KiCad プラグイン

July 30, 2017

Contents

_KiCad プラグインシステム _

* 貢献者 *

Cirilo Bernardo

翻訳

starfort <starfort AT nifty.com>, 2017.

フィードバック

バグ報告や提案はこちらへお知らせください:

- KiCad のドキュメントについて: https://github.com/KiCad/kicad-doc/issues
- KiCad ソフトウェアについて: https://bugs.launchpad.net/kicad
- KiCad ソフトウェアの国際化について: https://github.com/KiCad/kicad-i18n/issues

発行日とソフトウエアのバージョン

2016 年 1 月 29 日発行

1 KiCad プラグインシステムについて

KiCad プラグインシステムは、共有ライブラリを用いた KiCad の機能を拡張するためのフレームワークです。プラグインを使用する主な利点の一つは、プラグイン開発中に KiCad パッケージ一式を再構築する必要がないということです。実際、プラグインは、KiCad ソースツリーにあるヘッダのごく小さなセットを使ってビルドすることができます。開発者が直接プラグインに関するコードをコンパイルするだけで済むことが保証されると、その結果として各ビルトとテストのサイクルに必要な時間を減らすことができるので、プラグイン開発において KiCad をビルドする必要をなくすということは、生産性を大きく向上させることになります。

新しいモデル形式に対して KiCad ソースのメジャーな変更を伴うことなく、より多くの 3D モデル形式をサポートする目的で、プラグインは当初、3D モデルビューア用に開発されました。プラグインフレームワークは、将来の開発者がプラグインの異なったクラスを作成できるよう、後に一般化されたものです。今のところ、KiCad では 3D プラグインのみ実装されていますが、将来的にはデータのインポートとエクスポートをユーザーによって実装できるようにするための PCB プラグインが開発される見通しです。

1.1 プラグインのクラス

各プラグインはそれぞれ特定領域に関する問題を処理するので、その領域毎に別々のインターフェイスが必要となります。このため、プラグインはプラグインクラスへと分類されます。例えば、3D モデルプラグインはファイルから 3D モデルデータを読み込んで、3D ビューアで表示できるようなフォーマットへとデータを変換します。PCB インポート/エクスポートプラグインは PCB のデータを受け取って別の電気的あるいは機械的なデータフォーマットへとエクスポートしたり、外部フォーマットを KiCad PCB 形式に変換したりします。現時点では、3D プラグインクラスのみ開発されており、本文書でも集中して取り上げていきます。

プラグインクラスを実装するには、KiCad ソースツリー内でプラグインの読み込み管理を行うコードを作成することが必要です。全てのプラグインローダーに対する基本クラスは、KiCad ソースツリー内のファイル `plugins/ldr/pluginldr.h` で宣言されています。このクラスは、あらゆる KiCad プラグインで見つかるであろう (お約束のコードである) 最も基本的な関数を宣言しており、プラグインローダーと利用可能なプラグイン間のバージョン互換について最低限のチェックを行う機能を持っています。ヘッダ `plugins/ldr/3d/pluginldr3D.h` には、3D プラグインクラスのためのローダーが宣言されています。ローダーは与えられたプラグインの読み込みを担当し、プラグインが持っている関数を KiCad で利用可能にします。各プラグインローダーのインスタンスはプラグイン実装の実体であり、KiCad とプラグインの関数を透過的に橋渡しするよう振舞います。プラグインをサポートするために KiCad 内部で必要とされるコードはローダーだけではありません: プラグインを見つけるためのコード、プラグインローダー経由でプラグインの関数を呼び出すコードも必要です。3D プラグインの場合、この発見と呼び出しの関数は S3D_CACHE クラス内に全て含まれています。

新規のプラグインクラスを開発する場合以外、プラグインの開発者はプラグイン管理に関する KiCad 内部コードの詳細を知る必要はありません; プラグインに自身が属するプラグインクラスで宣言されている関数を定義していくだけで済みます。

ヘッダ `include/plugins/kicad_plugin.h` には、全ての KiCad プラグインで必要とされるジェネリック関数が宣言されています; これらの関数は、プラグインクラスを識別し、固有のプラグイン名、プラグインクラス API に関するバージョン情報、固有のプラグインに関するバージョン情報、プラグインクラス API 上での最低限のバージョン互換チェック機能を提供します。以下は、これらの関数についての要約です:

```
/* プラグイン クラス名を UTF-8 文字列で返す */
char const* GetKicadPluginClass( void );
```

```
/* プラグイン クラス API に関するバージョン情報を返す */
void GetClassVersion( unsigned char* Major, unsigned char* Minor,
     unsigned char* Patch, unsigned char* Revision );

/*
   プラグインに実装されたバージョンチェックが与えられた
   プラグイン クラス API との互換性を確認できた場合、true を返す
 */
bool CheckClassVersion( unsigned char Major,
    unsigned char Minor, unsigned char Patch, unsigned char Revision );

/* 固有のプラグイン名を返す、(例) "PLUGIN_3D_VRML" */
const char* GetKicadPluginName( void );

/* 固有のプラグインに関するバージョン情報を返す */
void GetPluginVersion( unsigned char* Major, unsigned char* Minor,
     unsigned char* Patch, unsigned char* Revision );
```

