工业数据转换引擎云服务(iDEE)

开发指南

文档版本 01

发布日期 2024-11-28

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 基于几何数据转换引擎将工业 CAD 模型转换至指定格式

1.1 方案概述

应用场景

由于传统的工业软件以单点工具为基础，各软件的底层数据表达形式不一致，导致上下游软件之间无法直接传递和共享异构几何数据格式。在企业进行工业设计、工业建模、工业制造的过程中，不同的建模软件和工具使用不同的模型格式，从而使得模型的跨平台使用变得困难。DEE SDK（几何数据转换引擎）能够兼容这些模型，并对商业私有格式、中立格式等进行有效地解析，并输出为通用的中立格式。用户可以指定所需的目标格式，并携带相关的参数，进行模型转换，并得到目标模型文件。

方案架构

图 1-1 DEE SDK 模型转换工作架构

DEE SDK提供封装好的各类接口，用户只需提供输入文件路径、指定输出文件的扩展名，即可将工业CAD模型转换至指定格式。

除了模型转换能力之外，DEE SDK还提供模型探索与分析、测量模型几何属性、网格划分等多种相关的模型查看、创建、修改、编辑能力，可覆盖多种下游应用场景。

方案优势

易于入手：无需下载与安装多个平台的软件，减少了对于各种格式学习和使用的时间，对操作人员来讲使用方式更为简单。

低成本：无需复杂的许可证申请过程，避免因申请各个厂家的软件许可造成的成本增加。

高效率：保证了设计、生成的连续性，提升了历史模型的可复用性，加速了产品的开发周期。

高兼容性：打通各个平台各个解决方案，实现了归一化、标准化、高兼容性的操作。

1.2 资源和成本规划

表 1-1 云服务列表

云服务	数量	规格	计费模式	预计成本
几何数据转换SDK	1	/	包年	480000元/年

以上是参考价格，具体价格以实际业务需求及官网实际情况为准。

1.3 实施步骤

准备工作

1. 开发技能
   - 熟悉C++语言，能够编写C++语言代码。
   - 熟悉CAD建模相关知识。
   - 了解OCCT几何建模相关知识。
   - 了解C++构建相关工具（cmake等）。
2. 开发环境
   - Windows 10+
     - Visual Studio 2017+或适用于Visual Studio 2017+的Microsoft Visual生成工具
     - CMake 3.7+
   - Linux：
     - gcc g++ gdb
     - CMake 3.7+
3. 许可准备

   开发和使用DEE SDK需要获取DEE SDK许可文件。

操作步骤

通过以下示例代码将一个模型文件从一种CAD格式转换为另一种CAD格式。

通过使用通用的ModelData_ModelReader和ModelData_ModelWriter，您可以处理任何支持格式的导入和导出。

步骤1 创建一个ModelData_ModelReader。

```
ModelData_ModelReader aReader;
```

步骤2 （可选）设置读（Reader）参数。

```
STEP_ReaderParameters aSTEPReaderParams;
aSTEPReaderParams.PreferredBRepRepresentationType() = STEP_ReaderParameters::AdvancedBRep;
aReader.SetReaderParameters (aSTEPReaderParams);

JT_ReaderParameters aJTReaderParams;
aJTReaderParams.LayerConversionMode() = JT_ReaderParameters::LayerFilter;
aReader.SetReaderParameters (aJTReaderParams);
```

步骤3 读取模型数据并将其数据读取到ModelData_Model对象。

```
ModelData_Model aModel;
if (!aReader.Read (aSource, aModel)) {
    cerr << "Failed to open and convert the file " << aSource << endl;
    return 1;
}
```

步骤4 （可选）访问ModelData_Model对象，并在进一步的转换和操作中使用它。

```
cout << "Model name: " << aModel.Name().ToUTF8().Data() << endl;
cout << "Number of roots: " << aModel.NumberOfRoots() << endl;
```

