Kicad 插件

May 15, 2020

Contents

KiCad 插件系统

Contributors

Cirilo Bernardo

翻译

taotieren <admin@taotieren.com>, 2019

Telegram 简体中文交流群: https://t.me/KiCad_zh_CN

反馈

请将任何错误报告、建议或新版本引导到此处:

- 关于 KiCad 文档: https://gitlab.com/kicad/services/kicad-doc/issues
- 关于 KiCad 软件: https://gitlab.com/kicad/code/kicad/issues
- 关于 KiCad 软件 i18n: https://gitlab.com/kicad/code/kicad-i18n/issues

出版日期和软件版本

2016 年 1 月 29 日出版。

1 KiCad 插件系统简介

KiCad 插件系统是一个使用共享库扩展 KiCad 功能的框架。使用插件的一个主要优点是在开发插件时没有必要重建 KiCad 套件; 事实上，可以借助 KiCad 源代码树中的一小组标题构建插件。通过确保开发人员仅编译与正在开发的插件直接相关的代码，从而减少每个构建和测试周期所需的时间，在插件开发期间删除构建 KiCad 的要求极大地提高了工作效率。

插件最初是为 3D 模型查看器开发的，因此可以支持更多类型的 3D 模型，而无需对支持的每种新模型类型的 KiCad 源进行重大更改。插件框架后来被推广，以便将来开发人员可以创建不同类型的插件。目前，只有 3D 插件在 KiCad 中实现，但可以想象最终将开发 PCB 插件，以使用户能够实现数据导入器和导出器。

1.1 插件类

插件分为插件类，因为每个插件都解决了特定域中的问题，因此需要该域独有的接口。例如，3D 模型插件从文件加载 3D 模型数据并将该数据转换为可由 3D 查看器显示的格式。PCB 导入/导出插件将获取 PCB 数据并导出为其他电气或机械数据格式，或将外部格式转换为 KiCad PCB。目前只开发了 3D 插件类，它将成为本文档的重点。

实现插件类需要在 KiCad 源代码树中创建代码来管理插件代码的加载。在 KiCad 源代码树中，文件‘plugins/ldr/-pluginldr.h’声明了所有插件加载器的基类。这个类声明了我们期望在任何 KiCad 插件（样板代码）中找到的最基本的函数，它的实现提供了对插件加载器和可用插件之间的版本兼容性的基本检查。标题‘plugins/ldr/3d/pluginldr3D.h’声明了 3D 插件类的加载器。加载器负责加载给定的插件并使其功能可用于 KiCad。插件加载器的每个实例代表一个实际的插件实现，并充当 KiCad 和插件功能之间的透明桥梁。加载器不是 KiCad 中支持插件所需的唯一代码：我们还需要代码来发现插件和代码，以通过插件加载器调用插件的功能。在 3D 插件的情况下，发现和调用功能都包含在 S3D_CACHE 类中。

除非正在开发新的插件类，否则插件开发人员不需要关心 KiCad 管理插件的内部代码的细节; 插件只需要定义其特定插件类声明的函数。

标题‘include/plugins/kicad_plugin.h’声明了所有 KiCad 插件所需的泛型函数; 这些函数标识插件类，提供特定插件的名称，提供插件类 API 的版本信息，提供特定插件的版本信息，并提供插件类 API 的基本版本兼容性检查。简而言之，这些功能是：

```
/* 返回命名插件类的 UTF-8 字符串 */
/* Return a UTF-8 string naming the Plugin Class */
char const* GetKicadPluginClass( void );

/* 返回插件类API的版本信息 */
/* Return version information for the Plugin Class API */
void GetClassVersion( unsigned char* Major, unsigned char* Minor,
     unsigned char* Patch, unsigned char* Revision );

/*
   如果插件中实现了版本检查，则返回 true。
   确定给定的插件类 API 是否兼容。
   Return true if the version check implemented in the plugin
   determines that the given Plugin Class API is compatible.
 */
bool CheckClassVersion( unsigned char Major,
```

```
    unsigned char Minor, unsigned char Patch, unsigned char Revision );

/* 返回具体插件的名称，例如 “PLUGIN_3D_VRML” */
/* Return the name of the specific plugin, for example "PLUGIN_3D_VRML" */
const char* GetKicadPluginName( void );

/* 返回特定插件的版本信息 */
/* Return version information for the specific plugin */
void GetPluginVersion( unsigned char* Major, unsigned char* Minor,
     unsigned char* Patch, unsigned char* Revision );
```

1.1.1 插件类：PLUGIN_3D

标题‘include/plugins/3d/3d_plugin.h’声明了必须由所有 3D 插件实现的函数，并定义了插件所需的许多函数以及用户不得重新实现的函数。用户不得重新现的已定义函数是：

```
/* 返回插件类名 “PLUGIN_3D” */
/* Returns the Plugin Class name "PLUGIN_3D" */
char const* GetKicadPluginClass( void );

/* 返回 PLUGIN_3D API 的版本信息 */
/* Return version information for the PLUGIN_3D API */
void GetClassVersion( unsigned char* Major, unsigned char* Minor,
     unsigned char* Patch, unsigned char* Revision );

/*
   执行由的开发人员强制执行的基本版本检查。
   PLUGIN_3D 类的加载器，如果。
   检查通过
   Performs basic version checks enforced by the developers of
   the loader for the PLUGIN_3D class and returns true if the
   checks pass
 */
bool CheckClassVersion( unsigned char Major, unsigned char Minor,
     unsigned char Patch, unsigned char Revision );
```

用户必须实现的功能如下：

```
/* 返回插件支持的扩展字符串数 */
/* Return the number of extension strings supported by the plugin */
int GetNExtensions( void );

/*
   返回请求的扩展字符串；有效值为 0 到
   GetNExtensions() - 1
   Return the requested extension string; valid values are 0 to
   GetNExtensions() - 1
 */
```

```
char const* GetModelExtension( int aIndex );

/* 返回插件支持的文件过滤器总数 */
/* Return the total number of file filters supported by the plugin */
int GetNFilters( void );

/*
   返回请求的文件筛选器；有效值为 0 到。
   GetNFilters() - 1
   Return the file filter requested; valid values are 0 to
   GetNFilters() - 1
 */
char const* GetFileFilter( int aIndex );

/*
    如果插件可以渲染这种类型的 3D 模型，则返回 true。
    在某些情况下，插件可能尚未提供可视模型。
    并且必须返回 false。
    Return true if the plugin can render this type of 3D model.
    In some cases a plugin may not yet provide a visual model
    and must return false.
 */
bool CanRender( void );

/* 加载指定的模型并返回指向其可视化模型数据的指针 */
/* Load the specified model and return a pointer to its visual model data */
SCENEGRAPH* Load( char const* aFileName );
```

2 教程：3D 插件类

本节包含对 PLUGIN_3D 类的两个非常简单的插件的描述，并引导用户完成代码的设置和构建。

2.1 基本的 3D 插件

本教程将引导用户开发一个名为“PLUGIN_3D_DEMO1”的非常基本的 3D 插件。本教程的目的只是为了演示一个非常基本的 3D 插件的构造，除了提供一些允许 KiCad 用户在浏览 3D 模型时过滤文件名的过滤字符串之外什么都不做。这里演示的代码是任何 3D 插件的绝对最低要求，可以用作创建更高级插件的模板。

为了构建演示项目，我们需要以下内容：

- CMake
- KiCad 插件头
- KiCad 场景图库‘kicad_3dsg’

要自动检测 KiCad 标头和库，我们将使用 CMake FindPackage 脚本; 如果相关的头文件安装到中，则本教程中提供的脚本应该适用于 Linux 和 Windows。

