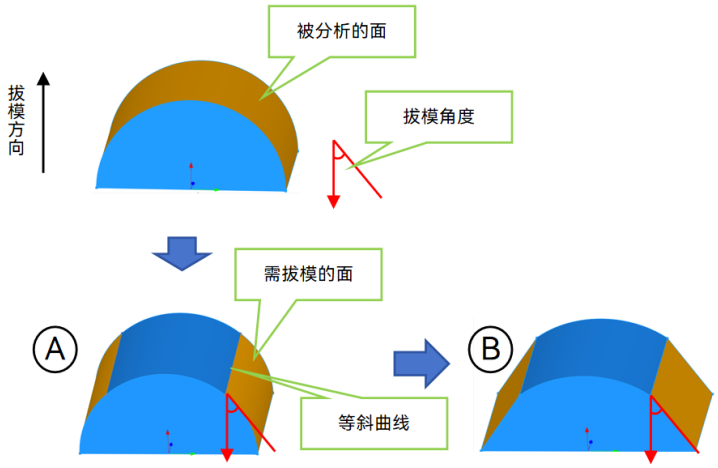
实体	描述
一条边	在拔模过程中，参考边保持固定，而拔模面的其他边则会被修改以完成拔模操作。如果在分析阶段已经压印了等斜曲线，那么生成的边可以作为拔模阶段的参考边使用。
一个面	拔模面会获得与参考面相同的拔模曲面。详细内容请参见示例三：以邻接面作为参考实体。
一个曲面	在拔模过程中，拔模面与参考曲面的交线会保持固定。

如果一个面被用作参考实体，它必须同时出现在面数组和参考实体数组中，这样面数组中的每个面都会映射到参考实体数组中对应位置上的单个参考实体。可以通过在面数组中复制面来指定多个参考实体。

以下示例将从不同方面说明此功能。

示例一：将单个面进行分割并对得到的面进行拔模

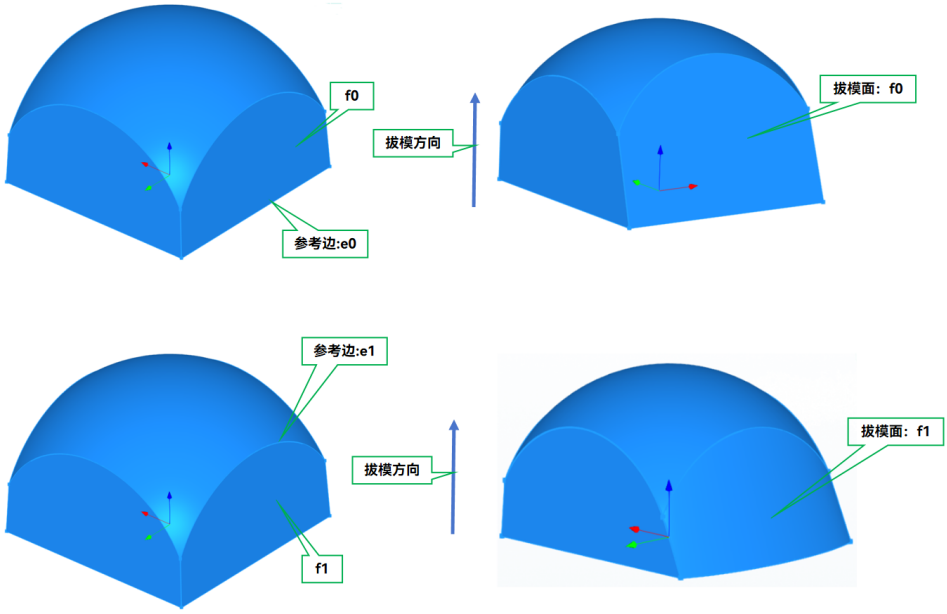
图 6-3 分割一个面并拔模两个分割面



A 小图中的曲面已经在分析阶段被两条等斜曲线分割成三个面，其中的两个面需要进行拔模处理。B 小图中的曲面已经完成了拔模，且等斜曲面与原始曲面之间平滑过渡。

示例二：选择作为参考实体的边

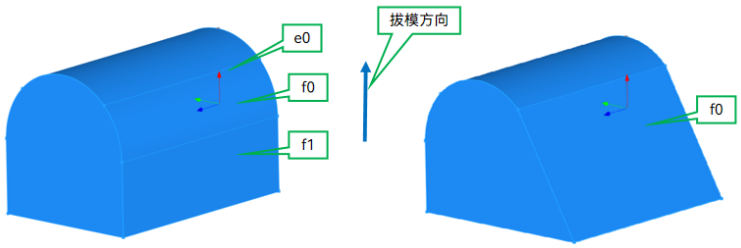
图 6-4 选择 e0 和 e1 做为参考实体



面 f0 和面 f1 都需要进行拔模处理。在这种情况下，可以将边 e0 和 e1 作为参考实体，它们在拔模处理后会保持几何形状不变。此示例中，面 f0 的拔模面是一个平面，面 f1 的拔模面是一个有界 B 曲面（ruled B-surface）。

示例三：以邻接面作为参考实体

图 6-5 使用邻接面作为参考实体进行拔模



在对 f0 和 f1 进行拔模时，选择 e0 作为 f0 的参考面，选择 f0 作为 f1 的参考面，使 f0 和 f1 具有相同的拔模曲面。由于 f0 和 f1 是相邻的，它们会被合并成一个面。

6.4 移动面

本章介绍如何旋转面、扫掠面和偏移面。

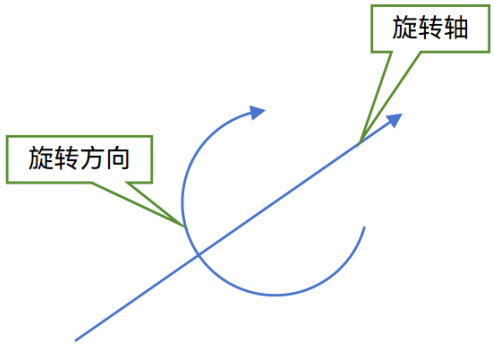
6.4.1 旋转面

spinFaces 用于旋转面。

旋转方向

旋转实体时，使用右手螺旋定则来确定旋转方向。

图 6-6 旋转轴和旋转方向



旋转和扫掠的使用

使用现有实体进行旋转通常比创建新实体来旋转更容易。例如，当旋转一个面时，可以通过在实体或片状体的面上绘制线条来创建新的边，从而形成一个新的面。如果这些线条将现有面分割成多个部分，只有一个部分为旧的面，其他部分都会变成新的面。然后，选择其中一个面进行旋转，以延伸原始实体。

在旋转时需要确保旋转面位于正确位置。

通过旋转创建的新实体

当实体被旋转时会创建新的边和面。

在创建新曲面的过程中，会尝试创建解析曲面（analytic surfaces）。解析曲面是指可以通过数学公式精确描述其形状的曲面，通常具有更高的计算效率和精度。

旋转操作对现有边的影响

通常情况下，旋转执行操作之前存在的边在操作执行之后仍然会存在。这些边可能会受到操作的影响从而发生旋转。

当执行 360°旋转时，如果与通过 360°旋转创建的面相关联的曲面是解析曲面或生成曲面（generated surface），那么原始实体的边将会消失。例如，当一个圆或椭圆绕与其法线正交的轴进行旋转，并且旋转的中心点位于圆或椭圆的平面内时，就会出现这种情况。在这种情况下，原始圆或椭圆的边界边将不再作为独立的边存在，而是融入了新形成的面的几何中。

在旋转操作完成后，旋转结果体中某条边的标签与原始主体中边的标签一样，但该边可能在一个或多个顶点处被分割。

6.4.2 拉伸面

extrudeFace 可以拉伸实心体的一个或多个面，也可以拉伸一个或多个片状体。

拉伸的使用

可以通过在原始实心体的面所在平面上创建一个片状体，然后使用拉伸片状体创建一个新的实心体，再用原始实心体和新的实心体执行布尔减操作即可在原始实心体上创建一个凹槽。

拉伸操作创建的新实体

当实体被拉伸时会产生额外的边和面。

在创建新曲面的过程中，如果可以，会创建解析曲面（analytic surfaces）。

拉伸操作对现有边的影响

通常情况下，执行拉伸操作之前存在的边在操作执行后仍然会存在。这些边可能会在拉伸操作的影响下发生偏移。

6.4.3 偏移面

详细内容请参见 11.2 偏移。

6.5 光顺

光顺的定义

光顺一词既可以是名词（即光顺性），也可以是动词（即光顺操作）。

从数学的角度来说，光顺的目标分为三种，即在给定约束条件下：

- 曲线或曲面的能量最小（mininum energy curve）。
- 曲线或曲面的变化最小（mininum variation curve）。
- 前两种目标按比例混合。

曲线能量最小就是曲线的曲率积分最小，即在满足约束条件下，曲线曲率总体绝对值的和尽量小。

在实际应用中要达到哪种目标，可由用户自己通过设置这两种目标的权重（比如一个占 25%，另一个占 75%）来控制。

成功评判标准

在做光顺操作时，用户需要对输入形状添加改动约束，否则可能会产生不良结果，一个较为极端的例子是：所有控制点合并在一个点，导致曲率、曲线长度等都为 0。常见的约束有固定端点和限制光顺结果与原来的形状之间的最大偏差。

一个光顺算法是否成功首先要看结果是否满足约束条件，如果不满足可以直接判定为失败。在满足约束的基础上，还需要满足以下条件：

- 从凹凸性上讲有更少的单调曲线段，即拐点变少，不会出现扭来扭去的结果。
- 局部小特征会变少，原曲线上一些局部变化比较大的区域会得到改善。
- 左求右求曲率不连续的节点数量降低。

这三个条件在用户给的约束较为宽松时，光顺结果会与原曲线有更大的差异。如果给的约束限制比较大，那么可能原曲线上的某些缺陷就无法通过光顺来修复。

总的来说，不论给出怎样的约束，如果光顺后的结果比原曲线差，即可判定为光顺失败。

6.5.1 曲线光顺

可使用 **fairBCurve** 对曲线进行光顺。

对曲线进行光顺时的约束条件有：

- m_maxTol：偏差最大值。
- m_fairG1Discontinuity：是否过滤 G1 不连续 B 曲线。
- m_fairG2Continuity：是否过滤 G2 连续 B 曲线。

光顺方法有：

- 曲线近似拟合。
- 曲线形状修改。

图 6-7 光顺曲率变化较大的 B 曲线为曲率变化较大的 B 曲线的光顺示例，灰色线为光顺前的 B 曲线（G1 不连续），蓝色线为光顺后的 B 曲线。图 6-8 光顺前后曲线的曲率梳为该曲线光顺前后的曲率梳。

图 6-7 光顺曲率变化较大的 B 曲线

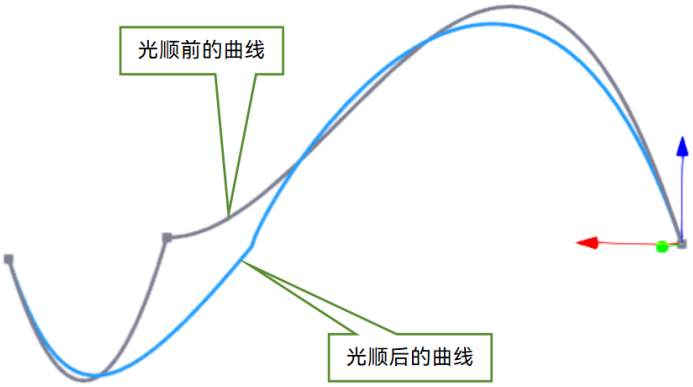
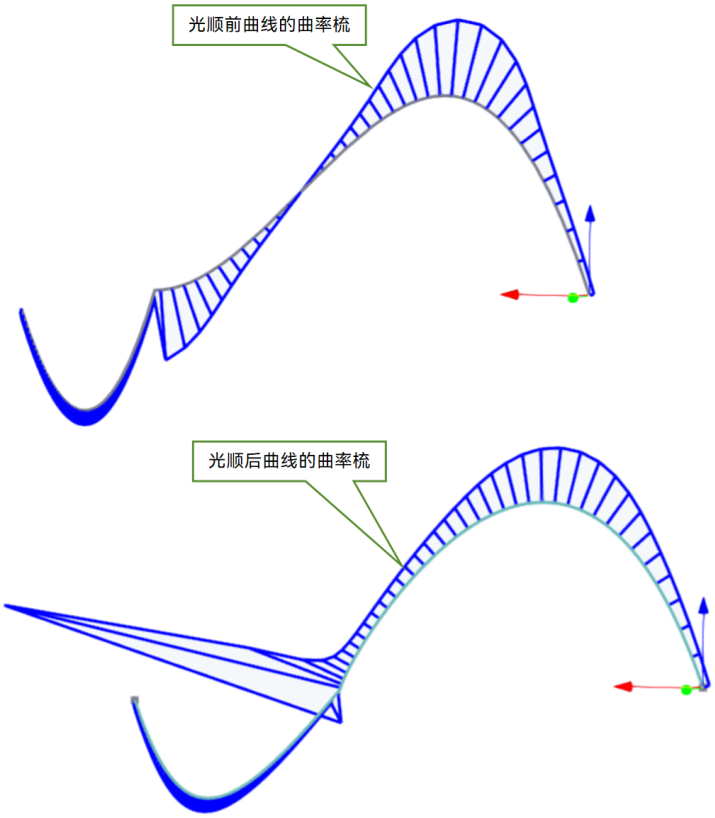


图 6-8 光顺前后曲线的曲率梳



曲线被光顺后能量变小。曲线内的顶点被光顺，该点原来的曲率为无穷大，光顺后变为有穷大。

6.5.2 曲面光顺

曲面的光顺只有曲面内的光顺，即光顺曲面内部的偏离点。可以使用 **fairBSurface** 对曲面进行光顺。

对曲面进行光顺时的约束条件有：

- m_maxTol：偏差最大值。
- m_fairG1Discontinuity：是否过滤 G1 不连续 B 曲面。
- m_fairG2Continuity：是否过滤 G2 连续 B 曲面。

图 6-9 光顺普通 B 样条曲面示例为对普通 B 样条曲面进行光顺的示例，灰色为光顺前的 B 样条曲面，蓝色为光顺后的 B 样条曲面。图 6-10 光顺普通 B 样条曲面为该曲面光顺前后的斑马条纹。该曲面在光顺后曲面上所有位置点的曲率和减小。

图 6-9 光顺普通 B 样条曲面示例

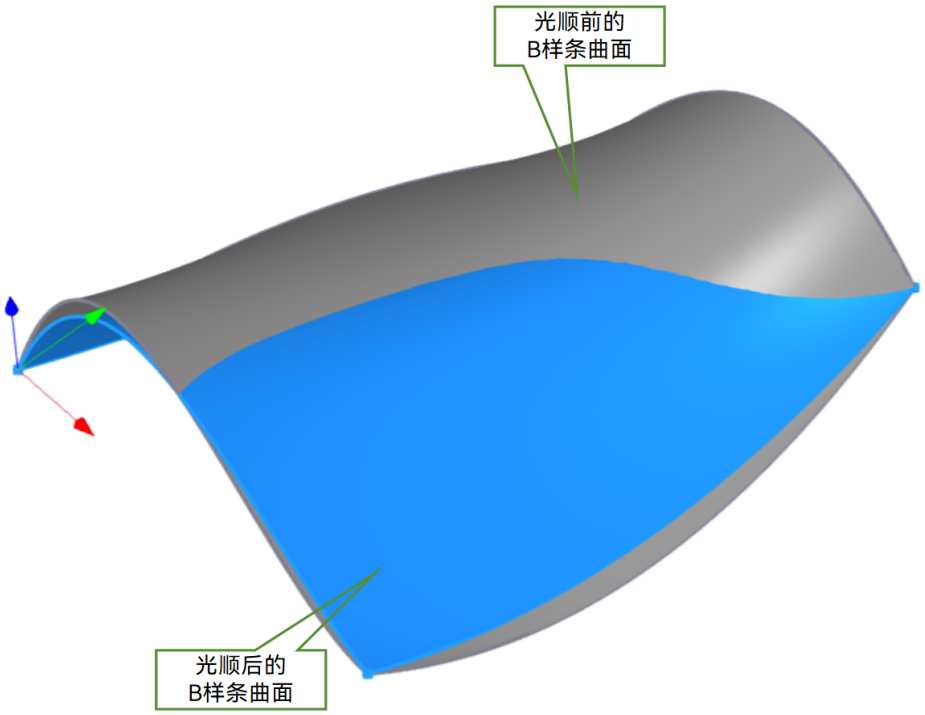
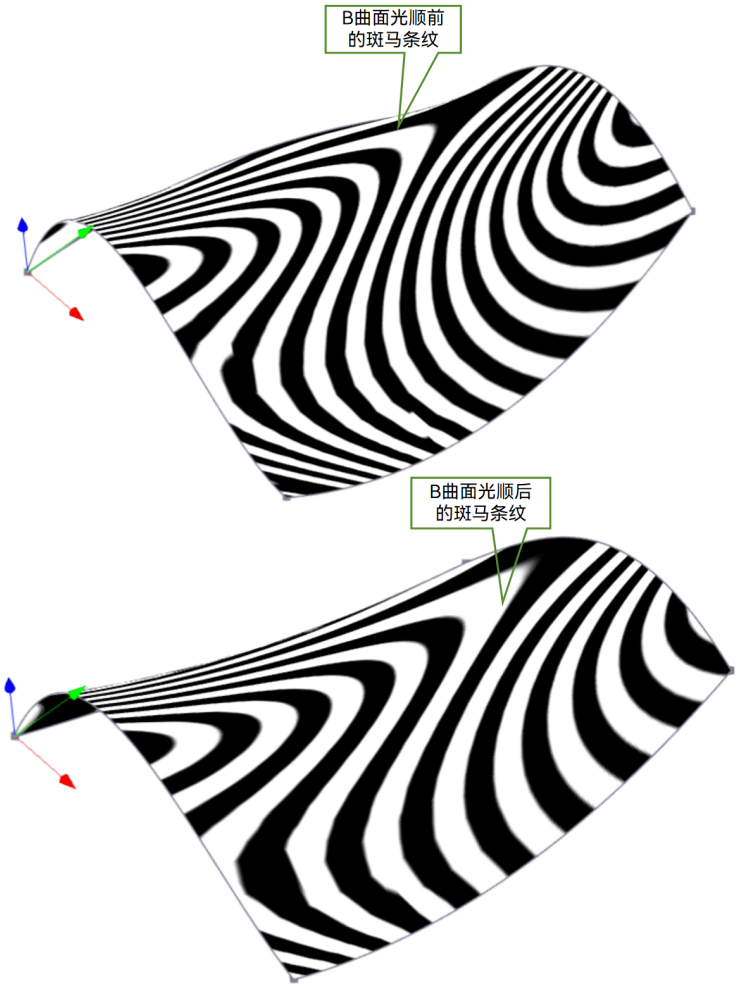


图 6-10 光顺普通 B 样条曲面



7 轮廓建模与曲面建模

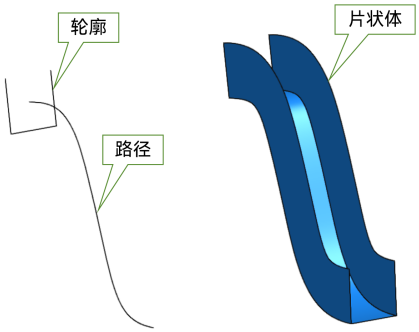
7.1 内容简介

本建模引擎软件提供了一系列曲面建模功能，这些功能可以通过轮廓体（profile body）和线框体（wire body）创建复杂的主体。这些功能在拓扑层面进行了集成，保证了软件的功能性、可靠性和性能。本章介绍如何将一组线框沿线性方向拉伸为一个主体、如何绕轴旋转轮廓、沿路径扫掠轮廓或在一组轮廓之间进行放样来创建主体。

本章包含以下内容：

- 7.2 拉伸：描述了如何通过线性拉伸一个轮廓来创建一个主体。
- 7.3 扫掠：描述了如何通过沿着路径扫掠实体来创建新主体。
- 7.4 放样：描述了如何通过拟合一系列曲面来创建一个片状体或实心体。

图 7-1 通过扫掠开放线框体轮廓创建片状体



7.2 拉伸

可以使用 **extrudeBody** 来拉伸轮廓。此接口通过线性拉伸轮廓来创建一个主体。轮廓可以是最小体、线框体或片状体，它们可以分别被拉伸为线框体 、片状体、实心体。拉伸的起始位置和终止位置可以通过指定拉伸距离来确定。

拉伸的方向是由一个单位向量来定义的。拉伸的起始位置和终止位置必须沿着此向量进行排序，以确保起始位置在终止位置前面。

拉伸的起始位置和终止位置可以通过以下边界类型来定义：

边界类型	描述
距离（distance）	拉伸是通过轮廓体的副本沿特定方向移动一定距离，从而界定出拉伸的范围，原始轮廓体所在位置为起始位置，副本所在位置为终止位置。

更多内容，请参见 7.2.2 边界类型。

7.2.1 拉伸参数

本建模引擎软件为 **extrudeBody** 提供以下参数：

参数	描述
profileTag	用于拉伸的轮廓体的标签，轮廓体可以是最小体、线框体或片状体。详细内容请参见 7.2.1.1 轮廓类型
pathDir	定义拉伸方向的单位向量。
option	拉伸选项，可用于指定以下内容： ● 拉伸的起始位置和终止位置，详细内容请参见 7.2.1.2 起始位置和终止位置

7.2.1.1 轮廓类型

以下是轮廓类型和对应的拉伸结果体类型。

轮廓类型	结果体类型
最小体	线框体
线框体	片状体
片状体	实心体

说明
轮廓不能为一般实体。片状体轮廓必须为连接的流形片状体，它可能包含多条线性边界（multiple laminar boundaries）。

7.2.1.2 起始位置和终止位置

拉伸的起始位置和终止位置是通过两个边界来指定的。这两个边界的类型可以由 PSGMApiExtrudeBodyOption 中的 PSGMApiBoundDefine 来指定，并且它们的指定方式相同，指定操作涉及以下参数：

参数	参数取值
m_bound	边界类型，可取以下值之一： ● Distance
m_isForward	指定终止位置位于轮廓的哪一侧，即拉伸是向前拉伸，还是向后拉伸。可取以下值之一： ● true：向前拉伸。 ● false：向后拉伸。
m_dist	拉伸距离。默认值：0.0。

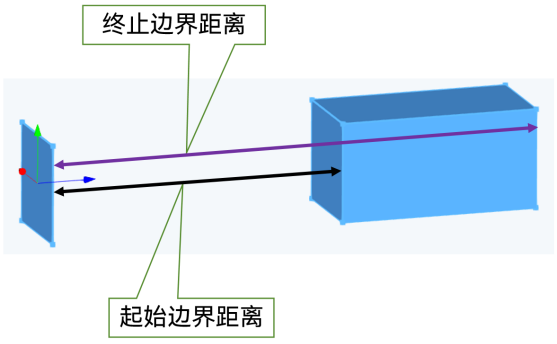
7.2.2 边界类型

本节将详细介绍不同的边界类型。

距离边界

边界是通过在路径向量方向上将给定轮廓的副本移动一定距离来产生的。这个距离必须是非负的。若要在与路径向量相反的方向上进行拉伸，则需要将边界的 m_isForward 字段设置为 false。下图为此边界类型的实例：

图 7-2 将指定轮廓按指定距离拉伸



7.3 扫掠

扫掠功能通过将实体沿着任意路径移动来创建新实体。
可以使用 **sweepProfileBodies** 沿指定路径扫掠一个线框体或片状体。

说明
此功能不支持离散几何。

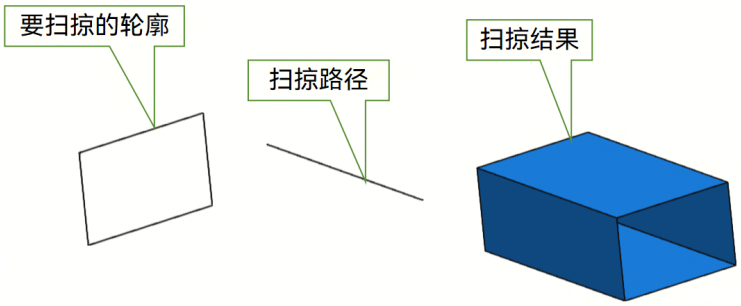
7.3.1 简介

7.3.1.1 扫掠的基础知识

在执行扫掠操作前需要提供一个或多个轮廓和一个扫掠路径，扫掠路径以线框体形式提供，轮廓则作为一组线框体或片状体来提供。

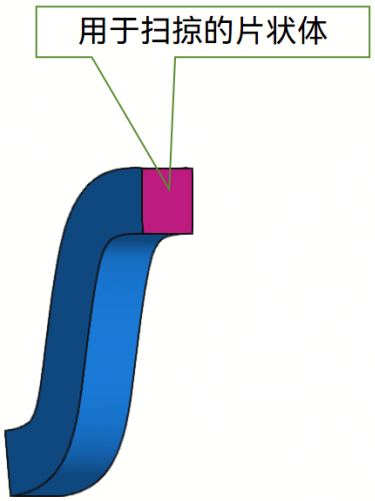
下图为线框体的扫掠示例。

图 7-3 扫掠线框体



下图为片状体的扫掠示例。

图 7-4 扫掠片状体



7.3.1.2 控制扫掠

可以使用 **sweepProfileBodies** 中的 m_alignment 选项来控制对齐方式，

7.3.2 无引导线扫掠

本节介绍通过 **sweepProfileBodies** 进行无引导线扫掠。

7.3.2.1 提供轮廓

轮廓以单个或多个线框体或者单个或多个片状体的形式提供，提供的片状体要求是闭合片状体。

如果仅指定了一个轮廓，那么生成的扫掠体在沿着路径的方向上其横截面将是恒定的（除非使用了缩放功能），并且与指定的轮廓保持一致。

当指定多个轮廓时，生成的扫掠体的横截面将在这些指定轮廓的形状之间平滑转换。

只提供线框轮廓

如果提供的轮廓只有线框体，扫掠结果将是片状体，片状体的法向（外向）与路径的切线方向保持垂直。

图 7-5 扫掠开放线框体

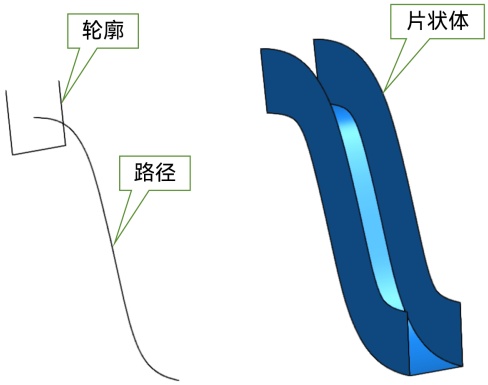
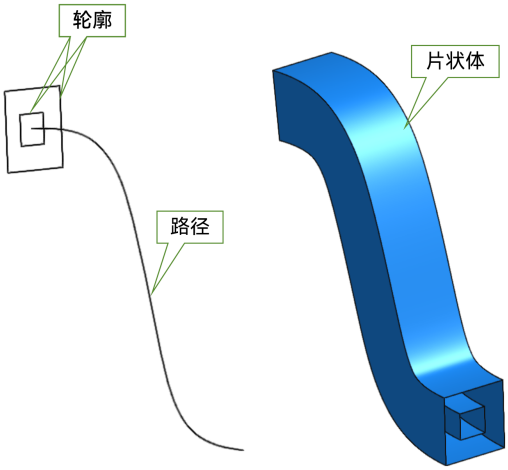


图 7-6 扫掠线框体组合



当提供多个轮廓时必须确保所有轮廓方向一致。

当提供的轮廓为不相交轮廓时，扫掠结果体可能会产生干扰：

- 如果提供的轮廓包含不连接的片状体，本建模引擎软件会检查是否存在干涉，任何干扰部件在返回之前都要进行统一。
- 如果提供的轮廓只包含不连接的线框体，本建模引擎软件不会执行干涉检查。

7.3.2.2 提供路径

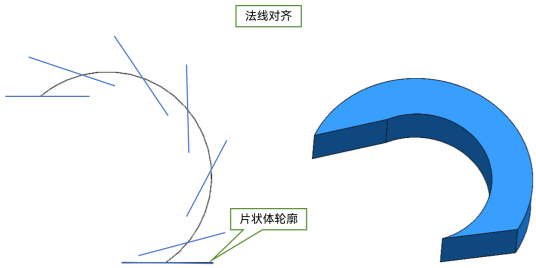
提供的路径定义了扫掠的方向和长度。这个路径是以线框体的形式给出的，它必须是流形的且带有方向。扫掠的方向遵循线框体的方向，扫掠从路径的起始顶点开始，到结束顶点结束。

📖 说明
路径不能包含任何容差顶点。

7.3.2.3 控制轮廓对齐

m_alignment 用于控制轮廓在沿路径移动时的方向。当没有引导线存在时，可以选择保持方向不变，或者改变扫掠方向以跟随路径的形状，如下图所示：

图 7-7 扫掠对齐



m_alignment 可取以下值：

值	说明
Normal	沿着路径扫掠时保持轮廓的方向不变。

7.4 放样

放样（loft）是通过一系列轮廓来拟合曲面，从而创建出一个片状体或实心体。可以使用 **loftProfileBody** 来放样。放样可以是周期性的（起始轮廓和结束轮廓相同），也可以是非周期性的（起始轮廓和结束轮廓不同）。可以分段放样，这样会重复使用轮廓。

📖 说明
此功能不支持离散几何。

可以通过指定以下任何一项来进一步约束放样中跨轮廓的几何：

- 末端条件（end condition）：控制放样在起始或结束处的几何。
- 导数条件（derivative condition）：控制任何轮廓中间点（intermediate point）的几何。

导数条件以及包含不同数量边的相邻轮廓如何匹配都由选项控制。

7.4.1 提供轮廓

轮廓是作为线框体数组、片状体数组或最小体数组提供给 **loftProfileBody** 的。片状体和最小体只能用作起始或结束轮廓。所有中间轮廓都必须是线框体。

周期性和非周期性放样所需的最少轮廓数量如下：

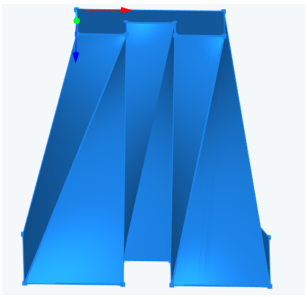
是否具有周期性	所需轮廓的最小数量
否	2
否	1 个线框体轮廓或 2 片状体轮廓
是	3
是	1

如果起始轮廓或结束轮廓是线框体，那么生成的主体将是片状体。

在生成的主体中，不会显示任何轮廓的几何或拓扑结构，当不再需要某些轮廓时，由您决定是否删除它们。

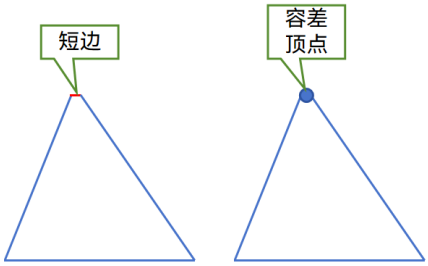
您必须为每个轮廓提供一个起始顶点。这些起始顶点应该对齐，以避免结果中出现扭曲，如下图所示。