1.1.1 プラグインクラス: PLUGIN_3D

ヘッダ include/plugins/3d/3d_plugin.h には、全ての 3D プラグインで実装されなければならない関数が宣言されており、プラグインにとって必要な及びユーザーが再実装する必要がない幾つかの関数が定義されています。ユーザーが再実装する必要がない定義済み関数は以下のとおりです:

```
/* プラグイン クラス名 "PLUGIN_3D" を返す */
char const* GetKicadPluginClass( void );

/* PLUGIN_3D API に関するバージョン情報を返す */
void GetClassVersion( unsigned char* Major, unsigned char* Minor,
     unsigned char* Patch, unsigned char* Revision );

/*
   PLUGIN_3D クラスに関し、ローダーの開発者による強制的な
   最低限のバージョンチェックを行なう。チェックにパスした場合、
   true を返す
 */
bool CheckClassVersion( unsigned char Major, unsigned char Minor,
     unsigned char Patch, unsigned char Revision );
```

ユーザーが実装しなければならない関数は次のとおりです:

```
/* プラグインでサポートされている拡張子文字列の数を返す */
int GetNExtensions( void );

/*
   要求された拡張子の文字列を返す; 有効な値は 0 から
   GetNExtensions() - 1
```

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

/* プラグインでサポートされているファイル フィルタの合計数を返す */
int GetNFilters( void );

/*
   要求されたファイル フィルタを返す; 有効な値は 0 から
   GetNFilters() - 1
 */
char const* GetFileFilter( int aIndex );

/*
    この 3D モデル タイプをレンダリングできる場合、true を返す
    プラグインがビジュアル モードを提供しないことがあるかも知れないが、その場合は
    false を返さなければならない
 */
bool CanRender( void );

/* 指定されたモデルを読み込み、ビジュアル モデル データへのポインタを返す */
SCENEGRAPH* Load( char const* aFileName );
```

2 チュートリアル: 3D プラグインクラス

この章では２つのとても単純な PLUGIN_3D クラスプラグインについて解説し、ユーザーがセットアップとコードのビルドを習得できるよう予行演習を行います。

2.1 基本的な 3D プラグイン

このチュートリアルでは、”PLUGIN_3D_DEMO1” という名前の非常に基本的な 3D プラグインを開発することで予行演習を行います。このチュートリアルの目的は、KiCad ユーザーが 3D モデルをブラウズする際に使用可能なファイル名をフィルタリングする幾つかの文字列を提供するだけという、非常に基本的な 3D プラグインの作成方法をデモすることです。ここでデモしているコードは任意の 3D プラグインに対する最低必要条件であり、より高度なプラグインを作成するためのテンプレートとして使用することができます。

デモプロジェクトをビルドするために必要なものは以下のとおりです:

- CMake
- KiCad プラグインヘッダ
- KiCad Scene Graph ライブラリ `kicad_3dsg`

自動的に KiCad のヘッダとライブラリを検出するためには、CMake FindPackage スクリプトを使うべきでしょう; 関連するヘッダファイルが `${KICAD_ROOT_DIR}/kicad` に、KiCad Scene Graph ライブラリが `${KICAD_ROOT_DIR}/`

lib にインストールされているなら、このチュートリアルで提供されているスクリプトは Linux および Windows 上で動作するはずです。

まず手始めに、プロジェクトディレクトリと FindPackage スクリプトを作成してみましょう:

```
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" )

    # sg_version.h からバージョン情報の取得を試みる
    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" )

        # "#define KICADSG_VERSION*" 行を抜き出す
        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()

        # NOT SG3D_VERSION* が '0' と評価されることを保証する
        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
```

KiCad とそのプラグインヘッダーをインストールしておく必要があります; もし Linux でそれらがユーザーディレクトリまたは `/opt` の下にインストールされているか、或いは Windows を使っているならば、KiCad の `include` と `lib` ディレクトリを含むディレクトリを指すように `KICAD_ROOT_DIR` 環境変数をセットする必要があります。OS X では、ここに示された FindPackage スクリプトを多少調整する必要があるでしょう。

チュートリアルのコードを組んでビルドするため、CMake を使用して `CMakeLists.txt` スクリプトファイルを作成します:

```
cd ${DEMO_ROOT}
cat > CMakeLists.txt << _EOF
# declare the name of the project
```

```
project( PLUGIN_DEMO )

# 必要な機能を全て備えた CMake のバージョンかどうかチェックする
cmake_minimum_required( VERSION 2.8.12 FATAL_ERROR )

# FindKICAD スクリプトがどこにあるか CMake に通知する
set( CMAKE_MODULE_PATH ${PROJECT_SOURCE_DIR}/CMakeModules )

# インストール済の KiCad ヘッダとライブラリを見つけるよう試みる
# そして変数にセットする:
#     KICAD_INCLUDE_DIR
#     KICAD_LIBRARY
find_package( KICAD 1.0 REQUIRED )

# コンパイラの検索パスに KiCad include ディレクトリを追加する
include_directories( ${KICAD_INCLUDE_DIR}/kicad )

# s3d_plugin_demo1 という名前のプラグインを作成する
add_library( s3d_plugin_demo1 MODULE
    src/s3d_plugin_demo1.cpp
    )

_EOF
```

最初のデモプロジェクトはとても基本的なものです; コンパイラのデフォルト以外は外部リンクの依存関係を持たない単一ファイルで構成されています。まずはソースディレクトリを作成することから始めます:

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

ではプラグインソース自体を作成しましょう:

s3d_plugin_demo1.cpp

```
#include <iostream>

//  3d_plugin.h ヘッダに 3D プラグインで必要とされる関数を定義する
#include "plugins/3d/3d_plugin.h"