步骤5 创建一个ModelData_ModelWriter。

```
ModelData_ModelWriter aWriter;
```

步骤6 （可选）设置写（Writer）参数。

```
OBJ_WriterParameters anOBJParams;
anOBJParams.LengthUnit() = Base_LengthUnit::Base_LU_Centimeters;
anOBJParams.ToGenerateMtlFile() = true;
aWriter.SetWriterParameters (anOBJParams);

JT_WriterParameters aJTParams;
aJTParams.SetFileSplitMode (JT_WriterParameters::PerPart);
aWriter.SetWriterParameters (aJTParams);
```

步骤7 将模型数据转换为选定的格式（指定文件扩展名）并保存。

```
if (!aWriter.Write (aModel, aDest)) {
    cerr << "Failed to convert and write the file to specified format" << aDest << endl;
    return 1;
}
```

----结束

常见问题

Q：为什么使用SDK时，抛出了许可证相关的异常?

A：请检查许可证的有效性，例如产品名称是否匹配，是否超过有效期限，并使用有效的许可证。

Q：为什么导入文件时，出现异常?

A：请检查文件是否能在指定的路径下查找到，并检查输入文件是否为DEE SDK支持的文件类型。

Q：为什么转换后得到的结果文件中，不包含PMI的信息?

A：此参数在默认情况下被设置为false，如果您需要此信息，请使用Base_WriterParameters::WritePMI()方法更改此项输出参数为true。

1.4 附录

更多内容请参考：

表 1-2 参考附录

序号	参考内容
01	产品介绍

1.5 文档修订历史

文档版本	发布日期	修改说明
01	2024年11月	首次发布

2 基于轻量化渲染引擎构建工业渲染应用

2.1 方案概述

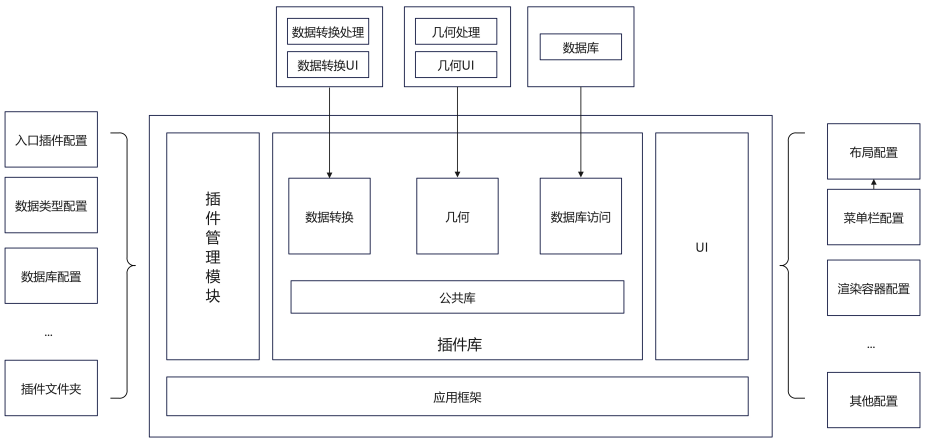
应用场景

1. 产品设计评估：在汽车、机械装备、电子产品等行业，可以利用轻量化渲染引擎快速创建产品的三维模型，并通过渲染生成逼真的外观图像。这有助于设计师从不同角度观察产品的外观效果，评估设计的美观性和合理性，及时发现并修改设计中的缺陷。
2. 结构设计与优化：对于复杂的机械结构或电子产品内部结构设计，通过对模型进行渲染，可以查看不同工况下的结构应力、变形等情况，从而对结构进行优化，提高产品的性能和可靠性。
3. 设备可视化管理：对于大型工业设备、工厂设施等，利用渲染引擎可以创建设备的三维模型，并将设备的运行状态、维护记录等信息与模型进行关联。管理人员可以通过可视化的界面实时监控设备的运行情况，及时发现设备的故障和异常，提高设备的管理效率和可靠性。

方案架构

采用插件式架构：

图 2-1 插件式 CAx 软件架构



1. 主应用框架

   构建桌面端C++应用的主体框架，包括应用的初始化、窗口管理、事件循环等基本功能。例如，使用MFC（Microsoft Foundation Classes）、Qt、Tk/Tcl等框架搭建应用的基本结构，确保应用能够正常运行并提供一个容器来集成3D查看器插件。