图 7-8 扭曲放样



为了确保最佳结果，轮廓中的任何一条边的长度都应该至少是放样操作所指定的容差的十倍。如果某条边的长度小于这个值，可能会导致放样失败，这取决于该边的相邻边的相对长度，以及与其他轮廓匹配的边的长度。如果轮廓中包含小于此长度的边，您可以使用容差顶点替换它，并在相邻轮廓中指定一个退化的匹配（在起始或结束放样的情况下），如下图所示。

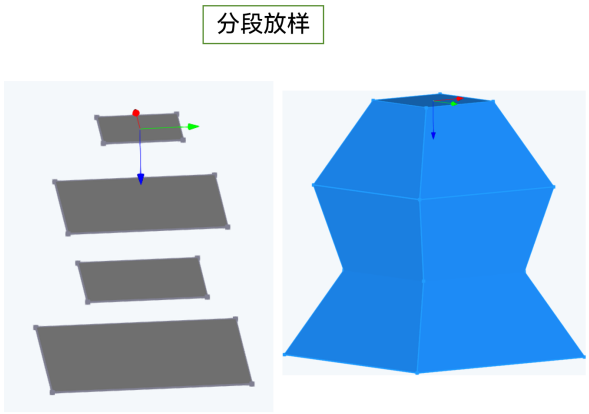
图 7-9 用容差顶点替换短边



分段放样

可以将重复的轮廓提供给 **loftProfileBody**，以生成分段放样体。在分段放样过程中，放样操作会在不同的部分中分别完成，从而创建一个在重复轮廓之间非 G1 光滑（non-G1 smooth）的放样体，如下图所示。

图 7-10 重复轮廓分段放样



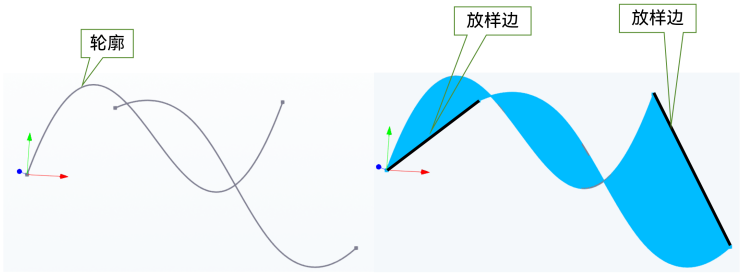
轮廓限制

许多限制适用于所提供的轮廓，不同类型的轮廓有不同的使用限制：

轮廓类型	限制
线框体轮廓	每个轮廓都必须有方位。生成的片状体是按照以下方式构建的：其向外方向（outward normal）为轮廓方向上的切线与放样方向的叉积。为了确保结果不出现扭曲，所有轮廓应具有相同的方向。 每个轮廓至少必须有一个顶点。如果需要，可以使用 **imprintPointOnEdge** 来添加一个顶点。
片状体轮廓	片状体只能有一个层流边界。 轮廓的层流边界必须拥有至少一个顶点。如果需要，请使用 **imprintPointOnEdge** 来添加一个顶点。

您还需要确保生成的主体不会发生自交。例如，当两个轮廓共面且其中一个发生了较大的方向变化时，就可能会发生自交的情况，如下图所示：

图 7-11 由两个共面轮廓创建的自交片状体



7.4.2 放样选项

m_endConditions

m_endConditions 用于控制放样曲面在终止轮廓处的行为。目前只支持 m_periodic，当起始轮廓与终止轮廓相同，并且放样曲面在放样方向上需要具有周期性时，则需要将 m_periodic 设置为 true；当起始轮廓与终止轮廓不同，并且放样曲面在放样方向上不需要具有周期性时，则需要将 m_periodic 设置为 false。

m_verticesMatches

当放样体中的所有轮廓都具有相同数量的边和相同数量的顶点时，顶点会以一一对应的方式映射到各个轮廓上。所有的起始顶点会相互匹配，然后每个轮廓上的后续顶点会按顺序匹配。

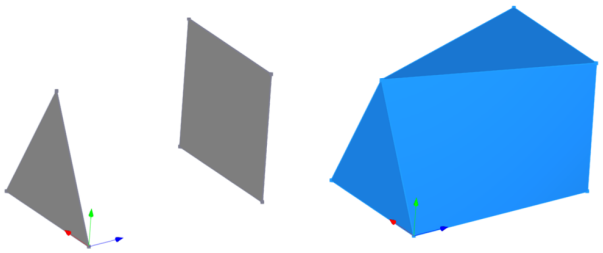
如果轮廓具有不同数量的边，则应用程序需要提供有关如何匹配轮廓的信息。这可以通过以下两种方式之一来完成：

- 使用 m_verticesMatches 提供顶点之间的映射关系。
- 使用 **imprintPointOnEdge** 为轮廓添加额外的顶点。

在提供顶点之间的映射时，则必须遵循以下约束：

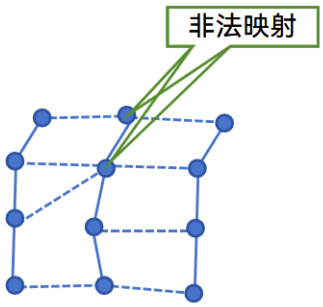
- 顶点必须映射到前一个或后一个轮廓上的顶点。
- 对于非周期性的放样体，起始轮廓上的的顶点必须映射到后一个轮廓上的顶点；终止轮廓上的顶点必须映射到前一个轮廓上的顶点。如下图所示：

图 7-12 合法顶点映射



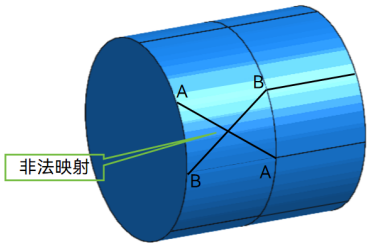
- 对于所有其他轮廓，轮廓上的每个顶点必须恰好映射到当前轮廓的前一个和后一个轮廓上的一个顶点。下图展示了一个非法顶点映射的示例。

图 7-13 非法顶点映射示例一



- 在从一个轮廓过渡到下一个轮廓时，路径不得交叉；如果在一个轮廓上顶点“A”位于顶点“B”之前，那么顶点 A 映射到前一个和后一个轮廓上的顶点也必须位于顶点 B 映射到的顶点之前，下图展示了一个非法顶点映射的示例。

图 7-14 非法顶点映射示例二



8 线框体与片状体

8.1 内容简介

本章介绍了线框体和片状体的创建，以及如何使用线框体和片状体进行建模。同时，也介绍了如何延伸曲面。这些功能有助于在建模过程中实现更加灵活和精准的操作，从而满足不同的设计需求。

本章包含以下章节：

- 8.2 线框体建模：介绍线体建模的相关功能。
- 8.3 片状体建模：介绍片状体建模的相关功能。
- 8.4 延伸曲面：介绍延伸曲面。

图 8-1 连接片状体

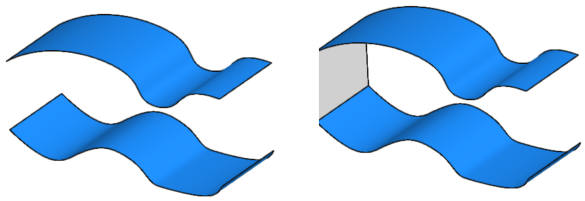
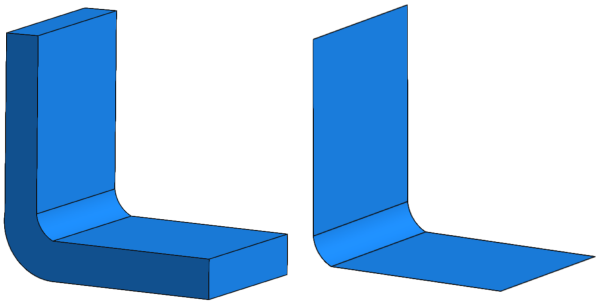


图 8-2 从实心体中创建片状体



8.2 线框体建模

本建模引擎软件提供了用于处理线框体和通过其他实体来创建线框体的功能。具体功能如下：

- 创建与线框体适配的面。
- 偏移平面线框体。
- 改变边的方向。
- 拆分线框体。
- 从曲线中创建线框体。
- 从边中创建线框体。

创建与线框体适配的面

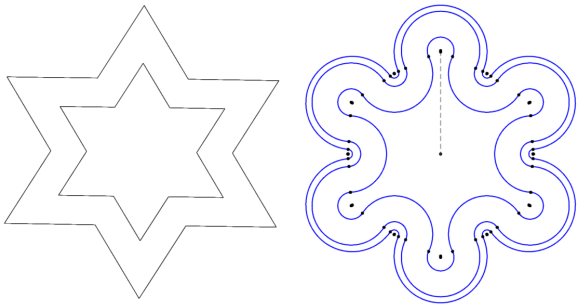
createFacesByWireEdges 根据指定的线框体创建一个面，面的边界由线框体的边所构成。新创建的面没有附加任何曲面，可以使用 **faceAttachSurfaces** 或 **fitSurfToFace** 来为其附加一个曲面。

此接口允许创建带有孔洞的面或具有分离轮廓的主体。新面中的每个环都是由线框体中边的闭合环创建的，您可以通过指定创建多少个面以及指定每个环相对于边的方向来灵活地处理复杂的线框体结构，创建出具有不同拓扑和几何特性的面或体。

偏移平面线框体

您可以使用 **offsetPlanarWireBody** 对平面内的线框体进行偏移，生成新的线框体。这个偏移可以是向内的也可以是向外的，如下图所示：

图 8-3 线框体的偏移



说明
此接口不支持离散几何。

可以通过将线框边的曲线偏移指定距离来创建新的线框体，此过程中在原主体上的顶点可能会被拆分，这会导致在偏移主体中出现间隙，**offsetPlanarWireBody** 提供以下选项来控制创建结果：

- m_gapFill：指定以哪种方式填充间隙：
  - Round：通过延伸圆弧填充。
  - Linear：通过切线延伸填充。
  - Natural：通过延伸自然曲线填充。
- m_localCheck：是否进行局部检查。

- m_tol：将一般曲线转换为 B 样条曲线（B-curves）时所使用的容差。如果此字段设置为零，接口将自动选择一个合适的容差。

offsetPlanarWireBody 通过 track 输出新线框体的边与原始线框体边或顶点（在新边段的情况下）配对的信息。

新边的几何只有在原始边的曲线是直线或圆时，才与原始边的几何相同；否则，新边的几何是近似生成的。您可以使用容差选项 m_tol 来控制近似程度。

线框体上边的方向

可以通过 **reverseEdge** 改变线框体的边的方向，该接口可以反转边以及边上的几何。此接口还支持片状体、实心体、一般实体。

拆分一个线框体

vertexRemoveEdge 可以将线框体拆分为成两个独立的体。此接口需要传入一个顶点和一个要被拆分的边，然后在该边的末端添加一个新顶点，从而在体上创建一个切口来拆分此线框体。

通过曲线创建线框体

createWireBodyByCurves 可以从一组曲线中创建一个线框体。创建的线框体可以是打开的、闭合的和不相交的。此接口的输入参数如下：

- curveTagCount 和 curveTags：用于生成线框体的曲线数量及曲线数组。如果提供的任何曲线来自当前分区以外的分区，则将使用该曲线的副本。
- intervals：区间数组，每个区间都描述了曲线数组中相应曲线的边界。
- options：生成线框体用到的一组选项。

此接口的输出参数如下：

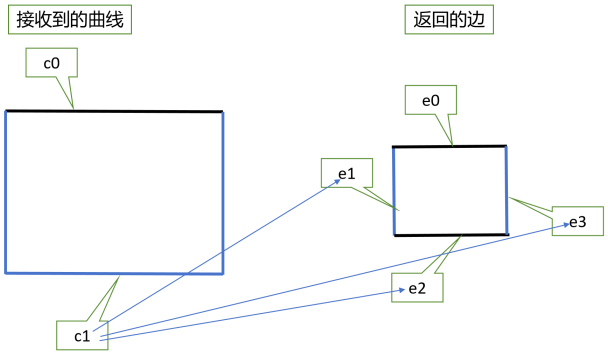
- bodyTag：创建的线框体的标签，该线框体在当前分区中输出，即使某些曲线不是当前分区中的曲线。
- newEdgeTagCount：线框体中的边数和边的数组。

🕮 说明
提供的曲线不能是二维 B 曲线。

将边映射到原始曲线上

edgeIndexes 数组返回原始曲线数组中的曲线与结果体中的边的映射信息。如下图所示：

图 8-4 原始曲线与边的映射



选项

您可以使用 option 来设置相关的创建选项：

- m_version：选项版本号。
- m_tol：曲线的近似容差，两条连接曲线之间的最大距离，默认值：$1.0e^{-6}$。
- m_wantEdges：是否返回新创建的线框中体的边的标签列表（默认值：false）。
- m_wantIndices：是否返回映射关系，如果设置为 false，则不返回 edgeIndexes。newEdgeTags 数组中的边会映射到原始曲线数组中的对应曲线，edgeIndexes 用于保存这些映射关系。

通过边创建线框体

createWireBodyByEdges 通过一组边创建一个线框体。所提供的边可以来自不同的模型，但它们不能相交或重合。此接口的输入参数如下：

- edgeTagCount：需要输入的边的数量。
- edgeTags：边的标签列表。
- option：用于创建线框体的选项。

createWireBodyByEdges 的输出参数如下：

- bodyTag：所创建的线框体的标签。

如果建模引擎软件启用了一般拓扑，则 **createWireBodyByEdges** 可以创建一个不相交的线框体。

选项

您可以使用 options 来设置相关的创建选项：

- m_version：选项版本号。
- m_allowDisjoint：是否可以创建不相交的主体。
- m_copyDepGeometry：复制曲线的时候是否把依赖几何也一起复制。
- m_tol：曲线近似时使用的容差。默认值：$1.0e^{-5}$。

8.3 片状体建模

本建模引擎软件提供了以下片状体建模功能：

- 创建片状体。
- 裁剪片状体。

裁剪片状体时通常采用布尔运算，裁剪多面片状体（multifaced sheet body）时则通过压印实体然后根据压印产生的边来进行裁剪。

创建片状体

您可以使用 **createSheetBodiesByFaces** 从一个面集中创建一个或多个片状体。此接口包含以下选项：

- m_version：选项版本号。
- m_allowDisjoint：将不相交部件作为单个不相交主体输出。默认值：false。
- m_makeFrom：是通过输入的面还是输入面的副本创建片状体。
- m_transferAttribs：片状体是否从创建其所用的拓扑中继承属性。
- m_groupTransfer：否将指定的组传输或复制到新的片状体中。
- m_trackEdges：是否在跟踪数据中返回创建片状体时生成的边。
- m_trackVertices：是否在跟踪数据中返回创建片状体时生成的顶点。

说明
如果您向 **createSheetBodiesByFaces** 提供来自不同实体的面，您需要确保它们不相交。如果提供的面相交，**createSheetBodiesByFaces** 可能会创建一个无效的片状体。

裁剪片状体

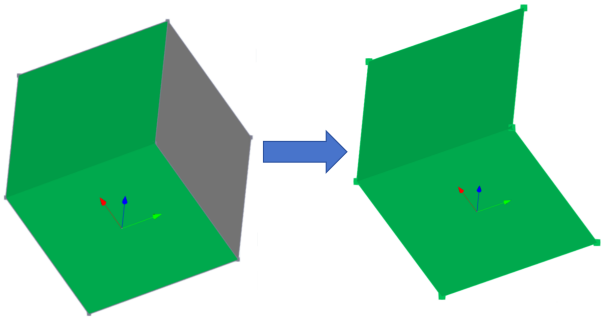
可以使用 **trimBody** 来裁剪主体。为保证裁剪成功，需要对主体上的面进行分组（至少分为两组）。

调用此接口时需要输入以下参数：

- sheetBodyTag：要裁剪的片状体标签。
- edgeTagCount：边的数量，这些边用于定义片状体上的区域。
- edgeTags：边的标签列表。
- faceTagCount：面的数量，这些边用于指示裁剪操作要保留或丢弃的区域。
- faceTags：面的标签列表，与 isKeep 配合使用。
- isKeep：用于指示保留哪些面集。当其为 true 时，只保留包含 faceTags 中的面的面集，删除所有其他面集；当其为 false 时，删除包含 faceTags 中的面的面集，保留所有其他面集。

图 8-5 裁剪片状体为裁剪片状体的一个示例，其中被裁剪的片状体为灰色，裁剪时要保留的部分为绿色。

图 8-5 裁剪片状体



8.4 延伸曲面

extendSurface 可以延伸除外来几何曲面之外的所有曲面。此接口接收以下参数：

- surfTag：待修改曲面的标签。
- option：延伸选项，用于指定所需的延伸类型。

接口执行完毕后，会返回一个状态标志，用于指示操作的结果。

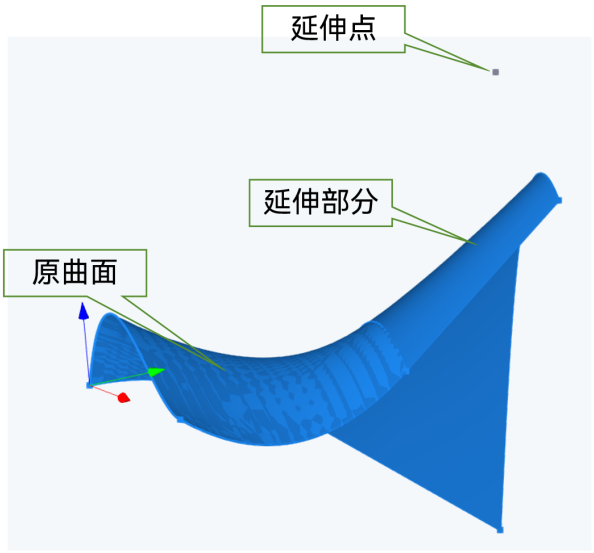
说明
此接口为离散几何提供部分支持。

可以通过以下四种方式来延伸曲面：

- 延伸到一个指定的点。
- 延伸到一个指定的包围盒。
- 延伸到一个指定的参数空间包围盒。
- 按照当前参数范围的比例延伸参数边界。

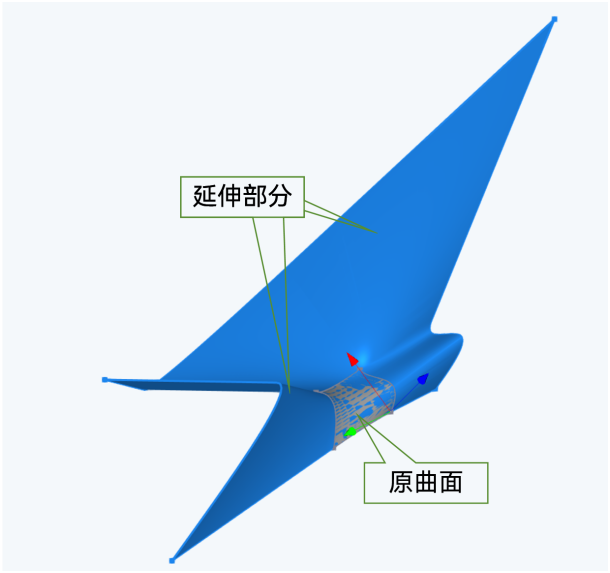
将曲面延伸到一个指定的顶点。

图 8-6 延伸到一个指定的顶点



将曲面延伸到一个指定的包围盒。

图 8-7 延伸到一个指定的包围盒



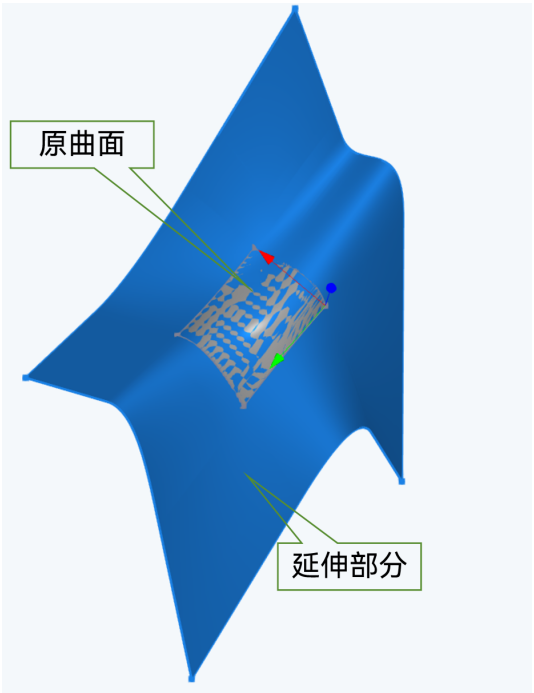
可以通过 option 中的字段来选择延伸方式，option 中包含的字段如下：

- m_version：选项版本号。
- m_type：用于指定延伸类型，可以取以下值：
  - None：未要求延伸。
  - Point：延伸到一个点。
  - Box：延伸到一个包围盒。
  - UVBox：延伸到一个参数空间包围盒。
  - Ratio：按照当前参数范围的比例延伸参数边界。

- m_vector：要延伸到的点。如果指定了要延伸到一个点，则忽略其他用于延伸的参数。
- m_box：要延伸到的包围盒，如果指定了要延伸到一个包围盒，则忽略其他用于延伸的参数。
- m_uvBox：要延伸到的参数空间包围盒，如果指定了要延伸到一个参数空间包围盒，则忽略其他用于延伸的参数。
- m_minURatio：U 参数范围的比例，用于曲面低 U 边界的延伸，如果指定了要使用比例延伸，则忽略其他用于延伸的参数。
- m_maxURatio：U 参数范围的比例，用于曲面高 U 边界的延伸，如果指定了要使用比例延伸，则忽略其他用于延伸的参数。
- m_minVRatio：V 参数范围的比例，用于曲面低 V 边界的延伸，如果指定了要使用比例延伸，则忽略其他用于延伸的参数。
- m_maxVRatio：V 参数范围的比例，用于曲面高 V 边界的延伸，如果指定了要使用比例延伸，则忽略其他用于延伸的参数。
- m_allowPartialExtension：在完全延伸会导致曲面无效的情况下，是否需要在保持曲面有效的情况下进行部分延伸。
- m_shape：指定要使用的延伸形状。

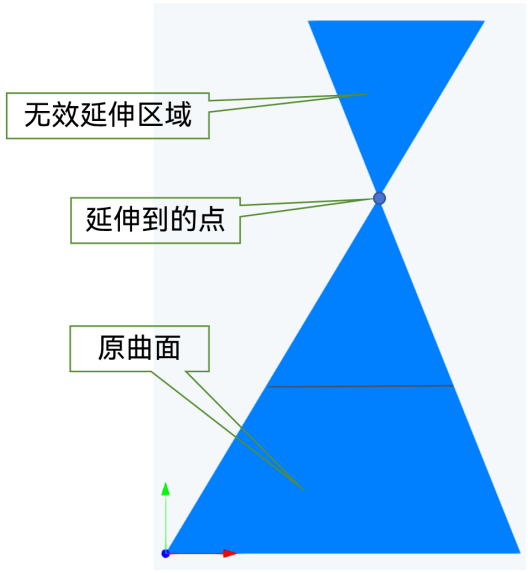
对于每个 ratio 字段，比例为 1.0 表示将相关边界扩展 100%。下图展示通过不同比例值延伸不同边界的效果：

图 8-8 通过比例延伸曲面



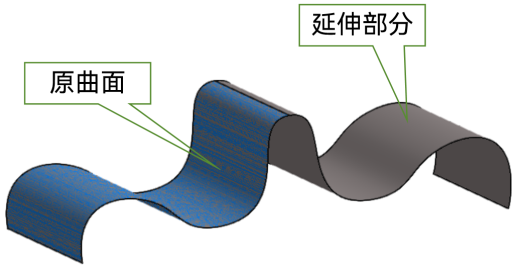
m_allowPartialExtension 的作用如下图所示，将指定的曲面延伸到所示的点会导致曲面自交，从而生成一个无效的曲面。如果 m_allowPartialExtension 设置为 false，则不执行延伸；如果 m_allowPartialExtension 设置为 true，曲面将延伸到自交点。

图 8-9 局部延伸的曲面



PSGMApiExtendSurfType 取 None 时，将使用自然延伸。

图 8-10 自然延伸



状态标志

extendSurface 返回一个状态标志，用于指示操作的结果。该状态标志有以下值：

- Ok：延伸成功。
- Unextended：无需延伸。
- Partial：仅执行了部分延伸。
- Invalid：延伸失败，导致曲面无效。
- Failure：内部算法故障。

9 拓扑操作

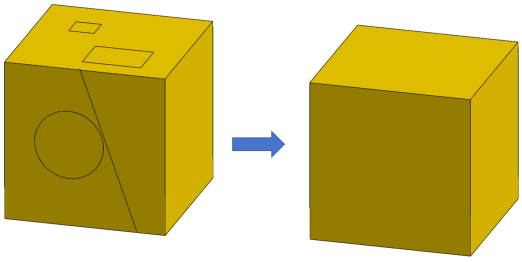
9.1 内容简介

本章将介绍本建模引擎软件中操作模型的拓扑结构的功能，主要包含：欧拉操作，识别和删除冗余拓扑，以及拆分拓扑。

本章包含以下章节：

- 9.2 欧拉操作：介绍小范围修改拓扑结构的低级接口。
- 9.3 管理冗余拓扑：介绍识别和删除一个完整的模型定义中的冗余拓扑。
- 9.4 拆分拓扑：介绍拆分拓扑结构。

图 9-1 删除冗余拓扑



9.2 欧拉操作

欧拉操作是一种可以修改小范围拓扑结构的低级接口，可以与附加和分离几何的接口一起使用。

通过欧拉操作创建的拓扑不会附带任何几何数据，是无效拓扑，所以需要手动为其附加有效几何数据。通过欧拉操作删除拓扑时，首先会删除它的几何。

说明
欧拉操作与建模引擎软件其他功能不同的是欧拉操作可能会产生无效主体。

欧拉操作会经常用到边和顶点。

边

边有以下类型：

- 线框边：没有半边。
- 层流边：有一条半边。
- 流形边：有两条方向相反的半边。
- 一般边：有两条方向相同的半边。
- 开放边：有两个顶点。
- 开口环边（split ring）：有一个顶点。
- 环边（ring）：没有顶点。

顶点

在某个顶点处的定向边数量可以使用 **vertexGetOrientedEdges** 来查询，没有连接定向边的顶点被成为孤立顶点。

当顶点满足以下条件之一时为流形顶点：

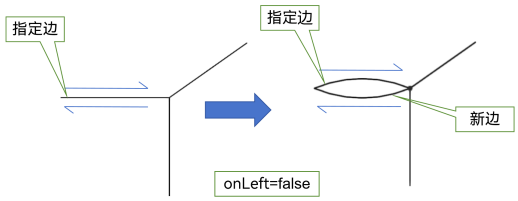
- 连接到一个孤立环。
- 未连接孤立环，与其连接的边都是流形边且这些边能够形成单个面连接集合，即在该顶点处，边界表示（B-rep）是局部流形的。

以下将介绍欧拉操作提供的接口及接口的作用。

edgeSlit

此接口纵向切割（slit）一条边，创建一个新的切口面。onLeft 参数指定新切口面是位于指定边的左侧还是右侧。

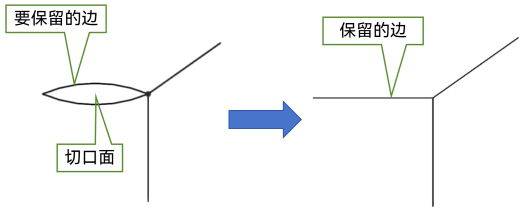
图 9-2 纵向切割边



faceUnslit

此接口用于删除一个切口面，要删除的切口面必须有一个环和两条边，或者有两个环，每个环都有一条环边。

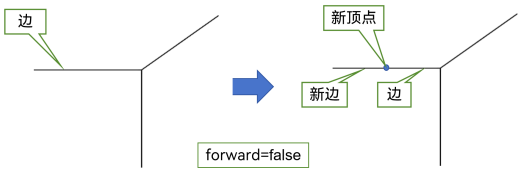
图 9-3 删除切口面



edgeSplit

此接口通过添加一个顶点来拆分（split）一条边。参数 forward 用于确定新顶点是边的前向顶点还是后向顶点。

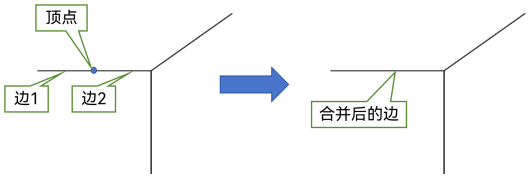
图 9-4 拆分一条边



vertexMergeEdges

通过从一条边中删除顶点来合并一条边。

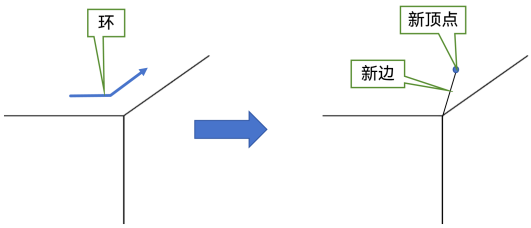
图 9-5 合并一条边



loopMakeEdge

在指定的环中添加一条后沿边（trailing edge）和一个顶点。使用的顶点是所提供半边的前向顶点（forward vertex）。新添加的边的方向朝向新添加的顶点。