// このプラグインのバージョン情報を定義する; 3d_plugin.h で定義されている
// プラ??グイン クラス バージョンと混同しないでください
#define PLUGIN_3D_DEMO1_MAJOR 1
#define PLUGIN_3D_DEMO1_MINOR 0
#define PLUGIN_3D_DEMO1_PATCH 0
#define PLUGIN_3D_DEMO1_REVNO 0

// このプラグイン名をユーザーに提供する関数を実装する
const char* GetKicadPluginName( void )
```

```
{
    return "PLUGIN_3D_DEMO1";
}

// このプラグインのバージョンをユーザに提供する関数を実装する
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;
}

// サポートされている拡張子の数; *NIX システムでは拡張子が
// 倍になります - 一つは小文字、もう一つは大文字です
#ifdef _WIN32
    #define NEXTS 7
#else
    #define NEXTS 14
#endif

// サポートされているフィルタ セットの数
#define NFILS 5

// このプラグインがユーザーに提供するフィルタ文字列と
// 拡張子の文字列を定義する
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

// 拡張子とフィルタ文字列のリストへのアクセスに便利な
// 構造体をインスタンス化する
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;


// このプラグインでサポートされている拡張子の数を返す
int GetNExtensions( void )
{
    return NEXTS;
}

// インデックス化された拡張子文字列を返す
char const* GetModelExtension( int aIndex )
{
    if( aIndex < 0 || aIndex >= NEXTS )
        return NULL;

    return file_data.extensions[aIndex];
}

// このプラグインで提供されるフィルタ文字列の数を返す
int GetNFilters( void )
{
    return NFILS;
}

// インデックス化されたフィルタ文字列を返す
char const* GetFileFilter( int aIndex )
{
    if( aIndex < 0 || aIndex >= NFILS )
        return NULL;

    return file_data.filters[aIndex];
}

// このプラグインは可視化されたデータを提供しないので、false を返す
bool CanRender( void )
{
    return false;
}

// このプラグインは可視化されたデータを提供しないので、NULL を返す
SCENEGRAPH* Load( char const* aFileName )
{
    // this dummy plugin does not support rendering of any models
    return NULL;
```

```
}
```

このソースファイルは 3D プラグインを実装するための最低必要条件を満足しています。このプラグインはモデルをレンダリングするためのいかなるデータも作り出しませんが、3D モデルファイル選択ダイアログを機能強化してサポートされているモデル拡張子のリストとファイル拡張子フィルタを KiCad に提供します。KiCad 内で拡張子文字列は特定の 3D モデルを読み込むために使われるプラグインの選択で使用されます; 例えば、もしプラグインが `wrl` ならば、KiCad はプラグインが可視化されたデータを返すまで拡張子 `wrl` をサポートすると宣言された各プラグインを呼び出すでしょう。各プラグインが提供するファイルフィルタはブラウジング UI を改良すべく 3D ファイル選択ダイアログへと渡されます。

プラグインをビルドするには:

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

プラグインはビルドされますが、インストールはされません; プラグインを読み込みたいのであれば、KiCad のプラグインディレクトリにプラグインファイルをコピーする必要があります。

2.2 高度な 3D プラグイン

このチュートリアルでは、”PLUGIN_3D_DEMO2” という名前の 3D プラグインを開発することで予行演習を行います。このチュートリアルの目的は、レンダリング可能な KiCad プレビューアで非常に基本的なシーングラフの構造図をデモすることです。このプラグインは `txt` タイプのファイルを要求します。キャッシュマネージャーがプラグインを呼び出すためにはファイルが存在しなければなりませんが、ファイルの中身はプラグインでは処理されません; 代わりにプラグインは単に四面体のペアを含んだシーングラフを作るだけです。このチュートリアルは、最初のチュートリアルを完了しており CMakeLists.txt と FindKICAD.cmake スクリプトファイルが既に作られている状況を想定して書かれています。

前のチュートリアルのソースファイルと同じディレクトリに新しいソースファイルを置き、このチュートリアルをビルドするため前のチュートリアルの CMakeLists.txt ファイルを拡張しましょう。このプラグインは KiCad のシーングラフを作るので、KiCad のシーングラフライブラリ `kicad_3dsg` へのリンクが必要となります。KiCad のシーングラフライブラリはシーングラフオブジェクトをビルドするために使われるクラスのセットを提供します。シーングラフオブジェクトは 3D キャッシュマネージャで使われる可視化フォーマットの中間データです。モデルの可視化をサポートする全てのプラグインは、このライブラリ経由でモデルデータをシーングラフへと変換しなければなりません。

最初のステップは、このシュートリアルプロジェクトをビルドできるように `CMakeLists.txt` を拡張することです:

```
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
```

ではソースディレクトリへと移動してソースファイルを作成しましょう:

```
cd ${DEMO_SRC}
```

s3d_plugin_demo2.cpp

```
#include <cmath>
// 3D Plugin Class declarations
#include "plugins/3d/3d_plugin.h"
// interface to KiCad Scene Graph Library
#include "plugins/3dapi/ifsg_all.h"

//  このプラグインのバージョン情報
#define PLUGIN_3D_DEMO2_MAJOR 1
#define PLUGIN_3D_DEMO2_MINOR 0
#define PLUGIN_3D_DEMO2_PATCH 0
#define PLUGIN_3D_DEMO2_REVNO 0

// このプラグインの名前を提供する
const char* GetKicadPluginName( void )
{
    return "PLUGIN_3D_DEMO2";
}