2. 3D查看器插件模块
   - 插件接口层：定义一组标准的插件接口，用于与主应用框架进行通信和交互。这些接口包括加载模型、控制查看器、获取查看器状态等功能的函数指针或者抽象类。通过这些接口，主应用可以在不了解3D查看器内部实现细节的情况下，调用其功能。
   - SDK集成层：在插件内部，将SDK集成进来。这包括将SDK的库文件连接到插件项目中，以及处理SDK与插件接口层之间的适配。例如，将SDK的模型加载函数封装成符合插件接口层定义的函数，使得主应用可以通过统一的接口来加载不同格式的模型。
   - 资源管理子模块：管理3D查看器插件所需要的资源，如字体、模型文件等。对于字体资源，确保在桌面应用环境下能够正确加载和显示；对于模型文件，提供资源路径的管理和转换功能，以便插件能够根据主应用的资源管理策略正确地访问模型文件。

方案优势

1. 高可扩展性
   - 易于添加新功能：随着业务需求的不断变化和发展，系统需要不断添加新的功能模块。在插件式架构中，新功能可以以插件的形式独立开发和添加，无需对整个系统的代码进行大规模的修改。
   - 支持第三方开发：插件式架构允许第三方开发者为系统开发插件，可以较快地扩展了系统的功能范围。
2. 低耦合性
   - 功能模块独立：插件与系统的其他部分以及其他插件之间的耦合度较低。每个插件都可以独立地进行开发、测试和维护，不会影响到系统的其他部分。
   - 独立测试：由于插件的独立性，每个插件都可以单独进行测试，当某个插件需要维护或升级时，只需要对该插件进行修改和测试。
3. 高开发效率
   - 并行开发：插件式架构支持多团队的并行开发，不同的开发团队可以同时开发不同的插件，提高了开发效率。
   - 快速迭代：插件的开发和发布相对独立，这使得开发者可以快速地迭代和发布插件，及时响应市场需求和用户反馈。

2.2 资源和成本规划

表 2-1 云服务列表

云服务	数量	规格	计费模式	预计成本
几何数据可视化SDK	1	/	包年	600000元/年

以上是参考价格，具体价格以实际业务需求及官网实际情况为准。

2.3 实施步骤

准备工作

1. 开发技能
   - 熟悉C++语言，能够编写C++语言代码。
   - 熟悉CAX软件功能。
   - 了解渲染、可视化领域知识。
   - 了解C++构建相关工具（cmake等）。
2. 开发环境
   - Windows 10+
     - Visual Studio 2017+或适用于Visual Studio 2017+的Microsoft Visual生成工具
     - CMake 3.7+
   - Linux：
     - gcc g++ gdb
     - CMake 3.7+

操作步骤

可视化SDK无法独立工作，需要嵌入到所需应用中，提供渲染可视化能力，详细可见SDK Doxygen文档。

步骤1 生成视口。

```
Handle(Aspect_DisplayConnection) myDisplayConnection = new Aspect_DisplayConnection();
myGraphicDiver = new OpenGL_GraphicDriver(myDisplayConnection);
myViewer = new V3d_Viewer(myGraphicDiver);
myView = myViewer->CreateView();
```

步骤2 创建上下文。

```
WId window_handle = (WId)winId();
Handle(WNT_Window) wind = new WNT_Window((Aspect_Handle)window_handle)
myView->SetWindow(wind);
m_context = new AIS_InteractiveContext(m_viewer);
```

步骤3 设置默认视图属性。

```
myViewer->SetDefaultLights();
myViewer->SetLightOn();
myView->SetBackgroundColor(Quantity_NOC_GRAY60);
myVieww->MustBeResized();
myView->TriedronDisplay(Aspect_TOTP_LEFT_LOWER, Quantity_NOC_GOLD,0.08,V3d_ZBUFFER);
myContext->SetDisplayMode(AIS_Shaded,Standard_True);
```

----结束

常见问题

Q：Windows和Linux环境的编译兼容性问题?