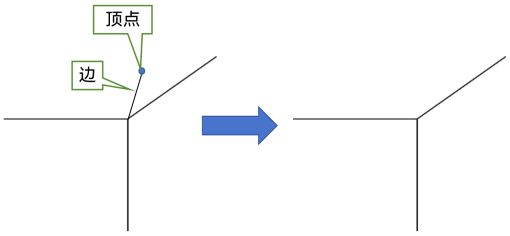
图 9-6 为指定环添加后沿边



vertexDelete

删除指定顶点以及与之相连的边。

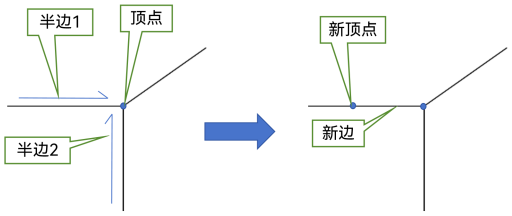
图 9-7 删除顶点以及与之连接的边



vertexSplit

拆分一个顶点，并在原顶点和新顶点之间添加一条边。该顶点必须是提供给接口的两个半边的共同前向顶点。

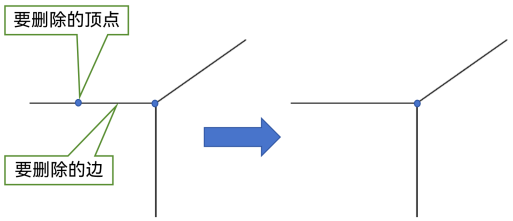
图 9-8 拆分一个顶点



edgeMergeVertices

合并两个顶点，并删除两个顶点之间的边。

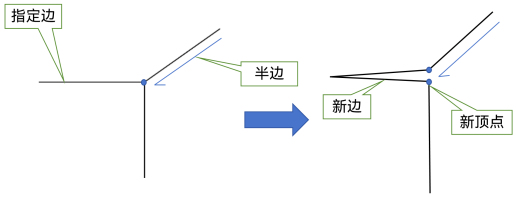
图 9-9 合并两个顶点



edgeOpenZip

从一条边的一端进行纵向拆分，拆分为两条边，这两条边在另一端相连。原边的顶点为所提供的半边的前向顶点。

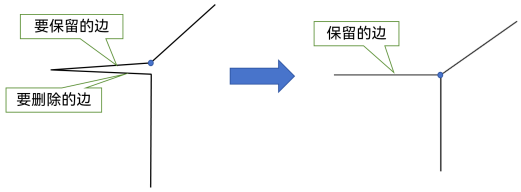
图 9-10 将一条边横向拆分为两条边



edgeCloseZip

把拆分出来的两条边进行合并，通过删除一条边，保留一条边的方式，把它们合并在一起。同时需要确保涉及到合并的顶点上有一条以上的边。

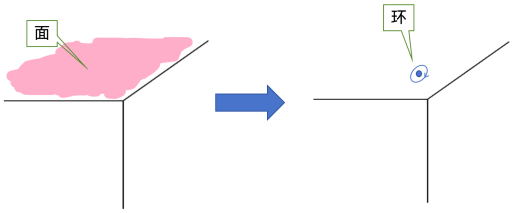
图 9-11 合并两条边



faceMakeLoop

向一个面添加一个孤立顶点和一个环。

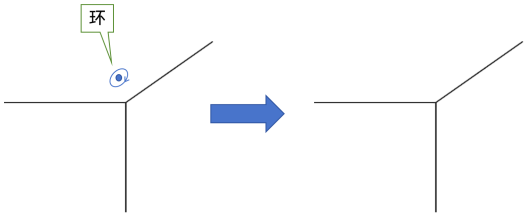
图 9-12 向一个面添加一个孤立顶点和一个环



loopDeleteIsolated

删除一个面上的一个孤立顶点和一个环。

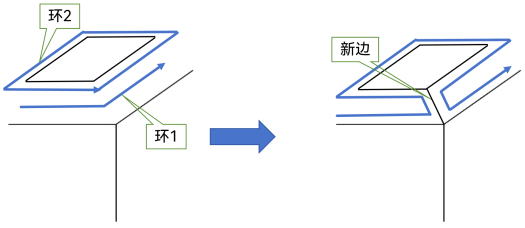
图 9-13 删除一个面上的一个孤立顶点和一个环



loopDeleteMakeEdge

将位于同一面上的两个环合并成一个，通过一条边将它们的顶点连接起来。在此过程中，“环 2”将被删除。

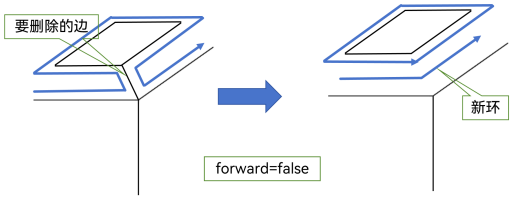
图 9-14 合并一个面上的两个环



edgeDeleteMakeLoop

从一个环中删除一条边，并将其分割为两个环。由逻辑前向决定哪个环是新的环。

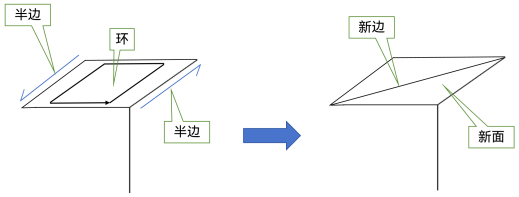
图 9-15 通过删除边来创建新的环



loopMakeEdgeFace

通过连接一个环中的两个顶点来创建一个新的面。新面在新边的右边。顶点为所提供半边的前向顶点。

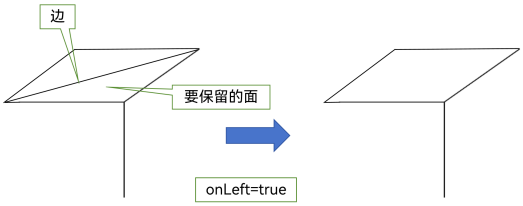
图 9-16 通过连接环中的两个顶点创建新的面



edgeDeleteWithFace

通过删除一条边，将两个面和环合并为一个面和一个环。参数 onLeft 决定保留哪个面。

图 9-17 通过删除边合并环与面



loopMakeEdgeLoop

此接口类似于 **loopMakeEdgeFace**，但它没有把面一分为二，而是在面上创建一个新的环，增加面的亏格（genus）。

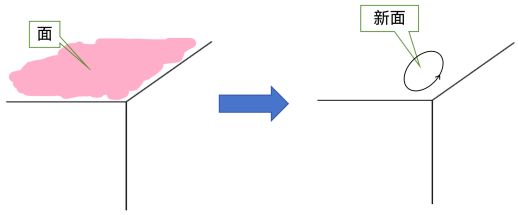
edgeDeleteWithLoop

此接口与 **edgeDeleteWithFace** 类似，只是它删除了一条边，该边的每个面都有不同的环。它将这些环合并成一个环，减少面的亏格。

faceMakeRingFace

此接口通过在面上添加一个环边来创建新的面。新的面总是在边的右边。

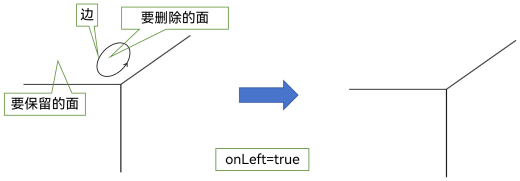
图 9-18 通过添加一个环边来拆分面



edgeDeleteRingFace

此接口用于删除一个环边和一个面。

图 9-19 删除一个环边和一个面



faceMakeRingLoop

此接口类似于 **faceMakeRingFace**，它不是把面一分为二，而是在面上创建了一个新的环，从而增加了它的亏格。

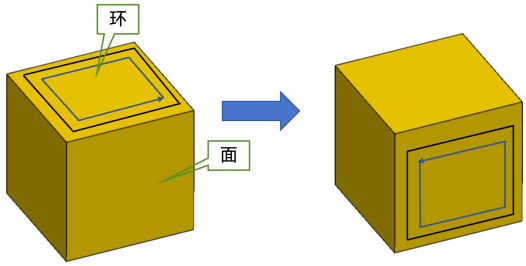
edgeDeleteRingLoop

此接口类似于 **edgeDeleteWithFace** ，它删除一个环边（ring edge），该环边在两侧分别属于同一面上的不同环。此操作会减少该面的亏格。

loopTransfer

此接口将一个环从一个面转移到另一个面。

图 9-20 将一个环转移到另一个面



coedgeGlue

此接口通过合并两个边的半边来连接边。传递给接口的第一个半边的边会保留下来，而另一个边则会被删除。

与其他欧拉操作的接口不同，**coedgeGlue** 在删除某个边时，并不会删除附着在该边上的曲线。

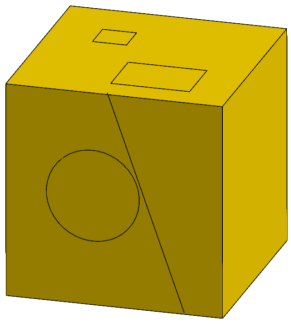
9.3 管理冗余拓扑

主体中有时可能包含一些完整模型定义不需要的边或顶点，这样的拓扑结构被称为冗余拓扑（redundant topology）。冗余拓扑可能包括以下几种情况：

- 附着在分裂环边上的顶点。
- 拆分两条边的的顶点，这两条边共享同一曲线。
- 线框边。
- 拆分两个面的边，这两个面共享同一曲面。

在将数据导出到另一个应用程序之前，通常需要处理冗余拓扑。本建模引擎软件为您提供了从实体中识别和删除冗余拓扑的功能。下图展示了主体上冗余拓扑的一些简单示例。

图 9-21 主体上的冗余拓扑



您可以使用以下接口来管理主体中的冗余拓扑：

接口	描述
identifyRedundant	识别主体中存在的冗余拓扑结构。更多内容请参见 9.3.1 识别冗余拓扑。
deleteRedundant	从一个模型中识别冗余拓扑并删除，然后合并可合并的实体。更多内容请参见 9.3.2 删除冗余拓扑。

9.3.1 识别冗余拓扑

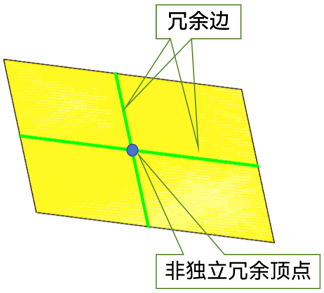
如果您需要识别出模型中的冗余拓扑，可以使用 **identifyRedundant**。此接口接收一组拓扑和一组选项，返回被发现的冗余拓扑。您可以将冗余属性附加到被发现的拓扑实体上以便后续操作。

选项中包含以下字段：

- m_version：选项版本号。用于控制选项的版本。

- m_maxTopoDimension：冗余拓扑的识别维度。例如，从主体中识别冗余的顶点时，需要将 m_maxTopoDimension 设置为 Zero。
- m_scope：冗余拓扑的识别范围。例如，需要识别主体边界上的冗余拓扑是，可以将 m_scope 设置为 On。
- m_propagateRedundancy：是否识别依赖冗余拓扑，即该拓扑本身不是冗余的，删除其他冗余拓扑会使其变为冗余的拓扑。下图中冗余边相交处的顶点并不是冗余的，如果删除冗余边，此顶点将冗余。

图 9-22 依赖冗余拓扑



9.3.2 删除冗余拓扑

deleteRedundant 接收一组拓扑和一组选项。它删除被发现的冗余拓扑，合并删除冗余拓扑后的其他可合并拓扑。您可以使用以下选项来控制此接口的行为：

- m_version：选项版本号。
- m_maxTopoDimension：冗余拓扑的删除维度。
- m_scope：冗余拓扑的删除范围。
- m_protectedTopoTagCount：受保护的拓扑数量，即操作期间不会被删除的拓扑。
- m_protectedTopoTags：受保护的拓扑数组。

m_scope

m_scope 选项可以用来控制删除的拓扑的数量，下图展示了不同的范围值对应的操作结果。

- In：对模型的影响最小，只有当删除某个拓扑元素后，不会影响到那些未在拓扑列表中明确指定的关联拓扑时，这个拓扑元素才会被删除。在这种情况下，非层流边界上的冗余顶点不会被删除，因为它们具有未在拓扑列表中指定的关联拓扑。
- On：默认范围值，在删除冗余拓扑时，移除指定拓扑内的冗余元素，同时避免与相邻拓扑合并。层流边界和非层流边界上的冗余顶点也会被删除，但会保留必要的内部边以防止合并。这种处理方式旨在优化模型拓扑结构，同时保持模型的完整性和稳定性。
- Out：对模型的影响最大，在删除冗余拓扑时，即使这会导致指定的拓扑与相邻拓扑合并，也会进行删除操作。不仅指定拓扑内部的冗余拓扑会被移除，层流边界上和非层流边界上的所有冗余顶点也会被删除，并且冗余的非层流边界也会被删除，导致相应的面合并。

不同范围值会产生不同的结果：

图 9-23 m_scope=In

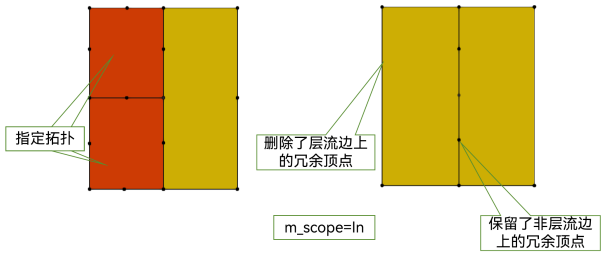


图 9-24 m_scope=On

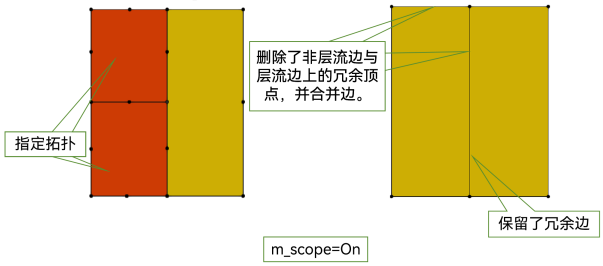
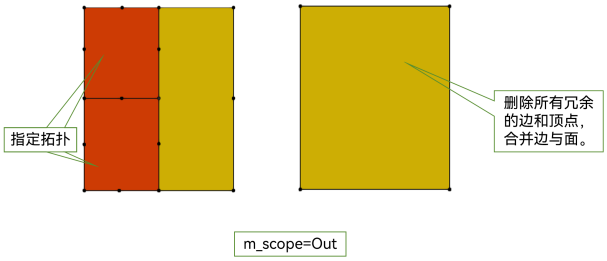


图 9-25 m_scope=Out



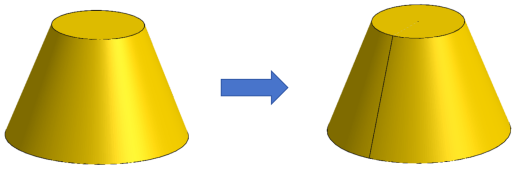
9.4 拆分拓扑

以下接口用于拆分拓扑实体：

接口	描述
edgeSplitAtParam	在指定的参数处拆分一条边。
faceSplitAtParam	沿着一条常量参数线拆分一个面。

您可以使用这些接口在模型中创建新的边或面。例如，默认情况下，本建模引擎软件不会在周期边界上拆分拓扑，比如：圆柱面不会有接缝。您可以调用 **faceSplitAtParam** 来创建一个接缝，如下图所示。

图 9-26 通过拆分拓扑创建的接缝



在指定的参数处拆分一条边

需要为 **edgeSplitAtParam** 输入以下参数：

- edgeTag：要拆分的边。
- param：用于拆分边的参数。您可以使用 **edgeGetGeometry** 找到参数范围。

edgeSplitAtParam 输出以下参数：

- newVertexTag：通过拆分边创建的新顶点的标签。
- newEdgeTag：通过拆分边创建的新边的标签。

edgeSplitAtParam 可以拆分容差边和精确边。为了确保拆分成功，以下条件必须均为真：

- param 在边的范围内。
- param 不与边上的顶点重合。
- 边上有几何附着。

沿常量参数线拆分面

需要为 **faceSplitAtParam** 输入以下参数：

- faceTag：要拆分的面。
- param：用于拆分面的参数。
- paramDir：面的拆分方向。
- option：拆分面的选项。

faceSplitAtParam 输出以下参数：

- newEdgeTagCount
- newEdgeTags
- newFaceTagCount
- newFaceTags

说明
此接口不支持离散几何。

---

10 压印与布尔

10.1 内容简介

本建模软件提供了压印功能和布尔运算功能，这些功能可以用来比较或组合实体。本章介绍了如何将实体压印到片状体上，并描述了用于创建流形体和一般体的布尔操作。

本章包含以下内容：

- 10.2 压印：介绍如何将曲线、面和平面压印到片状体。
- 10.3 布尔运算：介绍布尔运算。
- 10.4 阵列：介绍在如何模型上快速创建阵列。
- 10.5 分割：介绍如何通过曲面、面或片状体将主体切割成不同部分。
- 10.6 求交：介绍如何使用求交。

10.2 压印

您可以使用本建模引擎软件的压印功能，将实体压印到主体、片状体、曲面或一组面上，也可以使用投影功能将实体投影到主体、片状体、曲面或一组面上。

10.2.1 曲线压印和曲线投影

可以通过 **projectCurves** 进行法线投影。生成的投影曲线和点是孤立几何。

接收参数

projectCurves 接收以下参数：

- curveTagCount：工具曲线的数量。
- curveTags：工具曲线的标签列表。它不能包含自交曲线或外部几何。
- intervals：曲线区间的数组。
- targetEntityTagCount：目标实体的数量。
- targetEntityTags：目标实体的标签列表。
- option：一组用于控制曲线投影或压印结果的选项。

输出参数

projectCurves 输出以下参数：

- result：生成的投影几何实体。

选项摘要

以下是 **projectCurves** 中提供的选项。

- m_method：投影方法。

曲线的投影方法

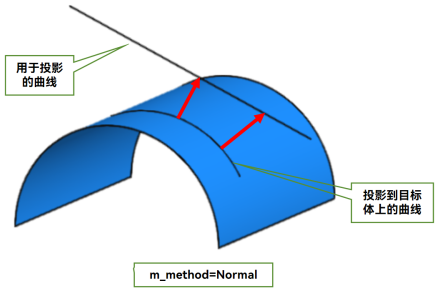
可以通过 m_method 来选择投影方式，投影方式有：法线投影。

法线投影

要使用法线投影，m_method 必须设置为：Normal。

虽然法线投影是沿着面的法线方向进行的，但它们实际上是双向的。因此，如果投影曲线位于面的法线场内但位于其后方，它们也会被投影到目标实体上。法线投影适合用于将曲线投影到与之紧密相关且相邻的面上。

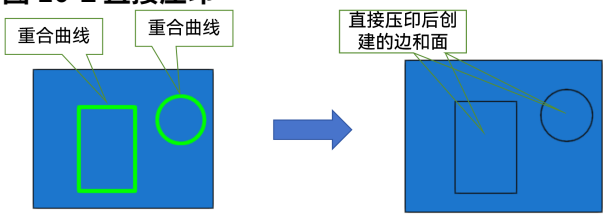
图 10-1 法线投影



10.2.2 直接压印

本建模引擎软件提供了 **imprintCurvesOnFace** 接口来实现直接压印功能。直接压印要求被压印的曲线集合与它们要压印到的面的曲面重合。直接压印的使用场景：通过在平面片状体上压印一组平面曲线来创建一组轮廓。

图 10-2 直接压印



10.3 布尔运算

10.3.1 布尔运算简介

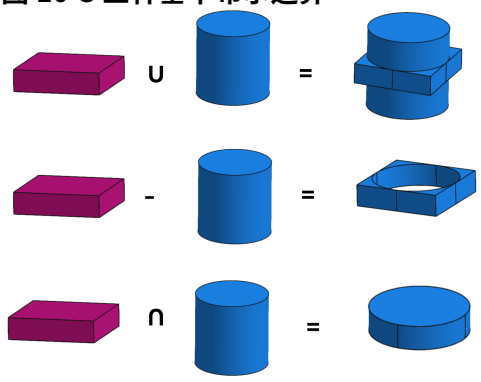
本章介绍建模引擎软件中可用的布尔运算。

对一般体或流形体进行布尔运算时，如果产生了一般体，则为一般布尔运算，如果产生了流行体则为流形布尔运算。如果在会话中禁用一般拓扑，则只能流形布尔运算，这种类型的布尔运算的输入必须是流形体，产生结果体也是流形体。

在执行布尔运算时，建议将局部坐标系与本建模引擎软件的坐标系对齐。

此功能对离散几何体提供部分支持。具体内容请参见*《Geoshape 几何建模引擎软件接口开发文档》*。

图 10-3 三种基本布尔运算



10.3.2 术语介绍

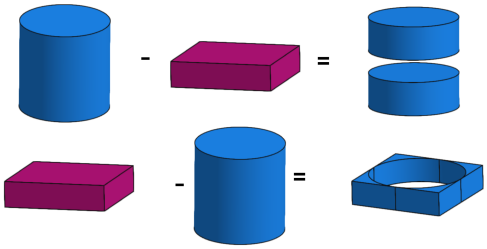
本节介绍在本建模引擎软件的布尔运算中使用的术语。

10.3.2.1 目标体和工具体

布尔运算的操作对象称为目标体和工具体。布尔值的目标体是开始使用的主体，工具体是操作目标体的主体。工具体修改目标体，在布尔运算结束时删除工具体。本建模引擎软件会尽可能保留目标体中实体的标记。

下图显示了在执行布尔减时交换目标体和工具体产生的不同效果：

图 10-4 布尔减的目标体、工具体和结果体



说明

如果使用多个工具体对单个目标体执行布尔运算，建议每次使用一个工具体对目标体进行操作。

10.3.2.2 布尔操作期间的标签持久化

标签中描述的标签持久化规则也适用于布尔操作。如果注册并使用普通属性回调，则禁用标记持久化。

10.3.2.3 全局布尔运算

全局布尔运算会比较目标体和工具体的所有面。

在使用 **booleanBodies** 执行全局布尔运算时，需要提供一个目标体、一个或多个工具体和一个选项结构。如果提供了多个工具体，则首先将具有重叠部分的工具体进行合并，然后执行目标体和工具体之间的布尔运算。对于相交布尔运算（intersect boolean operation），一个目标体和两个工具体的交集是目标体与工具体合并结果的交集，而不是三个主体之间的相互交集。

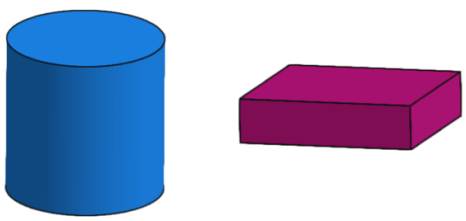
在执行布尔运算时需要确保目标体和工具体中没有橡子顶点（acorn vertex），目标体和工具体可以是流形实心体、片状体、线框体、一般体。

10.3.2.4 边界区域

在执行布尔运算时，会将边压印在目标体和工具体上以指出两个实体的相交部分。这些压印的边将实体的边界划分为边界区域（boundary region）。

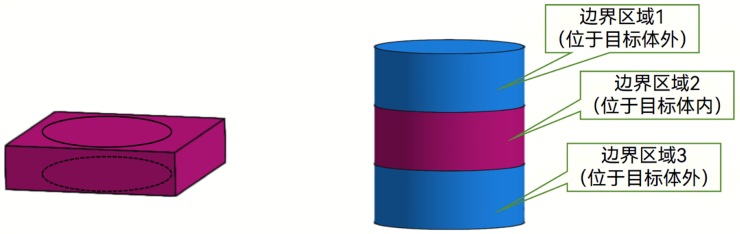
下图展示了后续执行全局布尔并运算时用到的主体。

图 10-5 后续布尔并运算使用的主体



对于全局布尔运算，工具体的所有边界区域要么完全位于目标体内部，要么完全位于目标主体外部，如下图所示。

图 10-6 全局布尔运算的边界区域



10.3.3 返回信息

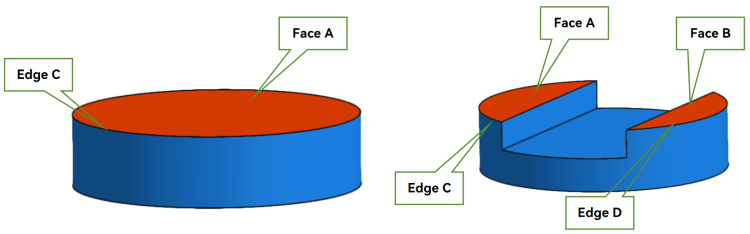
PSGMApiBoolResultType 会返回关于布尔操作的总体成功或失败的状态。

值	描述
Success	布尔运算执行成功，结果体在主体数组中返回。
NoClash	布尔运算执行成功，但目标体和工具体并没有相互作用： ● 合并：工具体完全在目标体之外，所得到的结果体是不相交的主体。 ● 相交：工具体和目标体都被删除，没有可返回的主体。
Failed	布尔运算执行失败。

10.3.4 布尔运算中的共享几何

下图展示了一个使用布尔运算从一个立方体中减去一个槽（slot）的例子。在这个例子中，“Face A”被分割，创建了新的面“Face B”，“Face A”和“Face B”两个面共享原始曲面。同样，边“Edge C”和“Edge D”共享同一条曲线。

图 10-7 两个面共享同一个曲面



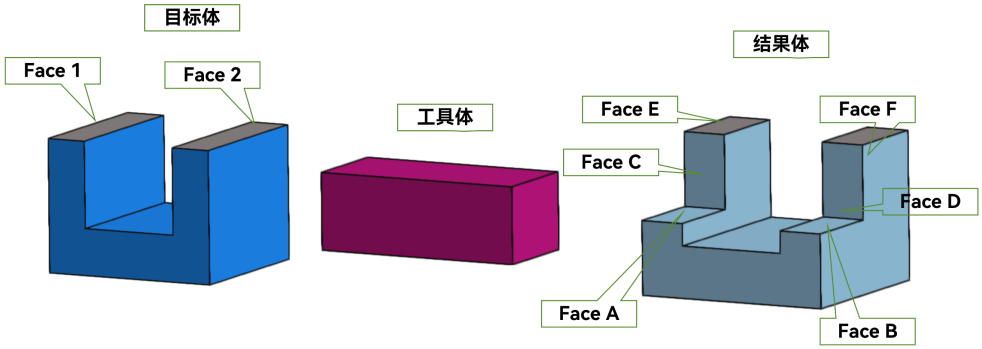
说明

如果上图的例子使用了离散几何，那么面“Face A”和“Face B”将不会共享曲面，而是会有单独的网格，边“Face C”和“Face D”将有单独的线。

10.3.4.1 布尔减导致的共享几何

在下图中“Face E”与面“Face 1”具有相同的曲面，面“Face F”与“Face 2”具有相同的曲面。因此，如果“Face 1”和“Face 2”从一开始就共享曲面，那么“Face E”和面“Face F”在结果体中共享曲面。在布尔减执行后，“Face A”和“Face B”共享工具体底部的曲面，“Face C”和“Face D”共享工具体背部的曲面。

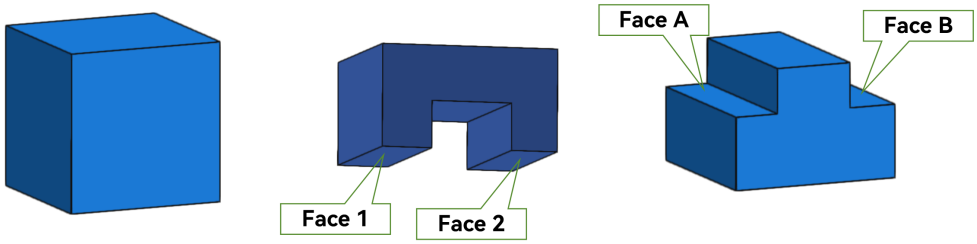
图 10-8 从目标体中减去工具体



10.3.4.2 布尔减保留的共享几何

下图中的例子对调了前一个例子中的目标体和结果体。在这个例子中，“Face A”和“Face B”是否共享曲面由面“Face 1”和“Face 2”决定; 如果面“Face 1”和“Face 2”共享曲面，那么“Face A”和“Face B”也共享曲面。

图 10-9 从目标体中减去工具体



10.3.5 流行体的布尔运算

本节将介绍更多关于使用流行体执行布尔运算的具体信息。

流行布尔运算是在会话中禁用一般拓扑时执行的布尔运算。全局和局部布尔操作都可以尝试使用流形拓扑。

流形布尔运算的类型如下：

类型	描述
布尔并（Union）	通过将位于目标体外部的工具体的所有面粘合（glue）到目标体上来延伸目标体。
布尔减（Subtraction）	通过移除目标体与工具体中重叠的面来修改目标体。
布尔交（Intersection）	只保留目标体和工具体中重叠的面。

10.3.5.1 布尔运算使用建议

当对流形体执行布尔运算时，建议执行以下操作：

- 将局部坐标系与本建模引擎软件坐标系对齐。
- 如果可以，建议使用多工具体布尔运算（a multi-tool boolean）而不是多个单独的工具体（multiple single tools）。
- 在执行布尔并和布尔减时使用大号工具体。

10.4 阵列

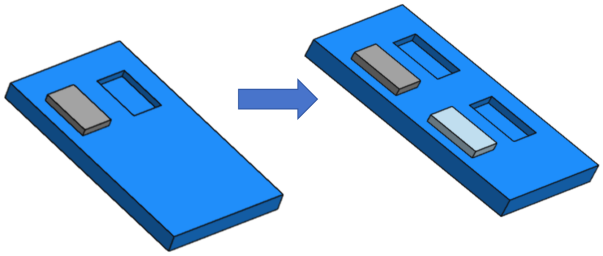
阵列（pattern）与布尔操作有点类似，但是它只在单个主体上操作。

说明
此功能不支持离散几何。

patternFaces 接收一组用于定义特征的面和一组变换。每个变换都被应用于构成特征的面的副本，以创建特征的新实例。通常，**patternFaces** 假设面的变换间隔足够远，新实例之间不会发生碰撞。也可以选择使用诊断来帮助应用程序识别实例之间的是否发生了碰撞。这些选项将在本节中进一步讨论。

下图展示了一个简单的阵列示例。提供的面为凹槽特征和按钮特征的中的面。只应用了一个变换，结果展示在右侧。此示例不能由单个布尔运算完成，因为这会同时进行布尔加和布尔减操作。

图 10-10 一个凹槽和一个凸台的阵列示例

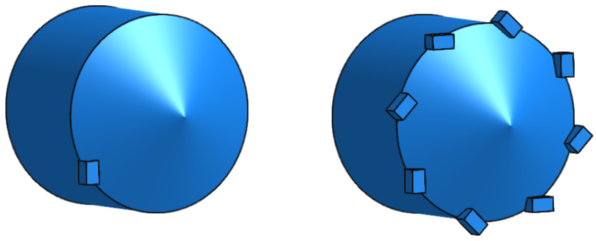


可被阵列的特征有类型限制，与可被实例化的特征的限制一样。

特征可以是不连贯的、开放的、闭合的，可能来自实心体、片状体、一般体，可能位于目标体的一个或多个面上。

阵列功能只能复制所提供的面，完成有限的拓扑变换。阵列操作通过分割目标体上的目标面和边来完成。有关拓扑更改限制的更多详细说明，请参见*《Geoshape 几何建模引擎软件 接口开发文档》*。

图 10-11 涉及常量拓扑和常量几何的阵列示例



阵列有以下选项：

选项	描述
m_sameFace	控制边界环在变换后是否必须保留在同一个面上。如果选择新的特征实例与原始特征在相同的面上，可提高阵列操作性能。更多内容请参见 10.4.1 提高阵列操作性能。
m_conicFace	控制边界环在变换后是否必须与面保持重合。转换后的特征的边界环是否完全位于它们各自的目标面上。

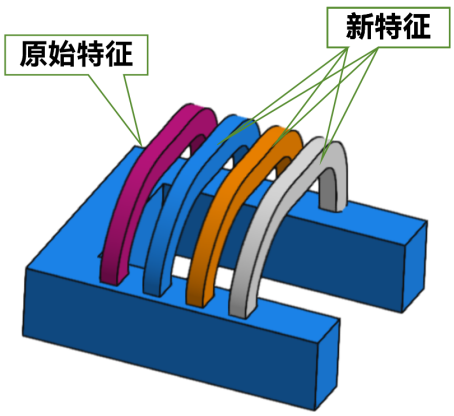
需要根据目标体和工具体的实际情况来设置 m_sameFace 和 m_conicFace，错误设置会导致阵列操作失败和主体损坏。

10.4.1 提高阵列操作性能

当特征的新实例的所有边界环都需要位于原始特征所在的面内时，可以使用 m_sameFace 选项来提高阵列操作的性能。

如果 m_sameFace 为 Yes，则只需要在包含原始特征的面上创建新实例，阵列操作的性能就会得到改善。如下图所示：

图 10-12 新实例位于与原始特征相同的面中



10.4.2 返回的信息

在阵列操作执行完成后会返回一个结果结构体 result 其中 m_type 用来返回操作的执行状态，有以下值：

- Ok：执行过程中没有出现错误。
- Fail：检测到执行过程中出现错误，并且没有创建新的实例。

如果实例创建成功，则会返回新实例的数量和新实例的数据。

10.5 分割

分割操作（sectioning operations）支持具有曲面的圆柱、平面和片状体。分割操作类似于布尔减法，不同的是分割操作接收的选项和布尔操作略有差异，返回的信息也比布尔操作多。分割操作的标记持久性的规则与布尔操作的规则相同。