要开始，让我们创建一个项目目录和 FindPackage 脚本：

```
mkdir demo && cd demo
export DEMO_ROOT=${PWD}
mkdir CMakeModules && cd CMakeModules
cat > FindKICAD.cmake << _EOF
find_path( KICAD_INCLUDE_DIR kicad/plugins/kicad_plugin.h
    PATHS ${KICAD_ROOT_DIR}/include $ENV{KICAD_ROOT_DIR}/include
    DOC "Kicad plugins header path."
    )

if( NOT ${KICAD_INCLUDE_DIR} STREQUAL "KICAD_INCLUDE_DIR-NOTFOUND" )

    # 尝试从 sg_version.h 中提取版本信息
    # attempt to extract the version information from sg_version.h
    find_file( KICAD_SGVERSION sg_version.h
        PATHS ${KICAD_INCLUDE_DIR}
        PATH_SUFFIXES kicad/plugins/3dapi
        NO_DEFAULT_PATH )

    if( NOT ${KICAD_SGVERSION} STREQUAL "KICAD_SGVERSION-NOTFOUND" )

        # 提取 “#define KICADSG_VERSION*” 行
        # extract the "#define KICADSG_VERSION*" lines
        file( STRINGS ${KICAD_SGVERSION} _version REGEX "^#define.*KICADSG_VERSION.*" )

        foreach( SVAR ${_version} )
            string( REGEX MATCH KICADSG_VERSION_[M,A,J,O,R,I,N,P,T,C,H,E,V,I,S]* _VARNAME $ ↩
                {SVAR} )
            string( REGEX MATCH [0-9]+ _VALUE ${SVAR} )

            if( NOT ${_VARNAME} STREQUAL "" AND NOT ${_VALUE} STREQUAL "" )
                set( _${_VARNAME} ${_VALUE} )
            endif()

        endforeach()

        # 确保 NOT SG3D_VERSION* 的计算结果为 “0”
        # ensure that NOT SG3D_VERSION* will evaluate to '0'
        if( NOT _KICADSG_VERSION_MAJOR )
            set( _KICADSG_VERSION_MAJOR 0 )
        endif()

        if( NOT _KICADSG_VERSION_MINOR )
            set( _KICADSG_VERSION_MINOR 0 )
        endif()
```

```
        if( NOT _KICADSG_VERSION_PATCH )
            set( _KICADSG_VERSION_PATCH 0 )
        endif()

        if( NOT _KICADSG_VERSION_REVISION )
            set( _KICADSG_VERSION_REVISION 0 )
        endif()

        set( KICAD_VERSION ${_KICADSG_VERSION_MAJOR}.${_KICADSG_VERSION_MINOR}.${ ↩
            _KICADSG_VERSION_PATCH}.${_KICADSG_VERSION_REVISION} )
        unset( KICAD_SGVERSION CACHE )

    endif()
endif()


find_library( KICAD_LIBRARY
    NAMES kicad_3dsg
    PATHS
        ${KICAD_ROOT_DIR}/lib $ENV{KICAD_ROOT_DIR}/lib
        ${KICAD_ROOT_DIR}/bin $ENV{KICAD_ROOT_DIR}/bin
    DOC "Kicad scenegraph library path."
    )

include( FindPackageHandleStandardArgs )
FIND_PACKAGE_HANDLE_STANDARD_ARGS( KICAD
    REQUIRED_VARS
        KICAD_INCLUDE_DIR
        KICAD_LIBRARY
        KICAD_VERSION
    VERSION_VAR KICAD_VERSION )


mark_as_advanced( KICAD_INCLUDE_DIR )
set( KICAD_VERSION_MAJOR ${_KICADSG_VERSION_MAJOR} CACHE INTERNAL "" )
set( KICAD_VERSION_MINOR ${_KICADSG_VERSION_MINOR} CACHE INTERNAL "" )
set( KICAD_VERSION_PATCH ${_KICADSG_VERSION_PATCH} CACHE INTERNAL "" )
set( KICAD_VERSION_TWEAK ${_KICADSG_VERSION_REVISION} CACHE INTERNAL "" )
_EOF
```

必须安装 Kicad 及其插件标头; 如果将它们安装到用户目录或 Linux 上的‘/opt’下，或者您使用的是 Windows，则需要将‘KICAD_ROOT_DIR’环境变量设置为指向包含 KiCad ‘include’和‘lib’目录的目录。对于 OS X，此处显示的 FindPackage 脚本可能需要进行一些调整。

要配置和构建教程代码，我们将使用 CMake 并创建一个‘CMakeLists.txt’脚本文件:

```
cd ${DEMO_ROOT}
cat > CMakeLists.txt << _EOF
```

```
# 声明项目名称
# declare the name of the project
project( PLUGIN_DEMO )

# 检查我们是否有具有所有必需功能的 CMake 版本
# check that we have a version of CMake with all required features
cmake_minimum_required( VERSION 2.8.12 FATAL_ERROR )

# 通知 CMake 何处可以找到 FindKICAD 脚本
# inform CMake of where to find the FindKICAD script
set( CMAKE_MODULE_PATH ${PROJECT_SOURCE_DIR}/CMakeModules )

# 尝试发现已安装的 kicad 标头和库
# attempt to discover the installed kicad headers and library
# 并设置变量: (and set the variables:)
#     KICAD_INCLUDE_DIR
#     KICAD_LIBRARY
find_package( KICAD 1.0 REQUIRED )

# 将 kicad include 目录添加到编译器的搜索路径中
# add the kicad include directory to the compiler's search path
include_directories( ${KICAD_INCLUDE_DIR}/kicad )

# 创建名为 s3D_plugin_demo1 的插件
# create a plugin named s3d_plugin_demo1
add_library( s3d_plugin_demo1 MODULE
    src/s3d_plugin_demo1.cpp
    )

_EOF
```

第一个演示项目非常基础; 它由一个文件组成，除了编译器默认值之外没有外部链接依赖项。我们首先创建一个源目录:

```
cd ${DEMO_ROOT}
mkdir src && cd src
export DEMO_SRC=${PWD}
```

现在我们自己创建插件源:

s3d_plugin_demo1.cpp

```
#include <iostream>

// 3d_plugin.h 标头定义了 3D 插件所需的功能
// the 3d_plugin.h header defines the functions required of 3D plugins
#include "plugins/3d/3d_plugin.h"

// 定义这个插件的版本信息，不要混淆
```

```
// define the version information of this plugin; do not confuse this
// 使用在 3d_plugin.h 中定义的插件类版本
// with the Plugin Class version which is defined in 3d_plugin.h
#define PLUGIN_3D_DEMO1_MAJOR 1
#define PLUGIN_3D_DEMO1_MINOR 0
#define PLUGIN_3D_DEMO1_PATCH 0
#define PLUGIN_3D_DEMO1_REVNO 0

// 实现为用户提供此插件名称的函数
// implement the function which provides users with this plugin's name
const char* GetKicadPluginName( void )
{
    return "PLUGIN_3D_DEMO1";
}

// 实现为用户提供此插件版本的功能
// implement the function which provides users with this plugin's version
void GetPluginVersion( unsigned char* Major, unsigned char* Minor,
    unsigned char* Patch, unsigned char* Revision )
{
    if( Major )
        *Major = PLUGIN_3D_DEMO1_MAJOR;

    if( Minor )
        *Minor = PLUGIN_3D_DEMO1_MINOR;

    if( Patch )
        *Patch = PLUGIN_3D_DEMO1_PATCH;

    if( Revision )
        *Revision = PLUGIN_3D_DEMO1_REVNO;

    return;
}

// 支持的扩展名数量; 在 *NIX 系统上, 扩展名为
// number of extensions supported; on *NIX systems the extensions are
// 提供两次-一次小写, 一次大写
// provided twice - once in lower case and once in upper case letters
#ifdef _WIN32
    #define NEXTS 7
#else
    #define NEXTS 14
#endif

// 支持的筛选器集数量
// number of filter sets supported
#define NFILS 5
```

```
// 定义此扩展字符串和筛选器字符串
// define the extension strings and filter strings which this
// 插件将提供给用户
// plugin will supply to the user
static char ext0[] = "wrl";
static char ext1[] = "x3d";
static char ext2[] = "emn";
static char ext3[] = "iges";
static char ext4[] = "igs";
static char ext5[] = "stp";
static char ext6[] = "step";

#ifdef _WIN32
static char fil0[] = "VRML 1.0/2.0 (*.wrl)|*.wrl";
static char fil1[] = "X3D (*.x3d)|*.x3d";
static char fil2[] = "IDF 2.0/3.0 (*.emn)|*.emn";
static char fil3[] = "IGESv5.3 (*.igs;*.iges)|*.igs;*.iges";
static char fil4[] = "STEP (*.stp;*.step)|*.stp;*.step";
#else
static char ext7[] = "WRL";
static char ext8[] = "X3D";
static char ext9[] = "EMN";
static char ext10[] = "IGES";
static char ext11[] = "IGS";
static char ext12[] = "STP";
static char ext13[] = "STEP";

static char fil0[] = "VRML 1.0/2.0 (*.wrl;*.WRL)|*.wrl;*.WRL";
static char fil1[] = "X3D (*.x3d;*.X3D)|*.x3d;*.X3D";
static char fil2[] = "IDF 2.0/3.0 (*.emn;*.EMN)|*.emn;*.EMN";
static char fil3[] = "IGESv5.3 (*.igs;*.iges;*.IGS;*.IGES)|*.igs;*.iges;*.IGS;*.IGES";
static char fil4[] = "STEP (*.stp;*.step;*.STP;*.STEP)|*.stp;*.step;*.STP;*.STEP";
#endif

// 实例化一个方便的数据结构来访问
// instantiate a convenient data structure for accessing the
// 扩展和筛选字符串列表
// lists of extension and filter strings
static struct FILE_DATA
{
    char const* extensions[NEXTS];
    char const* filters[NFILS];

    FILE_DATA()
    {
        extensions[0] = ext0;
        extensions[1] = ext1;
```

```
        extensions[2] = ext2;
        extensions[3] = ext3;
        extensions[4] = ext4;
        extensions[5] = ext5;
        extensions[6] = ext6;
        filters[0] = fil0;
        filters[1] = fil1;
        filters[2] = fil2;
        filters[3] = fil3;
        filters[4] = fil4;

#ifndef _WIN32
        extensions[7] = ext7;
        extensions[8] = ext8;
        extensions[9] = ext9;
        extensions[10] = ext10;
        extensions[11] = ext11;
        extensions[12] = ext12;
        extensions[13] = ext13;
#endif
        return;
    }

} file_data;


// 返回该插件支持的扩展数量
// return the number of extensions supported by this plugin
int GetNExtensions( void )
{
    return NEXTS;
}

// 返回索引的扩展字符串
// return the indexed extension string
char const* GetModelExtension( int aIndex )
{
    if( aIndex < 0 || aIndex >= NEXTS )
        return NULL;

    return file_data.extensions[aIndex];
}

// 返回该插件提供的筛选字符串数量
// return the number of filter strings provided by this plugin
int GetNFilters( void )
{
    return NFILS;
```

```
}

// 返回索引筛选字符串
// return the indexed filter string
char const* GetFileFilter( int aIndex )
{
    if( aIndex < 0 || aIndex >= NFILS )
        return NULL;

    return file_data.filters[aIndex];
}

// 返回 false，因为此插件不提供可视化数据
// return false since this plugin does not provide visualization data
bool CanRender( void )
{
    return false;
}

// 返回 null，因为此插件不提供可视化数据
// return NULL since this plugin does not provide visualization data
SCENEGRAPH* Load( char const* aFileName )
{
    // 此伪插件不支持渲染任何模型
    // this dummy plugin does not support rendering of any models
    return NULL;
}
```