A：

- Windows环境
  - 不同的C++编译器（如Visual C++、MinGW等）可能会与SDK的构建系统产生兼容性问题。例如，SDK可能在构建过程中依赖特定的编译器特性或标准库实现，而Visual C++和MinGW在某些C++标准支持程度上有所差异。
  - 对于依赖的其他库（如FreeType），其头文件和库文件路径也需要正确地添加到项目的包含路径和库路径环境变量中。如果这些路径设置不正确，在编译过程中会出现头文件无法找到或链接库错误的情况。
  - 一些库可能以动态链接库（DLL）的形式提供，在Windows上运行时，可能会出现DLL缺失或版本不匹配的问题。例如，本SDK依赖于某些系统级的DLL（如OpenGL相关的DLL），如果这些DLL版本过旧或者不存在，可能会导致应用程序在运行时崩溃或出现图形渲染错误。
- Linux环境
  - 包管理器依赖问题：
    - SDK需要特定版本的运行时g++库支持C++语言特性，但系统默认的版本可能较旧，导致在构建过程中出现语法错误。
    - 在Linux不同发行版存在不同，这可能会导致在安装和配置过程中出现混淆。
  - 权限问题：

    对于使用多用户环境的情况，不同用户可能对SDK相关的文件和目录有不同的权限。这可能会导致一些用户能够成功构建和运行应用，而其他用户则因为权限问题无法进行相同的操作。
  - 库路径和链接问题：
    - Linux系统在链接库文件时，依赖于特定的库路径设置（如LD_LIBRARY_PATH环境变量）。如果这些路径没有正确设置，应用可能无法找到所需的共享库。例如，在运行时应用可能无法加载OCCT库，因为系统无法在指定的库路径中找到该库。
    - 与Windows不同，Linux上的库文件命名和版本管理遵循一定的规则（如.so文件的命名和版本号约定）。如果库文件的名称或版本不符合这些规则，或者在链接过程中指定了错误的库名称或版本，会导致链接错误。

Q：Windows和Linux环境的性能和优化问题

A：

- Windows环境
  - 图形驱动和硬件加速问题：
    - Windows上的图形驱动程序对于3D图形渲染的性能有很大影响。如果图形驱动没有正确安装或者版本过旧，可能会导致3D查看器的性能下降，如出现画面卡顿、渲染错误等问题。
    - 不同的硬件设备（如NVIDIA、AMD或Intel的显卡）在Windows上支持的OpenGL版本可能存在不同。
  - 系统资源管理问题：

- 与Linux相比，Windows在某些情况下可能会对文件I/O操作进行更多的缓存和预读处理。这可能会对SDK的数据交换过程产生影响，例如在频繁读取和写入模型文件时，Windows的缓存策略可能会导致数据不一致或者性能下降。

- Linux环境
  - 图形栈和兼容性问题：
    - Linux系统有多种图形栈（如X11、Wayland等）可供选择。不同的图形栈可能对3D查看器的性能和兼容性产生不同的影响。例如，某些图形库（如OpenGL）在Wayland下的性能可能不如在X11下，或者可能会出现一些兼容性问题，如无法正确显示某些图形元素。
    - 不同Linux发行版对于图形驱动的安装和配置方式也有所不同。
  - 内存和缓存优化问题：
    - Linux系统提供了多种工具和机制来优化内存和缓存的使用。然而，对于SDK来说，可能需要根据具体的应用场景进行适当的调整。例如，在处理大型3D模型时，可能需要调整系统的内存缓存参数（如/proc/sys/vm相关的参数）来提高数据访问速度，但如果调整不当，可能会导致系统性能下降或者内存泄漏。
    - 与Windows不同，Linux在文件I/O缓存方面可能有更灵活的配置选项。例如，可以使用不同的缓存策略（如direct I/O、O_DIRECT标志等）来优化模型文件的读取和写入，但这些策略需要根据具体的硬件和应用需求进行选择，否则可能会影响数据交换的性能。