本建模引擎软件支持全局分割操作和局部分割操作。这些内容将在本章的其余部分中进行描述。

说明
分割操作为离散几何提供部分支持。

10.5.1 全局分割

全局分割会比较目标实体和工具实体之间的所有面对。这些操作由 **sectionBodyBySheet** 和 **sectionBodyBySurface** 提供支持，需要为这些接口提供一个目标体、一个工具体和一组选项。

用曲面进行分割

sectionBodyBySurface 支持用曲面分割具有平面曲面或圆柱曲面的主体。要分割的目标体可以是实心体。

此接口接收一组选项，其中 m_fence 用来指定返回分割工具体前侧、后侧或两侧的结果实体。

如果启用了一般拓扑，那么目标体可能是一个一般实体。在无法成功分割的情况下，错误信息保存在日志报告中。有关更多细节，请参见*《Geoshape 几何建模引擎软件接口开发文档》*。

用片状体进行分割

sectionBodyBySheet 支持用片状体分割主体。要分割的目标体可以是实心体，也可以是片状体。分割完成后，将删除用于分割的片状体。

sectionBodyBySheet 接收一组选项，其中 m_fence 用来指定返回分割工具体前侧、后侧或两侧的结果实体。

如果启用了一般拓扑，则目标体和结果体可能是一般体，工具体必须是一般体。如果无法将结果体分为分割工具体前侧或后侧部分，将生成 PSGMApiSectReport 类型的报告来记录此信息。有关更多细节，请参见*《Geoshape 几何建模引擎软件 接口开发文档》*。

全局分割选项

全局分割有以下选项：

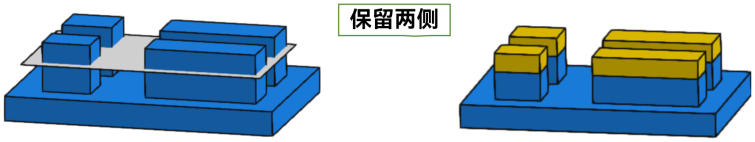
- m_version：选项版本号。
- m_fence：用来指定返回分割工具体前侧、后侧或两侧的结果实体。
- m_defTol：实体可被假定重合的默认容差。
- m_maxTol：可以应用的最大容差。

m_fence 选项在分割操作中的使用

m_fence 选项用于控制返回目标体的哪些区域。

示例一：在使用片状体分割实心体时将 m_fence 设置为 Both。

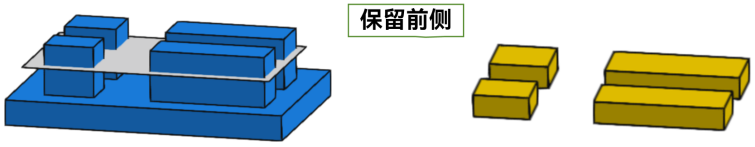
图 10-13 分割效果以及返回结果演示



返回的内容：四个正面主体和一个背面主体

示例二：在使用片状体分割实心体时将 m_fence 设置为 Front。

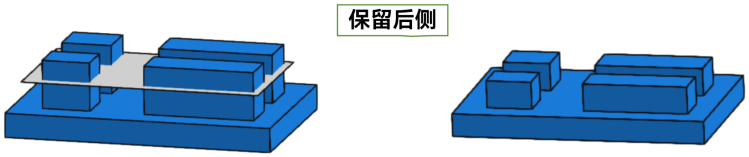
图 10-14 分割效果以及返回结果演示



返回的内容：四个正面主体

示例三：在使用片状体分割实心体时将 m_fence 设置为 Back。

图 10-15 分割效果以及返回结果演示



返回的内容：一个背面主体

共享几何

使用分割可能会破坏几何实体，将一个实体分割成两个可能导致每个实体上出现一些原本共享同一曲面的面。分割接口会为这些面创建共享几何，另一个实体将使用该共享几何的副本。

10.6 求交

与求交功能相关的接口如下：

接口	求交对象	说明
intersectCurves	两条曲线	所有位于重合区域边界上的点，包括完全与曲线重合的端点，都将作为重合交点返回。在使用 SP 曲线时，此接口可以间接地在曲面的参数空间中进行曲线与曲线的求交操作。
intersectSurfWithCurve	一个曲面和一条曲线	位于重合区域边界上的所有点，包括与曲面重合的有界曲线的端点，都将作为重合交点返回。
intersectFaceWithCurve	一个面和一条曲线	交点沿着有界曲线进行排序，并根据曲线的方向进行分类。
intersectFaces	两个面	如果这些面来自同一主体，则返回的任何曲线都将作为主体中的构造几何。如果面来自不同的主体，并且返回的任何曲线都是相交曲线，则这些曲线为面中曲面的副本之间的相交曲线。

接口	求交对象	说明
intersectSurfaces	两个曲面	这两个曲面必须都是孤立曲面或来自同一主体。在第二种情况下，生成的任何曲线都将作为主体上的构造几何。 🕮 说明 此接口不会定义部分重合的曲面区域，如果面的曲面完全重合，则不会返回相交数据。
intersectFaceWithSurface	一个面和一个曲面	曲面必须是孤立曲面或与面来自同一主体。如果曲面是孤立曲面，并且返回的任何曲线都是相交曲线，则这些曲线为该曲面和面中曲面的副本之间的相交曲线。如果曲面不是孤立曲面，则任何相交曲线都将作为主体上的构造几何。

🕮 说明
这些接口为离散几何提供部分支持。有关更多信息，请参见*《Geoshape 几何建模引擎软件 接口开发文档》*。

除 **intersectFaceWithCurve** 外，所有求交接口都提供了选项结构，可以用于提高求交性能，也可用于指定主体中您想求交的区域，详细内容请参见*《Geoshape 几何建模引擎软件 接口开发文档》*。

11 偏移

11.1 内容简介

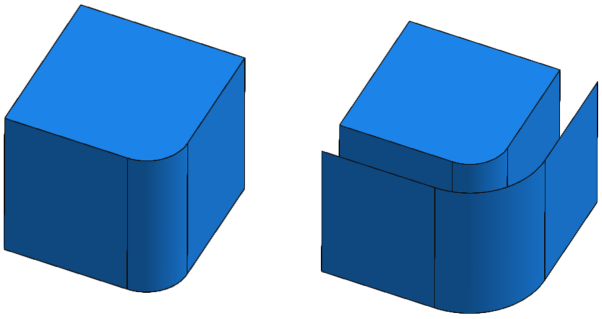
根据现有面来创建新面的操作为偏移操作，新面中的每个点都与原始面中的点相距一定的距离。 本章包含以下内容：

- 11.2 偏移：介绍如何通过已存在的面创建新的面。

11.2 偏移

偏移是根据现有面创建新面的过程，新面在每个点都与原始面相距一定的距离。如下图所示：

图 11-1 偏移示例



可以使用 **offsetBody** 来偏移一个实体中的所有面，偏移距离可以是正数或负数：

- 如果偏移距离为正数，那么面将会向外扩张，即远离实体的中心。
- 如果偏移距离为负数，那么面将会向内收缩，即靠近实体的中心。

通过选项 m_offsetFaceTags 和 m_offsetValues 可以实现将指定面偏移指定距离。

也可以使用 **offsetFaces** 来偏移一组面，通过参数 faceTags 和 offsets 来指定偏移面集和偏移距离。

offsetBody 提供的选项如下：

- m_offsetFaceTagCount：要偏移的面的数量。
- m_offsetFaceTags：要偏移的面的标签栏列表。
- m_offsetValues：每个面要偏移的距离。

偏移接口的使用

本建模引擎软件提供了两个偏移接口，**offsetBody** 用于偏移整个实体的面，**offsetFaces** 用于偏移实体内部特定的面。可根据实际情况决定选用接口：

- 如果需要用一个统一的偏移距离来偏移一个实体中的大多数面，使用 **offsetBody** 会更合适。
- 如果只需要偏移一个实体中的少数几个面，或者需要使用多个不同的偏移距离，使用 **offsetFaces** 会更合适。

12 混合

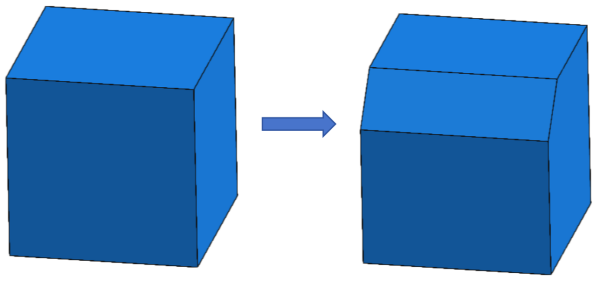
12.1 内容简介

本建模引擎软件为倒角复杂几何实体中的面和边提供了一套全面的倒角功能，本章将解释如何沿着边链（chains of edges）或在两个及两个以上的面之间添加倒角，从而平滑模型中的尖锐边。此外，还将解释本建模引擎软件中用于控制倒角面的是否保留和倒角面外观的选项，以及倒角过程中常见错误码和错误码的产生原因。

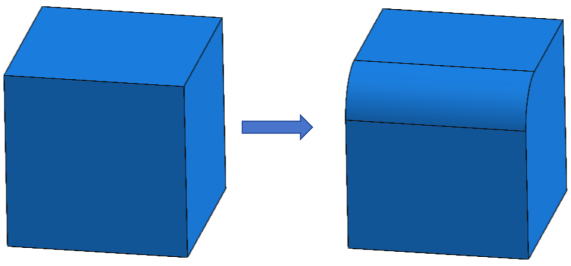
本章包含以下内容：

- 12.3 边倒角：介绍边倒角的概念、边倒角的类型以及如何进行边倒角。

倒斜角



倒圆角



12.2 查询混合曲面

可以使用 **getBlendSurface** 查询混合曲面（blend surface）的数据。混合曲面用于平滑面上的边或顶点。在内部，只有一种类型的倒角曲面，即混合边得到的混合曲面。例如，当两个倒角增材而另一个倒角移除材料时，可以使用这种倒角边来平滑一个由三条边组成的顶点。

混合曲面

倒角曲面中包含以下数据：

- m_geomTag1，m_geomTag2：原始倒角边两侧的曲面。
- m_radii[2]：对应曲面上的范围。
- m_spineTag：倒角的脊线。
- m_spineExtent：脊线的有效区域。

倒斜角时不会生成混合曲面，因为倒斜角时引入的曲面只有平面、圆柱曲面、圆锥曲面或 B 曲面。

简化倒角曲面

如果可以建模引擎软件会将倒圆角时生成的混合曲面简化为圆环面、圆柱面和球面。

getBlendSurface 只能用于输出 **PSGMApiBlendSurface** 类型的曲面。

12.3 边倒角

边倒角功能通过向主体引入新的边来平滑一条或多条边，生成的倒角面通常与相邻的面切线连续。

说明
边倒角为离散几何提供了部分支持。

12.3.1 已应用的倒角和未应用的倒角

为指定边设置倒角时，会将倒角以属性形式附着在指定边上。这样的倒角为未应用的倒角，当倒角操作执行后，生成的倒角面为已应用的倒角。

应用倒角后，原始边被生成的倒角面替代，原始边的数据已经被删除，可以通过回滚来恢复原始边的数据。关于回滚的更多内容，请参见 13.5 回滚。

说明
在设置或应用倒角时，修改模型的几何或拓扑会导致模型无效。

12.3.2 边倒角的类型

边倒角有两种类型：

- 倒斜角（chamfer）
- 滚球圆倒角（rolling-ball blend）

每种倒角都有对应的接口来设置，下面将详细介绍边倒角类型以及对应的设置接口。

说明
可以使用倒角固定接口 **blendApplyBody** 同时应用倒斜角和倒圆角。

倒斜角

可以使用 **blendSetChamferEdges** 来设置倒斜角。

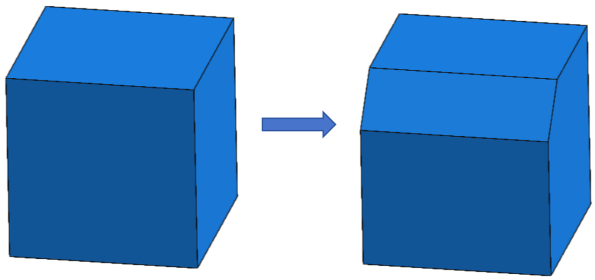
可以通过

blendSetChamferEdges
中的

range1
和

range2
定义的偏移曲面之间的交集来确定倒角范围。图 12-1 倒斜角为倒斜角示例。

图 12-1 倒斜角

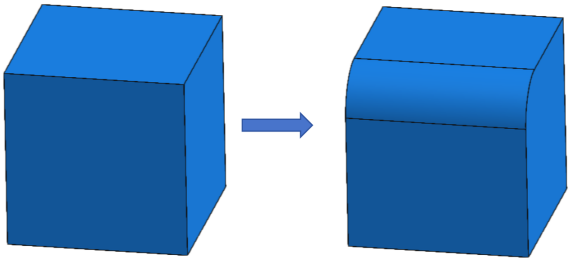


滚球圆倒角

滚球圆倒角可以使用 **blendSetConstantEdges** 来设置。

滚球圆是通过一个球体沿着两个与边相邻的面滚动而形成的，滚球圆的范围是由滚球圆的半径来确定的，此半径大小恒定。

图 12-2 滚球圆倒角



其他功能

除了创建未应用的倒角，还可以查询已应用与未应用的倒角，移除未应用的倒角。

查询未应用的倒角属性

有两种查询方式：

- 通过 **blendGetEdgeProperty** 查询指定边上未应用的倒角的参数信息：
  - 添加到边的倒角类型。
  - 位于倒角边左右两侧的面。
  - 倒角的形状。
  - 倒角的属性。
- 通过 **getBlendSurface** 查询滚球圆倒角曲面的标准形式。

移除未应用的倒角

可以使用 **blendResetEdge** 移除指定边上的未应用的倒角。

应用倒角

可以使用 **blendApplyBody** 将主体中所有具有倒角属性的边更改为适当倒角面。

12.3.3 边倒角的限制

边倒角存在以下影响因素：

- 倒角边的基本拓扑结构。
- 倒角边上附着的几何。
- 相邻边上是否存在倒角。
- 与倒角边相邻的已倒角边数量。

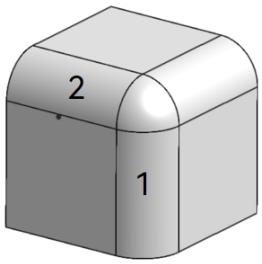
涉及到极其复杂的几何结构变化时，之前提到的倒角规则将不再适用。

12.3.3.1 一般限制

在对边进行倒角时，有以下一般限制：

- 不能对终止于一个切点的边使用非对称倒角（asymmetric chamfer）。
- 不能对终止于一个切点的边使用带有非称范围的圆锥截面倒角（conic cross-section blend）。
- 用作范围的值必须与倒角面的几何一致。例如，必须以指定的最小距离远离第二个面上的所有点。
- 需要的边所连接的面上的相关偏移量（由相关范围确定）不能自交，应相交以产生的脊线。

一个例外情况是：滚球圆倒角（边 1）的尺寸小于边 2 的倒角尺寸时，倒角失败。如下图所示：



倒角离散边或混合边

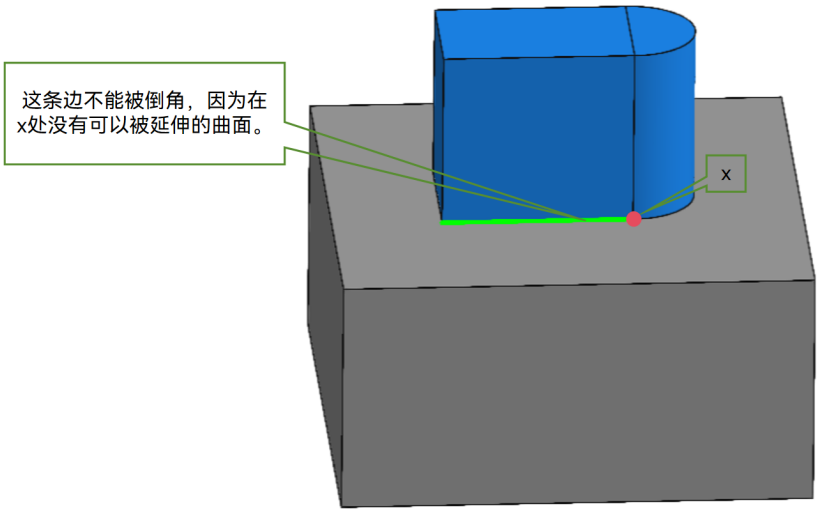
倒角离散边（facet edge）或混合边（mixed edge）时有以下限制：

- 如果一个顶点上的边数等于或大于四条，则只能倒角该顶点的一条边或两条边。
- 一个顶点上倒角的边不超过三条。
- 将一个三边顶点上的所有边进行倒角时，如果它们具有不同的凹凸性，则其中两条凹凸性相同的边必须具有相同的半径。
- 倒角一个边数等于或大于四的顶点上的两条边时，其他边必须是平滑的。
- 每个边链中倒角边的凹凸性必须一致，要么全是凸起（convex），要么全是凹陷（concave）。

12.3.3.2 特定限制

如果三条边相交于一个顶点，且末端曲面已定义，则可以对其中一条边进行倒角。末端曲面未定义时，则无法对其中的边进行倒角。示例如下：

图 12-3 在一个顶点上倒角一条边的限制



12.3.4 边倒角规则

以下小节介绍保证边倒角合法的相关规则。

说明
最好以成组的形式创建和应用倒角。

在一个多边顶点上倒角一条边

如果有超过三条以上的边在一个顶点处相交，可以对其中的一条边进行倒角，前提是：

- 倒角范围必须足够小，参与倒角的边最终要连接在同一个顶点上，并且它们的末端恰好终止于其他相邻的边，以保证倒角区域的连续性和平滑性。
- 现有面能够覆盖倒角，最多延伸两个相邻的面。

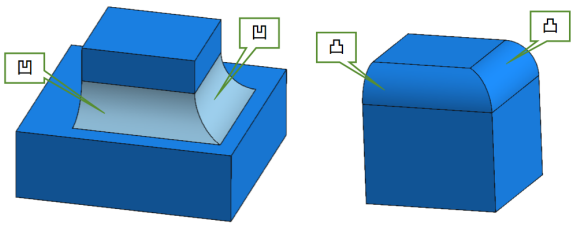
在三边顶点上倒角两条边

在三边顶点上倒角两条边的前提是：

- 三条边中必须有两条边凹凸性一致。
- 如果要倒角的两条边凹凸性不一致，则只有当其中一条边的倒角半径不变时，才能将其与另一条进行倒角。

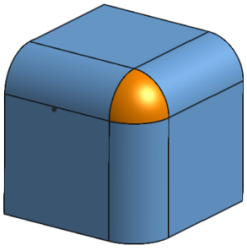
具有相同凸性的倒角边。如下图所示：

图 12-4 相同凸性倒角示例



在三边顶点上倒角三条边

对三边顶点上的三条边同时进行倒角（将它们作为一个组进行倒角的设置和固定），会添加一个额外的倒角面来平滑顶点。



12.3.5 边倒角错误

一般来说，要完全正确的完成倒角操作是较为困难的，因为很难提供一套适用于所有情况的的倒角规则。理解这些错误码是纠正不合法的倒角操作以及确保倒角正确的关键。这些错误码可以分为三组：

- 严重错误
- 一般配置错误
- 重叠倒角导致的配置错误

12.3.5.1 严重错误

严重错误表明目前指定的倒角边边永远无法倒角。这些严重错误要么反映了倒角功能上的严格限制，要么是在本建模引擎软件无法对错误进行分类的情况下产生。

不确定的倒角错误

这意味着倒角算法无法对检测到的错误进行分类。通常是内部算法计算错误。当此错误发生时，可以尝试改变倒角半径，以继续对边进行倒角。

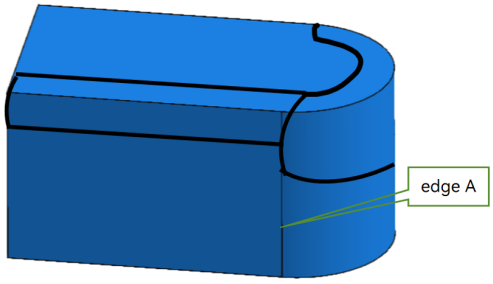
12.3.5.2 一般错误

配置错误会报告各种无效的倒角组合（ combinations of blends），这些错误可以通过以下方式来消除：

- 更改倒角半径（可能会更改多条边上的倒角半径）。
- 对边应用适当的倒角。
- 移除边的倒角属性。

相邻倒角边的范围不一致

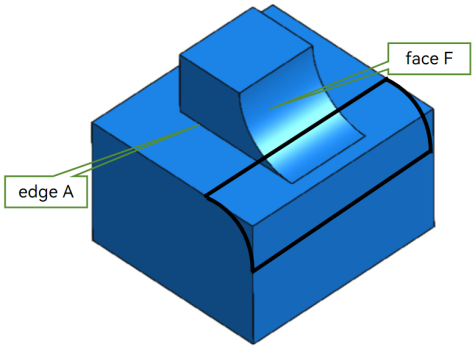
图 12-5 范围错误示例



上图展示了一个在指定的边上具有不同大小的未应用的倒角的实心体。如果执行倒角操作将产生范围误差，因为第三条边，边 A 是切线。

倒角面与未倒角的边重叠

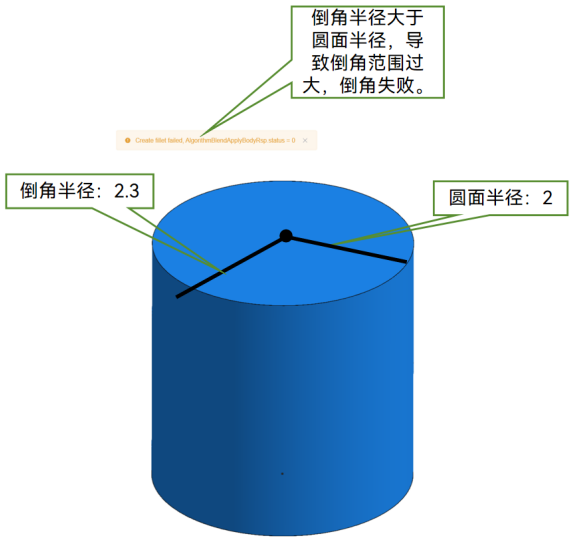
图 12-6 倒角重叠示例



上图展示的实心体示例中未应用的倒角的范围与未倒角边“edge A”有重叠，因为“face F”无法通过延伸来和倒角相交。虽然这种情况是不允许的，但如果倒角范围小到不出现重叠，则可能会继续进行倒角。

面上的倒角范围过大

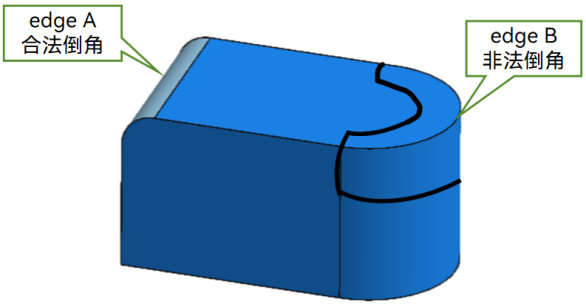
图 12-7 倒角范围大于面的示例



当没有任何边可以终止倒角的边界曲线时，可能会发生此错误，这可能是因为倒角半径太大，或者被检查边附近的一条或多条边也需要倒角，如上图所示。

另一个边上的非法倒角阻止了全面检查

图 12-8 非法倒角阻止全面检查的示例



上图展示了一个带有两个未应用的倒角的实心体。“edge A”是合法的，但同一个面上“edge B”的倒角是不合法的，所以导致重叠检查未执行。

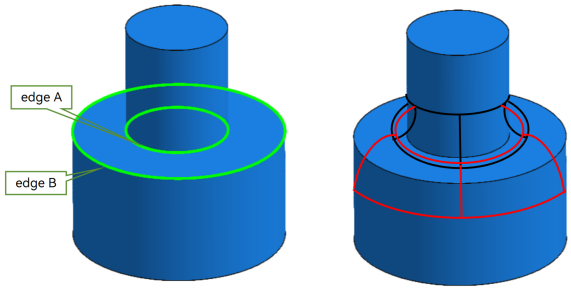
12.3.5.3 配置错误

当对一条边进行倒角时，倒角可能会与原始倒角边两侧的面之外的其他面进行交互。例如，倒角的范围很大时，它可能会覆盖一个面的部分区域并延伸到相邻的面。在这种情况下，倒角必须受到限制，以确保它与这个相邻面进行切向接触。

重叠倒角

下图展示了两个可以在非相邻面之间倒角的例子，这种情况被称为重叠倒角。右边的示意图中红色线表示“edge B”的倒角范围，黑色线表示“edge A”的倒角范围。

图 12-9 重叠倒角示例

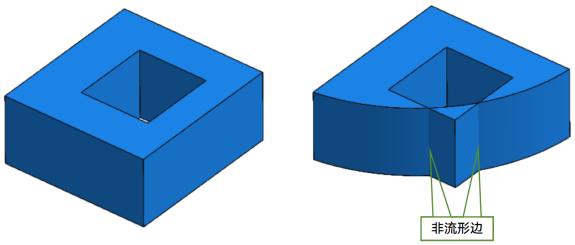


可以通过调用 **blendApplyBody** 同时应用重叠倒角中的两个倒角，如果应用失败，则可以使用 **blendSetConstantEdges**、**blendSetChamferEdges** 和 **blendApplyBody** 重新单独设置和应用每个倒角。

调用一次 **blendApplyBody** 来应用重叠倒角时可能会出现如下错误：

- OpBlendSurfOverlap
- OpBlendSurfNoEndBoundary

倒角与未参与倒角的边相交



在这个例子中，未应用的倒角与未参与倒角的边重叠。需要先减少倒角的范围，以便它不干扰未参与倒角的边，然后才能应用倒角。

12.3.5.4 面与面之间不一致

这个错误只有在应用倒角时执行了面与面之间不一致性检查才会出现。

12.3.5.5 自交曲面

如果倒角导致一个或多个倒角曲面发生了自交，就会出现此错误。

13 应用程序支持

13.1 内容简介

建模引擎软件提供了一系列功能，以高效的在应用程序中集成，从而支持复杂的应用程序特定功能。本章介绍了以下信息：

- 13.2 属性定义：介绍如何在应用程序中定义属性。
- 13.3 属性：介绍如何使用属性。
- 13.4 分区：介绍如何在会话中组织和保存实体。
- 13.5 回滚：介绍如何将会话返回至历史状态。
- 13.6 组：介绍如何创建关联实体的集合。
- 13.7 存储：介绍如何保存和加载本软件的模型和分区。

13.2 属性定义

可使用属性向实体附加额外数据来补充实体的数据结构。属性定义可设定属性的以下内容：

- 属性的类别；
- 属性可以附加到的实体类别；
- 属性字段的顺序和类型。

这些属性决定了建模引擎软件对属性应用的处理过程，可参见 13.3 属性。

应用程序也可通过回调函数指定一种替代的处理方法，该方法根据以下情况调用不同的回调函数：

- 属性定义类型；
- 对带有该类型属性的实体所应用的操作类型。

13.2.1 属性定义介绍

属性定义用于指定单个属性的行为。每个定义包含以下内容：

- 名称：用于在传输和接收过程中区分该定义。
- 属性类别：对属性所有者进行建模操作时，属性的行为方式。
- 可以拥有该定义属性的实体类别，包括：装配体、实例、主体、块、壳、面、环、边、半边、顶点、曲面、曲线、点、组。

  说明
  属性不能附加到孤立几何体上，无论该几何体属于什么类别。更多关于建模操作和附加到几何体上的属性的信息，请参见附加到几何体的属性处理。

- 属性的字段顺序和数据类型在定义中至关重要。一个定义中可以包含任意数量的字段，每个字段可以包含任意数量的类型的数据项，包括零项。可包含数据类型为：int、real、string、pointer、vector、axis。

属性定义属于会话，不与任何分区相关联。属性定义创建后不能被修改或删除，即使将会话回滚到属性定义创建之前的状态，属性定义也不会被删除。

说明
在会话中可创建多个不同名称的属性定义，如果您需要在产品中更改属性定义，可创建一个新的属性定义，并将其标识为现有属性定义的“新版本”。更多关于属性定义名称的信息，请参见属性定义名称。

属性定义函数

建模引擎软件提供如下属性定义相关的函数：
表 13-1 属性定义函数

函数	说明
createAttrDefinition	创建一个属性定义。
getAttrDefinition	查询属性定义的标准格式（**PSGMApiAttrDefinition**）。
getAttrDefTag	根据属性名称查询对应的属性定义。
hasAttrDefinition	检查实体是否拥有指定的属性定义。
modelGetAllAttrDefinitions	查询附加到指定模型中实体的所有属性的属性定义，如果模型中存在多个相同的属性定义，该定义只返回一次。
getAttrDefinitions	查询会话中用户自定义的属性定义。

属性定义名称

为属性定义指定一个专有的名称前缀，可以有效避免属性数据在不同应用程序中发生碰撞，建议您使用统一的命名规范作为属性定义名称的前缀（例如以应用程序或公司名称）。例如，在两个应用程序中都有一个名称为“Density”的属性定义，但分别使用了 string 类型和 double 类型。此种情况下，在一个应用程序中创建的模型无法加载到另一个应用程序中。

如果指定的属性定义是现有属性定义的新版本，可在名称中体现这一信息。

兼容性

当两个属性定义的名称不同时可同时存在于同一会话，即这两个属性定义相互兼容；若属性定义名称相同，则需要具有完全相同定义才可相互兼容。因此在后续的版本升级场景中，低版本的建模引擎软件模型文件可直接加载到高版本软件中，不需要修改属性定义。

说明
如果具有相同名称的两个属性定义之间的唯一区别是一个可以由半边拥有，而另一个不能，则它们也被视为兼容。此种情况在版本升级场景中，属性只能附加至高版本的半边。

保存和接收属性定义

- 保存属性定义
  - 保存一个模型时，与模型中实体相关联的所有属性及属性定义都将同时被保存至模型文件，未使用的属性定义不会被保存。
  - 在版本升级场景中，使用低版本格式保存模型时，若存在该版本存在不支持的属性定义，其相关的属性都会从实体中移除。
- 接收属性定义
  - 在接收模型文件时，只有当前会话中的属性定义与模型文件的属性定义兼容时，模型才能加载成功。加载成功后将自动创建模型中实体所附带的属性定义。
  - 如果模型的属性定义与当前会话中的定义不兼容，则模型无法成功加载。例如，两个具有不兼容的“密度”定义的应用程序无法共享任何带有密度属性的模型。

属性定义类型

属性定义的类型用于决定：

- 具有该定义的属性在其所附着的实体进行建模操作时的行为。
- 是否可以将具有该定义的多个属性附加到一个实体。

建模引擎软件提供如下属性类型：

表 13-2 属性定义类型

属性定义类型	说明
Internal	属性仅在内部创建和使用。该类的定义和这些定义的属性不能被创建，它们没有用户可见的字段。
AttrDefClassType01	属性与其所附着实体的物理尺寸和位置无关。例如密度。
AttrDefClassType02	属性取决于实体尺寸，但不取决于位置。例如重量。