// このプラグインのバージョンを提供する
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;
}


// サポートされている拡張子の数
#ifdef _WIN32
#define NEXTS 1
#else
#define NEXTS 2
```

```
#endif

// サポートされているフィルタ セットの数
#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];
}


// このプラグインは可視化されたデータを提供するので、true を返す
bool CanRender( void )
{
    return true;
}


// 可視化データを作成する
SCENEGRAPH* Load( char const* aFileName )
{
    // このデモでは、四面体の各面をなす４つの SGSHAPE (VRML Shape)
    // オブジェクトを含んだ SCENEGRAPH (VRML Transform) から構成
    // される四面体 (tx1) を作ります。各 SGSHAPE は色 (SGAPPEARANCE)
    // と SGFACESET (VRML Geometry->indexedFaceSet) を伴います。
    // 各 SGFACESET は vertex list (SGCOORDS)、per-vertex normals
    // list (SGNORMALS) と coordinate index (SGCOORDINDEX) に紐付け
    // られています。
    // 頂点法線と面法線を使えるようにするため、一つのシェイプが各面を表すように
    // 使用されます。
    //
    // 連続した四面体は、四面体 tx1 (rotation + translation) のトランスフォームである
    // ２つ目の子供 SCENEGRAPH (tx2) を持つトップ レベル SCENEGRAPH (tx0) の
    // 子供です。これはシーン グラフ階層内でコンポーネントを再利用する デモとなって
    // います。

    // 四面体の頂点定義
    // 面 1: 0, 3, 1
    // 面 2: 0, 2, 3
    // 面 3: 1, 3, 2
    // 面 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 );


// トップ レベル トランスフォームを作成する; これは全ての異なった
// scenegraph オブジェクトを保持するだろう; あるトランスフォームは
// 別のトランスフォームとシェイプを保持してもよい
IFSG_TRANSFORM* tx0 = new IFSG_TRANSFORM( true );

// シェイプを保持するためのトランスフォームを作成する
IFSG_TRANSFORM* tx1 = new IFSG_TRANSFORM( tx0->GetRawPtr() );

// 四面体の一つの面を定義するために用いられるシェイプを追加する;
// シェイプはファイス セットと外見 (appearance) を保持している
IFSG_SHAPE* shape = new IFSG_SHAPE( *tx1 );

// フェイス セットを追加する; これらは、座標リスト、座標位置、
// 頂点リスト、頂点位置を含み、色のリストとその位置を
// 含んでもよい

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] );

// 座標位置 - 注記: 強制的に三角形となる;
// 実際のプラグインでは、三角形のどちらの面から
// 見えるかを決定できるようにする必要はなく、
// 2 点の順番で各三角形を指定しなければならない
IFSG_COORDINDEX* coordIdx = new IFSG_COORDINDEX( *face );
coordIdx->AddIndex( 0 );
coordIdx->AddIndex( 1 );
coordIdx->AddIndex( 2 );

// 外見 (appearance) を作成する; 外見 (appearance) はマゼンタ色のシェイプによって

// 所有される
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 );
```

```
    // 法線
    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 );

    //
    // シェイプ2
    // 注記: 新しい構造体を作成して操作するために
    //  IFSG* ラッパーを再利用する
    //
    shape->NewNode( *tx1 );
    face->NewNode( *shape );
    coordIdx->NewNode( *face );
    cp->NewNode( *face );
    np->NewNode( *face );

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

    // 位置
    coordIdx->AddIndex( 0 );
    coordIdx->AddIndex( 1 );
    coordIdx->AddIndex( 2 );

    // 法線
    nval = S3D::CalcTriNorm( vert[0], vert[2], vert[3] );
    np->AddNormal( nval );
    np->AddNormal( nval );
    np->AddNormal( nval );
    // 色 (赤)
    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 );

    //
    // シェイプ3
    //
    shape->NewNode( *tx1 );
    face->NewNode( *shape );
    coordIdx->NewNode( *face );
    cp->NewNode( *face );
    np->NewNode( *face );
```

```
// 頂点
cp->AddCoord( vert[1] );
cp->AddCoord( vert[3] );
cp->AddCoord( vert[2] );

// 位置
coordIdx->AddIndex( 0 );
coordIdx->AddIndex( 1 );
coordIdx->AddIndex( 2 );

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

// 色 (緑)
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 );

//
// シェイプ4
//
shape->NewNode( *tx1 );
face->NewNode( *shape );
coordIdx->NewNode( *face );
cp->NewNode( *face );
np->NewNode( *face );

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

// 位置
coordIdx->AddIndex( 0 );
coordIdx->AddIndex( 1 );
coordIdx->AddIndex( 2 );

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

```
    // 色（青）
    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 );

    // Z+2 シフトされ、2/3PI 回転した完全な四面体のコピーを作成する
    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();

    // ラッパーを削除する
    delete shape;
    delete face;
    delete coordIdx;
    delete material;
    delete cp;
    delete np;
    delete tx0;
    delete tx1;
    delete tx2;

    return (SCENEGRAPH*)data;
}
```

3 アプリケーションプログラミングインタフェース (API)

プラグインはアプリケーションプログラミングインタフェース (API) の実装により開発されます。各プラグインクラスは固有の API を持っており、3D プラグインチュートリアルでは `3d_plugin.h` ヘッダで宣言された 3D プラグイン API の実装例を見てきました。プラグインはまた KiCad ソースツリーで定義された別の API にも依存しています; 3D プラグインの例では、モデルの可視化をサポートする全てのプラグインは `ifsg_all.h` ヘッダとそれ自身が包んでいるヘッダとで宣言された Scene Graph API と密接に関わらなければなりません。