此源文件满足实现 3D 插件的所有最低要求。该插件不会为渲染模型生成任何数据，但它可以为 KiCad 提供支持的模型文件扩展名和文件扩展名过滤器列表，以增强 3D 模型文件选择对话框。在 KiCad 中，扩展字符串用于选择可用于加载指定模型的插件; 例如，如果插件是‘wrl’，那么 KiCad 将调用声称支持扩展‘wrl’的每个插件，直到插件返回可视化数据。每个插件提供的文件过滤器将传递到 3D 文件选择器对话框，以改善浏览 UI。

要构建插件：

```
cd ${DEMO_ROOT}
# export KICAD_ROOT_DIR if necessary
mkdir build && cd build
cmake .. && make
```

该插件将被构建但未安装; 如果要加载插件，必须将插件文件复制到 KiCad 的插件目录。

2.2 高级 3D 插件

本教程将引导用户开发名为“PLUGIN_3D_DEMO2”的 3D 插件。本教程的目的是演示 KiCad 预览器可以渲染的非常基本的场景图的构造。该插件声称处理‘txt’类型的文件。虽然文件必须存在，以便缓存管理器调用插件，但此插件不处理文件内容; 相反，插件只是创建一个包含一对四面体的场景图。本教程假定第一个教程已完成，并且已创建 CMakeLists.txt 和 FindKICAD.cmake 脚本文件。

将新的源文件放在与上一个教程的源文件相同的目录中，我们将扩展上一个教程的 CMakeLists.txt 文件来构建本教程。由于这个插件会为 KiCad 创建一个场景图，我们需要链接到 KiCad 的场景图库‘kicad_3dsg’。KiCad 的场景图库提供了一组可用于构建场景图对象的类; 场景图对象是 3D 缓存管理器使用的中间数据可视化格式。所有支持模型可视化的插件都必须通过此库将模型数据转换为场景图。

第一步是扩展‘CMakeLists.txt’来构建这个教程项目：

```
cd ${DEMO_ROOT}
cat >> CMakeLists.txt << _EOF
add_library( s3d_plugin_demo2 MODULE
    src/s3d_plugin_demo2.cpp
    )

target_link_libraries( s3d_plugin_demo2 ${KICAD_LIBRARY} )
_EOF
```