属性定义类型	说明
AttrDefClassType03	属性可能随位置或方向而变化。例如转动惯量。
AttrDefClassType04	属性随其所有者一起变换，但在其他方面与所有者的大小和形状无关。例如切割面的工具体的移动起点或方向。
AttrDefClassType05	属性随其所有者进行变换，前提是其所有者不会以其他方式更改。例如重心。
AttrDefClassType06	属性与其所附着实体的物理尺寸和位置无关。与 AttrDefClassType01 不同，此类属性支持多个值，即一个实体可以附加一个相同类型的属性列表。
AttrDefClassType07	属性随其所有者一起变换，但在其他方面与所有者的大小和形状无关。与 AttrDefClassType04 不同，此类属性支持多个值，即一个实体可以附加一个相同类型的属性列表。

几何实体只能附加第 AttrDefClassType01、AttrDefClassType04、AttrDefClassType06 和 AttrDefClassType07 类型的属性。有关建模操作和附加到几何体上的属性的更多信息，请参见附加到几何体的属性处理。

13.2.2 属性回调

属性回调为应用程序提供了一种比常规属性处理更强大的属性处理方法。这些函数与属性定义相关联，当具有该类型属性的实体发生以下事件时，建模引擎软件会调用这些函数：

- 分割
- 合并
- 删除
- 复制
- 传输
- 接收

在调用回调函数之前，需先进行注册。使用 **attrDefRegisterCallback** 注册应用程序中提供的回调函数指针，使用 **attrDefGetCallbacks** 获取这些指针。

说明
当一个实体具有多个不同定义的属性时，不存在任何特定的处理顺序。

属性回调包含如下两种类型：

- 普通回调：此类回调函数可以根据需要修改和查询模型的属性或其他信息。当调用普通回调函数时，它所执行的操作会替代建模引擎软件的常规属性处理。普通回调函数可以调用只读内核函数。

- 只读回调：此类回调函数不能以任何方式修改模型的属性或其他信息。在只读回调函数返回后，建模引擎软件会根据属性的类型以常规的方式处理这些属性。只读回调函数只能调用只读内核函数。

您可以使用 **attrDefRegisterCallback** 中的字段 option 来指定回调函数是普通回调函数（Normal）还是只读回调函数（ReadOnly）。

每个属性定义都可以与多个回调函数相关联（每个事件类型一个），不同的事件可以触发对不同的回调函数的调用。每个属性定义都有一组逻辑标志，这些标志决定在事件发生时是否调用回调函数。

使用 **attrDefSetCallbackFlags** 设置属性回调函数的开启/关闭标志，使用 **attrDefGetCallbackFlags** 查询这些标志。

📖 说明
在会话中的任何时候注册了普通回调函数，那么在该会话的剩余时间里，标签持久性行为将被永久禁用。这适用于所有可能拆分实体然后删除其中一个拆分实体的建模操作，包括布尔、压印、倒角和局部操作。有关标签和标签持久性的更多信息，请参见标签。
注册只读属性回调函数，不会导致标签持久性被禁用。

事件

当某个事件发生时，若存在针对该事件注册了属性回调的属性定义，且参与事件的实体至少具有一个该类型的属性，则会调用（仅一次）相关的属性定义。

回调函数会在主体、块、壳、面、环、边、半边、顶点、几何体、变换、组、装配体和实例上被调用。所有类型的实体都可能发生所有事件，但几何体、变换、装配体和实例永远不会合并或分割。

当事件发生时执行的具体操作取决于为该事件的属性定义注册的回调类型：

- 注册了普通回调：该回调函数将替换属性的现有行为（由属性的类定义）。
- 注册了只读回调：该回调函数将补充属性的现有行为，一旦回调函数完成，建模引擎软件将按常规方式处理属性。
- 没有注册属性回调：属性的行为由其类决定。

实现回调函数

调用普通回调函数时，建模引擎软件将不执行必要操作（即从实体中删除的属性）之外的常规属性处理。如果需要进行这种处理（如 13.3 属性所述），则需在普通回调函数的定义中包含它。

属性回调函数接收一个或多个属性数组，这些数组始终包含与属性回调相关联的属性定义。

属性回调函数和标志不会通过 **saveModel** 或 **savePartition** 进行传输。

属性回调接口

回调函数的调用方式如下表所示。在每种情况下，建模引擎软件传递的属性列表仅包含具有相关属性定义的属性。

回调类型	函数调用时间点	通过函数传递的数据
分割	分割之后	● 原始实体及其属性 ● 新实体（不包含属性）
合并	即将合并时	● 保留的实体及其属性 ● 删除的实体及其属性
删除	即将删除时	删除的实体及其属性
复制	复制之后	● 原始实体及其属性 ● 复制的实体（不包含属性）
传输	开始进行传输时	传输的实体及其属性
接收	开始进行接收时	接收的实体及其属性

13.3 属性

您可使用属性向建模引擎软件实体附加额外数据来补充数据结构。属性的特性在属性定义中指定。

13.3.1 属性介绍

属性是通过指定属性定义和要附加的实体来创建的。属性只能附加到其属性定义中指定的类别的实体上，在这个阶段，属性的字段是空的。

然后，可以设置和查询属性的每个字段，每个字段可以包含任意数量的适当类型的数据项：

- 属性中的每个字段都有一个索引号和名称。
- 字段内的每个数据项也有一个索引号。
- 数据作为完整的项目列表（或作为字符串）写入字段，无法单独写入单个项目。
- 查询函数返回字段中的所有项目或单个项目。
- 向量字段处理函数用于访问类型为向量、坐标和方向的字段。

📖 说明
如果要给会话中大部分的实体添加附加数据，用户字段可能比属性更合适。用户字段和属性的差异对比请参见表 15-1 用户字段和属性对比。

创建空属性

使用 **attrCreateEmpty** 创建一个新的属性，这个属性会附加到指定的实体上（通过 **attrGetOwner** 获取），并且具有指定的属性定义（通过 **attrGetAttrDefinition** 获取），但不包含任何数据。

创建空属性并将其附加到实体上，只是将有效属性与实体关联的第一阶段。第二阶段则是用适当的信息填充该属性。如果一个模型的任何实体上都关联了空属性，那么该模型被视为处于中间状态，并且不应进一步用于建模操作。

检查属性有效性

您可以检查属性的有效性，既可以作为对一组实体进行常规检查的一部分，也可以作为对附加到指定实体的属性的特定检查。更多信息，请参见 5.6.2.1 检查系统属性缺陷。

添加数据到属性字段并进行检索

建模引擎软件提供多个接口将数据数组写入到指定类型的属性字段，并检索该数据。具体接口请参见表 13-3 通过字段索引添加属性数据和表 13-4 通过字段名称添加属性数据。

表 13-3 通过字段索引添加属性数据

写入字段	读取字段	读取字段中的第 n 个实体
attrSetAxes	**attrGetAxes**	**attrGetNthAxis**
attrSetDoubles	**attrGetDoubles**	**attrGetNthDouble**
attrSetIntegers	**attrGetIntegers**	**attrGetNthInteger**
attrSetString	**attrGetString**	不支持。
attrSetVectors	**attrGetVectors**	**attrGetNthVector**
attrSetPointers	**attrGetPointers**	**attrGetNthPointer**

表 13-4 通过字段名称添加属性数据

写入字段	读取字段
attrSetNamedAxes	**attrGetNamedAxes**
attrSetNamedDoubles	**attrGetNamedDoubles**
attrSetNamedIntegers	**attrGetNamedIntegers**
attrSetNamedString	**attrGetNamedString**
attrSetNamedVectors	**attrGetNamedVectors**
attrSetNamedPointers	**attrGetNamedPointers**

删除属性

通过如下接口删除属性：

表 13-5 删除属性

接口	说明
deleteAttributes	在指定实体中删除指定属性定义相关的属性。
modelDeleteAttributes	在指定模型中删除指定的属性定义相关的属性。

查询属性

表 13-6 查询属性

接口	说明
getAttributes	通过指定的属性定义查询附加到实体的属性。
getFirstAttribute	通过指定的属性定义查询附加到实体的第一个属性。
modelGetAllAttributes	查询指定模型中指定属性定义的所有属性。
modelGetAttrsByCallback	从指定模型的实体中查询指定属性及其所有者，使用回调函数从其定义和字段值中选择属性。
topoGetEntitiesByAttrDefinition	查询指定拓扑的实体集合，并且这些实体可能具有指定属性。是否具有指定属性由 hasAttr 参数控制。

使用回调函数从模型中选择属性

查询模型上的属性以及这些属性所附加的实体的最灵活方式是使用 **modelGetAttrsByCallback**。这个接口允许你在应用程序代码中指定一个回调函数，该函数可以根据属性定义以及属性字段自身的值来返回模型中选定的属性。**modelGetAttrsByCallback** 的参数如表 13-7 输入参数和表 13-8 输出参数所示。

表 13-7 输入参数

输入参数	说明
modelTag	需要查询的模型的标签。
callback	用于确定返回哪些属性的回调函数，由应用程序提供。
context	供回调使用的上下文数据，此参数可以包含回调所需的任何数据。例如，如果要返回属性值等于特定字符串的所有属性，请将该字符串作为上下文传递。
getAttrOption	选项参数。可选择是否返回属性和相关实体，并将查询限制在特定的属性定义中。

表 13-8 输出参数

输出参数	说明
attrTagCount	查询到的属性的数量。
attrTags	查询到的属性标签列表。
ownerTags	属性所有者的标签列表。

您定义的任何回调函数都需包含如下参数：

- 属性定义的标准形式；
- 属性字段值的数组及其长度；
- 调用 **modelGetAttrsByCallback** 时指定任何上下文。

使用回调函数选择属性时，模型中每个属性都会被检查：

- 属性定义是否存在字段。
- 相关字段类型是否正确。
- 是否存在足够的条目需要检查。
- 值是否匹配。

13.3.2 建模操作对属性的影响

属性可以作为建模操作的结果，在实体之间移动、修改或删除，这些操作对属性的影响首先取决于属性定义是否注册了普通回调函数。

使用注册的回调函数处理属性

当出现以下情况时，调用回调函数：

- 附加了属性的实体经历拆分、合并、删除、复制、传输或接收事件。
- 与该属性相关联的属性定义有一个为正在发生的事件注册的回调函数（不是 NULL）。
- 回调的标志设置为 true。

在普通回调下，调用回调函数，替代建模引擎软件常规的属性处理。

在只读回调下，调用回调函数，并进行常规的属性处理。

属性回调函数请参见 13.2.2 属性回调。

没有注册属性回调时处理属性

如果未调用回调函数，则应用建模引擎软件常规的属性处理。它取决于：

- 属性定义的类型。
- 建模操作期间应用于实体的事件的组合。
- 属性所包含的数据字段的类型。
- 数据字段的内容（比较两个属性时）。

13.3.3 事件对属性的影响

高级建模操作（如布尔运算、局部操作、倒角等）可以视为一系列应用于单个实体的基本建模操作或事件的连续过程。在建模操作下，属性的行为是其所附加的实体上所应用事件的基本影响的综合结果。

本节描述了这些事件以及它们可能对附加到拓扑实体上的属性产生的影响。这些影响会根据属性定义的类别以及是否已注册属性回调而有所不同。

🕮 说明
这些影响不适用于附加到几何实体上的属性。

回调函数和类

当未调用属性回调时，建模引擎软件按常规方法处理属性（根据属性类中定义的方法）。当调用了属性回调函数时：

- 当为属性定义注册了普通回调函数时，这些回调函数会覆盖当前由属性类定义的属性的现有行为。在事件发生时，属性的行为将遵循回调函数中定义的逻辑，而不是根据属性类自动执行的操作。
- 当为属性定义注册了只读回调函数时，正常的属性处理仍然会进行，并且还会调用只读回调函数。在读取属性值时，除了正常的处理外，还会执行只读回调函数中定义的任何附加操作。
- 如果只为某些事件注册了回调函数，那么在未注册回调函数的事件发生时，属性的行为将继续由属性的类定义。

变换

一个实体可能会被转换，在这种情况下，模型组成部分的所有实体（如面的曲面、体的边等）也会相应地转换。变换的效果取决于变换组合中包含的变换，有如下四种基本类型：

- 镜像
- 旋转
- 平移
- 缩放

每种类型的变换都可能被独立的应用于被转换实体上的属性，也可能对其无影响，或导致属性被删除。

🕮 说明
如果实体的变换除了简单地移动之外还对其产生了其他影响（例如，实心体中面的曲面被转换，并且边和顶点需要重新计算），应属于更改事件而不是变换事件。

当对一个属性应用变换时，它的影响会根据字段类型以不同的方式作用在每个字段上。只有四种几何字段类型（向量、方向、坐标和轴）会受到影响，并且它们各自的变换方式是不同的。为了确定复杂变换的影响，我们需要将其分解为基本变换的组合，并查看它们对各种字段类型的个别影响，具体如下：

- 坐标字段会受到所有变换的影响。
- 向量和方向字段会受到旋转和镜像的影响，但不会受到缩放或平移的影响。
- 轴字段的行为是一个坐标字段和一个方向字段的组合。

分割

当实体被分割时，它的属性要么被删除，要么被传递到两个或多个新生成的实体上。

说明
这与倒角不同，因为在倒角时，通常会创建一个新的面，而不是分割相邻的面，因此不存在属性的传递。

例如：

- 一个面可以通过布尔运算、划线或局部操作进行分割。
- 从一个实体的面创建的片状体或实心体将接收该面上属性的副本。
- 参与布尔减法或截面操作的目标体上的属性会受到基本操作分割的影响；但是工具体及其属性将被删除。

此外，分割一个实体可能会调用附加到该实体的属性的回调函数。

合并

两个同类型的拓扑实体可以合并成一个。在合并操作中，两个实体上相等的属性（属性定义相同且所有字段中的值也相同）可能会被删除，也可能会保留。具体行为取决于属性类是否允许存在多个副本，如下表所示：

属性类	属性行为
属性类 02、03、05	不允许存在多个副本，属性将被删除。
属性类 01、04	不允许存在多个副本，当： ● 合并的实体中只有一个实体附加了属性，该属性将被删除。 ● 合并的两个实体都附加了属性，并且这些属性是相等的，只保留该属性的一个副本。 ● 合并的两个实体都附加了属性，并且这些属性不相等，这两个属性都将被删除。
属性类 06、07	允许存在多个副本，当： ● 合并的两个实体中任意一个实体附加了属性，则该属性将被保留。 ● 合并的两个实体都附加了属性，并且这些属性是相等的，只保留该属性的一个副本。 ● 合并的两个实体都附加了属性，并且这些属性不相等，这两个属性都将被保留。

属性类的更多信息，请参见表 13-2 属性定义类型。

此外，合并一个实体可能会调用附加到该实体的属性的回调函数。

转移

一个实体可以从一个模型转移至另一个模型，也可从一个模型中移入或移出。例如在布尔运算期间，壳、面、环等可以在主体之间转移。当几何体从一个模型中移出但并未移入另一个模型时，它将会失去其所有属性。

创建/删除

您可通过直接调用建模引擎软件接口来创建或删除实体，也可间接创建或删除，例如执行布尔运算时可能会创建或删除多个实体。新创建的实体（即不是从现有实体中分割出来的）没有任何属性。删除一个实体将同时删除其属性。

例如：

- 在合并操作中删除实体（例如通过合并两个面删除边）时，附着到该实体的属性将被删除。
- 混合边会删除该边及其附加的属性。

此外，创建或删除实体可能会调用附加到该实体的属性的回调函数。

传输/接收

传输和接收实体都可能会调用附加到该实体的属性的回调函数。

拓扑体的其他变更

面的方向：面方向的改变本身并不会触发事件（例如仅仅反转面的方向），附加到该面上的属性不会受到影响。但是，导致方向反转的操作很可能会影响这些属性。

例如：从一个长方体块中减去一个圆柱体，会在长方体块中形成一个圆柱形的孔，此时圆柱面的方向会被反转。此种情况下，附加到圆柱面上的属性会受到影响。

13.3.4 其他属性处理

常规属性处理

不同事件发生时，建模引擎软件对不同类的属性的常规处理如表 13-9 常规属性处理所示。若调用了普通回调，则回调中定义的属性行为将替代常规属性处理。

表 13-9 常规属性处理

属性类型	事件	属性行为
01	镜像、旋转、变换、缩放、转移、更改	无影响
	分割	传递
	合并	当属性相等时保留一个副本
02	旋转、变换	无影响
	镜像、缩放、分割、合并、转移、变更	删除

属性类型	事件	属性行为
03	所有事件	删除
04	镜像、旋转、变换、缩放	应用
	分割	传递
	合并	当属性相等时保留一个副本
	转移、变更	无影响
05	旋转、变换、缩放	应用
	镜像、分割、合并、转移、变更	删除
06	镜像、旋转、变换、缩放、转移、更改	无影响
	分割	传递
	合并	全部保留（如果属性相等则只保留一个副本）
07	镜像、旋转、变换、缩放	应用
	分割	传递
	合并	全部保留（如果属性相等则只保留一个副本）
	转移、更改	无影响

属性类的详细说明请参见表 13-2 属性定义类型；事件对属性的影响请参见 13.3.3 事件对属性的影响。

组合事件

许多高级建模操作会对操作的实体应用多个事件。这些事件对附加到这些实体的属性的影响是每个事件影响的组合。

例如，当以下操作对面或边有影响时，它们将作为分割、转移和更改事件的组合来实现这些影响：

- 对主体进行分割操作，影响了面/边。
- 布尔运算中影响了面/边。
- 在面上绘制一条线，其中边受到影响。

这些操作会导致类型为 02、03 或 05 的属性从面/边上被删除，而类别为 01、04、06 和 07 的属性则被保留，并从原始面/边传递到拆分出来的任何新面/边上。

附加到几何体的属性处理

属性可以附加到几何实体（点、曲线、曲面）上，这些几何实体本身附加到拓扑结构上（即使仅作为构造几何体）。仅允许对类别为 01、04、06 和 07 的属性进行此操作。此类属性仅受到属性本身删除、拥有该属性的几何体删除或使用相同定义的新属性覆盖的影响，其他建模操作不会影响附加到几何实体的属性。

几何实体只有在其本身属于某个模型时拥有属性，例如它们是模型的构造几何体，或者是模型中某个对应拓扑实体的几何体。不能将属性附加到孤立的几何体上，如果带有属性的几何体从一个模型中转移出来但没有转移到另一个模型中，它将失去其属性。

附加到组的属性处理

建模操作对附加到组的属性影响取决于组中实体的类别。可以使用一个简单的方法来判断建模操作对附加到组的属性影响：

- 假设组中的每个实体都附加了一个属性的副本。
- 确认建模操作对每个属性副本的影响。
- 如果所有副本都保留下来并且最终具有相同的值，那么原始属性就保留下来并具有这个值；否则，属性将被删除。

13.4 分区

应用程序可通过分区将建模会话中的实体组织成相关模型集，并将这些模型保存为单个项目。创建分区后，应用程序可根据分区标记将建模引擎软件会话回滚至历史状态。回滚的更多信息请参见 13.5 回滚。

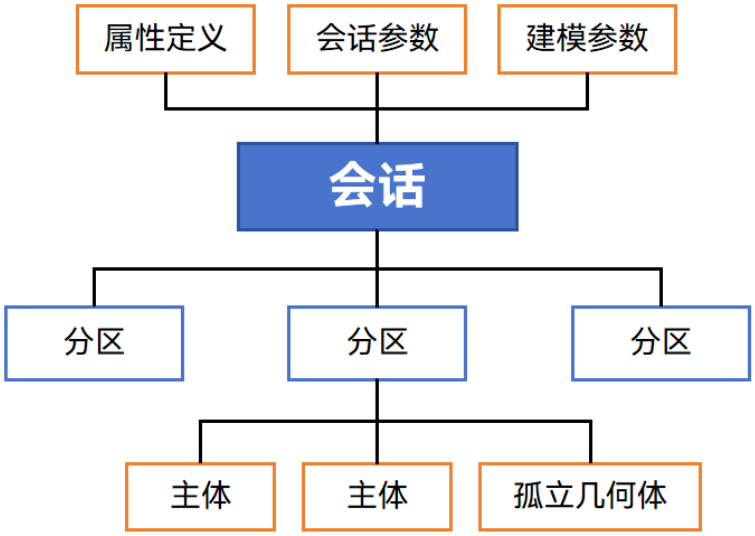
分区和回滚使用的对象如表 13-10 使用对象所示。
表 13-10 使用对象

对象	类型	说明
PARTITION	分区	可以独立回滚或前滚的实体集合。
PMARK	分区标记（partition mark）	分区的状态记录。
MARK	会话标记	会话的状态记录。
DELTA	分区标记之间实体的变化	两个相邻的分区标记间创建、修改、删除实体的记录。

分区是一个包含主体、孤立几何体和变换的独立集合。它可以独立于其他分区进行回滚和前滚操作。如图 13-1 会话结构所示，会话中的所有实体都位于某个分区中，除了以下例外：

- 属性定义
- 建模和会话参数

图 13-1 会话结构



这些对象存在于会话级别，它们在创建时不考虑当前分区，并且不受无分区间引用规则的约束（参见分区间引用），不受分区回滚的影响。

一些常用的分区概念如下：

- 当前分区：建模引擎软件当前所处的分区为当前分区，当前分区的值受回滚的影响。当会话启动时，建模引擎软件会新建一个分区并使其成为当前分区，新建的、未引用的实体将被放入当前分区中。
- 轻量型分区：轻量分区在第一个非初始的 pmark 处没有向后的 delta，其他功能与标准分区完全相同。更新或删除轻量级分区的第一个非初始 pmark 比标准分区要更快，在以下情况中，您可选择使用轻量型分区：
  - 仅使用两个 pmark，并且第一个非初始的 pmark 经常更新。例如，为方便在操作失败后进行错误恢复，每隔一段时间就使用 pmark 来保存分区的状态。此时可只设置两个 pmark，其中第一个非初始 pmark 将在每次保存分区状态时更新。
  - 使用三个或更多的 pmark，并且第一个非初始的 pmark 经常被删除。

分区间引用

通常情况下，为避免在使用分区回滚时可能出现的不一致性，建模引擎软件不支持跨分区引用。例如：将不同分区中的主体进行合并，将几何体附加到不同分区中的主体上，或在不同分区中的曲面进行相交操作。

如果创建了一个由现有实体引用的实体，创建的实体与现有实体处于同一分区。如果创建的实体没有被引用（例如使用 **createLine** 创建线），创建的实体将处于当前分区。因此，当两个现有实体处于同一分区时，才能通过调用接口创建它们之间的引用。例如：布尔运算只能在同一分区中的面和主体上执行。

如果调用接口进行操作的过程中引用了其他分区的实体（例如临时访问其他分区的数据），但在操作完成后没有创建实际的跨分区引用关系，这种情况下跨分区引用是允许的。根据跨分区引用的情况，函数分为如下类型：

- 只读函数：只读函数只是读取数据，并不涉及任何创建引用的操作。即使其输入的实体处于不同分区，也不影响其调用。
- 不在不同输入模型（或孤立几何体）之间创建引用的建模函数：这类函数不会创建跨分区的引用关系，输入模型（或孤立几何体）的实体可以处于不同分区。
- 创建未被任何现有实体引用的实体（例如模型或孤立几何体）的建模函数，会在当前分区中创建这些实体。
- 接收一个模型会在当前分区中创建体。
- 其他内核函数可能会在实体之间创建引用关系，此时输入的实体必须位于同一分区，并且创建的实体也位于该分区。

分区操作

建模引擎软件中，常见的分区操作如表 13-11 分区操作所示：

表 13-11 分区操作

操作	说明
在分区之间复制实体	使用 **copyEntity** 将实体复制到指定分区，复制行为在选项结构 **PSGMApiEntityCopyOption** 中设定。
在分区之间移动主体	对于最近一次分区回滚操作之后创建的主体（其内部所有实体都是新建的），可使用 **bodyChangePartition** 将主体移动到其他分区，同时保留其所有标签，主体和其内部实体的标签将保持不变。 **bodyChangePartition** 可以用于多种情况，比如接收一个包含多个主体的传输文件，然后将这些体分发到其他分区。通过该函数可确保每个体都被放置到预期的分区中，同时保持它们的属性和标签不变。
复制分区	使用 **copyPartition** 复制整个分区。设置 type 的值，控制如何复制分区中的增量，包含如下选项： ● None：不复制增量。 当复制一个分区时，分区中的模型和附加到分区的孤立几何体都会被复制，但是，原始分区中的用户字段不会被复制。复制的分区中的内容在顺序上（由查询函数返回）与原始分区相同，以便应用程序可以将复制的实体与原始实体关联起来。 📖 说明 使用 **copyPartition** 时，要复制的分区不能处于其初始 pmark。若在指定会话标记处不存在的任何分区，也将被视为处于其初始 pmark。

操作	说明
删除分区	使用 **deletePartition** 删除指定分区，当前分区不能被删除。在创建会话标记时，会话标记将关联当前会话所有分区的当前 pmark（若在创建会话标记后新建分区，则会话标记将关联该分区的初始 pmark）。若需删除分区的非初始 pmark 关联的所有会话标记均未与当前分区的初始 pmark 关联，删除成功，否则删除失败。 设置 **deletePartition** 的 delNonEmpt 来选择是否强制删除非空分区。默认情况下只能删除空分区（即不包含任何装配体、主体、几何实体、变换），若设置为 true，则可在每个会话标记处删除非空分区，以及分区中当前的所有模型。 使用 **gotoPMark** 回滚至初始 pmark 处可使分区处于非活动状态。此时分区不能被设置为当前分区，也不能在其中创建实体。将分区回滚至其他 pmark 后，非活动状态解除。通过此方法可隐式删除该分区。
合并分区	使用 **mergePartition** 来合并两个或更多的分区，将每个分区中的模型数据组合起来，并在过程中交错处理 pmark 的增量。每个要合并的分区必须包含一个线性的 pmark 图（即 pmark 序列是连续且没有分支的）。 📖 说明 轻量型分区不能被合并。 合并分区的更多信息请参见合并分区。
修改分区类型	使用 **partitionGetType** 查询分区的类型，使用 **partitionSetType** 设置分区的类型： ● 从标准型改为轻量型：可在任何时候将标准型分区改为轻量型分区，此时初始 pmark 和其之后 pmark 之间的反向增量（backward delta）将被删除。这种修改不能通过回滚会话来撤销，只能重新设置分区类型为标准型。 ● 从轻量型改为标准型：只有在当前 pmark 是初始 pmark 或初始 pmark 后一个 pmark 时可执行该操作。 📖 说明 ● 为避免反复创建和删除反向增量，可在创建 pmark 时设置其为轻量型 pmark。pmark 的更多信息请参见 13.5.1 分区回滚，增量的更多信息请参见 13.5.2 增量。 ● 这些函数应在调用 **deltaRegisterCallbacks** 启用分区回滚后使用。

合并分区

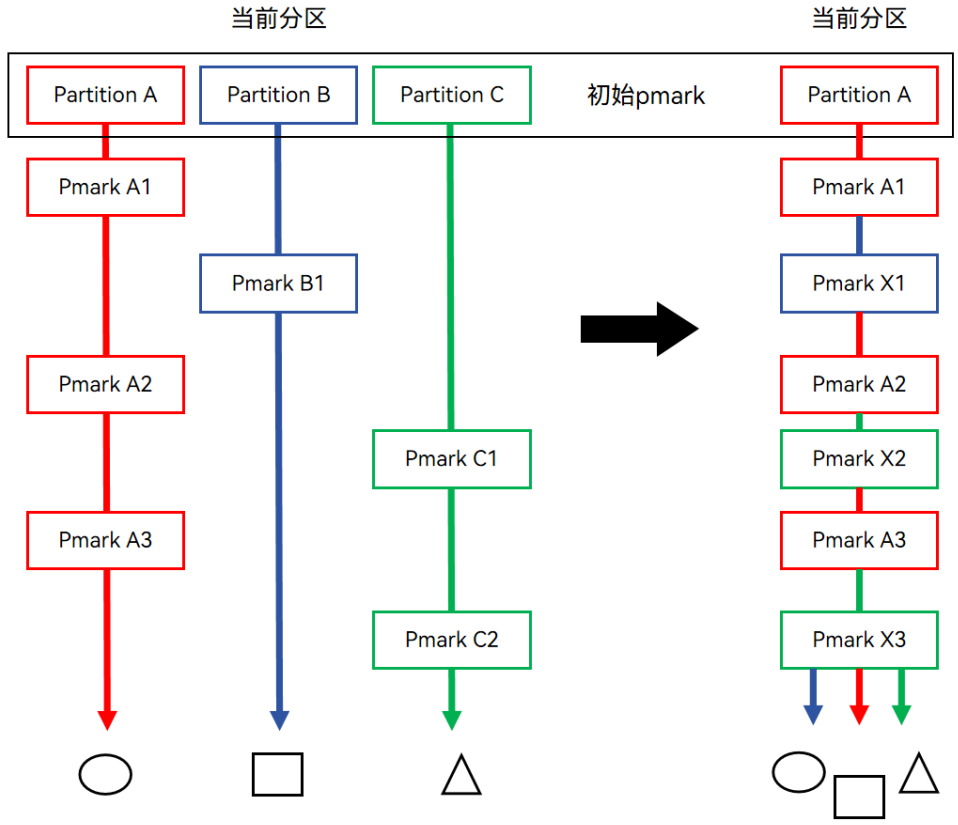
分区合并函数 **mergePartition** 接收一个分区数组和 pmark 数组作为输入参数。其中分区数组中第一个分区将被保留，其余分区将被删除。若提供的分区包含当前分区，则保留下的分区将成为新的当前分区。

pmark 数组包含指定分区的所有 pmark（初始 pmark 除外），pmark 的提供顺序定义了这些 pmark 在保留下来的分区中出现的顺序，同一分区 pmark 的提供顺序与在单独分区时的顺序相同。

如图 13-2 合并三个分区所示，使用 **mergePartition** 将三个分区合并，其中传递给函数的分区数组顺序为 A、B、C，pmark 数组顺序为 A1、B1、A2、C1、A3、C2，则：

- 分区 A 在操作中保留下来。它的三个 pmark 除了连接性外保持不变。
- 分区 B 和 C 在操作中被销毁。它们包含的 pmark 在保留下来的分区 A 中被赋予新的标识符（但具有相同的标签）。
- 分区 B 是操作前的当前分区，分区 A 是操作后的当前分区。

图 13-2 合并三个分区



在以上示例中，pmark 可以按照各种顺序提供给 **mergePartition**，最终保留的顺序也将随之改变。但同一分区 pmark 的顺序与在原始分区顺序中相同，例如 A3 不能在 A2 前提供，B2 不能在 B1 前提供。

分区函数

以下函数也与分区相关，更多信息可参见*《Geoshape 几何建模引擎软件 接口开发文档》*。

函数	说明
getPartition	查询实体所在的分区。
pmarkGetPartition	查询 pmark 所在的分区。
getCurrentPartition	查询当前分区。
getPartitions	查询会话中所有分区。

13.5 回滚

您可在以下情况时使用回滚：

- 当操作未产生预期结果时，使用回滚撤销操作或一系列操作；
- 建模操作失败导致状态不一致时，使用回滚回到失败之前的状态；
- 还可使用回滚回到之前的状态，尝试其他的建模方法和策略，寻找更合适的方法，以提高特征模型更新的性能。