2.4 附录

更多内容请参考：

表 2-2 参考附录

序号	参考内容
01	产品介绍

2.5 文档修订历史

文档版本	发布日期	修改说明
01	2024 年 11 月	首次发布

3 基于 NCAD 几何处理引擎处理几何数据应用

3.1 方案概述

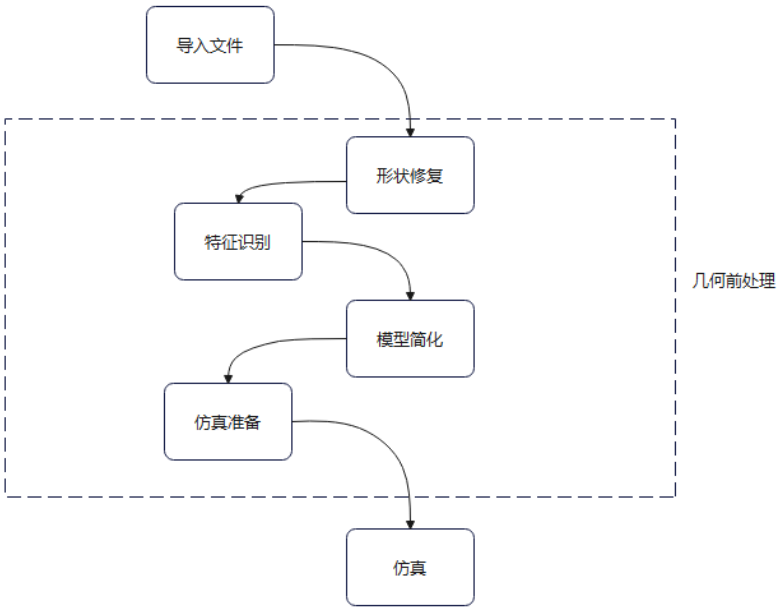
应用场景

NCAD是一款功能强大的开发工具包，专为CAD（计算机辅助设计）、CAE（计算机辅助工程）和CAM（计算机辅助制造）领域中的几何建模、修复与优化任务设计。它提供了完整的解决方案，涵盖了从几何形状修复、特征识别、简化到仿真准备及网格修复等多个功能，帮助开发者快速构建高效的工程应用，提升设计、分析与制造过程的精准度和效率。

NCAD在多个行业中有着广泛的应用，尤其适用于以下场景：

- **CAD系统开发**：为开发高效的计算机辅助设计系统提供全面的几何建模、修复和特征识别支持，简化复杂设计任务。
- **CAE仿真准备**：对复杂的CAD模型进行简化与优化，包括几何形状的修复和生成网格的修复，生成符合仿真分析需求的网格，减少仿真计算时间和资源消耗。
- **3D打印准备**：自动修复和优化3D打印模型的几何数据，确保STL或其他文件格式符合3D打印要求，避免错误打印。

方案架构



NCAD提供了模块化的架构，能够灵活地集成到现有的CAD、CAE或CAM工作流中。其核心架构包括以下几个模块：

1. **几何修复模块**：
   - 自动检测并修复几何模型中的错误，如非流形边界、重叠面、无效面等。
   - 修复错误的拓扑关系，确保模型的可用性。
   - 完全可定制的管道，支持灵活排序修复的顺序
2. **特征识别模块**：
   - 对CAD模型中的特征（如孔、槽、壁等）进行自动识别和提取。
   - 支持多种复杂的几何特征识别，包括非规则形状的特征提取。
3. **几何简化模块**：
   - 对复杂的CAD模型进行简化，保留关键几何信息的同时减少冗余数据。
   - 提供多种简化策略，适用于不同精度和计算需求。
4. **仿真准备模块**：
   - 提供干扰修复、对称检测、规范几何体检测等通用仿真的前期准备算法。
   - 提供中面查找、梁模型、表面焊接和点焊接等结构分析的前期准备算法，也提供部分CFD的准备算法。
5. **网格修复模块**：
   - 自动修复生成的网格文件，确保没有漏洞、非流形边界等问题。
   - 优化网格质量，提高仿真和分析的精度与稳定性。
6. **数据交换模块**：
   - 支持常见的BRep、STEP、STL格式的导入与导出，确保与其他软件系统的兼容性。
   - 支持多种数据格式之间的转换，简化数据交换过程。