现在我们切换到源目录并创建源文件：

```
cd ${DEMO_SRC}
```

s3d_plugin_demo2.cpp

```
#include <cmath>
// 3D 插件类声明
// 3D Plugin Class declarations
#include "plugins/3d/3d_plugin.h"
// KiCad 场景图形库接口
// interface to KiCad Scene Graph Library
#include "plugins/3dapi/ifsg_all.h"

// 该插件的版本信息
// version information for this plugin
#define PLUGIN_3D_DEMO2_MAJOR 1
#define PLUGIN_3D_DEMO2_MINOR 0
#define PLUGIN_3D_DEMO2_PATCH 0
#define PLUGIN_3D_DEMO2_REVNO 0

// 提供此插件的名称
// provide the name of this plugin
const char* GetKicadPluginName( void )
{
    return "PLUGIN_3D_DEMO2";
}

// 提供此插件的版本
// provide the version of this plugin
void GetPluginVersion( unsigned char* Major, unsigned char* Minor,
    unsigned char* Patch, unsigned char* Revision )
{
    if( Major )
```

```
        *Major = PLUGIN_3D_DEMO2_MAJOR;

    if( Minor )
        *Minor = PLUGIN_3D_DEMO2_MINOR;

    if( Patch )
        *Patch = PLUGIN_3D_DEMO2_PATCH;

    if( Revision )
        *Revision = PLUGIN_3D_DEMO2_REVNO;

    return;
}


// 支持的扩展数量
// number of extensions supported
#ifdef _WIN32
#define NEXTS 1
#else
#define NEXTS 2
#endif

// 支持的筛选器集数量
// number of filter sets supported
#define NFILS 1

static char ext0[] = "txt";

#ifdef _WIN32
static char fil0[] = "demo (*.txt)|*.txt";
#else
static char ext1[] = "TXT";

static char fil0[] = "demo (*.txt;*.TXT)|*.txt;*.TXT";
#endif


static struct FILE_DATA
{
    char const* extensions[NEXTS];
    char const* filters[NFILS];

    FILE_DATA()
    {
        extensions[0] = ext0;
        filters[0] = fil0;
```

```
#ifndef _WIN32
        extensions[1] = ext1;
#endif
        return;
    }

} file_data;


int GetNExtensions( void )
{
    return NEXTS;
}


char const* GetModelExtension( int aIndex )
{
    if( aIndex < 0 || aIndex >= NEXTS )
        return NULL;

    return file_data.extensions[aIndex];
}


int GetNFilters( void )
{
    return NFILS;
}


char const* GetFileFilter( int aIndex )
{
    if( aIndex < 0 || aIndex >= NFILS )
        return NULL;

    return file_data.filters[aIndex];
}


// 返回 true, 因为此插件可以提供可视化数据
// return true since this plugin can provide visualization data
bool CanRender( void )
{
    return true;
}


// 创建可视化数据
```

```
// create the visualization data
SCENEGRAPH* Load( char const* aFileName )
{
    // 对于本演示，我们创建一个四面体( TX1)，包括
    // For this demonstration we create a tetrahedron (tx1) consisting
    // SCENEGRAPH (VRML 变换)，它依次包含 4
    // of a SCENEGRAPH (VRML Transform) which in turn contains 4
    // SGSHAPE (VRML Shape) 对象表示
    // SGSHAPE (VRML Shape) objects representing each of the sides of
    // 四面体。每个形状都与一种颜色关联 (SGAPPEARANCE)
    // the tetrahedron. Each Shape is associated with a color (SGAPPEARANCE)
    // 和一个 SGFACESET (VRML Geometry->indexedFaceSet)。每个 SGFACESET 是
    // and a SGFACESET (VRML Geometry->indexedFaceSet). Each SGFACESET is
    // 与顶点列表 (SGCOORDS) 关联，每顶点法线
    // associated with a vertex list (SGCOORDS), a per-vertex normals
    // list(SGNORMALS) 和坐标索引 (SGCOORDINDEX)。一个形状
    // list (SGNORMALS), and a coordinate index (SGCOORDINDEX). One shape
    // 用于表示每个面，以便我们可以使用逐顶点逐面
    // is used to represent each face so that we may use per-vertex-per-face
    // 法线。
    // normals.
    //
    // 四面体又是顶级 SCENEGRAPH(Tx0) 的子级
    // The tetrahedron in turn is a child of a top level SCENEGRAPH (tx0)
    // 它有第二个 SCENEGRAPH子(TX2)，这是一个转换
    // which has a second SCENEGRAPH child (tx2) which is a transformation
    // 四面体 TX1 (旋转+平移)。这表明
    // of the tetrahedron tx1 (rotation + translation). This demonstrates
    // 场景图层次结构中组件的重用。
    // the reuse of components within the scene graph hierarchy.

    // 定义四面体的顶点
    // define the vertices of the tetrahedron
    // 面 1: 0, 3, 1
    // face 1: 0, 3, 1
    // 面 2: 0, 2, 3
    // face 2: 0, 2, 3
    // 面 3: 1, 3, 2
    // face 3: 1, 3, 2
    // 面 4: 0, 1, 2
    // face 4: 0, 1, 2
    double SQ2 = sqrt( 0.5 );
    SGPOINT vert[4];
    vert[0] = SGPOINT( 1.0, 0.0, -SQ2 );
    vert[1] = SGPOINT( -1.0, 0.0, -SQ2 );
    vert[2] = SGPOINT( 0.0, 1.0, SQ2 );
    vert[3] = SGPOINT( 0.0, -1.0, SQ2 );
```

```
// 创建顶级转换；这将保存所有其他
// create the top level transform; this will hold all other
// Scenegraph 对象；转换可能包含其他转换和
// scenegraph objects; a transform may hold other transforms and
// 形状
// shapes
IFSG_TRANSFORM* tx0 = new IFSG_TRANSFORM( true );

// 创建将容纳形状的转换
// create the transform which will house the shapes
IFSG_TRANSFORM* tx1 = new IFSG_TRANSFORM( tx0->GetRawPtr() );

// 添加一个形状，我们将使用该形状定义四面体的一个面；
// add a shape which we will use to define one face of the tetrahedron;
// 形状保存 facesets 和外观
// shapes hold facesets and appearances
IFSG_SHAPE* shape = new IFSG_SHAPE( *tx1 );

// 添加 faceset；这些包含坐标列表，坐标索引，
// add a faceset; these contain coordinate lists, coordinate indices,
// 顶点列表、顶点索引，还可能包含颜色列表和
// vertex lists, vertex indices, and may also contain color lists and
// 它们的索引。
// their indices.

IFSG_FACESET* face = new IFSG_FACESET( *shape );

IFSG_COORDS* cp = new IFSG_COORDS( *face );
cp->AddCoord( vert[0] );
cp->AddCoord( vert[3] );
cp->AddCoord( vert[1] );

// 坐标索引-注意：强制三角形；
// coordinate indices - note: enforce triangles;
// 在不一定可能的实际插件中
// in real plugins where it is not necessarily possible
// 若要确定三角形从哪一侧可见，请执行以下操作：
// to determine which side a triangle is visible from,
// 2 必须为每个三角形指定点序
// 2 point orders must be specified for each triangle
IFSG_COORDINDEX* coordIdx = new IFSG_COORDINDEX( *face );
coordIdx->AddIndex( 0 );
coordIdx->AddIndex( 1 );
coordIdx->AddIndex( 2 );

// 创建外观；外观归形状所有
// create an appearance; appearances are owned by shapes
```

```
// 文件名将更改为
// magenta
IFSG_APPEARANCE* material = new IFSG_APPEARANCE( *shape);
material->SetSpecular( 0.1, 0.0, 0.1 );
material->SetDiffuse( 0.8, 0.0, 0.8 );
material->SetAmbient( 0.2, 0.2, 0.2 );
material->SetShininess( 0.2 );

// 法线
// normals
IFSG_NORMALS* np = new IFSG_NORMALS( *face );
SGVECTOR nval = S3D::CalcTriNorm( vert[0], vert[3], vert[1] );
np->AddNormal( nval );
np->AddNormal( nval );
np->AddNormal( nval );

//
// 形状 2
// Shape2
// 注意: 我们重用 IFSG* 包装器来创建和操作新的
// Note: we reuse the IFSG* wrappers to create and manipulate new
// 数据结构。
// data structures.
//
shape->NewNode( *tx1 );
face->NewNode( *shape );
coordIdx->NewNode( *face );
cp->NewNode( *face );
np->NewNode( *face );

// 顶点
// vertices
cp->AddCoord( vert[0] );
cp->AddCoord( vert[2] );
cp->AddCoord( vert[3] );

// 索引
// indices
coordIdx->AddIndex( 0 );
coordIdx->AddIndex( 1 );
coordIdx->AddIndex( 2 );

// 法线
// normals
nval = S3D::CalcTriNorm( vert[0], vert[2], vert[3] );
np->AddNormal( nval );
np->AddNormal( nval );
```

```
    np->AddNormal( nval );
    // 颜色 (红色)
    // color (red)
    material->NewNode( *shape );
    material->SetSpecular( 0.2, 0.0, 0.0 );
    material->SetDiffuse( 0.9, 0.0, 0.0 );
    material->SetAmbient( 0.2, 0.2, 0.2 );
    material->SetShininess( 0.1 );

    //
    // 形状 3
    // Shape3
    //
    shape->NewNode( *tx1 );
    face->NewNode( *shape );
    coordIdx->NewNode( *face );
    cp->NewNode( *face );
    np->NewNode( *face );

    // 顶点
    // vertices
    cp->AddCoord( vert[1] );
    cp->AddCoord( vert[3] );
    cp->AddCoord( vert[2] );

    // 索引
    // indices
    coordIdx->AddIndex( 0 );
    coordIdx->AddIndex( 1 );
    coordIdx->AddIndex( 2 );

    // 法线
    // normals
    nval = S3D::CalcTriNorm( vert[1], vert[3], vert[2] );
    np->AddNormal( nval );
    np->AddNormal( nval );
    np->AddNormal( nval );

    // 颜色 (绿色)
    // color (green)
    material->NewNode( *shape );
    material->SetSpecular( 0.0, 0.1, 0.0 );
    material->SetDiffuse( 0.0, 0.9, 0.0 );
    material->SetAmbient( 0.2, 0.2, 0.2 );
    material->SetShininess( 0.1 );

    //
    // 形状 4
```

```
// Shape4
//
shape->NewNode( *tx1 );
face->NewNode( *shape );
coordIdx->NewNode( *face );
cp->NewNode( *face );
np->NewNode( *face );

// 顶点
// vertices
cp->AddCoord( vert[0] );
cp->AddCoord( vert[1] );
cp->AddCoord( vert[2] );

// 索引
// indices
coordIdx->AddIndex( 0 );
coordIdx->AddIndex( 1 );
coordIdx->AddIndex( 2 );

// 法线
// normals
nval = S3D::CalcTriNorm( vert[0], vert[1], vert[2] );
np->AddNormal( nval );
np->AddNormal( nval );
np->AddNormal( nval );

// 颜色 (蓝色)
// color (blue)
material->NewNode( *shape );
material->SetSpecular( 0.0, 0.0, 0.1 );
material->SetDiffuse( 0.0, 0.0, 0.9 );
material->SetAmbient( 0.2, 0.2, 0.2 );
material->SetShininess( 0.1 );

// 创建整个四面体移位 Z+2 并旋转 2/3PI 的副本
// create a copy of the entire tetrahedron shifted Z+2 and rotated 2/3PI
IFSG_TRANSFORM* tx2 = new IFSG_TRANSFORM( tx0->GetRawPtr() );
tx2->AddRefNode( *tx1 );
tx2->SetTranslation( SGPOINT( 0, 0, 2 ) );
tx2->SetRotation( SGVECTOR( 0, 0, 1 ), M_PI*2.0/3.0 );

SGNODE* data = tx0->GetRawPtr();

// 删除包装器
// delete the wrappers
delete shape;
delete face;
```

```
    delete coordIdx;
    delete material;
    delete cp;
    delete np;
    delete tx0;
    delete tx1;
    delete tx2;

    return (SCENEGRAPH*)data;
}
```

3 应用程序编程接口（API）

插件通过应用程序编程接口（API）实现实现。每个插件类都有其特定的 API，在 3D 插件教程中，我们已经看到了由标题 “3d_plugin.h” 声明的 3D 插件 API 实现的示例。插件也可能依赖于 KiCad 源代码树中定义的其他 API; 在 3D 插件的情况下，支持模型可视化的所有插件必须与标题‘ifsg_all.h’及其包含的标题中声明的 Scene Graph API 交互。

本节描述了插件类实现可能需要的插件类 API 和其他 KiCad API 的详细信息。

3.1 插件类 API

目前只有一个为 KiCad 声明的插件类：3D 插件类。所有 KiCad 插件类都必须实现头文件‘kicad_plugin.h’中声明的一组基本函数; 这些声明称为 Base Kicad 插件类。不存在 Base Kicad 插件类的实现; 头文件的存在纯粹是为了确保插件开发人员在每个插件实现中实现这些定义的函数。

在 KiCad 中，插件加载器的每个实例都实现了插件提供的 API，就像插件加载器是提供插件服务的类一样。这是通过 Plugin 加载器类实现的，该类提供包含与插件实现的类似的函数名的公共接口; 如果例如没有加载插件，则参数列表可以变化以适应向用户通知可能遇到的任何问题的需要。在内部，插件加载器使用存储的指针指向每个 API 函数，以代表用户调用每个函数。