この章では利用可能なプラグインクラス API とプラグインクラスの実装に必要と思われる別の KiCad API の詳細について説明します。

3.1 プラグインクラス API

今のところ、KiCad で宣言されているプラグインクラスは次の一つだけです: 3D プラグインクラス。全ての KiCad プラグインクラスは `kicad_plugin.h` で宣言されている基本的な関数のセットを実装しなければなりません; これ

らの宣言はベース KiCad プラグインクラスとして参照されます。ベース KiCad プラグインクラスの実装は存在していません; ヘッダファイルはプラグイン開発者が各プラグインでこれらの定義済み関数を確実に実装するためだけに存在しています。

KiCad では、プラグインローダーの各インスタンスはプラグインが公開する API を提供します。プラグインローダーはプラグインのサービスを提供するクラスのようなものです。これはプラグインで実装されたものと同様な関数名を含んだパブリックインターフェイスをプラグインローダークラスが提供することで実現されています; 引数リストは、例えばプラグインが読み込まれていないというような予見される問題をユーザーに知らせる必要に応じて、適宜変更しても構いません。内部的には、プラグインローダーはユーザーに代わって各関数を呼び出すために各 API 関数へのストアドポインタを使用します。

3.1.1 API: ベース KiCad プラグインクラス

ベース KiCad プラグインクラスはヘッダファイル `kicad_plugin.h` で定義されます。このヘッダは全ての異なるプラグインクラスの宣言に含まれなければなりません; 例としてヘッダファイル `3d_plugin.h` で宣言された 3D プラグインクラスを見てみましょう。これらの関数のプロトタイプは Plugin Classes内で簡潔に記述されています。API は `pluginldr.cpp` で定義されているようにベースプラグインローダーによって提供されます。

ベース KiCad プラグインヘッダで要求される関数を理解するには、ベースプラグインローダークラスで何が起こるのか調べる必要があります。プラグインローダークラスは読み込まれるプラグインのフルパスを引数とする virtual 関数 `Open()` を宣言します。特定のプラグインクラスローダーでの `Open()` 関数の実装では、ベースプラグインローダーの protected 関数 `open()` を呼び出します; このベース `open()` 関数は要求された基本的なプラグインの関数それぞれのアドレスを見つけようとします; 各関数のアドレスが取得されると、いくつかのチェックが実行されます:

プラグイン `GetKicadPluginClass()` が呼び出されると、プラグインローダーが提供するプラグインクラス文字列との比較が行われます; もしこれらの文字列が一致しなければ、開かれたプラグインはこのプラグインローダーインスタンス向けのものではありません。プラグイン `GetClassVersion()` はプラグインが実装しているプラグインクラス API Version を取得するために呼び出されます。プラグインローダーvirtual 関数 `GetLoaderVersion()` はローダーが実装しているプラグインクラス API Version を取得するために呼び出されます。プラグインとローダーが報告するプラグインクラス API Version は同じメジャーバージョン番号を持っている必要があります。もし違っていれば互換性はないと考えられます。これは最も基本的なバージョンのテストで、ベースプラグインローダーによって強制的に実行されます。プラグイン `CheckClassVersion()` はプラグインローダーのプラグインクラス API Version information を呼び出します; もしプラグインが与えられたバージョンをサポートしていれば、成功を意味する `true` が返ります。成功した場合、ローダーは `GetKicadPluginName()` と `GetPluginVersion()` の結果を基に PluginInfo 文字列を作成し、plugin loading procedure がプラグインローダーの `Open()` 実装部の中で続けて実行されます。

3.1.2 API: 3D プラグインクラス

3D プラグインクラスはヘッダファイル `3d_plugin.h` で宣言されており、Plugin Class: PLUGIN_3D 内で記述されるような必要とされるプラグイン関数を拡張します。対応するプラグインローダーは `pluginldr3D.cpp` 内で定義され、ローダーは要求された API 関数に加えて public 関数を提供します。

```
/* フル パス "aFullFileName" で指定されたプラグインを開く */
bool Open( const wxString& aFullFileName );

/* 現在開かれているプラグインを閉じる */
```

```
void Close( void );

/* このプラグイン ローダーが提供する プラグイン クラス API Version を取得する */
void GetLoaderVersion( unsigned char* Major, unsigned char* Minor,
    unsigned char* Revision, unsigned char* Patch ) const;
```

要求された 3D プラグインクラス関数は、以下の関数を経由して公開されます:

```
/* プラグイン クラスを返す。プラグインが読み込まれていなければ NULL */
char const* GetKicadPluginClass( void );

/* プラグインが読み込まれていなければ false を返す */
bool GetClassVersion( unsigned char* Major, unsigned char* Minor,
    unsigned char* Patch, unsigned char* Revision );

/* クラスのバージョン チェックが失敗またはプラグインが読み込まれていなければ、false を返す */
bool CheckClassVersion( unsigned char Major, unsigned char Minor,
    unsigned char Patch, unsigned char Revision );

/* プラグイン名を返す。プラグインが読み込まれていなければ NULL */
const char* GetKicadPluginName( void );

/*
   プラグインが読み込まれていない場合、false を返す。これ以外は、
   引数は GetPluginVersion() の戻り値に含まれる
 */
bool GetVersion( unsigned char* Major, unsigned char* Minor,
    unsigned char* Patch, unsigned char* Revision );

/*
   プラグインが読み込まれていない場合、空文字列を aPluginInfo にセットする。
   これ以外は、下記フォームの文字列が aPluginInfo にセットされる:
   [NAME]:[MAJOR].[MINOR].[PATCH].[REVISION] ここで
   NAME = name provided by GetKicadPluginClass()
   MAJOR, MINOR, PATCH, REVISION = version information from
   GetPluginVersion()
 */
void GetPluginInfo( std::string& aPluginInfo );
```