📖 说明
回滚的使用方式是应用程序与其用户之间接口的基础，它影响应用程序在出现错误后的处理方式，以及是否允许用户在会话中来回滚动。

回滚用于撤消在某个标记点之后所做的更改，该标记由应用程序或用户设置。根据标记的不同回滚分为以下两种类型：

- 分区回滚；
- 会话回滚。

接口函数

回滚涉及如下接口函数：

接口名称	说明
getRollbackEnabled	查询会话是否启用回滚。
setRollForwardEnabled	设置会话是否可以前滚。
getRollForwardEnabled	查询会话是否启用前滚。
getCurrentPartition	查询当前分区。
getCurrentMark	查询当前会话标记。
getPartitions	查询当前会话所有的分区。

13.5.1 分区回滚

分区回滚（Partitioned rollback）使应用程序在模型参数发生变化后能够更高效地更新模型，它允许将模型回滚到添加某个特征的点，重新应用修改后的特征，以生成更

新后的模型。单个主体或主体集合可以独立地回滚，因此在一次会话中更新一个模型不会影响其他模型。此外，还可支持对指定主体的分支（多种可替代的更新）进行回滚。

在使用分区回滚之前，你需要使用 **deltaRegisterCallbacks** 注册一个 delta 回调函数。要停止分区回滚时，需停止建模引擎软件会话。有关 delta 函数的更多信息，请参见 13.5.2 增量。

分区标记

您可在分区中创建分区标记，以对分区进行来回滚动。分区标记可以被视为记录分区的一个特定状态，对应的对象是 PMARK。

在任何时候，分区都有一个当前 pmark，是最近创建或回滚到的 pmark。若 pmark 成为当前状态后，分区没有被修改（创建、修改或删除实体，或执行回滚操作），则分区一直处于该 pmark 状态。

每个分区都有且仅有一个初始 pmark，它不能被删除；处于初始 pmark 状态的分区为非活动状态，无法被使用（进行建模等操作），虽然新创建的分区只有一个初始 pmark，但其并不处于初始 pmark 状态。

pmark 序列图

分区所有的 pmark 一起构成了它的 pmark 序列图，应用程序可以查询其中包含的 pmark 序列：

- 序列图主线是从初始 pmark 到当前 pmark 的路径。
- 新创建的分区具有一个只包含初始 pmark 的序列图。

图 13-3 分区的 pmark 序列图

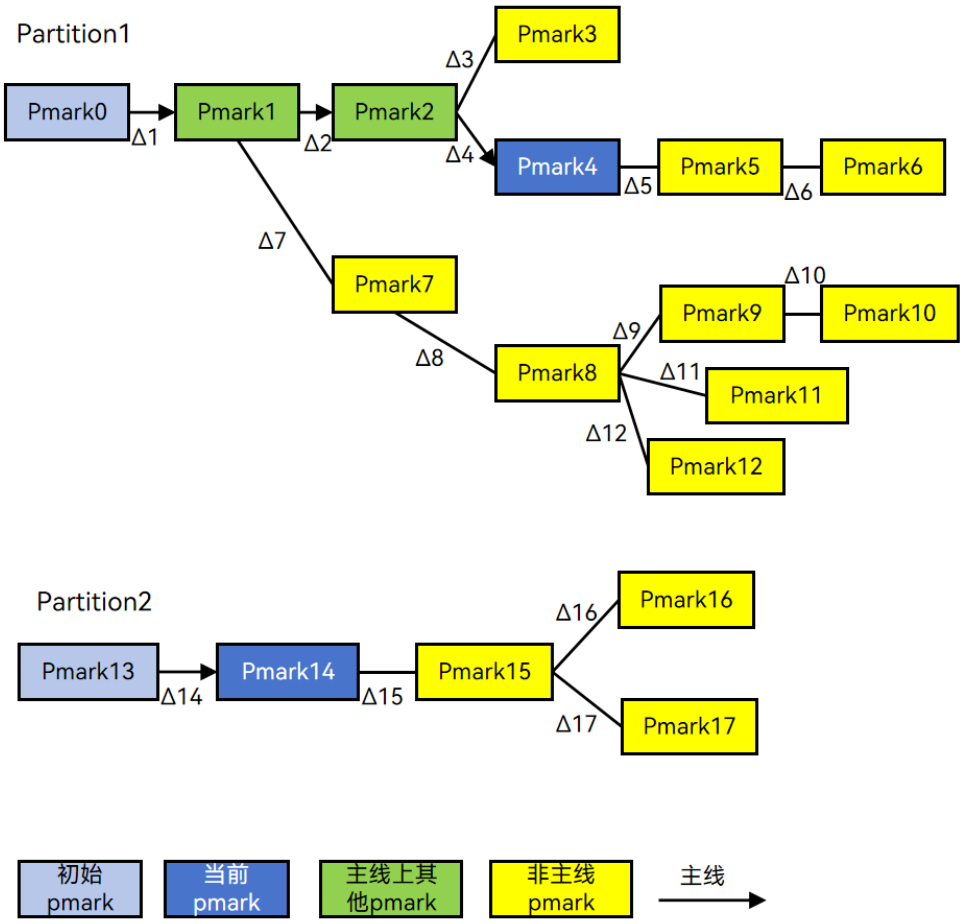


图 13-3 分区的 pmark 序列图展示了两个不同分区的 pmark 序列图，每个分区都包含了当前 pmark。在分区 1 中：

- Pmark0~Pmark3 最先创建，随后分区回滚至 Pmark2 处。
- 回滚至 Pmark2 后进行了建模操作，Pmark4 被创建。
- 创建了 Pmark5 和 Pmark6 后，分区回滚至 Pmark1。
- 回滚至 Pmark1 后进行了建模操作，Pmark7 被创建。
- 分区最终回滚（或前滚）至 Pmark4 即当前 pmark 处。
- 序列图的主线顺序是 Pmark0、Pmark1、Pmark2、Pmark4。

两个分区都可在任意时间回滚（或前滚）至任意 pmark，可以进行进一步的建模操作，或创建新的 pmark。

滚动到分区标记

使用 **gotoPMark** 将分区滚动到指定 pmark 处，滚动操作将可能导致实体被创建、修改或删除。本节介绍如何控制这些实体的报告以用于跟踪。

gotoPMark 将返回滚动操作影响的实体数量，默认情况下，还将返回如下数组：

表 13-12 返回数组

数组	说明
newEntityTags	创建的实体。滚动操作前不存在，操作后存在的实体。
modEntityTags	修改的实体。滚动操作前后均存在，但可能被滚动操作修改的实体。
delEntityTags	删除的实体。滚动操作前存在，操作后不存在的实体。

您可通过以下选项控制是否返回以上数组及其包含的数据。

表 13-13 选项参数

选项	说明
m_wantNewEntities m_wantModEntities m_wantDelEntities	控制是否返回 newEntityTags、modEntityTags 或 delEntityTags 数组。
m_wantAttrModify m_wantLoggedModify	控制 modEntityTags 数组的内容。
m_delAttrCallback m_delAttrDefTagArray	指定删除实体时，属性调用的回调函数及传递给回调函数的属性定义。
m_newEntityClassArray m_modEntityClassArray m_delEntityClassArray	控制在返回数组中包含哪些类的实体。

您还可使用 **pmarkGetEntities** 在不实际修改建模引擎软件会话的情况下查询分区回滚至指定 pmark 时，将创建、修改、删除的实体。**pmarkGetEntities** 的输入参数和输出参数与 **gotoPMark** 相同，选项也类似。

说明

- 表 13-13 选项参数中，除 m_delAttrCallback、m_delAttrDefTagArray 和 m_wantAttrModify 外，其他选项也适用于 **pmarkGetEntities**。
- 此外，表 13-13 选项参数中，除 m_delAttrCallback 和 m_delAttrDefTagArray 外，其他选项也适用于 **createPMark**。

将分区回滚到其初始 pmark 后，分区处于非活动状态，此时无法进行进一步的建模或分析操作；将分区回滚至其他 pmark 可解除该状态。当前分区不支持回滚至初始 pmark。

gotoPMark 中参数的使用场景如所示。

使用场景	使用说明
modEntityTags 数组的使用	**gotoPMark** 返回的实体数组与建模引擎软件模型相关，与应用程序并不完全一致。与 newEntityTags、delEntityTags 不同，不建议使用 modEntityTags 数组中返回的数据进行跟踪。 对于应用程序而言，modEntityTags 数组可能不包含已修改的实体，也可能包含未修改的实体，例如： ● 由于内部指针引用，布尔运算可能会导致未经几何或拓扑修改的面出现在返回数组中。 ● 对主体曲面的修改可能不会引起其面的变化，这些面也可能出现在返回数组中。
返回请求删除实体、创建实体和修改实体的数组	m_wantXXXEntities 的默认值均为 true，将任意 m_wantXXXEntities 设置为 false，可不输出对应的实体数据。例如不需要输出修改实体的数组时，可将 modEntityTags 设置为 false，以节省输出参数返回时间，降低回滚期间内存使用率。 也可设置 m_wantLoggedModify 为 true（默认值：false），仅返回通过建模修改的实体数组。
返回指定类的实体	使用 m_XXXEntityClassArray 选项控制要返回的数组中包含的实体类。指定的实体类可包含 Entity 及其子类（Attribute 除外），但如果同时指定了超类及其任意子类（例如 Curve 和 Circle），则会引发错误。
控制返回属性修改的类型	对属性的更改始终包含在 modEntityTags 数组中，您将 m_wantAttrModify 设置为 false（默认值：true），以不返回以下数据： ● 只添加了或移除了属性的实体。 ● 值没有发生变化的属性。 🕮 说明 某些场景下，即使 m_wantAttrModify 设置为 false，以上类型的数据仍然会被返回。例如，如果回滚到某个 pmark 导致属性被附加到一个之前没有任何属性的主体上，主体本身因为附加了属性而发生了某种形式的变更，即使这种变更只是属性的添加。

使用场景	使用说明
删除实体时指定属性回调	当实体因为回滚到指定的 pmark 而被删除时，附加在这些实体上的属性也会被删除。您可以使用 m_delAttrCallback 选项来指定一个回调函数，该函数会在属性被删除之前被调用。例如，您可以使用此回调函数来释放您的应用程序中由属性内容指向的内存。 使用 m_delAttrDefTagArray 可设置在删除指定属性定义的属性时，才调用属性回调函数。 📖 说明 避免在 m_delAttrCallback 定义中使用 **attrGetOwner**。因为在调用属性回调时，可能已经删除了属性的所有者。

删除分区标记

使用 **deletePMark** 删除 pmark，并返回不能被删除的 pmark，包含以下几种类型：

- 初始 pmark；
- 会话标记使用 pmark；
- 具有多个直接后续 pmark 的 pmark。此种情况下，需将其后续 pmark 也放在删除数组中（可保留一个 pmark），但需确保这些后续 pmark 本身也可被删除；
- 若当前 pmark 后面任一 pmark 无法被删除，则当前 pmark 无法被删除。

删除一个 pmark 时，若需要读取回调函数数据以合并 pmark 两侧的更改（增量），可能会需要额外的磁盘空间。例如，在图 13-3 分区的 pmark 序列图中，如果删除了分区 1 中的 Pmark5，那么 n5 和 n6 将会被合并，以保持 pmark 序列图形的连通性。在这种情况下，合并操作可能需要额外的磁盘空间来存储整合后的数据。因此，在删除 pmark 之前，可能需要考虑磁盘空间的管理和分配。

更新分区标记

使用 **partitionAdvancePMark** 重置当前 pmark，使其代表指定分区的当前状态，该操作将覆盖该指定分区存储在 pmark 中之前的状态。相比设置新的 pmark 再删除历史 pmark 而言，**partitionAdvancePMark** 减少了 delta 回调的访问，更为高效。

以下 pmark 不能被更新：

- 初始 pmark；
- 会话标记使用的 pmark；
- 另一个 pmark 之前的 pmark（回滚之后）。

📖 说明
这些函数只能在调用 **deltaRegisterCallbacks** 启用分区回滚后才能使用。

pmark 接口函数

以下接口与 pmark 相关：

接口	说明
isPMark	查询指定对象是否为 pmark。
deletePMark	删除指定 pmark。
pmarkGetPartition	查询 pmark 所属分区。
pmarkGetMarks	查询哪些会话标记使用了指定 pmark。
pmarkHasUsedByMark	查询指定 pmark 是否被会话标记使用。
pmarkGetFollowing	查询指定 pmark 之后的所有 pmark。
pmarkGetPreceding	查询指定 pmark 的前一个 pmark。
pmarkGetEntities	查询从指定 pmark 滚动到目标 pmark 过程中将会创建、修改和删除的实体。
gotoPMark	将分区滚动至指定 pmark 处。
markGetPMarks	查询指定会话标记使用的 pmark。
partitionGetPMark	查询指定分区的当前 pmark，即最近设置或回滚到的 pmark。
partitionGetInitialPMark	查询指定分区的初始 pmark。
partitionGetPMarks	查询指定分区所有 pmark。
createPMark	在指定分区中创建 pmark。
partitionGetPMarkById	查询指定分区中具有指定标识符的 pmark。
partitionAdvancePMark	更新最近的 pmark 以记录指定分区的当前状态 。

13.5.2 增量

分区回滚机制通过存储 pmark 之间的增量（delta）来实现，delta 记录了一个 pmark 移动到相邻 pmark 时需要创建、修改、删除的实体。Delta 通过回调函数写出，在应用程序中存储，在回滚时读取，创建新的 pmark 时，写入零长度的 delta。

使用 **deltaRegisterCallbacks** 注册分区回滚回调函数，开启分区回滚功能。分区回滚回调函数应在会话启动前调用，在会话停止时关闭分区回滚并注销分区回滚回调函数。

Delta 回调函数

Delta 包含一个 pmark 到相邻 pmark 所需的更改信息。Delta 是单向的，即它们只保存从 pmark A 到 pmark B 的变更信息，而不保存返回信息。当建模引擎软件执行从

一个 pmark 到相邻 pmark 的滚动操作时，它会创建一个新的反向 delta 以生成反向路线。如果磁盘空间不足，则不会执行滚动操作。

如果删除了一个 pmark，则需要将两个 delta 合并为一个。此种情况下，在删除原始 delta 之前，需先写出合并的 delta。

应用程序在编写 delta 回调时，需符合要求：应用程序需为每个 delta 分配一个正整数值。当建模引擎软件请求创建一个 delta 时，应用程序需提供该 delta 相关的下一个 pmark。

delta 回调包含如下函数：

函数	说明
openForWrite	创建一个与指定 pmark 相关的 delta 进行输出。
openForRead	打开指定 delta 进行输入。
close	关闭指定 delta。
write	将字节数组中的指定字符串写入到指定 delta 中。
read	从指定 delta 中读取字符串到字节数组中。
deleete	删除指定 delta。

13.5.3 会话回滚

会话回滚提供了更改整个会话状态的能力，它通过在每个分区中同时设置 pmark 来协调会话中的分区。

会话标记

会话标记在建模引擎软件中的对象是 MARK，它记录了整个会话中各个阶段的状态。当会话回滚到某个会话标记时，会话中的实体与创建该标记时的状态完全相同（属性定义和某些不受回滚影响的会话参数除外）。

当前会话标记是最近创建或滚动到的标记，在创建或滚动到会话标记后，会话会立即处于当前会话标记处。当任何模型实体被修改、创建、删除，或者创建了 pmark 或滚动到 pmark 时，当前会话标记（以及标记序列）不会改变，但是会话不再处于当前会话标记处。

创建会话标记

使用 **createMark** 创建一个会话标记。会话标记通过在所有尚未处于 pmark 的分区中创建新的 pmark 来记录会话中所有分区的状态。

与分区标记不同，会话标记形成一个线性序列，而不是分支树。创建会话标记时，会首先删除先前当前会话标记之后的任何会话标记。

回滚到会话标记

使用 **gotoMark** 将会话返回至创建指定会话标记时的状态，当前分区设置为创建会话标记时的当前分区。如果会话前滚选项处于禁用状态，回滚到某一会话标记时，它之后的会话标记和新建 pmark 都将被删除。

会话参数不受分区回滚影响，但会通过会话回滚操作进行回滚，除以下参数外：

- 接口检查打开/关闭；
- 日志记录打开/关闭；
- 前滚打开/关闭；
- 标记限制。

属性定义一旦创建将始终存在于建模会话中，不受会话回滚影响。

删除会话标记

使用 **deleteMark** 删除指定会话标记，创建该会话标记时创建的 pmark 和其他会话标记依旧保留。

会话回滚对分区的影响

当会话回滚到创建分区之前的会话标记时，分区的处理策略如下：

- 开启会话前滚时：保留分区，将其置于初始 pmark，允许对其进行分区滚动。
- 关闭会话前滚时：删除分区及其所有 pmark。

会话回滚接口函数

其他与会话回滚相关的接口如下：

接口	说明
isMark	查询指定会话标记是否存在。
markGetFollowing	查询指定会话标记的下一个会话标记。
markGetPreceding	查询指定会话标记的前一个会话标记。
markGetPMarks	查询指定会话标记使用的 pmark。当会话回滚到该会话标记时，设置该 pmark 为当前 pmark。
pmarkGetMarks	查询使用了指定 pmark 的会话标记。
pmarkHasUsedByMark	查询指定 pmark 是否被会话标记使用。
getCurrentMark	查询当前会话标记，以及会话是否处于该标记。

13.6 组

在建模引擎软件中，组（Group）是实体的集合，支持应用程序在模型内组装和跟踪实体集，以补充建模引擎软件的数据结构。您可使用 **createGroup** 在指定模型中创建一个组，并通过 **groupAddEntities** 和 **groupRemoveEntities** 在组中添加或删除实体。

组特征

每个组都属于指定的模型，并且具有指定的类别。一个组只能包含其所属模型中的实体。在使用 **createGroup** 创建组时，通过 entityType 指定组的类别，也决定了组可以包含的实体类别，指定单一类别可以防止不适当的实体被添加到组中。

同一模型可以属于多个组，这些组可以是相同的类别或不同的类别。同样，特定的实体可以属于多个能够包含该实体的组。

组也是实体的一种，所有用于管理实体的通用函数都可以应用于组。

实体标志

您可使用 **createGroup** 中的 m_labels 和 m_labelledEntityTags 为组内不同实体分配标志（label），以便于识别和结构化组的内容。例如，将一个实体的侧面标记为“1”。实体标志具有以下特性：

- 实体标志不是唯一的，例如可将相同的标志分配给任何实体的顶部面。
- 每个组的实体标志是独立的，如果一个实体存在于多个组中，在一个组中设置其标志不会影响它在其他组中的标志。
- 同一组中的多个实体可以被分配相同的标志。例如，为了便于识别，一个实体的所有“侧面”都可以标记为“1”。
- 实体标志在传输和接收过程中是持久的，未设置的实体标志将初始化为零。

相关接口函数

组相关的接口函数如下表所示：

接口	说明
createGroup	在指定模型中创建一个组，并将一组实体添加至该组。
groupAddEntities	在组中添加实体。
groupRemoveEntities	删除组中的实体。
groupHasEntity	查询指定实体是否包含在指定组中。
groupFindEntities	查询组中包含的实体，可选择是否通过实体标签来查找。
modelGetGroups	查询模型中包含的组。

13.7 存储

建模引擎软件的归档功能主要是将软件内部存储的数据保存到外部存储设备中。应用程序可以将一个或多个模型，或者一个分区的内容，归档到一个单独的逻辑文件或数

据块中，以便于适配应用程序使用的归档系统（例如主机计算机上的受控目录结构，或某种数据库）。

应用程序需跟踪归档所使用的键，建模引擎软件并不支持从归档中删除模型或查询用于归档模型的键。同时，应用程序提供的回调函数也需避免将归档数据写入已存在的位置（即键的重复使用）。

13.7.1 模型保存

使用 **saveModel** 将模型保存至数据逻辑块，并提供一个键字符串（key string）作为文件名或数据库中的索引。这些数据块通过回调进行管理，当需使用这些数据时，通过键进行检索。

使用模型保存函数时，需注意：

- 保存的模型不需要都在同一个分区。
- 一个模型指的是一个主体或一个装配体，从属的实体（如边和曲面）不能单独保存。

保存格式

文件的保存类型如表 13-14 文件保存格式所示，在应用程序的回调函数中提供。
表 13-14 文件保存格式

文件格式	文件类型
建模引擎软件模型文件（文本格式）。	建模引擎软件模型文件（二进制格式）。
.pmb	
.dmp	建模引擎软件模型文件（转储格式）。
.stl	模型 STL 文件（文本或二进制格式）。
.step	step 文件。
.json	● 日志文件，如操作日志文件，调试报告日志文件。 ● 边界表示文件。

保存版本

模型文件支持在不同版本的建模引擎软件中传输，您可将模型文件归档为新版本的格式，并在旧版本中打开。设置 **PSGMApiModelSaveOption** 中的 m_saveVersion 指定使用的版本。

保存内容

归档模型时，将保存以下项目：

- 模型中包含的所有拓扑和几何实体。

- 每个实体的用户字段（可选）。
- 属于模型的任何组。
- 属于模型中实体的属性。
- 网格数据。
- 属于模型中实体的标识符。

保存网格数据

设置 **PSGMApiModelSaveOption** 中的 m_saveMeshs 指定网格数据的保存方式：

- Separate：保存到单独文件中。
- Embedded：在模型或分区文件中保存。此时 m_saveFormat 需选择二进制格式。

保存标识符

标识符是附加到模型内所有实体（除半边和模型本身外）的整数值。一个模型内的标识符是唯一的。与标签不同，当模型被归档时，标识符会被保存，因此可以将标识符与模型键一起存储在外部数据库中，以便追踪特定的实体。

标识符的更多信息请参见标识符。

由于无法保证实体在加载到另一个会话时具有相同的标签，当应用程序需要在不同的建模会话中引用归档的特定实体时，可使用使用标识符进行检索。使用 **getIdentifier** 查询实体的标识符。

标识符与标签的差异如表 13-15 标识符与标签的差异所示。
表 13-15 标识符与标签的差异

差异点	标签	标识符
独特性：	在会话中唯一。	在模型中唯一。
模型归档时：	不归档。	随模型归档。
模型变更时：	不变更。	在拥有实体之间转移实体时发生变更。

13.7.2 模型加载

使用 **loadModel** 根据指定的键字符串在逻辑文件或数据块中搜索并加载一个或多个模型。

加载模型时模型是加载到当前分区中的，若同时加载了多个分区，则按加载时的顺序进行恢复。

用户字段

若要加载与模型实体一起存档的用户字段，存档用户字段的长度必须与当前会话设置的长度相同或更短（通过 **start** 设置）。

若存档的用户字段长度超出当前会话设置的用户长度时，可在 **loadModel** 中设置选项 m_loadUserFields 为 false，以不加载相关用户字段。模型加载后，建模引擎软件将为模型的实体创建当前会话所设置长度的用户字段，并置为 0。

用户字段的更多信息，请参见 15.2.4 用户字段。

属性

如果模型附加了用户定义的属性，属性及其定义将与模型一起保存。在加载这些模型到当前会话时：

- 若当前会话不存在相同名称的属性定义，在加载模型时将自动加载这些属性的属性定义；
- 若当前会话存在属性定义，且与归档模型的属性定语相匹配，则可正常进行模型加载；
- 若当前会话存在属性定义，且与归档模型的属性定义不匹配，则可通过 m_attrDefMismatch 控制处理方式，包括如下两种方式：
  - Fail：加载模型文件失败；
  - Ignore：不加载不匹配的属性定义及相关属性。

📖 说明
系统属性定义之间不会产生冲突，可随模型一起加载。

13.7.3 分区保存

使用 **savePartition** 保存分区，保存格式与模型保存的格式相同，具体请参见保存格式。轻量型分区不会被保存。

保存历史分区

默认情况下，**savePartition** 只保存指定分区的当前状态，当需要保存分区的历史状态时，可设置 **PSGMApiPartitionSaveOption** 中的 m_saveDeltas 和 m_pmarkArrays。

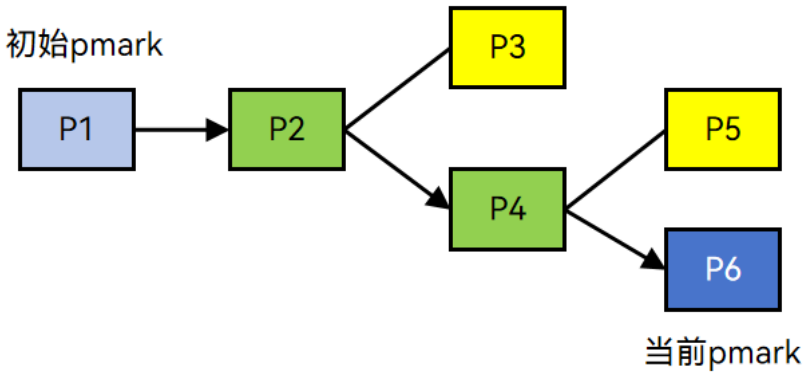
- m_saveDeltas：保存历史分区的基础控制。
- m_pmarkArrays：保存历史分区的辅助控制，可指定保存或不保存的 pmark。

m_saveDeltas 支持如下取值：

取值	说明
None	不保存 delta 数据。
All	保存所有 delta 数据。

取值	说明
Main	只保存主线的 delta 数据，即从初始 pmark 到当前 pmark。如图 13-4 保存主线 pmark 所示，当选择该选项时，只保存 P1、P2、P4 和 P6 的 delta 数据。
Later	不在当前调用时保存 delta。选择该选项时，将在后续调用 **saveDelta** 时保存分区的 delta 数据。

图 13-4 保存主线 pmark



当需要保存 delta 数据时：

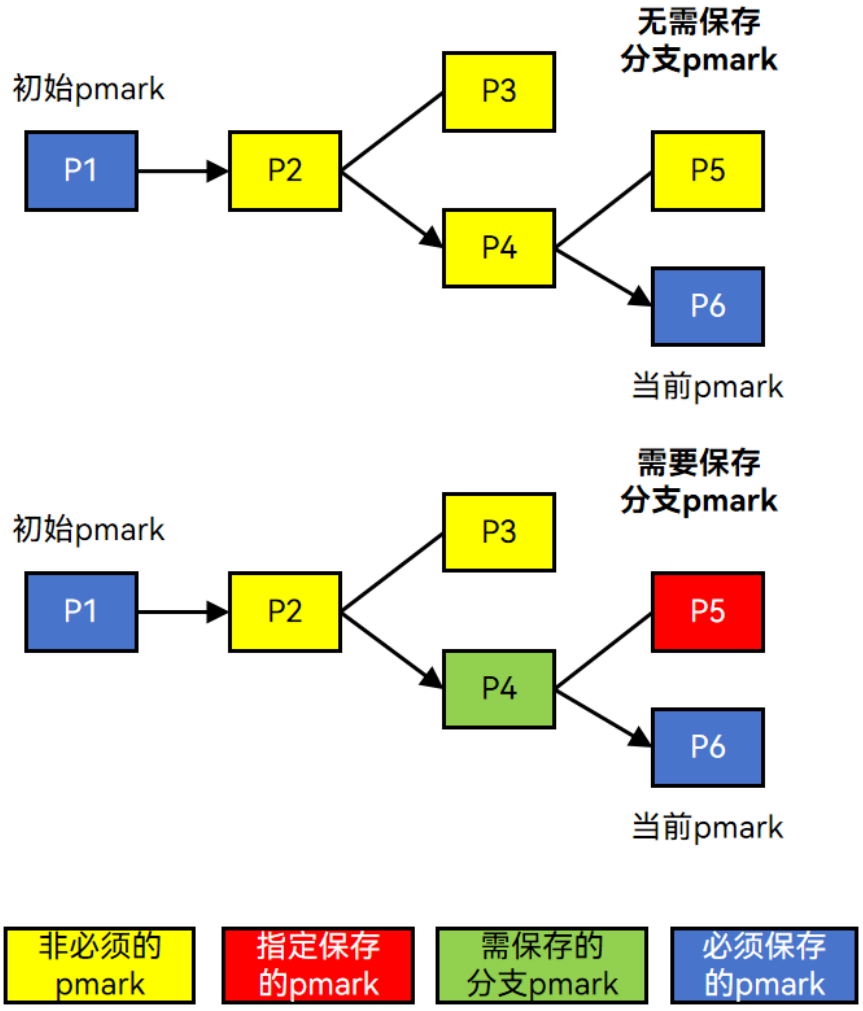
- 分区不能处于初始 pmark 状态；
- pmarks 和 delta 被写入一个独立的空间，使用相同的键；
- 使用 **pmarkGetIdentifier** 查询 pmark 的标识符，以便在新会话中跟踪。

m_pmarkArrays 包含 pmark 的数组，当 m_saveDeltas 取值不同时，m_pmarkArrays 中数组的含义也不同，如下表所示。

m_saveDeltas 取值	m_pmarkArrays 含义
None	m_pmarkArrays 中的 pmark 数组表示需要被保存的 pmark。
All	m_pmarkArrays 中的 pmark 数组表示不需要被保存的 pmark。
Main	m_pmarkArrays 中的 pmark 数组表示不需要被保存的主线 pmark。
Later	选择该选项时请勿设置 m_pmarkArrays，否则将返回错误。可在后续调用 **saveDelta** 时设置 pmark 选项。

请始终保存初始 pmark 和当前 pmark；若保存了多个分支 pmark，则需同时保存分支点的 pmark。如图 13-5 保存分支 pmark 所示，当不需要保存 P5 时，可不保存分支点的 pmark（P4）。但是，如果要保存 P5，则需保存 P4。这两种情况下，P1 和 P6 都需要保存，因为它们是初始 pmark 和当前 pmark。

图 13-5 保存分支 pmark



13.7.4 分区加载

使用 **loadPartition** 加载分区到当前会话，此时建模引擎软件将在当前会话中创建一个新分区，并将分区内容加载至该新分区，分区中的内容与保存时顺序相同，以便应用程序将其与保存的实体相关联，当前分区保持不变。

加载分区时，可通过如下选项控制加载的数据：

选项	说明
m_loadDeltas	选择如何加载 delta 数据： ● None：不加载 delta 数据； ● Later：稍后加载 delta 数据。选择该选项时，可在后续使用 **loadDelta** 加载 delta 数据。 ● Yes：加载分区中的 delta 数据。

选项	说明
m_loadAllAttrDefinitions	选择是否加载所有属性定义： ● ture：加载保存的所有属性定义； ● false：只加载分区及增量中属性相关的属性定义。分区中的属性定义立即加载，增量中的属性定义将在使用 **loadDelta** 后加载（仅在 m_loadDeltas 为 Later 时生效）。
m_allowMissingDeltas	选择是否加载没有 delta 文件的分区： ● true：加载该分区； ● false：不加载该分区，返回错误。 可通过如下方法检查加载的分区是否包含 delta 文件： ● 使用 **partitionGetPMarks** 查询分区中 pmark 的数量，当只有一个 pmark 时，分区中不存在 delta 文件； ● 使用 **partitionGetPMark** 查询分区的当前 pmark，若 delta 文件丢失，isAtPMark 将返回 false。