3.1.1 API：Base Kicad 插件类

Base Kicad 插件类由头文件‘kicad_plugin.h’定义。此标头必须包含在所有其他插件类的声明中; 例如，请参阅头文件‘3d_plugin.h’中的 3D 插件类声明。这些函数的原型在《plugin-classes（插件-类），Plugin Classes（插件类）》中简要描述。API 由 “pluginldr.cpp” 中定义的基本插件加载器实现。

为了帮助理解基本 KiCad 插件头所需的功能，我们必须查看基本插件 Loader 类中发生的情况。Plugin Loader 类声明了一个虚函数‘Open()’，它接受要加载的插件的完整路径。在特定的插件类加载器中实现‘Open()’函数最初将调用基本插件加载器的受保护的‘open()’函数; 这个基础‘open()’函数试图找到每个必需的基本插件函数的地址; 一旦检索到每个函数的地址，就会强制执行一些检查：

1. 调用插件‘GetKicadPluginClass()’，并将结果与插件加载器实现提供的插件类字符串进行比较; 如果这些字符串不匹配，则打开的插件不适用于 Plugin Loader 实例。
2. 调用插件‘GetClassVersion()’来检索插件实现的插件类 API 版本。

3. 插件加载器虚拟‘GetLoaderVersion()’函数被调用以检索由加载器实现的插件类 API 版本。
4. 插件和加载器报告的插件类 API 版本必须具有相同的主版本号，否则它们被认为是不兼容的。这是最基本的版本测试，它由基本插件加载器强制执行。
5. 使用插件加载器的插件类 API 版本信息调用插件‘CheckClassVersion()’；如果插件支持给定版本，则返回“true”表示成功。如果成功，加载器根据‘GetKicadPluginName()’和‘GetPluginVersion()’的结果创建一个 PluginInfo 字符串，插件加载过程在 Plugin Loader 的‘Open()’实现中继续。

3.1.2 API：3D 插件类

3D 插件类由头文件‘3d_plugin.h’声明，它扩展了所需的插件函数，如《class-plugin-3d（类-插件-3d），Plugin Class（插件类）：PLUGIN_3D（插件 _3D）》中所述。相应的插件加载器在‘pluginldr3D.cpp’中定义，除了所需的 API 函数之外，加载器还实现了以下公共函数：

```
/* 打开完整路径 "aFullFileName" 指定的插件 */
/* Open the plugin specified by the full path "aFullFileName" */
bool Open( const wxString& aFullFileName );

/* 关闭当前打开的插件 */
/* Close the currently opened plugin */
void Close( void );

/* 检索该插件加载器实现的插件类 API 版本 */
/* Retrieve the Plugin Class API Version implemented by this Plugin Loader */
void GetLoaderVersion( unsigned char* Major, unsigned char* Minor,
    unsigned char* Revision, unsigned char* Patch ) const;
```

所需的 3D 插件类功能通过以下功能公开：

```
/* 如果没有加载插件，则返回插件类或 NULL */
/* returns the Plugin Class or NULL if no plugin loaded */
char const* GetKicadPluginClass( void );

/* 如果没有加载插件，则返回 FALSE */
/* returns false if no plugin loaded */
bool GetClassVersion( unsigned char* Major, unsigned char* Minor,
    unsigned char* Patch, unsigned char* Revision );

/* 如果类版本检查失败或未加载插件，则返回 FALSE */
/* returns false if the class version check fails or no plugin is loaded */
bool CheckClassVersion( unsigned char Major, unsigned char Minor,
    unsigned char Patch, unsigned char Revision );

/* 返回插件名称，如果没有加载插件，则返回 NULL */
/* returns the Plugin Name or NULL if no plugin loaded */
const char* GetKicadPluginName( void );

/*
```

```
   如果未加载任何插件，则返回 False，否则返回参数。
   包含 GetPluginVersion() 的结果。
*/
/*
   returns false if no plugin is loaded, otherwise the arguments
   contain the result of GetPluginVersion()
 */
bool GetVersion( unsigned char* Major, unsigned char* Minor,
    unsigned char* Patch, unsigned char* Revision );

/*
   如果没有加载插件，则将 aPluginInfo 设置为空字符串，
   否则，将 aPluginInfo 设置为以下形式的字符串：
   [名称]：[主要].[次要].[修补程序].[修订版本]其中。
   name = 由 GetKicadPluginClass() 提供的名称。
   主要、次要、修补程序、修订=版本信息来自。
   GetPluginVersion()。
*/
/*
   sets aPluginInfo to an empty string if no plugin is loaded,
   otherwise aPluginInfo is set to a string of the form:
   [NAME]:[MAJOR].[MINOR].[PATCH].[REVISION] where
   NAME = name provided by GetKicadPluginClass()
   MAJOR, MINOR, PATCH, REVISION = version information from
   GetPluginVersion()
 */
void GetPluginInfo( std::string& aPluginInfo );
```

在典型情况下，用户将执行以下操作：

1. 创建‘KICAD_PLUGIN_LDR_3D’的实例。
2. 调用‘Open(“/path/to/myplugin.so”)’来打开一个特定的插件。必须检查返回值以确保根据需要加载插件。
3. 调用由‘KICAD_PLUGIN_LDR_3D’公开的任何 3D 插件类调用。
4. 调用‘Close()’来关闭（取消链接）插件。
5. 销毁‘KICAD_PLUGIN_LDR_3D’实例。

3.2 场景图类 API

Scenegraph 类 API 由标题‘ifsg_all.h’及其包含的标题定义。API 由许多辅助例程组成，命名空间为‘S3D’，在‘ifsg_api.h’中定义，包装类由各种‘ifsg_*.h’标题定义；包装器支持底层的场景图类，它们一起形成一个与 VRML2.0 静态场景图兼容的场景图结构。标题，结构，类及其公共函数如下：

sg_version.h

```
/*
   定义 SceneGraph 类的版本信息。
   所有使用 Scenegraph 类的插件都应该包含这个头。
   并对照所报告的版本检查版本信息。
   S3D::GetLibVersion() 以确保兼容性。
*/
/*
   Defines version information of the SceneGraph Classes.
   All plugins which use the scenegraph class should include this header
   and check the version information against the version reported by
   S3D::GetLibVersion() to ensure compatibility
 */

#define KICADSG_VERSION_MAJOR         2
#define KICADSG_VERSION_MINOR         0
#define KICADSG_VERSION_PATCH         0
#define KICADSG_VERSION_REVISION      0
```

sg_types.h

```
/*
   定义 SceneGraph 类类型; 这些类型。
   与 VRML2.0 节点类型密切相关。
*/
/*
   Defines the SceneGraph Class Types; these types
   are closely related to VRML2.0 node types.
 */

namespace S3D
{
    enum SGTYPES
    {
        SGTYPE_TRANSFORM = 0,
        SGTYPE_APPEARANCE,
        SGTYPE_COLORS,
        SGTYPE_COLORINDEX,
        SGTYPE_FACESET,
        SGTYPE_COORDS,
        SGTYPE_COORDINDEX,
        SGTYPE_NORMALS,
        SGTYPE_SHAPE,
        SGTYPE_END
    };
};
```

‘sg_base.h’ 头包含 scenegraph 类使用的基本数据类型的声明。

sg_base.h

```
/*
   这是相当于 VRML2.0 的 RGB 颜色模型。
   RGB 模型，其中每种颜色可能在。
   范围 [0..1]。
*/
/*
    This is an RGB color model equivalent to the VRML2.0
    RGB model where each color may have a value within the
    range [0..1].
 */

class SGCOLOR
{
public:
    SGCOLOR();
    SGCOLOR( float aRVal, float aGVal, float aBVal );

    void GetColor( float& aRedVal, float& aGreenVal, float& aBlueVal ) const;
    void GetColor( SGCOLOR& aColor ) const;
    void GetColor( SGCOLOR* aColor ) const;

    bool SetColor( float aRedVal, float aGreenVal, float aBlueVal );
    bool SetColor( const SGCOLOR& aColor );
    bool SetColor( const SGCOLOR* aColor );
};


class SGPOINT
{
public:
    double x;
    double y;
    double z;

public:
    SGPOINT();
    SGPOINT( double aXVal, double aYVal, double aZVal );

    void GetPoint( double& aXVal, double& aYVal, double& aZVal );
    void GetPoint( SGPOINT& aPoint );
    void GetPoint( SGPOINT* aPoint );

    void SetPoint( double aXVal, double aYVal, double aZVal );
    void SetPoint( const SGPOINT& aPoint );
};
```

```
/*
   SGVECTOR 有 3 个分量(x, y, z)类似于一个点; 但是。
   向量确保存储的值是规范化的, 并且。
   防止直接操作组件变量。
*/
/*
    A SGVECTOR has 3 components (x,y,z) similar to a point; however
    a vector ensures that the stored values are normalized and
    prevents direct manipulation of the component variables.
 */
class SGVECTOR
{
public:
    SGVECTOR();
    SGVECTOR( double aXVal, double aYVal, double aZVal );

    void GetVector( double& aXVal, double& aYVal, double& aZVal ) const;

    void SetVector( double aXVal, double aYVal, double aZVal );
    void SetVector( const SGVECTOR& aVector );

    SGVECTOR& operator=( const SGVECTOR& source );
};
```