典型的な状況では、ユーザーは以下のような使い方をするでしょう:

1. KICAD_PLUGIN_LDR_3D のインスタンスを作成する。特定のプラグインを読み込むために Open("/path/to/myplugin.so") を呼ぶ。望み通りのプラグインが読み込まれたかどうか確かめるためには、戻り値をチェックする必要がある。
2. KICAD_PLUGIN_LDR_3D で公開されているような、何れかの 3D プラグインクラスコールを呼ぶ。
3. プラグインを閉じる (リンクを外す) ために Close() を呼ぶ。
4. KICAD_PLUGIN_LDR_3D インスタンスを破棄する。

3.2 シーングラフクラス API

シーングラフクラス API はヘッダファイル ifsg_all.h とそれに含まれるヘッダで定義されています。この API は、ifsg_api.h で定義されている名前空間 S3D にある幾つかのヘルパールーチンと ifsg_*.h ヘッダ各種で定義されているラッパークラスからなっています; ラッパーは、VRML2.0 のスタティックシーングラフと互換があるシーングラフ構造体をまとめたものである、下層のシーングラフクラスをサポートします。ヘッダ、構造体、クラスおよびその public 関数は次の通りです:

sg__version.h

```
/*
   シーングラフ クラスのバージョン情報を定義する。
   シーングラフ クラスで使用される全てのクラスは本ヘッダを含む必要があり、
   また互換性を保証するため S3D::GetLibVersion() によって報告される
   バージョンに対してバージョン情報チェックを行わなければならない。
 */

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

sg__types.h

```
/*
   シーングラフ クラス タイプを定義する; これらのタイプは
   VRML2.0 ノード タイプと密接な関係がある。
 */

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 ヘッダはシーングラフクラスで使われる基本的なデータ型の宣言を含んでいる。

sg__base.h

```
/*
    これは、各色が範囲 [0..1] の数を持つ
    VRML2.0 RGB モデルと同等な RGB
    色モデルである。
 */

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 は点と同じく３つの成分 (x,y,z) を持っています; しかしながら
    ベクトルは保存された値が正規化されていることが保証されており、
    成分の値を直接操作できないようになっています。
 */
```

```
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 クラスは全てのシーングラフノードの基本クラスです。全てのシーングラフオブジェクトはこのクラスの public 関数として実装されますが、いくつかのケースでは、ある関数は特定のクラスでは意味がないかも知れません。

ifsg_node.h

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

    /**
     * Destroy 関数
     * このラッパーで保持されているシーングラフ オブジェクトを削除する
     */
    void Destroy( void );

    /**
     * Attach 関数
     * このラッパーに SGNODE* を関連付ける
     */
    virtual bool Attach( SGNODE* aNode ) = 0;

    /**
     * NewNode 関数
     * このラッパーに関連付ける新しいノードを作成する
     */
    virtual bool NewNode( SGNODE* aParent ) = 0;
    virtual bool NewNode( IFSG_NODE& aParent ) = 0;

    /**
     * GetRawPtr() 関数
     * 元々の内部 SGNODE ポインタを返す
     */
```

```
    SGNODE* GetRawPtr( void );

    /**
     * GetNodeType 関数
     * このノード インスタンスのタイプを返す
     */
    S3D::SGTYPES GetNodeType( void ) const;

    /**
     * GetParent 関数
     * このオブジェクトの親 SGNODE へのポインタを返す。
     * もしオブジェクトが親を持っていない (例. トップ レベル transform) か、
     * ラッパーが現在の SGNODE と関連付けられていない場合は NULL
     */
    SGNODE* GetParent( void ) const;

    /**
     * SetParent 関数
     * このオブジェクトの親 SGNODE をセットする。
     *
     * @param aParent [in] はセットしたい親ノード
     * @return 関数が成功した場合は true; 与えられた
     * ノードが派生オブジェクトへのペアレントを許されて
     * いない場合は false
     */
    bool SetParent( SGNODE* aParent );

    /**
     * GetNodeTypeName 関数
     * ノード タイプを表すテキストを返す。
     * もし何故かノードが無効なタイプであれば NULL
     */
    const char * GetNodeTypeName( S3D::SGTYPES aNodeType ) const;

    /**
     * AddRefNode 関数
     * このノードが所有していない (子ノードではない)
     * 既存ノードへの参照を追加する。
     *
     * @return 成功した場合は true
     */
    bool AddRefNode( SGNODE* aNode );
    bool AddRefNode( IFSG_NODE& aNode );

    /**
     * AddChildNode 関数
     * このノードが所有する子ノードとして追加する。
     *
```

```
     * @return 成功した場合は true
     */
    bool AddChildNode( SGNODE* aNode );
    bool AddChildNode( IFSG_NODE& aNode );
};
```

IFSG_TRANSFORM は VRML2.0 Transform ノードと同等です; これは、子ノード IFSG_SHAPE と IFSG_TRANSFORM および参照ノード IFSG_SHAPE と IFSG_TRANSFORM を大量に含んでいます。有効なシーングラフは、ルートに単一の IFSG_TRANSFORM オブジェクトを持っている必要があります。

ifsg_transform.h

```
/**
 * IFSG_TRANSFORM クラス
 * VRML 互換 TRANSFORM ブロック クラス SCENEGRAPH のラッパー
 */

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 );

    /* いくつかのベース クラス関数はここにありません */
};
```