方案优势

NCAD工具包为开发者提供了一整套高效、可靠的解决方案，具有以下几大优势：

- **高度集成与模块化**：NCAD提供了一套完整的工具集，所有功能模块之间无缝衔接，用户可根据实际需求选择性集成，节省开发时间和资源。
- **自动化与智能化**：NCAD在几何修复、特征识别及网格修复方面具备高度自动化的能力，减少人工干预，提高工作效率。特征识别与几何简化过程也经过优化，能够自动调整，适应不同类型的模型。
- **高精度与可靠性**：NCAD内部采用高精度的数学算法，保证几何修复和网格生成的准确性，为复杂的工程仿真提供稳定的基础数据。
- **高效的数据处理能力**：优化的算法使得NCAD在大规模模型和复杂几何体的处理上表现出色。无论是在处理高精度零件模型还是在生成超大规模的仿真网格时，NCAD都能保持高效的处理速度。
- **用户友好与易于集成**：NCAD提供清晰易懂的文档、示例代码及开发工具包，帮助开发人员快速上手，并且其API设计简洁，易于与现有项目或软件进行集成。

3.2 资源和成本规划

表 3-1 云服务列表

云服务	数量	规格	计费模式	预计成本
几何操作SDK	1	/	包年	780000元/年

以上是参考价格，具体价格以实际业务需求及官网实际情况为准。

3.3 实施步骤

准备工作

1. 开发技能
   - 熟悉C++语言，能够编写C++语言代码。
   - 熟悉CAX软件功能。
   - 了解OCCT几何建模相关知识。
   - 了解C++构建相关工具（cmake等）
2. 开发环境
   - Windows 10+
     - Visual Studio 2017+或适用于Visual Studio 2017+的Microsoft Visual生成工具
     - CMake 3.7+
   - Linux：
     - gcc g++ gdb

- CMake 3.7+

操作步骤

1. 读取STEP文件。

```
Ncad::Exchange::STEPReader reader;
reader.ReadFile("model_name.stp");
const ShapePtr inputShape = reader.GetOneShape();
```

2. （可选）创建基础实体（如盒子、锥体、圆柱、球体、圆环）。

```
// Prepare data.
const DirPtr oz = std::make_shared<Ncad::Base::Dir>(0.0, 0.0, 1.0);
const Ax2Ptr cylAxis = std::make_shared<Ncad::Base::Ax2>(origin, oz);

// Create cylinder.
Ncad::Prim::Cylinder cylBuilder(cylAxis, 5.0, 10.0);
cylBuilder.Perform();
const ShapePtr cyl = cylBuilder.GetResult();
```

3. （可选）执行布尔操作。

   布尔操作是指一组允许使用类似于布尔代数的方法融合、切割或分割形状的操作。NCAD支持的布尔操作类型包括：FUSE（融合）、CUT（切割）、COMMOM（并集）、SPILT（分割）。以下是布尔CUT的调用示例：

```
Ncad::Modify::Boolean bopAlgo(box, cyl, Ncad::Modify::Boolean::BOP_CUT);
const ShapePtr bopRes = bopAlgo.GetResult();
```

4. 创建圆角和倒角。
   - 圆角是在两个或多个相交或相邻边缘之间创建平滑过渡或曲率的过程。
   - 倒角是沿着形状的边缘或角落创建平面或有角度的切口的过程。

```
// Edges to be filleted.
ListOfShape edges;
std::set<int> edgeIndices = {3, 9, 13};
Ncad::Base::ShapeExplorer exp(bopRes, Ncad::Base::Shape::EDGE);
for (int index = 1; exp.More(); exp.Next(), ++index)
    if (edgeIndices.find(index) != edgeIndices.end())
        edges.push_back(exp.Current());

// Fillet creation.
static constexpr double radius = 1.0;
Ncad::Modify::Fillet filletAlgo(bopRes, edges, radius);
ShapePtr result = filletAlgo.GetResult();
```

5. 缺陷检测。

```
 // detection of problems
 Ncad::Analysis::DefectConfig config(1.0e-4);
 Ncad::Analysis::CheckShape checker(config);
 const bool isValid = checker.IsValid(inputShape);
 if (isValid) {
      std::cout << "Input shape is valid" << std::endl;
      return 0;
 }

 std::list<FaultyShapePtr> faulties =
checker.GetFaultiesByID(Ncad::Analysis::DefectID::GEOM_SELF_INTERSECTION);
 std::cout << "Input shape has faulties. Launching shape healing." << std::endl;
```