'IFSG_NODE' 类是所有场景图节点的基类。所有 scenegraph 对象都实现此类的公共函数, 但在某些情况下, 特定函数可能对特定类没有意义。

ifsg_node.h

```
class IFSG_NODE
{
public:
    IFSG_NODE();
    virtual ~IFSG_NODE();

    /**
     * 功能销毁。
     * 删除此包装器持有的 Scenegraph 对象。
     */
    /**
     * Function Destroy
     * deletes the scenegraph object held by this wrapper
     */
    void Destroy( void );

    /**
     * 函数附加。
     * 将给定S GNODE* 与此包装器关联。
```

```
     */
    /**
     * Function Attach
     * associates a given SGNODE* with this wrapper
     */
    virtual bool Attach( SGNODE* aNode ) = 0;

    /**
     * 函数 NewNode。
     * 创建与此包装器关联的新节点。
     */
    /**
     * Function NewNode
     * creates a new node to associate with this wrapper
     */
    virtual bool NewNode( SGNODE* aParent ) = 0;
    virtual bool NewNode( IFSG_NODE& aParent ) = 0;

    /**
     * 函数 GetRawPtr()。
     * 返回原始内部SGNODE指针。
     */
    /**
     * Function GetRawPtr()
     * returns the raw internal SGNODE pointer
     */
    SGNODE* GetRawPtr( void );

    /**
     * 函数 GetNodeType。
     * 返回此节点实例的类型。
     */
    /**
     * Function GetNodeType
     * returns the type of this node instance
     */
    S3D::SGTYPES GetNodeType( void ) const;

    /**
     * 函数 GetParent。
     * 返回指向此对象的父 SGNODE 的指针。
     * 如果对象没有父对象，则为 NULL(即 顶级变换)。
     * 或者如果包装器当前未与 SGNODE 关联。
     */
    /**
     * Function GetParent
     * returns a pointer to the parent SGNODE of this object
     * or NULL if the object has no parent (ie. top level transform)
```

```
     * or if the wrapper is not currently associated with an SGNODE.
     */
    SGNODE* GetParent( void ) const;

    /**
     * 函数 SetParent。
     * 设置此对象的父 SGNODE。
     *
     * @param aParent [in] 是所需的父节点。
     * @return 如果操作成功，则返回 true；如果操作成功，则返回 false。
     * 给定节点不允许为父节点。
     * 派生对象。
     */
    /**
     * Function SetParent
     * sets the parent SGNODE of this object.
     *
     * @param aParent [in] is the desired parent node
     * @return true if the operation succeeds; false if
     * the given node is not allowed to be a parent to
     * the derived object.
     */
    bool SetParent( SGNODE* aParent );

    /**
     * 函数 GetNodeTypeName。
     * 返回节点类型的文本表示。
     * 如果节点不知何故具有无效类型，则为 NULL。
     */
    /**
     * Function GetNodeTypeName
     * returns the text representation of the node type
     * or NULL if the node somehow has an invalid type
     */
    const char * GetNodeTypeName( S3D::SGTYPES aNodeType ) const;

    /**
     * 函数 AddRefNode。
     * 添加对不属于的现有节点的引用。
     * (不是此节点的子节点)。
     *
     * @return 成功返回 true。
     */
    /**
     * Function AddRefNode
     * adds a reference to an existing node which is not owned by
     * (not a child of) this node.
     *
```

```
     * @return true on success
     */
    bool AddRefNode( SGNODE* aNode );
    bool AddRefNode( IFSG_NODE& aNode );

    /**
     * 函数 AddChildNode。
     * 添加一个节点作为此节点拥有的子节点。
     *
     * @return 成功返回 true。
     */
    /**
     * Function AddChildNode
     * adds a node as a child owned by this node.
     *
     * @return true on success
     */
    bool AddChildNode( SGNODE* aNode );
    bool AddChildNode( IFSG_NODE& aNode );
};
```

‘IFSG_TRANSFORM’类似于 VRML2.0 Transform 节点; 它可以包含任意数量的子 IFSG_SHAPE 和 IFSG_TRANSFORM 节点以及任意数量的引用的 IFSG_SHAPE 和 IFSG_TRANSFORM 节点。有效的场景图必须有一个“IFSG_TRANSFORM”对象作为根。

ifsg_transform.h

```
/**
 * IFSG_Transform 类。
 *是 VRML 兼容转换块类 SCENEGRAPH 的包装。
 */
/**
 * Class IFSG_TRANSFORM
 * is the wrapper for the VRML compatible TRANSFORM block class SCENEGRAPH
 */

class IFSG_TRANSFORM : public IFSG_NODE
{
public:
    IFSG_TRANSFORM( bool create );
    IFSG_TRANSFORM( SGNODE* aParent );

    bool SetScaleOrientation( const SGVECTOR& aScaleAxis, double aAngle );
    bool SetRotation( const SGVECTOR& aRotationAxis, double aAngle );
    bool SetScale( const SGPOINT& aScale );
    bool SetScale( double aScale );
    bool SetCenter( const SGPOINT& aCenter );
    bool SetTranslation( const SGPOINT& aTranslation );
```

```
    /* 此处未显示的各种基类函数 */
    /* various base class functions not shown here */
};
```

‘IFSG_SHAPE’ 类似于 VRML2.0 Shape 节点; 它必须包含单个子节点或引用 FACESET 节点，并且可以包含单个子节点或引用 APPEARANCE 节点。

ifsg_shape.h

```
/**
 * IFSG_SHAP E类。
 * 是 SGSHAPE 类的包装。
 */
/**
 * Class IFSG_SHAPE
 * is the wrapper for the SGSHAPE class
 */

class IFSG_SHAPE : public IFSG_NODE
{
public:
    IFSG_SHAPE( bool create );
    IFSG_SHAPE( SGNODE* aParent );
    IFSG_SHAPE( IFSG_NODE& aParent );

    /* 此处未显示的各种基类函数 */
    /* various base class functions not shown here */
};
```

‘IFSG_APPEARANCE’ 类似于 VRML2.0 Appearance 节点，但目前它只代表包含 Material 节点的 Appearance 节点的等价物。

ifsg_appearance.h

```
class IFSG_APPEARANCE : public IFSG_NODE
{
public:
    IFSG_APPEARANCE( bool create );
    IFSG_APPEARANCE( SGNODE* aParent );
    IFSG_APPEARANCE( IFSG_NODE& aParent );

    bool SetEmissive( float aRVal, float aGVal, float aBVal );
    bool SetEmissive( const SGCOLOR* aRGBColor );
    bool SetEmissive( const SGCOLOR& aRGBColor );

    bool SetDiffuse( float aRVal, float aGVal, float aBVal );
    bool SetDiffuse( const SGCOLOR* aRGBColor );
    bool SetDiffuse( const SGCOLOR& aRGBColor );

    bool SetSpecular( float aRVal, float aGVal, float aBVal );
```

```
    bool SetSpecular( const SGCOLOR* aRGBColor );
    bool SetSpecular( const SGCOLOR& aRGBColor );

    bool SetAmbient( float aRVal, float aGVal, float aBVal );
    bool SetAmbient( const SGCOLOR* aRGBColor );
    bool SetAmbient( const SGCOLOR& aRGBColor );

    bool SetShininess( float aShininess );
    bool SetTransparency( float aTransparency );

    /* 此处未显示的各种基类函数 */
    /* various base class functions not shown here */

    /* 以下函数在。
       外观节点，并始终返回失败代码
    /* the following functions make no sense within an
       appearance node and always return a failure code

        bool AddRefNode( SGNODE* aNode );
        bool AddRefNode( IFSG_NODE& aNode );
        bool AddChildNode( SGNODE* aNode );
        bool AddChildNode( IFSG_NODE& aNode );
     */
};
```

‘IFSG_FACESET’ 类似于包含 IndexedFaceSet 节点的 VRML2.0 Geometry 节点。它必须包含单个子节点或引用 COORDS 节点，单个子 COORDINDEX 节点以及单个子节点或引用 NORMALS 节点; 另外可能有一个子节点或引用 COLORS 节点。提供简化的法线计算功能以帮助用户将正常值分配给表面。与 VRML2.0 模拟的偏差如下:

1. 法线始终是每个顶点。
2. 颜色总是每个顶点。
3. 坐标索引集必须仅描述三角形面。

ifsg_faceset.h

```
/**
 * IFSG_FACESET 类。
 * 是 SGFACESET 类的包装。
 */
/**
 * Class IFSG_FACESET
 * is the wrapper for the SGFACESET class
 */

class IFSG_FACESET : public IFSG_NODE
{
public:
```

```
    IFSG_FACESET( bool create );
    IFSG_FACESET( SGNODE* aParent );
    IFSG_FACESET( IFSG_NODE& aParent );

    bool CalcNormals( SGNODE** aPtr );

    /* 此处未显示的各种基类函数 */
    /* various base class functions not shown here */
};
```

ifsg_coords.h

```
/**
 * IFSG_COORDS 类别。
 * 是 SGCOORDS 的包装器。
 */
/**
 * Class IFSG_COORDS
 * is the wrapper for SGCOORDS
 */

class IFSG_COORDS : public IFSG_NODE
{
public:
    IFSG_COORDS( bool create );
    IFSG_COORDS( SGNODE* aParent );
    IFSG_COORDS( IFSG_NODE& aParent );

    bool GetCoordsList( size_t& aListSize, SGPOINT*& aCoordsList );
    bool SetCoordsList( size_t aListSize, const SGPOINT* aCoordsList );
    bool AddCoord( double aXValue, double aYValue, double aZValue );
    bool AddCoord( const SGPOINT& aPoint );

    /* 此处未显示的各种基类函数 */
    /* various base class functions not shown here */

    /* 以下函数在。
       协调节点并始终返回失败代码
    /* the following functions make no sense within a
       coords node and always return a failure code

        bool AddRefNode( SGNODE* aNode );
        bool AddRefNode( IFSG_NODE& aNode );
        bool AddChildNode( SGNODE* aNode );
        bool AddChildNode( IFSG_NODE& aNode );
     */
};
```

‘IFSG_COORDINDEX’ 类似于 VRML2.0 coordIdx[] 集，除了它必须专门描述三角形面，这意味着索引的总数可以被 3 整除。

ifsg_coordindex.h

```
/**
 * IFSG_COORDINDEX 类。
 * 是 SGCOORDINDEX 的包装。
 */
/**
 * Class IFSG_COORDINDEX
 * is the wrapper for SGCOORDINDEX
 */

class IFSG_COORDINDEX : public IFSG_INDEX
{
public:
    IFSG_COORDINDEX( bool create );
    IFSG_COORDINDEX( SGNODE* aParent );
    IFSG_COORDINDEX( IFSG_NODE& aParent );

    bool GetIndices( size_t& nIndices, int*& aIndexList );
    bool SetIndices( size_t nIndices, int* aIndexList );
    bool AddIndex( int aIndex );

    /* 此处未显示的各种基类函数 */
    /* various base class functions not shown here */

    /* 以下函数在。
       协调节点并始终返回失败代码
    /* the following functions make no sense within a
       coordindex node and always return a failure code

        bool AddRefNode( SGNODE* aNode );
        bool AddRefNode( IFSG_NODE& aNode );
        bool AddChildNode( SGNODE* aNode );
        bool AddChildNode( IFSG_NODE& aNode );
     */
};
```

‘IFSG_NORMALS’ 相当于 VRML2.0 Normals 节点。

ifsg_normals.h

```
/**
 * IFSG_Normal 类。
 * 是 SGNORMALS 类的包装。
 */
/**
 * Class IFSG_NORMALS
```

```
 * is the wrapper for the SGNORMALS class
 */

class IFSG_NORMALS : public IFSG_NODE
{
public:
    IFSG_NORMALS( bool create );
    IFSG_NORMALS( SGNODE* aParent );
    IFSG_NORMALS( IFSG_NODE& aParent );

    bool GetNormalList( size_t& aListSize, SGVECTOR*& aNormalList );
    bool SetNormalList( size_t aListSize, const SGVECTOR* aNormalList );
    bool AddNormal( double aXValue, double aYValue, double aZValue );
    bool AddNormal( const SGVECTOR& aNormal );

    /* 此处未显示的各种基类函数 */
    /* various base class functions not shown here */

    /* 以下函数在。
       法线节点并始终返回失败代码
    /* the following functions make no sense within a
       normals node and always return a failure code

        bool AddRefNode( SGNODE* aNode );
        bool AddRefNode( IFSG_NODE& aNode );
        bool AddChildNode( SGNODE* aNode );
        bool AddChildNode( IFSG_NODE& aNode );
     */
};
```

‘IFSG_COLORS’ 类似于 VRML2.0 colors[] 集。

ifsg_colors.h

```
/**
 * IFSG_COLOR 类。
 * 是 SGCOLORS 的包装器。
 */
/**
 * Class IFSG_COLORS
 * is the wrapper for SGCOLORS
 */

class IFSG_COLORS : public IFSG_NODE
{
public:
    IFSG_COLORS( bool create );
    IFSG_COLORS( SGNODE* aParent );
    IFSG_COLORS( IFSG_NODE& aParent );
```

```
    bool GetColorList( size_t& aListSize, SGCOLOR*& aColorList );
    bool SetColorList( size_t aListSize, const SGCOLOR* aColorList );
    bool AddColor( double aRedValue, double aGreenValue, double aBlueValue );
    bool AddColor( const SGCOLOR& aColor );

    /* 此处未显示的各种基类函数 */
    /* various base class functions not shown here */

    /* 以下函数在。
       法线节点并始终返回失败代码
    /* the following functions make no sense within a
       normals node and always return a failure code

        bool AddRefNode( SGNODE* aNode );
        bool AddRefNode( IFSG_NODE& aNode );
        bool AddChildNode( SGNODE* aNode );
        bool AddChildNode( IFSG_NODE& aNode );
     */
};
```