IFSG_SHAPE は VRML2.0 シェイプノードと同等です; これは、単独の子ノードあるいは参照ノード FACESET を含んでいる必要があります。また、単独の子ノードあるいは参照ノード APPEARANCE を含んでいても構いません。

ifsg_shape.h

```
/**
 * IFSG_SHAPE クラス
 * SGSHAPE クラスのラッパー
 */

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

```
    /* いくつかのベース クラス関数はここにありません */
};
```

IFSG_APPEARANCE は VRML2.0 Appearance ノードと同等です。しかしながら、今のところ Material ノードを含んだ Appearance ノードと同じ意味しか持っていません。

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 );

    /* いくつかのベース クラス関数はここにありません */

    /* 以下の関数は appearance ノードでは意味がありません。
       また、常に失敗を表すコードを返します。

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

IFSG_FACESET は IndexedFaceSet ノードを含んだ VRML2.0 Geometry ノードと同等です。これは、単独の子ノードあるいは参照ノード COORDS、単独の子ノード COORDINDEX、単独の子ノードあるいは参照ノード NORMALS

を含んでいる必要があります。さらに、単独の子ノードあるいは参照ノード COLORS を含んでいても構いません。ユーザーが面に法線を配置できるように、単純化された法線を計算する関数が用意されています。VRML2.0 同等と異なっている部分は次のとおりです:

1. 法線は常に頂点毎である。
2. 色は常に頂点毎である。
3. 座標値の集合は三角形の面だけを記述しなければならない。

ifsg_faceset.h

```
/**
 * IFSG_FACESET クラス
 * SGFACESET クラスのラッパー
 */

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

    bool CalcNormals( SGNODE** aPtr );

    /* いくつかのベース クラス関数はここにありません */
};
```

ifsg_coords.h

```
/**
 * IFSG_COORDS クラス
 * SGCOORDS のラッパー
 */

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 );

    /* いくつかのベース クラス関数はここにありません */
```

```
    /* 以下の関数は coords ノードでは意味がありません。
       また、常に失敗を表すコードを返します。

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

IFSG_COORDINDEX は三角形の面のみを記述しなければならない点を除いて VRML2.0 coordIdx[] set と同等です。これは座標値の合計数が 3 で割り切れることを意味しています。

ifsg_coordindex.h

```
/**
 * IFSG_COORDINDEX クラス
 * SGCOORDINDEX のラッパー
 */

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 );

    /* いくつかのベース クラス関数はここにありません */

    /* 以下の関数は coordindex ノードでは意味がありません。
       また、常に失敗を表すコードを返します。

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

IFSG_NORMALS は VRML2.0 Normals ノードと同等です。

ifsg_normals.h

```
/**
 * IFSG_NORMALS クラス
```

```
 * SGNORMALS クラスのラッパー
 */

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 );

    /* いくつかのベース クラス関数はここにありません */

    /* 以下の関数は normals ノードでは意味がありません。
       また、常に失敗を表すコードを返します。

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

IFSG_COLORS は VRML2.0 colors[] set と同等です。

ifsg_colors.h

```
/**
 * IFSG_COLORS クラス
 * SGCOLORS のラッパー
 */

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 );

    /* いくつかのベース クラス関数はここにありません */
```

```
    /* 以下の関数は normals ノードでは意味がありません。
       また、常に失敗を表すコードを返します。

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

残りの API 関数は、以下のように ifsg_api.h で定義されています:

ifsg_api.h

```
namespace S3D
{
    /**
     * GetLibVersion 関数は、kicad_3dsg library の
     * バージョン情報を取得する
     */
    SGLIB_API void GetLibVersion( unsigned char* Major, unsigned char* Minor,
                                  unsigned char* Patch, unsigned char* Revision );

    // SGNODE ポインタから情報を得るための関数
    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 );

    /**
     * CalcTriNorm 関数
     * 頂点 p1, p2, p3 で表される三角形の法線ベクトルを返す
     */
    SGLIB_API SGVECTOR CalcTriNorm( const SGPOINT& p1, const SGPOINT& p2, const SGPOINT& p3 ↩
        );

    /**
     * WriteCache 関数
     * バイナリ キャッシュ ファイルへ SGNODE ツリーを書き込む
     *
     * @param aFileName 書き込まれるファイル名
     * @param overwrite 既存のファイルへ上書きするためには true をセットしなければならない
     * @param aNode 書き込まれるノード ツリー内にあるノードのいずれか
     * @return 成功した場合は true
     */
    SGLIB_API bool WriteCache( const char* aFileName, bool overwrite, SGNODE* aNode,
        const char* aPluginInfo );
```

```
/**
 * ReadCache 関数
 * バイナリ キャッシュ ファイルを読み込み、SGNODE ツリーを作成する
 *
 * @param aFileName 読み込まれるバイナリ キャッシュ ファイル名
 * @return 失敗した場合は NULL、成功した場合はトップ レベル SCENEGRAPH ノードへのポインタ;
 * 必要なら、このノードを IFSG_TRANSFORM::Attach() 関数経由で
 * IFSG_TRANSFORM ラッパーと関連付けることも可能
 */
SGLIB_API SGNODE* ReadCache( const char* aFileName, void* aPluginMgr,
    bool (*aTagCheck)( const char*, void* ) );