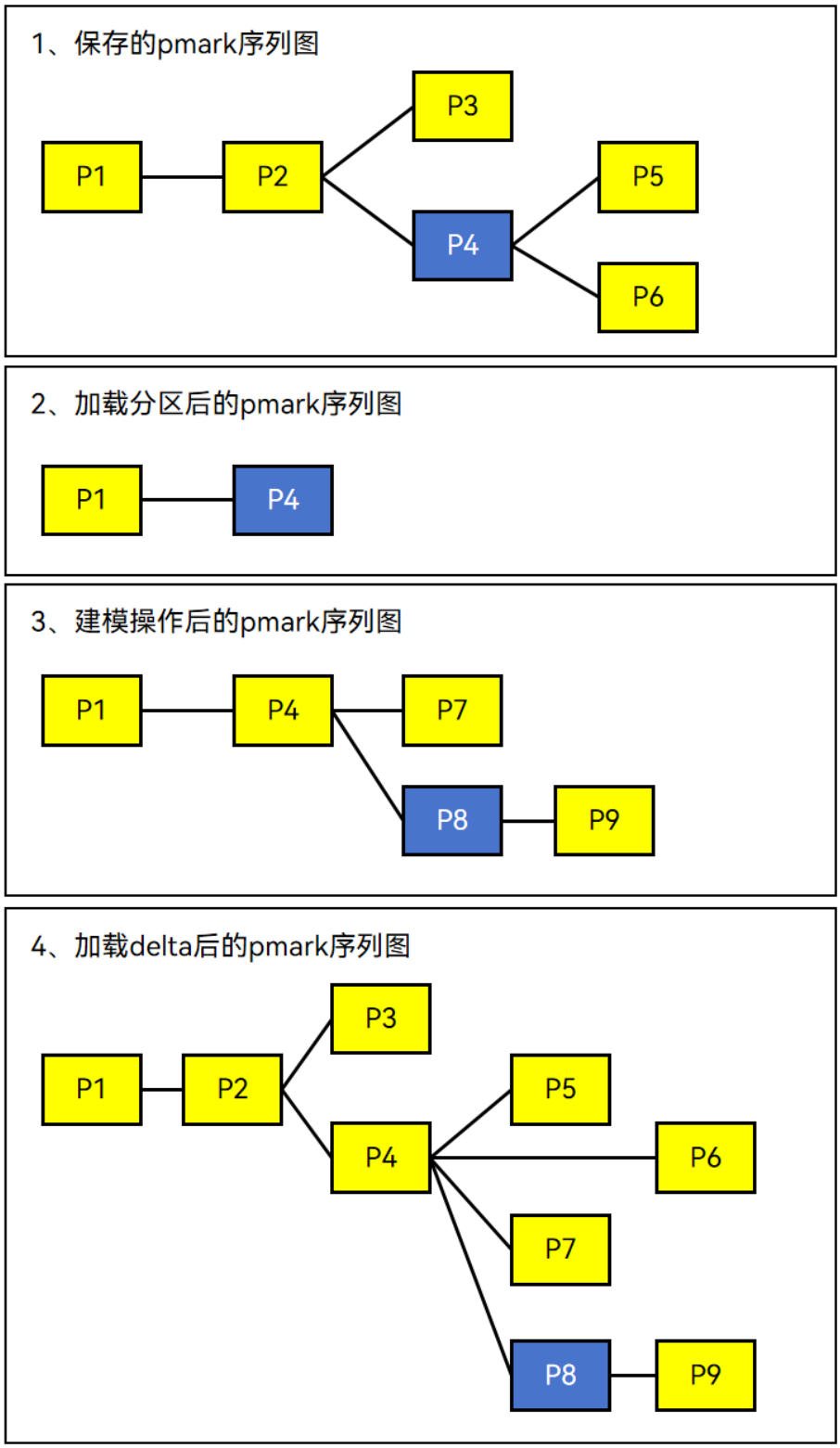
加载分区 pmark 和 delta 数据

使用 **loadDelta** 加载分区中的 pmark 和 delta 数据：

- 使用的键、加载格式、是否加载用户字段选项需与调用 **loadPartition** 时保持一致。
- delta 数据需在相同的建模引擎软件版本之间传输。
- pmark 的标签可能发生变更，请使用标识符进行跟踪。

当 m_loadDeltas 为 Later 时，只有在加载分区后没有删除 pmark，才可成功加载 delta 文件。在加载分区和加载 delta 之间，支持修改和创建 pmark，如图 13-6 传输时 pmark 的序列图所示。

图 13-6 传输时 pmark 的序列图



14 图形支持

14.1 内容简介

建模引擎软件支持广泛的图形输出功能，使用户能够在最小化内存使用的同时准确地渲染大型模型。本软件强大的模型面片化功能支持可视化、CAM、CAE、曲面分析和简化应用。本章介绍模型数据如何被渲染，以及如何通过面片化选项生成网格模型并输出到显示设备。此外，还介绍了模型的选择性和增量渲染机制。

14.2 渲染和离散简介

建模引擎软件提供模型的可视化包括渲染和离散功能，该功能可生成会话中实体、面和边的线框图片。以下内容将介绍这些过程。

渲染简介

渲染是生成表示特定实体或实体集的图形数据的过程。在渲染几何实体时，可通过 **renderGeometry** 生成独立于视图（view-independent，即不受观察者视角影响的属性）的几何图形。

在最简单的形式中，图形数据由多段线、圆形和椭圆形线段组成，这些线段近似于渲染实体的表面边界和边几何图形。用户提供的曲线容差控制渲染数据与原始模型实体的几何图形的匹配程度。

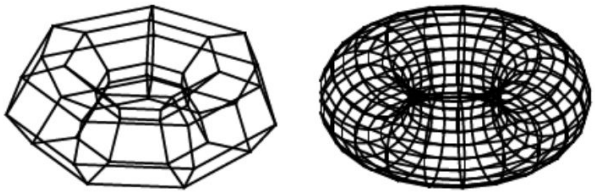
> 说明
> 由于渲染数据是根据曲线容差精度而非建模精度计算的，因此这些数据不应用于建模计算。

您可设置渲染选项用于生成其他几何数据（如面填充、倒角边界和轮廓线），并在输出几何数据的同时输出其他附加数据（如平滑度和区域分类值）。选项的功能请参见 14.3 渲染函数，选项结构请参见 14.4 渲染选项设置。

离散简介

离散是指使用多边形平面的网格来近似表示模型的表面，这些多边形平面称之为“面片”。应用程序提供的曲线容差控制每个面网格的边界与面的边几何图形的匹配程度，提供的曲面容差控制每个面网格与面的曲面几何图形的匹配程度。

图 14-1 圆环体的面片表示



建模引擎软件支持以下方式输出面片信息：

- 通过 GO（Graphical Output：图形输出）输出：调用 **renderTopoFacet** 将每个面片定义为闭合多边形，再通过 GO 接口输出面片信息。输入选项结构 **PSGMApiRenderTopoFacetOption**，其子结构如表 14-1 PSGMApiRenderTopoFacetOption 子结构所示。
- 以表格方式输出：使用 **discreteTopology** 返回模型面片拓扑的表格表示以及相关的几何信息。这些信息根据连接的面片、面片带、半边和数据索引定义了面片网格。输入选项结构 **PSGMApiDiscreteTopoOption**，其子结构如表 14-2 PSGMApiDiscreteTopoOption 子结构所示。

表 14-1 PSGMApiRenderTopoFacetOption 子结构

选项结构	控制项	参考信息
PSGMApiRenderTopoFacetMeshOption	控制如何生成面片网格。	14.6 面片网格生成
PSGMApiRenderTopoFacetGOOption	控制通过 GO 函数输出的数据。	14.7 通过 GO 进行离散输出

表 14-2 PSGMApiDiscreteTopoOption 子结构

选项结构	控制项	参考信息
PSGMApiDiscreteTopoMeshOption	控制如何生成面片网格。	14.6 面片网格生成
PSGMApiDiscreteTopoChoiceOption	控制以表格形式输出的数据。	14.8 离散表格输出

图形输出（GO）

当内核渲染函数调用注册的 GO 函数（openSegment、segmentFacet、segmentLine、closeSegment）时，输出图形数据。应用程序获取此图形数据后，在屏幕上生成图像或用于其他用途。

渲染选项的选择取决于应用程序的 GO 功能是否需要此类附加数据，在编写 GO 函数时，需注意如下事项：

- GO 的最小实现必须能够转换和显示（或后处理）基本渲染数据，如多段线和圆弧段。
- GO 的实现应该编写成能够忽略它不理解的渲染数据，并且在没有始终接收到额外的渲染数据时能够合理地表现。这允许对 GO 的最小实现进行后期增强，并引入新的渲染选项。

14.3 渲染函数

建模引擎软件提供 **renderGeometry** 接口用于渲染功能，支持独立于视图的几何体渲染。

14.3.1 实体绘制

通过 **renderGeometry** 绘制几何实体（如 B 曲线和 B 曲面）。

重复项

实体出现（entity occurrence）是不重合的，当向这些函数传递实体数组时，该数组不应包含重复项，除非同时也提供了转换实体的数组。

出现次数

几何或拓扑实体数组中的每个实体都由一个出现次数来标识。

这个出现次数会被传递给 GO，并在调用 openSegment 和 closeSegment 时作为 lineTypes 数组的第一个元素输出。该值等于源实体的数组索引加 1，即 GO 出现次数是从 1 开始输出的。当面或边出现在拓扑实体数组中时，会为拥有它们的实体调用 openSegment，并传递一个出现次数为 0 的值。

出现次数允许您将实体出现与在单个渲染函数调用内产生的 GO 函数调用序列相关联，可帮助您跟踪实体信息。例如当您按交错方式输出面时，可通过出现次数查看面与主体关联信息。有关输出交错数据的更多信息，请参见 14.7 通过 GO 进行离散输出中的“m_goInterleaved”。

14.3.2 渲染质量和性能的控制选项

renderGeometry 函数允许在渲染模型中的曲线时指定容差。这些近似容差用于控制曲线的多段线（polyline）表示的准确性。

容差的更多信息请参见 14.5 容差显示。

14.4 渲染选项设置

以下内容介绍渲染函数 **renderGeometry** 的选项设置。

14.4.1 **renderGeometry** 选项设置

本节介绍 **renderGeometry** 的选项设置。

m_boundary

值	描述
Yes	渲染曲线边界（默认值）。
No	不渲染曲线边界 。

m_param

值	描述
No	无参数填充（默认值）。
Yes	根据 U/V 方向的填充间距进行参数填充。

m_bcurve

值	描述
Polyline	将 B 曲线输出为折线（默认值）。
Bezier	以贝塞尔形式输出 B 曲线。
Nurbs	以非均匀有理 B 样条曲线形式输出 B 曲线。

14.5 容差显示

渲染函数 **renderGeometry** 允许用户指定容差，这些容差在渲染部件中的曲线时使用。这些近似容差控制曲线多段线表示的准确性。

渲染函数中设置的曲线近似容差会影响软件的输出质量和性能。

曲线近似容差

renderGeometry 中渲染的质量和性能由以下选项控制：

选项	描述
m_curveChordTol	曲线与其对应面片边之间的弦容差（使用模型单位），即弦到其近似曲线的距离上限（默认值：0.0） 。

选项	描述
m_curveChordMax	弦长的最大长度，即用于近似曲线边的弦的长度上限（默认值：0.0）。
m_curveChordAngle	相邻弦之间的最大角度，以弧度表示（默认值：0.0）。

这些选项控制着曲线作为多段线的表示方式，通过 GO 输出或在渲染函数内部传递，它们还控制着一般曲面上轮廓线的准确性。建模引擎软件所选的默认值旨在针对正常的显示目的优化性能和内存使用，请根据需要设置容差值。

说明

- 这些容差的值，尤其是弦容差，可能会影响渲染函数的性能，尤其是在包含复杂几何形状的模型上。
- 由于算法内部使用多段线近似来生成曲线，因此在图像中可能会出现与弦容差大小相等或更小的明显不一致性。通过减小弦容差，可以消除这些不一致性。

14.6 面片网格生成

本章介绍如何通过选项结构中的字段控制面片网格的生成。

- **PSGMApiRenderTopoFacetMeshOption** 由 **renderTopoFacet** 使用，后者通过 GO 输出每个面片的拓扑结构。更多信息请参见 14.7 通过 GO 进行离散输出。
- **PSGMApiDiscreteTopoMeshOption** 由 **discreteTopology** 使用，后者返回模型面片拓扑结构的表格表示。更多信息请参见 14.8 离散表格输出。

默认值

PSGMApiRenderTopoFacetMeshOption 和 **PSGMApiDiscreteTopoMeshOption** 中的默认值展示了如何为应用程序返回面片表示的基本拓扑数据以及相关的几何数据。关于选择返回哪些表格的更多信息，请参见选择返回表的类型。

设置容差值

部分面片网格选项可以接受一个容差值。如果没有提供明确的容差值，建模引擎软件可以自由计算其内部容差。

14.6.1 网格生成选项

以下内容介绍 **PSGMApiRenderTopoFacetMeshOption** 和 **PSGMApiDiscreteTopoMeshOption** 中的字段。

m_match

选项	描述
Trimmed	面片网格不考虑相邻面，这会导致网格中可能包含重叠的面片或面片之间存在间隙。
Geometry	相邻面之间的面片网格将精确相连，但在拓扑上是不相连的。
Topology	相邻面之间的面片网格精确相连，并具有相同的拓扑结构；位于一个网格边界上的面片顶点将始终与相邻网格边界上的顶点匹配。

几何匹配

几何匹配选项（Geometry）可以在为简单的渲染应用程序生成多面体输出时使用。

在几何匹配模式下，相邻的多面体网格的边界将精确匹配，但它们在拓扑上是分离的。如果正在使用 **discreteTopology** 来返回表格形式的多面体数据，那么 m_coedges 表只匹配属于同一面的半边（coedge）。

面片网格在使用较粗糙的弦容差进行面剖分时可能无法匹配。建模引擎软件会检查面是否包含非常接近外表面边界的内环，如果存在，建模引擎软件会在局部降低其弦容差，以确保边界弦不会与任何内环相交。如果启用了几何匹配，精细化的网格边界将不再与相邻面的相对粗糙的边界完全匹配（精细化边界的弦与粗糙边界的弦之间会出现小的三角形间隙）。您可以使用拓扑匹配来防止这种间隙的产生。

拓扑匹配

拓扑匹配选项（Topology）解决了几何匹配的微小间隙问题，它将整个实体或片状体模型的所有面作为一个单独的网格来处理。

如果 **discreteTopology** 被用来返回表格形式的面片数据，m_coedges 表描述了属于同一面的半边（指面片的边界部分）之间的匹配，以及属于相邻面边界的半边之间的匹配。

拓扑匹配选项会检查一个面片网格上的边界顶点是否总是与相邻网格的边界顶点相匹配（可能通过拆分相邻面片网格中的面片来满足这一条件）。

🕮 说明
拓扑匹配在单独对各个面进行剖分（即面实体被显式地作为参数提供给剖分函数）时不会产生影响。然而，当应用程序输入了一组面时，建模引擎软件会识别这组面中所有属于同一个实体的面，并对输入数组中相邻面之间的边界应用拓扑匹配。

面片裁剪

裁剪面片选项（Trimmed）将面片边界裁剪至模型边缘曲线。它不会匹配边两侧的面片，但确保这些面片之间的间隙或重叠不会超过提供的容差。

面片宽度

以下字段用于控制面片宽度的上下限。

开关字段	取值字段	说明
m_hasMinFacetWidth	m_hasMinFacetWidth	设置面片的最小宽度。打开开关后，面片会反复细化，直到每个面片的宽度小于指定的值。一旦达到这个条件，建模引擎软件将不再进一步细化面片以满足提供的曲线、曲面或平面度容差，这通常会导致面片的尺寸小于面片最小宽度。
m_hasMaxFacetWidth	m_maxFacetWidth	设置面片的最大宽度。

说明

- 如果同时设置了 m_hasMinFacetWidth 和 m_hasMaxFacetWidth，m_minFacetWidth 的值必须小于 m_maxFacetWidth。
- 在具有面片几何的拓扑结构中，m_minFacetWidth 将被忽略。

曲线弦长容差

以下字段控制面网格边界上的半边与原始边几何形状的匹配程度。

开关字段	取值字段	说明
m_hasCurveChordTol	m_curveChordTol	曲线与其对应的面片边之间的弦容差（使用模型单位），即弦到其近似曲线的距离上限。
m_hasCurveChordMax	m_curveChordMax	弦的最大长度（使用模型单位），即用于近似曲线边的弦长度上限。
m_hasCurveChordAngle	m_curveChordAngle	曲线和近似该曲线的弦（直线）之间最大角度（以弧度为单位），即在弦的两端测量的弦和曲线切线之间的角度之和的上限。

其中 3 个开关字段的默认值均为 false，表示不提供明确的曲线容差，由建模引擎软件在内部计算合适的容差值。当开关字段设置为 true 时，需要将其对应的取值字段设置为非零正值。

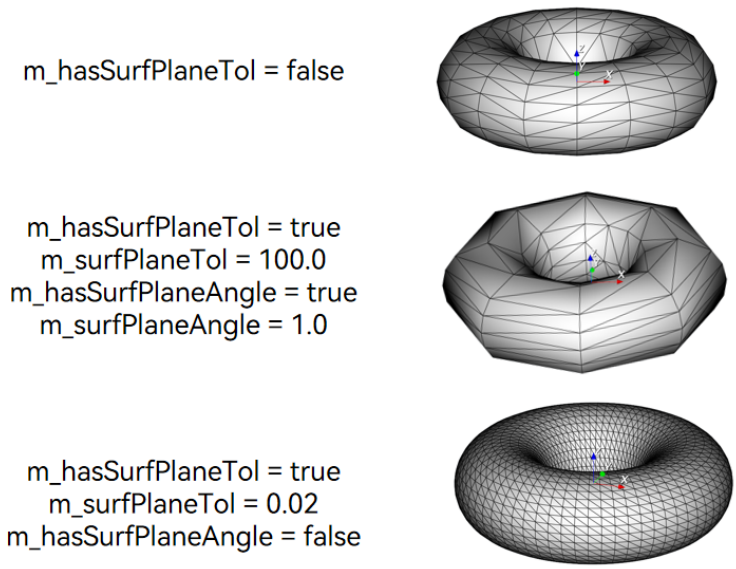
曲面平面容差

以下字段控制每个面网格与原始面的曲面几何图形的匹配程度。

开关字段	取值字段	说明
m_hasSurfPlaneTol	m_surfPlaneTol	曲面与其对应面片之间的距离容差，使用模型单位，即面片上的位置到曲面的距离上限。
m_hasSurfPlaneAngle	m_surfPlaneAngle	曲面及其面片之间的角度容差（单位为弧度），即面片下任意两个位置对应曲面上的点的法线之间角度偏差的上限。

其中开关字段的默认值均为 false，表示不提供明确的曲面容差，由建模引擎软件在内部计算合适的容差值。当开关字段设置为 true 时，需要将其对应的取值字段设置为非零正值。

图 14-2 设置示例



面片平面容差

以下字段用于控制面片坐标与公共中平面的匹配程度。

开关字段	取值字段	说明
m_hasFacetPlaneTol	m_facetPlaneTol	从面片到中平面的最大距离 。
m_hasFacetPlaneAngle	m_facetPlaneAngle	m_facetPlaneTol 和 m_maxFacetWidth 的最大比率。

平面度容差值仅在面包含三个以上边的情况下才生效。

14.7 通过 GO 进行离散输出

本章介绍通过图形输出（GO）接口输出面片几何图形。

说明
此函数为具有面片几何（网格和样条线）的拓扑提供部分支持。更多信息请参见 API 接口参考手册。

通过 GO 输出

面片网格输出由选项结构 **PSGMApiRenderTopoFacetOption** 控制，包含如下两个子结构：

- PSGMApiRenderTopoFacetMeshOption：控制面片网格的生成选项，详细信息请参见 14.6 面片网格生成。
- PSGMApiRenderTopoFacetGOOption：控制通过 GO 输出等数据类型，详细信息请参见 PSGMApiRenderTopoFacetGOOption 选项。

PSGMApiRenderTopoFacetGOOption 选项

PSGMApiRenderTopoFacetGOOption 中包含如下控制字段：

字段名	字段类型	控制项	取值范围
m_goNormals	PSGMApiFacetGONormalsType	是否输出面法线。	● No（默认值）：不输出曲面法线。 ● Yes：在面片顶点处输出曲面法线。
m_goParameters	PSGMApiFacetGOParametersType	是否输出曲面参数。	● No（默认值）：不输出曲面参数数据。 ● D0：输出顶点处的曲面参数。 ● D1：输出曲面参数以及所有一阶导数。 ● D2：输出曲面参数以及所有一阶、二阶导数。
m_goCurvatures	PSGMApiFacetGOCurvaturesType	是否输出面的主方向和曲率。	● No（默认值）：不输出面的主方向和曲率。 ● Yes：输出面片顶点处面的主方向和曲率。
m_goEdges	PSGMApiFacetGOEdgesType	是否输出边。	● No（默认值）：不输出边数据。 ● Yes：输出面片边的边实体。
m_goStrips	PSGMApiFacetGOStripsType	是否以面片带形式输出面片。	● No（默认值）：通过 GO 输出单独的面片。 ● Yes：通过 GO 输出面片，尽可能以面片带形式输出（一些单独的面片也可能被输出）。

字段名	字段类型	控制项	取值范围
m_goMaxFacetsPerStrip	int	每个面片带中的最大面片数量。	默认值为 0。
m_goInterleaved	PSGMApiFacetGOInterleavedType	是否交错输出来自不同主体的面。	● No（默认值）：在开始输出另一个主体的面片之前，先输出一个主体的所有面片。 ● Yes：面片是逐个面输出的。在存在多个实例或启用多线程的情况下，对于特定面的面片将尽早输出，因此可能会连续输出来自不同主体的面片。
m_splitStrips	PSGMApiFacetSplitStripType	是否拆分面片带。	● No（默认值）：不拆分面片带。面片带可以跨越曲面的多个周期。 ● Yes：拆分面片带以确保没有一个面片带跨越曲面的多个周期。
m_consistentParams	PSGMApiFacetConsistentParamsType	是否强制使每个面片顶点处的参数彼此之间保持在半个周期内。	● No（默认值）：不确保面片顶点具有一致的参数。 ● Su：每个面片或面片带内的一些面片顶点将在第一周期中具有参数，并且面片或面片带内的所有其他顶点将与此一致。 ● Fa：有在面片未覆盖整个周期时，才会产生与“Su”不同的结果。 — 每个面片顶点都被赋予唯一的参数值。 — 从 PSGMApiFaceFindUvbox 返回的面参数范围框内的面片顶点将具有位于该包围盒内的参数值。 — 所有其他面片顶点将与面框内的顶点参数保持一致。 — 在面片未覆盖整个周期的情况下，如果面的参数空间包围盒涵盖整个周期，则一些面片顶点可能不在面的参数空间包围盒内。

14.8 离散表格输出

建模引擎软件可以以表格格式创建面片信息，根据连接的面片带、面片、半边和顶点索引定义面网格。与通过 GO 接口输出面片不同，离散的表格输出允许您输出网格的连接信息。本章介绍如何使用 **discreteTopology** 以表格格式将面片数据返回到应用程序，以及如何控制返回的信息。

📖 说明
此函数为具有分面几何体（网格和样条线）的拓扑提供部分支持。更多信息请参见《*Geoshape 几何建模引擎软件 接口开发文档*》。

本章包含以下内容：

- 14.8.1 以表格形式返回面片数据：介绍什么是面片表格数据，以及如何控制这些数据。
- 14.8.2 拓扑信息：介绍 **discreteTopology** 返回的拓扑信息。
- 14.8.3 几何信息：介绍 **discreteTopology** 返回的几何信息。

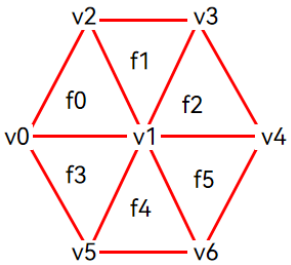
面片图形输出的相关信息，请参见 14.7 通过 GO 进行离散输出。

14.8.1 以表格形式返回面片数据

面片是将顶点连接在一起的半边的有序序列，面片带是连接面片的有序序列，面片网格是一组连接的面片或面片带。面片网格的拓扑结构可以根据面片网格中的面片或面片带、相关联的半边以及每个半边头部之间的连通性来表示。面片网格的几何结构源自模型的几何结构：面片中每个半边的头部都有一组几何信息（点、法线、参数、导数、主方向和曲率），这些信息源自原始模型中的相应位置。

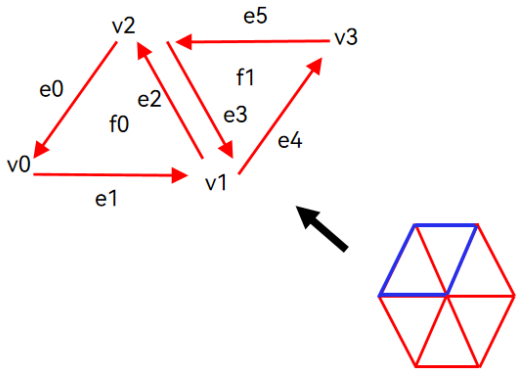
图 14-3 六边形网格图显示了由六个面片组成的六边形面片网格，编号为 f0 至 f5。这些面片将七个顶点连接在一起，编号为 v0 到 v6（为方便起见，未显示半边编号）。其中，顶点 v1 称为内部顶点。其他顶点位于网格边界上，称为外部顶点。

图 14-3 六边形网格图



显示了面片 f0 和 f1 的细节，并添加了半边信息（编号为 e0 至 e5）。当从面的外部观察面片网格时，面片周围的半边指向逆时针方向。每个半边只与一个顶点相关联：位于半边头部的顶点。

图 14-4 面片示意图



一个面片由形成其边界的半边所限定，而这些半边又指向它们所连接的顶点。可通过如下以表格的形式描述这一信息（以面 f0 和 f1 为例）：

面片	关联的半边	半边头部的顶点
f0	e0	v0
	e1	v1
	e2	v2
f1	e3	v1
	e4	v3
	e5	v2

与上面的示例不同，**discreteTopology** 返回的所有面片数据表仅关联两种类型的信息。建模引擎软件提供多种数据表，应用程序可根据需求选择需要的数据。以下内容介绍这些数据表的详细信息。

表类型

表格数据由 **discreteTopology** 以如下两种类型之一的表格返回。

表格类型	说明
索引表	索引表是简单的两列表，用于表示返回的表格数据中的一对一映射关系。表的第一列代表实际值的索引，而实际值则存储在第二列中。 索引表是通过一维数组实现的：数组的索引代表表的第一列，而值则存储在数组中。
查找表	查找表是更复杂的多列（两列或更多）表，可以在返回的表格数据中表示一对一、一对多或多对一的关系（即任何列中的数据都可以重复）。 查找表是通过结构数组实现的，其中每个结构包含两个或更多字段，数组本身的索引没有特定的含义。

选择返回表的类型

PSGMApiDiscreteTopoOption 包含两个子数据结构：

- PSGMApiDiscreteTopoMeshOption：控制如何生成面片表示的选项，详细信息请参见 14.6 面片网格生成。
- PSGMApiDiscreteTopoChoiceOption：控制如何输出面片表示并选择要返回的表格的选项，包含以下类别：
  - 拓扑信息
  - 几何信息

每个选项默认设置为不返回表格数据，应用程序需在初始化 **PSGMApiDiscreteTopoChoiceOption** 后设置对应的值以返回表格数据。

访问返回的数据

表格数据以 **PSGMApiDiscreteTopoResult** 结构返回，包含如下数据：

- m_facetCount：面片数量。
- m_stripCount：面片带数量。
- m_coedgeCount：半边数量。
- m_tableCount：表格数量。
- m_tables：返回的表格。

数据表本身在 **PSGMApiDiscreteTopoTable** 子结构中返回，该子结构包含指向数据阵列的指针，这些数据阵列定义了面片网格的面片拓扑、几何结构。

PSGMApiDiscreteTopoChoiceOption 中的选项名称与此返回结构中的字段名称匹配。因此，要返回 FacetCoedge 表时，需在选项结构中设置 m_facetCoedge 选项为“true”。

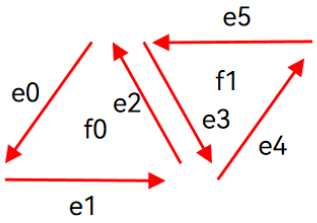
14.8.2 拓扑信息

您可以返回表示面片及半边之间关系的面片数据。

定义面片边界

面片与半边的连接关系在 FacetCoedge 表中返回。该表提供了哪些面片关联了哪些半边，因此提供了关于面片边界的信息。FacetCoedge 表属于查找表，它将属于同一个面片的半边集合分组在一起。图 14-5 返回面片和半边的关系展示了面片 f0 和 f1 的面与半边的连接关系。

图 14-5 返回面片和半边的关系



Facet	Coedges
f0	e0
f0	e1
f0	e2
f1	e3
f1	e4
f1	e5

说明
面片组（在第一列中）按数字顺序返回。

14.8.3 几何信息

本节介绍如何使用 **discreteTopology** 返回从原始模型继承的几何数据。本节中描述的每个表都是作为索引表实现的，您可以通过参考相应中间数据表值中的索引，将任何表中的数据与面片拓扑相关联。

获取几何信息

返回参数 **PSGMApiDiscreteTopoResult** 的 **PSGMApiDiscreteTopoTable** 中，包含多种数据表格，您可根据需求获取如下信息：

表 14-3 获取几何信息

几何信息	数据表	涉及步骤
坐标信息	Points	1. 从 CoedgeData 表中获取 PointIndices 表的索引。 2. 从 PointIndices 表中获取 Points 表的索引。 3. 从 Points 表中获取坐标。
面法线信息	Normals	1. 从 CoedgeData 表中获取 NormalIndices 表的索引。 2. 从 NormalIndices 表中获取 Normals 表的索引。 3. 从 Normals 表中获取法线向量。 说明 面法线指向远离原始面的方向，如果无法计算面法线，则返回零向量。

几何信息	数据表	涉及步骤
曲面参数信息	UVParams	1. 从 CoedgeData 表中获取 ParamIndices 表的索引。 2. 从 ParamIndices 表中获取 UVParams 表的索引。 3. UVParams 表中获取曲面参数信息。

14.9 拓扑拾取

使用 **pickBodies** 根据一组主体实例中的实体（面、边或顶点，或这些实体的组合）与一条射线的接近程度选取实体。如果边和顶点位于射线的指定距离内，则会拾取它们。这条射线是在与几何体相同的坐标系中定义的。如果提供了变换数组，那么将从这些变换定义的几何体实例中选择实体，而不是从几何体本身中选择。实例变换可以包含平移、镜像、旋转和均匀缩放，但不包含非均匀缩放或透视项。

在选择面时，射线必须始终与面相交。而在选择边和顶点时，射线应该与边或顶点在指定的距离内。有关控制所选实体的数量和类型的更多信息，请参见 14.9.1 实体数量和类型。

14.9.1 实体数量和类型

您可以使用 m_maxFaces、m_maxEdges 和 m_maxVertices 选项来控制您希望 **pickBodies** 返回的实体类型的最大数量。默认情况下，只返回一个面。

您可以更改这些选项来查找，例如：

查找对象	相关设置
最近的单个实体	将 m_maxFaces、m_maxEdges 或 m_maxVertices 设置为 1。
附近实体的数组	将 m_maxFaces、m_maxEdges 或 m_maxVertices 设置为任意上限（例如 20）。

当射线与面相交时，该面被选中；若需要选中边和顶点，则需通过 m_maxEdgeDist 和 m_maxVertexDist 设置边和顶点到射线的最大距离，当实际距离小于该值时，边和顶点被选中。