6. 执行形状修复。

```
std::list<Ncad::Guid> repairs =
{Ncad::Healing::repairSILoopBySplitting(),Ncad::Healing::repairSISurfaceBySplitting()};
config.SetDefectRepairs(Ncad::Analysis::DefectID::GEOM_SELF_INTERSECTION, repairs);
Ncad::Healing::FixShape healer(config);
const bool isFixed = healer.Fix(faulties);
ShapePtr resultShape = healer.GetResult();
std::cout << "healing is performed" << std::endl;
```

7. 几何特征识别。

NCAD支持钻孔检测、凹槽检测、空腔检测及隔离特征检测等检测算法。以下是钻孔识别检测的调用示例：

```
Ncad::Feature::DrilledHolesDetector dhDetector(shape);
dhDetector.Perform();
const std::vector<std::set<int>>& drilledHoles = dhDetector.GetAllFeatureIds();
```

8. 几何特征移除。

```
// read indices of faces to be removed
std::set<int> faceIndices;
for (int i = 3; i < argc; ++i)
  faceIndices.insert(atoi(argv[i]));

std::cout << "indices are loaded" << std::endl;

// convert indices into faces to remove
ListOfShape faces2remove;
Ncad::Base::ShapeExplorer exp(inputShape, Ncad::Base::Shape::FACE);
for (int index = 1; exp.More(); exp.Next(), ++index)
  if (faceIndices.find(index) != faceIndices.end())
    faces2remove.push_back(exp.Current());

// defeaturing
Ncad::Modify::AdvancedDefeaturing advDef(inputShape, faces2remove);
advDef.Perform();
ShapePtr resultShape = advDef.GetResult();

std::cout << "Defeaturing is performed" << std::endl;
```

9. 将结果文件写入成STEP文件。

```
Ncad::Exchange::STEPWriter writer;
writer.WriteShape(shape, "path_to_file");
```

常见问题

Q：Windows和Linux环境的编译兼容性问题

A：

- Windows环境
  - 编译器兼容性问题：
    - 在Windows上，不同的C++编译器（如Visual C++、MinGW等）可能会与SDK的构建系统产生兼容性问题。例如，SDK可能在构建过程中依赖特定的编译器特性或标准库实现，而Visual C++和MinGW在某些C++标准支持程度上有所差异。
    - 对于依赖的其他库（如FreeType），其头文件和库文件路径也需要正确地添加到项目的包含路径和库路径环境变量中。如果这些路径设置不正确，在编译过程中会出现头文件无法找到或链接库错误的情况。
  - DLL依赖问题：

    一些库可能以动态链接库（DLL）的形式提供，在Windows上运行时，可能会出现DLL缺失或版本不匹配的问题。
- Linux环境
  - 包管理器依赖问题：
    - SDK需要特定版本的运行时g++库支持C++语言特性，但系统默认的版本可能较旧，导致在构建过程中出现语法错误。
    - 在Linux不同发行版存在不同，这可能会导致在安装和配置过程中出现混淆。

- 权限问题：

  对于使用多用户环境的情况，不同用户可能对SDK相关的文件和目录有不同的权限。这可能会导致一些用户能够成功构建和运行应用，而其他用户则因为权限问题无法进行相同的操作。

- 库路径和链接问题：

  i. Linux系统在链接库文件时，依赖于特定的库路径设置（如LD_LIBRARY_PATH环境变量）。如果这些路径没有正确设置，应用可能无法找到所需的共享库。例如，在运行时应用可能无法加载OCCT库，因为系统无法在指定的库路径中找到该库。

  ii. 与Windows不同，Linux上的库文件命名和版本管理遵循一定的规则（如.so文件的命名和版本号约定）。如果库文件的名称或版本不符合这些规则，或者在链接过程中指定了错误的库名称或版本，会导致链接错误。

Q：为什么使用SDK时，抛出了许可证相关的异常?

A：请检查许可证的有效性，例如产品名称是否匹配，是否超过有效期限，并使用有效的许可证。

3.4 附录

更多内容请参考：

表 3-2 参考附录

序号	参考内容
01	产品介绍

3.5 文档修订历史

文档版本	发布日期	修改说明
01	2024 年 11 月	首次发布