/**
 * WriteVRML 関数
 * VRML2 ファイルへ与えられたノードとそのサブノートを書き込む
 *
 * @param filename 出力ファイル名
 * @param overwrite 既存の VRML ファイルへ上書きするためには true をセットしなければならない
 * @param aTopNode VRML シーンで表される SCENEGRAPH オブジェクトへのポインタ
 * @param reuse VRML DEF/USE 機能を使用する場合には true をセットしなければならない
 * @return 成功した場合は true
 */
SGLIB_API bool WriteVRML( const char* filename, bool overwrite, SGNODE* aTopNode,
                bool reuse, bool renameNodes );

// 注記: 以下の関数は、モジュールの各 SG* 表現のいろいろなインスタンスで用いられる
//  VRML アセンブリの作成とともに使用されます。
// 典型的な使用例:
// 1. グローバル ノード名のインデックスをリセットするために 'ResetNodeIndex()' を呼ぶ
// 2. 'S3DCACHE->Load()' で得られた各モデルのポインタに対して、一度 'RenameNodes()' を呼ぶ;
//    これにより、全てのノードが最終的な出力ファイルで一意的な名前を持つことが保証されます
//    内部的には、RenameNodes() は与えられたノードと全ての子サブノートをリネームするだけです;
//    参照されているだけのノードはリネームされないでしょう。'S3DCACHE->Load()' で得られた
//    ポインタを使うことで、返ってくるノード (トップ ノード) 以外の全てのノードが少なくとも1つのノードの
//    子供であることが保証されます。このため、全てのノードは一意的な名前を与えられることになります
// 3. もし SG* ツリーが S3DCACHE->Load() とは無関係に作られたなら、ユーザーは全てのノードが
//    一意的な名前を持つことを保証できるよう適宜 RenameNodes() を呼ばなければなりません
// 4. コンポーネントの各インスタンスに対して適切な IFSG_TRANSFORM ノードを新たに作成し、
//    アセンブリ構造体を作成します; S3DCACHE->Load() が返すコンポーネント ベース モデルは
//    'AddRefNode()' 経由でこれらの IFSG_TRANSFORM ノードを追加するかも知れません;
//    正確さを保証するために IFSG_TRANSFORM ノードのオフセット、回転角などをセットします
// 5. 最終的にノード名を決定して出力するための準備として、全ての新規 IFSG_TRANSFORM ノードが
//     トップ レベル IFSG_TRANSFORM ノード内で子ノードとして配置されていることを確認します
// 6. トップ レベル アセンブリ ノード上で RenameNodes() を呼ぶ
// 7. 一つの VRML ファイルに全てのアセンブリ構造体を書き込むために、
//    renameNodes = false として、WriteVRML() を呼ぶ
// 8. アセンブリ出力のためにだけ作られた全ての追加 IFSG_TRANSFORM ラッパーとそれらの下層 SG*
```

```
//     クラスを削除して後始末を行う

/**
 * ResetNodeIndex 関数
 * グローバル SG* クラス 値をリセットする
 *
 * @param aNode 有効な SGNODE のいずれか
 */
SGLIB_API void ResetNodeIndex( SGNODE* aNode );

/**
 * RenameNodes 関数
 * 現在のグローバル SG* クラス値を基にノードと
 * 全ての子ノードをリネームする
 *
 * @param aNode トップ レベル ノード
 */
SGLIB_API void RenameNodes( SGNODE* aNode );

/**
 * DestroyNode 関数
 * 与えられた SG* クラス ノードを削除する。この関数は、対応した IFSG*
 * ラッパーに関連付けられたノード以外の SG* ノードを安全に削除すること
 * を可能にします。
 */
SGLIB_API void DestroyNode( SGNODE* aNode );

// 注記: 以下の関数は、レンダリング時のデータ構造の生成と破棄を
// 容易にするためのものです。

/**
 * GetModel 関数
 * aNode (raw data, no transforms) の S3DMODEL 表現を作成する
 *
 * @param aNode S3DMODEL 表現へと変換されるノード
 * @return 成功した場合は aNode の S3DMODEL 表現、それ以外は NULL
 */
SGLIB_API S3DMODEL* GetModel( SCENEGRAPH* aNode );

/**
 * Destroy3DModel 関数
 * S3DMODEL 構造体で使われていたメモリを開放し、構造体へのポインタに
 * NULL をセットする
 */
SGLIB_API void Destroy3DModel( S3DMODEL** aModel );

/**
 * Free3DModel 関数
```

```
     * S3DMODEL 構造体で内部的に使われていたメモリを開放する
     */
    SGLIB_API void Free3DModel( S3DMODEL& aModel );

    /**
     * Free3DMesh 関数
     * SMESH 構造体で内部的に使われていたメモリを開放する
     */
    SGLIB_API void Free3DMesh( SMESH& aMesh );

    /**
     * New3DModel 関数
     * S3DMODEL 構造体を作成して初期化する
     */
    SGLIB_API S3DMODEL* New3DModel( void );

    /**
     * Init3DMaterial 関数
     * SMATERIAL 構造体を初期化する
     */
    SGLIB_API void Init3DMaterial( SMATERIAL& aMat );

    /**
     * Init3DMesh 関数
     * SMESH 構造体を作成して初期化する
     */
    SGLIB_API void Init3DMesh( SMESH& aMesh );
};
```

シーングラフ API の実際の使用例は、Advanced 3D Plugin tutorial の記述と KiCad VRML1, VRML2, X3D パーサーを参照のこと。