在选择面时，您可以使用 m_geomClasses 选项来过滤结果，以便仅选择附加到特定曲面类的面。

默认情况下，如果可选实体数量超过指定的最大值，建模引擎软件将在返回结构的 m_excessFaceCount、m_excessEdgeCount 和 m_excessVertexCount 字段中分别返回超出数量的面、边或顶点的数量。

在选择面时（m_maxFaces > 0），将 m_ignoreExcessEntities 选项设置为 true，可提高 **pickBodies** 的性能。该操作将把 m_excessFaceCount、m_excessEdgeCount 和 m_excessVertexCount 设置为 0，且在选中的面数量等于 m_maxFaces 时，停止搜索与面的进一步相交。

说明
您可以通过超出数量来判断 m_maxEdgeDist 和 m_maxVertexDist 的值是否过大。

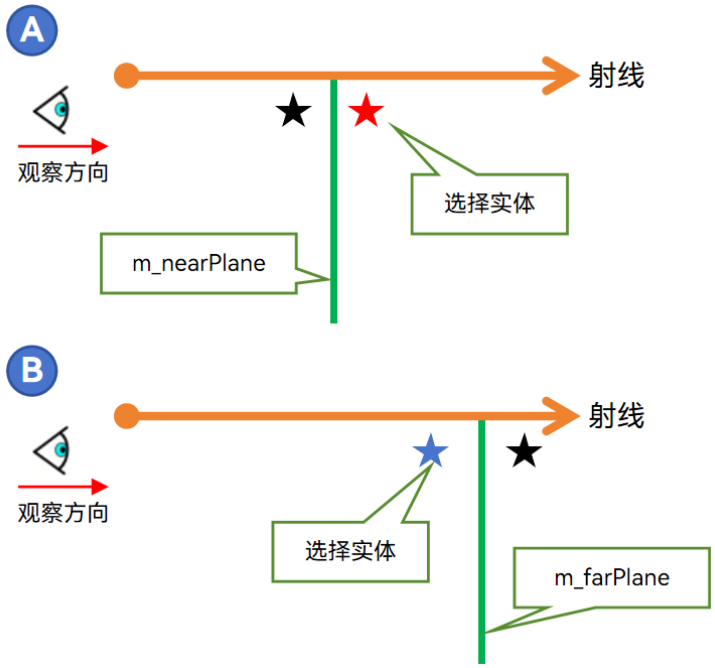
14.9.2 实体空间范围

通过 **pickBodies** 的选项 m_nearPlane 和 m_farPlane 限制哪些模型空间中的实体可以被选中。

选项	说明
m_nearPlane	是否选择近平面。设置为 true 后，仅选中指定平面后方的实体，如图 14-6 限制实体选择的空间范围的 A 示例所示，选中红色实体。
m_farPlane	是否选择远平面。 设置为 true 后，仅选中指定平面前方的实体，如图 14-6 限制实体选择的空间范围的 B 示例所示，选中蓝色实体。

选择拾取近平面或远平面，可提高 **pickBodies** 的性能，拾取效果如图 14-6 限制实体选择的空间范围所示。

图 14-6 限制实体选择的空间范围



14.9.3 实体排序

通过 **pickBodies** 的 m_method 来控制返回边和顶点的顺序，包含如下取值：

取值	说明
Axial （默认值）	轴向排序。按照射线遇到的顺序返回实体，请参见轴向排序。
Radial	径向排序。按照实体与射线的距离顺序返回实体，请参见径向排序。
Ratio	比率排序。使用轴向和径向组合方法返回实体，请参见比率排序。
Location	绝对位置排序。按照从绝对位置沿射线遇到的顺序返回实体，请参见绝对位置排序。

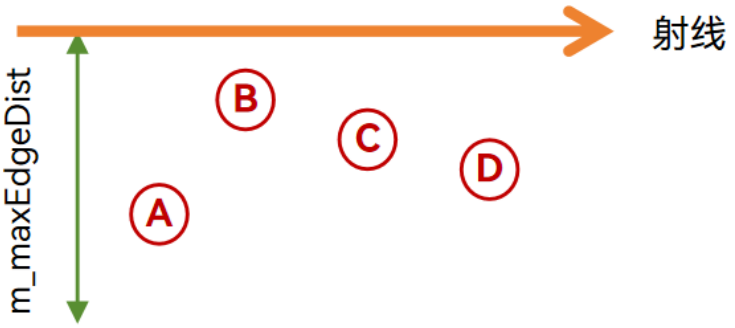
说明
Ratio 和 Radial 仅影响返回结构中边和顶点的顺序。当使用这些值时，面总是按照它们在射线上遇到的顺序返回（即轴向排序）。

轴向排序

pickBodies 返回结构中的默认实体排序方法称为轴向排序。使用此方法时，实体将按照它们在指定射线上相对于该射线的方向上的相对位置顺序返回。

如图 14-7 按顺序返回实体所示，当前有 4 条边（顶端视角）位于指定射线的 m_maxEdgeDist 范围内，当选择了 Axial 方法时，返回顺序为 ABCD。

图 14-7 按顺序返回实体



径向排序

根据边或顶点与射线之间的径向距离返回实体。径向排序方法可用于为用户提供一种从屏幕上的线框图中选择边和顶点的方法，应用程序可以优先返回那些与用户光标位置最接近的实体，从而提高用户选择实体的准确性和效率。

当选择了 Radial 方法，图 14-7 按顺序返回实体返回的实体顺序为 BCDA。

比率排序

比率排序结合了轴向排序和径向排序两种方法，您需要提供一个额外的比率值。根据所提供的比率值，径向和轴向方法将以不同的比例用于计算实体的排序。

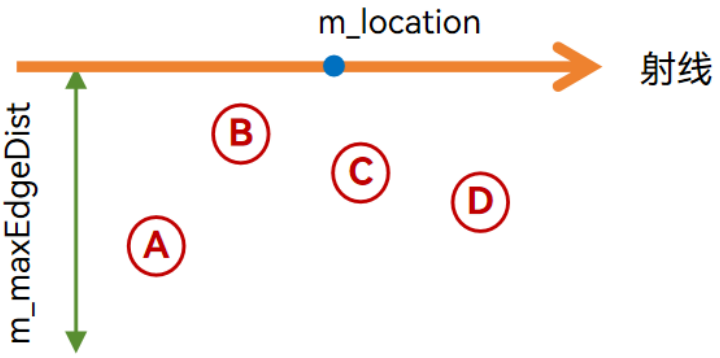
当比率值为 0 时，比率排序效果与轴向排序相同；比率值越大，其排序效果与径向排序越接近，直至与径向排序效果一致。当拾取既需要考虑实体在射线上的相对位置，又需要考虑它们与射线距离时，可通过调整比率值以调整返回顺序。

绝对位置排序

根据实体与指定位置 m_location 轴向距离的绝对值由小到大进行排序，m_location 通过 **pickBodies** 中的输入参数 ray 指定。

如图 14-8 绝对位置排序所示，当选择 Location 方法时的返回顺序为 CBDA，与使用轴向排序时的顺序不同。

图 14-8 绝对位置排序



14.9.4 最优拾取效果

本节介绍 **pickBodies** 相关的性能和可靠性问题。

近似拾取与精确拾取

默认情况下，**pickBodies** 使用模型中的曲线的近似表示，而不是精确的几何表示。您可以通过在 **PSGMApiPickBodiesOption** 中指定容差值来选择近似的程度。为了提高 **pickBodies** 的性能，建议您使用模型中的曲线近似表示。

说明
在使用近似选择时，返回的交点和距离值是近似值，不得用作建模函数的参数。

您可以通过将 **PSGMApiPickBodiesOption** 中的 m_pickApprox 选项设置为 No 来强制 **pickBodies** 使用精确几何表示而不是近似表示。

15 会话支持

15.1 内容简介

介绍建模引擎软件在建模会话期间提供的支持，包含以下内容：

- 15.2 会话支持：介绍建模引擎软件建模会话的特性以及如何接收的相关信息。

15.2 会话支持

介绍应用程序可以用来控制和获取建模引擎软件会话信息的功能。

15.2.1 会话统计信息

在会话期间的任何时间点，都可以返回建模会话状态的信息。

内存使用情况

- 使用 **getMemoryUsage** 查询模型数据结构当前占用的内存量。
- 使用 **bodyGetMemoryUsage** 查询主体的数据结构当前占用的内存量。

标签限制

您可以使用以下接口函数管理会话中的标签值：

接口函数	说明
setLimitTag	设置建模引擎软件标签号的上限，以控制标签的数值范围。
getLimitTag	查询建模引擎软件标签号的上限。
getHighestTag	查询建模引擎软件当前使用的最大标签值。
getRemainingTags	查询当前可供使用的标签数。

如果标签限制在会话期间被更改，它不会受到会话或分区滚动操作的影响，即保持在最近设置的值。

15.2.2 会话传输（快照）

会话传输文件或快照是特定时刻的会话映像，可以加载该映像以重新创建会话。通常用于如下场景：

- 防止会话意外停止时丢失数据。可在会话中的重要点进行快照，以便在会话意外停止时不会丢失到该点为止所做的工作。
- 在会话之间保留数据。可创建快照并停止会话，然后通过恢复快照恢复工作。
- 保留会话，以便快速重现故障。

保存会话

使用 **save** 创建会话的快照。您可使用 m_transmitFormat 选项选择文件的格式，包括：文本、机器依赖二进制、机器独立二进制、调用注册函数。

您还可以分别使用 m_transmitUserFields、m_transmitDeltas 和 m_transmitMarks 选项选择是否保存用户字段、pmarks、会话标记和增量信息。有关详细信息，请参见 **save** 的接口说明。

恢复会话

使用 **load** 恢复快照到指定版本，该函数中的可用选项请参见 **load** 的接口说明。

恢复快照将对会话产生如下影响：

- 将删除会话中以前存在的所有实体。
- 新分区和实体与保存快照时相同。
- 如果保存快照时 m_transmitMarks 选项设置为 All，所有会话标记与保存快照时相同；否则无可用的会话标记。
- 如果快照保存时 m_transmitDeltas 选项设置为 All，所有分区中的所有 pmark 都与快照保存时相同；否则只有初始 pmark 可用。
- 当使用会话标记保存会话时，可使用 **gotoMark** 将恢复的快照回滚到以前的状态。

会话传输版本

使用 **loadVersion** 返回建模引擎软件的版本信息，在通过指定文件加载会话时使用该信息。

15.2.3 日志文件

日志文件是对建模引擎软件函数接口调用及其参数的记录。开发人员可以检查日志文件以识别应用程序代码中的错误（日志文件通常以文本格式存储）。

说明
日志文件记录了大量信息，可能会影响软件性能。日志文件可在会话中打开或关闭，若应用程序需要在任何阶段创建日志文件，则必须在会话开始时启用日志记录，并在完成记录后关闭日志记录。

日志记录由以下动作控制：

- 将文件名传递到 **start**：打开文件并开始日志记录。
- **setJournalling**：在会话中打开和关闭日志记录。
- **getJournalling**：查询当前是否启用日志记录。

向应用程序返回错误的函数会记录错误码及错误级别（错误处理程序长时间跳过内核时除外）。

向日志文件添加注释

启用日志记录时，使用 **writeLogRecord** 将注释添加到日志文件中，该日志文件可用于将内核函数调用链接到应用程序正在执行的操作。

15.2.4 用户字段

用户字段用于存储应用程序所需的数据，由多个字节的内存组成，分配给会话中的每个实体。可以使用 **setUserField** 和 **getUserField** 分别设置和读取每个实体的用户字段中包含的数据。

用户字段和属性类似，都是用于存储会话中实体的信息数据，其特性对比如下：
表 15-1 用户字段和属性对比

对比项	用户字段	属性
性能	占用内存更小，速度更快。	-
字段长度	固定长度（最大 16 字节）。	可自定义长度和类型。
关联实体	与会话中每一个实体关联。一旦启用了用户字段，会话中的每个实体都将附加该字段，且无法为实体附加其他用户字段。	属性定义指定了属性可以存储的数据的顺序和类型。支持将任何组合和数量的属性附加到单个或多个实体上，但是每个定义的属性只能附加到一个实体（类 6 和类 7 的属性除外），并且只有特定类别的实体才能拥有属性。

对比项	用户字段	属性
数据继承	当实体被复制或拆分时，用户字段只保留在原始实体上，新实体的用户字段为空。	当实体被复制或拆分时，新实体会自动继承原始实体的属性数据。
数据加载	加载模型到会话时，可选择不加载与模型一起保存的用户字段。若选择加载用户字段，则需遵循以下规则： ● 当前会话没有使用用户字段：无法加载模型中的用户字段。 ● 当前会话存在使用的用户字段：只能加载长度相同或更短的用户字段。	加载模型到会话时，自动加载模型的属性，若存在与当前会话冲突的属性定义，则模型加载失败。

综上所述，如果需要将数据附加到会话中大部分实体时，可选择用户字段。反之，如果所需要存储的数据只与单个实体相关，那么将其作为该特定实体的属性来存储会更加节省内存空间。

用户字段通过向 **start** 传递所需的长度来启用，当前设置通过 **getUserFieldLength** 返回。

有关属性的更多详细信息，请参见 13.3 属性。

16 错误处理

16.1 内容简介

本建模引擎软件的所有接口都会返回错误码，这些错误码用于指示操作是否成功。本章将介绍如何根据返回的错误码处理本建模引擎软件生成的错误，以及如何向本建模引擎软件注册错误处理函数。

本章包含以下内容：

- 16.2 错误处理：描述了如何处理错误和失败状态码；

16.2 错误处理

应用程序必须拦截并处理本建模引擎软件返回的错误，本章将详细介绍如何处理这些错误。

当本软件的接口执行失败时，会返回一个非零错误码。例如，如果调用 **createSolidCylinder** 时使用负半径或高度，它将返回 GeomGenRadiusLessZero。请注意，在发生错误时，任何返回参数的值都是未定义的，不能被使用。

当接口未能按预期执行任务，需要返回更详细的问题诊断时，会返回一个失败状态码。如果接口返回的是零错误码，则会通过其输出参数中的状态码提供有关问题的信息。

可以在接口文档的错误码列表中找到本建模引擎软件所有错误码，以及每个错误码的描述和导致接口返回该错误码的原因。

处理本建模引擎软件错误的涉及到两个决策点：

- 是否需要向本建模引擎软件注册一个错误处理函数。
- 是否需要使用异常处理（exception）。

应用程序可以向本建模引擎软件注册一个特定的函数来处理错误，这个函数被称为错误处理器，由应用程序提供，并在发生错误时被调用，以执行某些恢复任务。在执行

失败的建模引擎软件接口返回之前，建模引擎软件会自动调用这个错误处理函数。如果控制权通过执行失败的建模引擎软件接口传回给应用程序，那么建模引擎软件接口的任何返回参数的值都是未定义的。

此外，应用程序可以选择在遇到错误时是否通过 C++中的 try/throw/catch 语句（或在 C 中使用等效的 setjmp/longjmp 函数）抛出异常。如果您的应用程序使用注册的错误处理器与异常相结合，那么它在继续执行之前无需检查每个建模引擎软件接口的返回状态。

不同的错误处理方法，最终都需要采取相同的恢复操作。这个操作取决于错误的严重程度，请参见 16.2.1.1 错误严重程度。

16.2.1 建模引擎软件接口错误

为了比较不同的错误处理方式，本节提供了一个简化的 C++驱动程序，用于交互式建模引擎软件应用程序。示例程序执行在一个无限循环中，等待用户输入指令。一旦接收到指令，程序会根据指令的类型进行处理，这可能会调用一个或多个建模引擎软件接口。

处理指令的代码在 try 语句中，并配有相应的 catch 块，以处理出现的错误。这些 catch 块会通知用户指令没有执行成功，并确保应用程序准备好接收下一条指令。这可能意味着用户可以使用不同的输入数据重新执行之前的指令。

```
while (true)
{
   // await input from the keyboard/mouse/etc.
   myApp->getNextInstruction( &instruction );
   // process instruction
   try
    {
       switch (instruction.type)
      {
       case i_type_1:
       ...
       case i_type_n:
       // processing instruction initiates (at some point)  a sequence of (related) PSGM
functions
       ...
       // PSGM function calls happen here
       // deal with any errors (from PSGM or otherwise)
       if ( myApp->isError() )
       {
          errorTypeN error = myApp->getError();
          throw error;
       }
       break;
       ...
       }
     }
   catch (errorType1 error)
       {
          // take appropriate recovery action and prepare to receive the next instruction
```



```
        ...
    }
  catch (errorType2 error)
    {
        ...
    }
  ...
}
```

16.2.1.1 错误严重程度

建模引擎软件错误可以根据其严重程度进行分类，严重程度有轻微、严重和致命。错误导致的后果以及错误恢复所需的步骤取决于错误的严重程度，详情参见下表：

严重程度	错误后果	恢复措施
轻微	操作失败，但涉及的模型没有改变。	应用程序可以继续正常运行。
严重	操作中涉及的模型可能已被更改，并可能因此无效。会话中的其余模型完好无损。	应用程序需要回滚到模型的有效状态。如果不能回滚，则需要关闭并重新启动会话，然后再重新注册增量回调函数。
致命	会话已损坏，回滚无效。	应用程序需要关闭并重新启动会话，然后重新注册增量回调函数。可能需要退出应用程序。

如果应用程序在发生严重或致命错误后没有执行必要的恢复操作，那么可能会出现数据结构损坏，从而导致后续的运行时错误。因此，建议在应用程序中实现回滚功能，以便在发生严重错误后恢复会话：详情请参见 13.5 回滚。请注意，如果会话中有多个分区，需要回滚所有包含受错误影响的实体的分区。

错误码由于引发原因的不同可被定性为轻微错误或严重错误。因此，还需要检查错误的严重性才能准确诊断问题。

16.2.1.2 错误处理策略

本节描述的错误处理策略包括：

- 不注册错误处理器。
- 注册不使用异常处理的错误处理器。
- 注册使用异常处理的错误处理器。

可以通过调用 **errorRegisterCallbacks** 向本建模引擎软件注册一个错误处理函数。错误处理器必须接收一个指向 PSGMApiError 结构的指针，并且返回类型必须为 PSGMApiErrorCode。PSGMApiError 结构包含错误的详细信息，如执行失败的接口名称、错误码以及错误的严重程度。

在使用异常的错误处理器时不需要为每种类型的错误都抛出异常。可以将某些错误返回给执行失败的建模引擎软件接口，而对于其他错误则抛出异常。

错误处理器不应尝试修改当前错误的任何详细信息（比如修改 PSGMApiError 结构或返回不同的错误码）：建模引擎软件会单独存储这些信息，任何此类修改都不会产生效果。

不注册错误处理器

一般情况下，应用程序在执行某个操作时会按顺序依次调用多个建模引擎软件接口，每个接口调用完成之后应用程序会检查接口的返回值和状态。如果某次调用出现错误，后续的 if 语句不会再调用后边的接口。在出现错误时，应用程序会调用 **errorGetLast** 来获取相关详细信息，并记录这些信息以供后续使用。然后，应用程序会调用自己的函数来处理错误，该函数不需要向建模引擎软件注册。

```
// call sequence of (related) PSGM functionsPSGMApiErrorCode error =
fucntion1(arg1, ..., argn);
// check the error return code each time before continuing
if (error == NoError)
error = fucntion2(arg1, ..., argm);
if (error == NoError)
error = fucntion3(arg1, ..., argr);
...
// perform error handling after PSGM calls
if (error != NoError)
{
    bool wasError;
    PSGMApiError error;
    // get details of the error
    mySession->threadGetLastError(&wasError, &error);
    // set error information for the application and take appropriate action
    myApp->setErrorStatus(true);
    myApp->setError( “PSGM”, error.severity, ...);
    myApp->handlePSGMError(&error);
}
```

```
void myApp::handlePSGMError(PSGMApiError *error)
{
    if (error->severity == Mild)
    {
        // warn user of a mild error
        Message( "Mild error in PSGM function %s", error->function );
        ...
        // perform any further application-specific clean-up tasks
    }
    else if (error->severity == Serious)
    {
        // warn user of a serious error
        Message( "Serious error in PSGM function %s", error->function );
        // roll back the current partition (and any other affected partitions)
        getCurrentPartition( &partition );
        partitionGetPMark( partition, &pmark1, ... );
        gotoPMark( pmark1, &n_new, &new_entities, ... );
        // free allocated memory where necessary
        if (n_new)
        freeMemory(new_entities);
        ...
        // perform any further application-specific clean-up tasks
    }
```



```
    else if (error->severity ==Fatal)
    {
        // warn user of a fatal error
        Message( "Fatal error in PSGM function %s", error->function );
        // stop the current session
        stop();
        //call application function containing PSGM calls to start a new session and re-
register the frustrum and delta frustrum
        start();
        ...
        // perform any further application-specific clean-up tasks
    }
}
```

注册不使用异常的错误处理器

下面的伪代码展示了已注册但不使用异常的错误处理程序的建模引擎软件调用序列(不包含错误处理器)。

```
// call sequence of (related) PSGM
functionsPSGMApiErrorCode error = fucntion1(arg1, ..., argn);
// check the error return code each time before continuing
if (error == NoError) error = fucntion2(arg1, ..., argm);
if (error == NoError) error = fucntion3(arg1, ..., argr);
...
// call error-processing PSGM functions outside of the error handler
if (error != NoError)
{
  bool wasError;
  PSGMApiError error;
  // get details of the error
  mySession->threadGetLastError(&wasError, &error);
  // process serious or fatal errors
if (error.severity == Serious)
  {
    // roll back the current partition (and any other affected partitions)
    getCurrentPartition(&partition);
    partitionGetPMark(partition, &pmark1, ...);
    gotoPMark(pmark1, &n_new, &new_entities, ...);
    // free memory where necessary if (n_new)
    freeMemory(new_entities);
  }
else if (error.severity == Fatal)
  {
     // stop the current session
     stop();
     // call application function containing PSGM calls to start a new session  and re-
register the frustrum and delta frustrum
     start();
  }
}
```

下面的伪代码为错误处理器示例，其中错误处理器提供了相关信息并执行了应用程序特定的清理任务，但没有调用建模引擎软件的接口。这是为了将错误信息返回给应用

程序之后再调用建模引擎软件的接口，以避免建模引擎软件的接口内核处于不一致的状态。实际的执行流程类似于前面没有注册错误处理器的情况。

```
PSGMApiErrorCode myApp::handlePSGMError(PSGMApiError *error)
{
   // set error information for the application
   myApp->setErrorStatus(true);
   myApp->setError( “PSGM”, error.severity, ...);
   // report severity of error and failing function
   if (error_sf->severity == Mild)
   {
      // warn user of a mild error
      Message("Mild error in PSGM function %s", error->function);
      ...
      // perform any further application-specific clean-up tasks
   }
   else if (error_sf->severity == Serious)
   {
      // warn user of a serious error: for safety, do not call the PSGM here
      Message("Serious error in PSGM function %s", error->function);
      ...
      // perform any further application-specific clean-up tasks
   }

   else if (error_sf->severity == Fatal)
   {
      // warn user of a fatal error: for safety, do not call the PSGM here
      Message("Fatal error in PSGM function %s", error->function);
      ...
      // perform any further application-specific clean-up tasks
   }
}
```

注册使用异常的错误处理器

下面的伪代码为使用异常来注册错误处理器的建模引擎软件调用序列。

```
try
 {
   // call sequence of (related) PSGM functions
   fucntion1(arg1, ..., argn);
   // no need to check the error return code each time
   fucntion2(arg1, ..., argm);
   fucntion3(arg1, ..., argr);
   ...
 }
 ...
catch (PSGMApiError *error)
 {
      if (error->severity == Mild)
          {
             // warn user of a mild error
             Message("Mild error in PSGM function %s", error->function);
             ...
             // perform any further application-specific clean-up tasks
```



```
        }
    else if (error->severity == Serious)
        {
          // warn user of a serious error
          Message("Serious error in PSGM function %s", error->function);
          // roll back the current partition (and any other affected partitions)
          getCurrentPartition(&partition); partitionGetPMark(partition, &pmark1, ...);
          gotoPMark(pmark1, &n_new, &new_entities, ...);
          // free memory where necessary
          if (n_new)
          freeMemory(new_entities);
          ...
          // perform any further application-specific clean-up tasks
        }
    else if (error->severity == Fatal)
        {
          // warn user of a fatal error Message("Fatal error in PSGM function %s",
error->function);
          // stop the current session stop();
          // call application function containing PSGM calls to start a new session and
re-register the frustrum and delta frustrum
          start();
          ...
          // perform any further application-specific clean-up tasks
        }
}
```

注册错误处理器的优点

错误处理器允许建模引擎软件立即报告错误，并自动执行应用程序特定的错误恢复措施。抛出异常的错误处理器可以完全自动处理建模引擎软件的错误；不再需要应用程序检查建模引擎软件接口的返回值。

16.2.1.3 错误处理接口

在建模引擎软件中提供的错误查询和处理接口如下表所述。

接口	描述
errorGetLast	此接口返回上次发生的建模引擎软件接口调用的错误信息。错误信息通过 PSGMApiError 结构返回，其中包括： ● 发生错误的建模引擎软件接口的名称。 ● 错误码。 ● 错误的严重程度。 ● 无效参数的编号和名称。 ● 应用错误的实体。 如果没有错误，或错误已被清除，PSGMApiError 结构会包含以下内容： ● 请求错误信息的建模引擎软件接口。 ● NoError。 ● None。 ● PSGM_API_NULL_TAG。 ● 没有参数（No arguments）。
errorClearLast	此接口清除上次发生的建模引擎软件接口错误的信息。 在发生下一个错误之前，调用 **errorGetLast** 将不会返回任何信息。
errorRaise	此接口根据输入的 PSGMApiError 人为引发建模引擎软件错误。
errorReraise	此接口与 errorRaise 的工作方式完全相同，只是它引发最后发生的错误（即通过 **errorGetLast** 返回的错误）。
errorRegisterCallbacks	此接口用于注册作错误处理器。错误处理器必须返回一个 PSGMApiErrorCode 并接收一个指向 PSGMApiError 结构的指针。
errorGetCallbacks	如果存在当前错误处理器，则此接口返回当前错误处理器作为指向 PSGMApiErrorHandler 结构的指针；如果不存在，则返回 NULL。

16.2.2 故障状态码

一些建模引擎软件接口在未按预期方式执行后，会返回详细的错误信息。在这种情况下，不可能通过通用的 PSGMApiError 结构返回这些信息；因此，建模引擎软件接口返回错误码 NoError，并通过状态码将错误信息输出。

📖 说明
如果接口的输出参数中有 status 来判断，则接口可以通过 status 来返回错误信息。

状态码可能明确地出现在接口的输出参数中，也可能作为描述操作结果的大型数据结构中的一个字段。

对于通过状态码返回错误信息的接口，必须在应用程序中显式检查其参数的内容，以确定操作的状态。这种情况下，不会调用已注册的错误处理器。

🕮 说明
如果一个接口通过状态码来输出错误信息，则该接口将返回错误码 NoError。

16.2.2.1 故障状态码类型

常用状态码类型如下：

- **blendApplyBody** 中的 PSGMApiBlendFault
- **booleanBodies** 中的 PSGMApiBoolResultType
- **checkBody** 中的 PSGMApiCheckState
- **extrudeBody** 中的 PSGMApiTopoLocalStatus

16.2.2.2 必要处理措施

大多数情况下，接口返回失败状态码时，模型并未改变，因此不需要执行任何特定的处理操作；这与建模引擎软件接口返回轻微错误的情况类似。然而，在局部操作的情况下，模型可能已经损坏，删除它可能会导致运行时错误。具体来说，返回 PSGMApiTopoLocalStatus 的接口可能在执行失败后损坏了模型。

如果接口返回类型为 PSGMApiTopoLocalStatus 的错误状态码，建议回滚到模型的有效状态，以避免后续出现运行时错误。

对于布尔运算，返回错误状态码 Failed，表示产生了额外的实体，导致模型发生预期之外的更改。数据结构可能不会被损坏，但调用 **checkBody** 或 **checkFace** 可能会显示模型无效。如果在调用 **booleanBodies** 时返回了 Failed，则应用程序应回滚到模型的有效状态。

如果在应用程序中处理多个线程，并且返回了错误状态码，则应该回滚会话。

17 附录

17.1 术语表

英文	中文	说明
Entity	实体	实体是模型中可以被应用程序访问的项，例如拓扑实体、属性、组等。
Topology	拓扑	一个实体的各个拓扑元素(顶点，边，面等)之间通过一个连通图所体现的相互连接或连通关系。
Lump	块	基本拓扑元素，其边界是顶点、边和面的集合。
Shell	壳	基本拓扑元素，表示一个面和边的集合，该集合中的面和边相互连接。
Face	面	基本拓扑元素，表示单个曲面的有界区域。
Loop	环	基本拓扑元素，是一个面的边界。
Coedge	半边	基本拓扑元素。表示边的一个方向，可引用一条边。
Edge	边	基本拓扑元素，表示单个曲线的有界区域。
Vertex	顶点	基本拓扑元素，表示空间中的一个点。
Geometry	几何元素	实体的几何元素是指定形状和位置的点、曲线和曲面。
Point	点	几何元素，主要附着在顶点上。也可作为构造几何体附着在主体上。
Curve	曲线	几何元素，附着在模型的边或半边上，也可作为构造几何添加到主体或装配体上，也可作为孤立几何存在于分区中。
Surface	曲面	几何元素，主要附着在面上，也可作为构造几何附着在主体上。

英文	中文	说明
Construction Geometry	构造几何	可以直接附加到主体或组件上作为额外信息的几何元素（曲面、曲线和点）。
Orphan Geometry	孤立几何	未附着在任何拓扑上的几何。
Body	主体	建模的基本“单位”。一个主体由一个或多个部件组成。 🕮 说明 如果一个主体包含一个以上的部件，它就是一个不相交的主体。
Acorn Body	橡子体	一个零维的主体，包含多个孤立顶点。
Minimum Body	最小体	最简单的主体类型，一个零维的体，只包含一个孤立的顶点。
Wire Body	线框体	线框体在拓扑上是一维的，包含一组边。
Sheet Body	片状体	零厚度的主体。
Solid Body	实心体	占据有限体积的三维主体。
General Body	一般体	实体（面、边和顶点）和被实体划分为空间的三维连接区域的集合，可包含非流形/跨维度等多种类型的主体。
B-curve	B 曲线	一种参数化定义的曲线，其形状由控制顶点控制。
Spline	样条	样条是一种特殊的函数，由多项式分段定义，样条曲线/曲面由多段函数表示组成部分的各个曲线段/曲面块，且曲线各段的连续性的级别由该函数定义。样条曲线段/曲面块之间由函数的某一段过渡到另一段的参数值被称为节点。
B-spline Curve	B 样条曲线	B 曲线形式的一种拓展，由一组控制点和节点向量定义。其形状由控制点的位置和节点向量的值来确定。
NURBS（Non-Uniform Rational B-spline）	非均匀有理 B 样条	NURBS 是根据 B 样条基函数定义的样条。分配给这些基函数的系数称为控制顶点。 NURBS 有： ● 完全有理，在这种情况下，每个控制顶点都与其关联一个权值。 ● 纯多项式，在这种情况下，顶点没有关联权值。
SP-curve	SP 曲线	用曲面参数空间中方程定义的三维曲线。

英文	中文	说明
B-surface	B 曲面	一种参数化定义的曲面，其形状由控制顶点的位置和节点向量的值控制。
Chord	弦	连接曲线上两个点之间的直线段。