其余的 API 函数在‘ifsg_api.h’中定义如下：

ifsg_api.h

```
namespace S3D
{
    /**
     * 函数 GetLibVersion 检索。
     * kicad_3dsg 库。
     */
    /**
     * Function GetLibVersion retrieves version information of the
     * kicad_3dsg library
     */
    SGLIB_API void GetLibVersion( unsigned char* Major, unsigned char* Minor,
                                  unsigned char* Patch, unsigned char* Revision );

    //从 SGNODE 指针提取信息的函数
    // functions to extract information from SGNODE pointers
    SGLIB_API S3D::SGTYPES GetSGNodeType( SGNODE* aNode );
    SGLIB_API SGNODE* GetSGNodeParent( SGNODE* aNode );
    SGLIB_API bool AddSGNodeRef( SGNODE* aParent, SGNODE* aChild );
    SGLIB_API bool AddSGNodeChild( SGNODE* aParent, SGNODE* aChild );
    SGLIB_API void AssociateSGNodeWrapper( SGNODE* aObject, SGNODE** aRefPtr );

    /**
     * 函数 CalcTriNorm
     * 返回顶点 p1, p2, p3 描述的三角形的法向量。
```

```
 */
/**
 * Function CalcTriNorm
 * returns the normal vector of a triangle described by vertices p1, p2, p3
 */
SGLIB_API SGVECTOR CalcTriNorm( const SGPOINT& p1, const SGPOINT& p2, const SGPOINT& p3 ↩
    );

/**
 * 函数 WriteCache
 * 将 SGNODE 树写入二进制缓存文件。
 *
 * @param aFileName 是要写入的文件的名称。
 * @param overwrite 必须设置为 true 才能覆盖现有文件。
 * @param 阳极是要写入的节点树中的任何节点。
 * @return 成功返回 true。
 */
/**
 * Function WriteCache
 * writes the SGNODE tree to a binary cache file
 *
 * @param aFileName is the name of the file to write
 * @param overwrite must be set to true to overwrite an existing file
 * @param aNode is any node within the node tree which is to be written
 * @return true on success
 */
SGLIB_API bool WriteCache( const char* aFileName, bool overwrite, SGNODE* aNode,
    const char* aPluginInfo );

/**
 * 函数 ReadCache
 * 读取二进制缓存文件并创建 SGNODE 树。
 *
 * @param aFileName 是要读取的二进制缓存文件的名称。
 * @return 失败返回 NULL，成功返回顶层 SCENEGRAPH 节点指针；
 * 如果需要，此节点可以通过以下方式与 IFSG_Transform 包装器关联。
 * IFSG_Transform::Attach() 函数。
 */
/**
 * Function ReadCache
 * reads a binary cache file and creates an SGNODE tree
 *
 * @param aFileName is the name of the binary cache file to be read
 * @return NULL on failure, on success a pointer to the top level SCENEGRAPH node;
 * if desired this node can be associated with an IFSG_TRANSFORM wrapper via
 * the IFSG_TRANSFORM::Attach() function.
 */
SGLIB_API SGNODE* ReadCache( const char* aFileName, void* aPluginMgr,
```

```
        bool (*aTagCheck)( const char*, void* ) );

    /**
     * 函数 WriteVRML
     * 将给定节点及其子节点写出到 VRML2 文件。
     *
     * @param filename 是输出文件的名称。
     * @param overwrite 应设置为 true 以覆盖现有 VRML 文件。
     * @param aTopNode 是指向代表 VRML 场景的 SCENEGRAPH 对象的指针。
     * @param reuse 应设置为 true 以使用 VRML DEF/USE 功能。
     * @return 成功返回 true。
     */
    /**
     * Function WriteVRML
     * writes out the given node and its subnodes to a VRML2 file
     *
     * @param filename is the name of the output file
     * @param overwrite should be set to true to overwrite an existing VRML file
     * @param aTopNode is a pointer to a SCENEGRAPH object representing the VRML scene
     * @param reuse should be set to true to make use of VRML DEF/USE features
     * @return true on success
     */
    SGLIB_API bool WriteVRML( const char* filename, bool overwrite, SGNODE* aTopNode,
                              bool reuse, bool renameNodes );

    // 注意: 以下函数组合用于创建 VRML。
    // 组装，可以使用模块的每个 SG* 表示的各种实例。
    // 典型的用例是:
    // 1.调用 'ResetNodeIndex()' 重置全局节点名索引。

    // 2.对于 'S3DCACHE->load()' 提供的每个模型指针，调用一次 'RenameNodes()';
    //     这可确保所有节点都有唯一的名称以呈现给最终输出文件。
    //     内部 RenameNodes() 只重命名给定节点和所有子节点;
    //     仅被引用的节点不会被重命名。使用提供的指针。
    //     通过 'S3DCACHE->load()' 确保除了返回的节点(顶层节点)之外的所有节点都是。
    //     至少一个节点的子节点，因此所有节点都有唯一的名称。
    // 3.如果 SG* 树是独立于 S3DCACHE 创建的 -> load()，则用户必须调用。
    //     RenameNodes() 以确保所有节点都具有唯一的名称。
    // 4.根据需要创建新的 ifsg_transform 节点来创建程序集结构。
    //     对于组件的每个实例; 由返回的组件基础模型。
    //     S3DCACHE->load() 可以通过 'AddRefNode()' 添加到这些 IFSG_Transform 节点;
    //     设置 IFSG_Transform 节点的偏移、旋转等，确保正确。
    // 5.确保所有新的 IFSG_Transform 节点都作为子节点放置在。
    //     顶层 IFSG_Transform 节点，为最终节点命名和输出做准备。
    // 6.在顶层组装节点上调用 RenameNodes()。
    // 7.正常调用 WriteVRML(), renameNodes=false, 写入整个程序集。
    //     结构转换为单个 VRML 文件。
    // 8.通过删除任何额外的 IFSG_Transform 包装器及其底层 SG* 进行清理。
```

```
//    专门为程序集输出创建的类
// NOTE: The following functions are used in combination to create a VRML
// assembly which may use various instances of each SG* representation of a module.
// A typical use case would be:
// 1. invoke 'ResetNodeIndex()' to reset the global node name indices
// 2. for each model pointer provided by 'S3DCACHE->Load()', invoke 'RenameNodes()' once;
//    this ensures that all nodes have a unique name to present to the final output file.
//    Internally, RenameNodes() will only rename the given node and all Child subnodes;
//    nodes which are only referenced will not be renamed. Using the pointer supplied
//    by 'S3DCACHE->Load()' ensures that all nodes but the returned node (top node) are
//    children of at least one node, so all nodes are given unique names.
// 3. if SG* trees are created independently of S3DCACHE->Load() the user must invoke
//    RenameNodes() as appropriate to ensure that all nodes have a unique name
// 4. create an assembly structure by creating new IFSG_TRANSFORM nodes as appropriate
//    for each instance of a component; the component base model as returned by
//    S3DCACHE->Load() may be added to these IFSG_TRANSFORM nodes via 'AddRefNode()';
//    set the offset, rotation, etc of the IFSG_TRANSFORM node to ensure correct
// 5. Ensure that all new IFSG_TRANSFORM nodes are placed as child nodes within a
//    top level IFSG_TRANSFORM node in preparation for final node naming and output
// 6. Invoke RenameNodes() on the top level assembly node
// 7. Invoke WriteVRML() as normal, with renameNodes = false, to write the entire assembly
//    structure to a single VRML file
// 8. Clean up by deleting any extra IFSG_TRANSFORM wrappers and their underlying SG*
//    classes which have been created solely for the assembly output

/**
 * 函数 ResetNodeIndex
 * 重置全局 SG* 类索引。
 *
 *@param 阳极可以是任何有效的 SGNODE。
 */
/**
 * Function ResetNodeIndex
 * resets the global SG* class indices
 *
 * @param aNode may be any valid SGNODE
 */
SGLIB_API void ResetNodeIndex( SGNODE* aNode );

/**
 * 函数 RenameNodes
 * 根据当前重命名节点和所有子节点。
 * 全局 SG* 的值类别索引。
 *
 * @param 阳极是顶层节点。
```

```
 */
/**
 * Function RenameNodes
 * renames a node and all children nodes based on the current
 * values of the global SG* class indices
 *
 * @param aNode is a top level node
 */
SGLIB_API void RenameNodes( SGNODE* aNode );

/**
 * 函数 DestroyNode。
 * 删除给定 SG* 类节点。此功能使其成为可能。
 * 安全删除 SG* 节点而不将节点与关联。
 * 其对应的 IFSG* 包装器。
 */
/**
 * Function DestroyNode
 * deletes the given SG* class node. This function makes it possible
 * to safely delete an SG* node without associating the node with
 * its corresponding IFSG* wrapper.
 */
SGLIB_API void DestroyNode( SGNODE* aNode );

// 注意：以下函数便于创建和销毁。
// 用于呈现的数据结构
// NOTE: The following functions facilitate the creation and destruction
// of data structures for rendering

/**
 * 函数 GetModel
 * 创建阳极的 S3DMODEL 表示（原始数据，无转换）。
 *
 * @param 阳极是要转录为 S3DMODEL 表示的节点。
 * @return 成功时返回阳极的 S3DMODEL 表示，否则为 NULL。
 */
/**
 * Function GetModel
 * creates an S3DMODEL representation of aNode (raw data, no transforms)
 *
 * @param aNode is the node to be transcribed into an S3DMODEL representation
 * @return an S3DMODEL representation of aNode on success, otherwise NULL
 */
SGLIB_API S3DMODEL* GetModel( SCENEGRAPH* aNode );

/**
 * 函数 Destroy3DModel
 * 释放 S3DMODEL 结构使用的内存并将指针设置为。
```

```
     * 结构为 NULL。
     */
    /**
     * Function Destroy3DModel
     * frees memory used by an S3DMODEL structure and sets the pointer to
     * the structure to NULL
     */
    SGLIB_API void Destroy3DModel( S3DMODEL** aModel );

    /**
     * 函数 Free3DModel
     * 释放由 S3DMODEL 结构内部使用的内存。
     */
    /**
     * Function Free3DModel
     * frees memory used internally by an S3DMODEL structure
     */
    SGLIB_API void Free3DModel( S3DMODEL& aModel );

    /**
     * 函数 Free3DMesh
     * 释放 SMESH 结构内部使用的内存。
     */
    /**
     * Function Free3DMesh
     * frees memory used internally by an SMESH structure
     */
    SGLIB_API void Free3DMesh( SMESH& aMesh );

    /**
     * 函数 New3DModel
     * 创建并初始化 S3DMODEL 结构。
     */
    /**
     * Function New3DModel
     * creates and initializes an S3DMODEL struct
     */
    SGLIB_API S3DMODEL* New3DModel( void );

    /**
     * 函数 Init3DMaterial
     * 初始化 SMATERIAL 结构。
     */
    /**
     * Function Init3DMaterial
     * initializes an SMATERIAL struct
     */
    SGLIB_API void Init3DMaterial( SMATERIAL& aMat );
```

```
     /**
      * 函数 Init3DMesh
      * 创建并初始化 SMESH 结构。
      */
     /**
      * Function Init3DMesh
      * creates and initializes an SMESH struct
      */
     SGLIB_API void Init3DMesh( SMESH& aMesh );
};
```

有关 Scenegraph API 的实际使用示例，请参阅上面的《advanced-3d-plugin（高级-3d-插件）,Advanced 3D Plugin tutorial（高级 3D 插件教程)》，以及 KiCad VRML1，VRML2 和 X3D 解析器。