OpenSceneGraph 场景节点

一、***OSG*** 场景节点简介及组合模式介绍

OSG 中的场景是树形结构表示的层次结构，如下图所示：

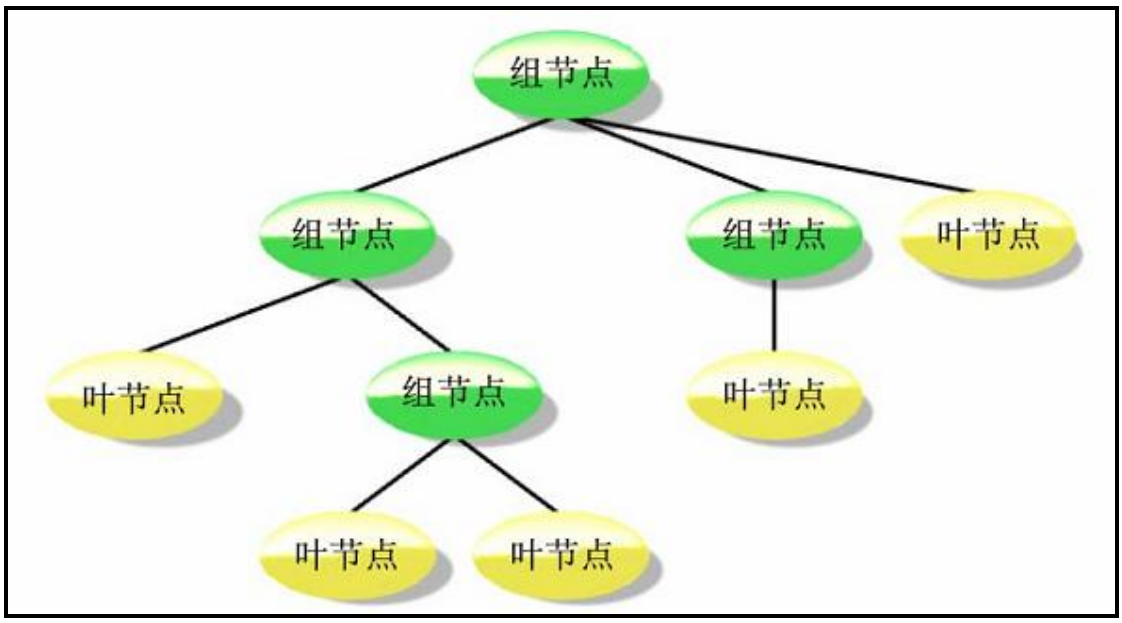

Figure 1.1 OpenSceneGraph 场景树形层次结构

根据其源码中的注释得知，***OSG*** 中场景节点的管理采用了组合（***Composite***）模式。先简要介绍一下组合模式，其类图为：

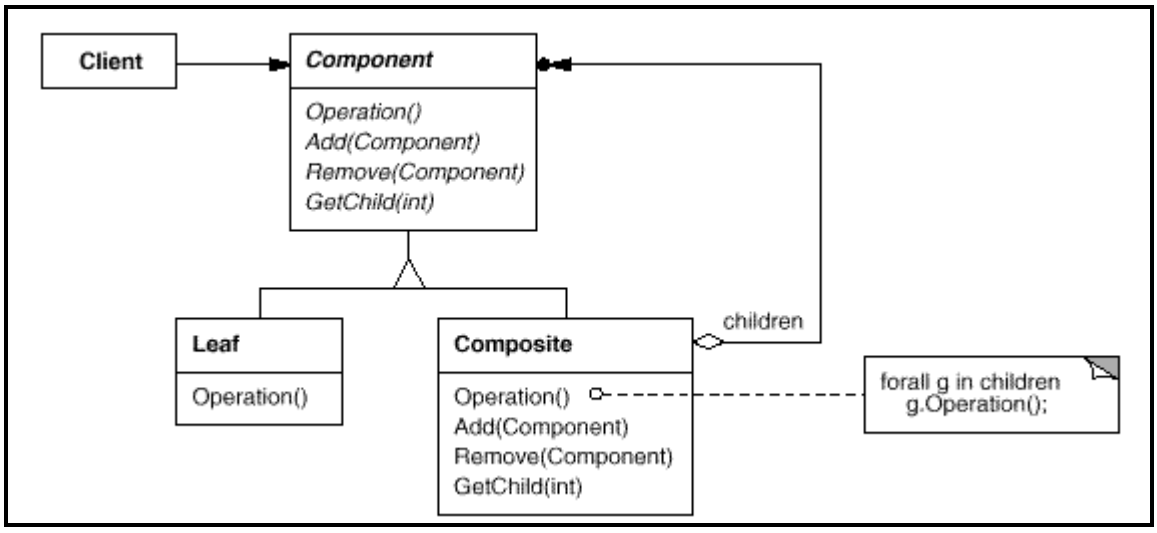

Figure 1.2 Composite Pattern's Structure

使用组合模式的目的是为了将对象合成树形结构以表示“部分—整体”的层次结构。***Composite*** 使得用户对单个对象和组合的使用具有一致性。组合模式通常用于以下情况：

- 你想表示对象的部分—整体层次结构；
- 你希望用户忽略组合对象与单个对象的不同，用户将统一地使用组合结构中的所有对象；

为了达到叶子节点与组合节点的一致性，也可以给叶子节点定义与组合节点一样的操作，但是这些操作什么也不做，与引入 ***Null Object*** 模式类似，这里引入 ***Null Operation***。

组合模式中主要的参与者有三个：

- ***Component***
  - ***Declares the interface for objects in the composition；***
  - ***Implements default behavior for the interface common to all classes, as appropriate;***
  - ***Declares and interface for accessing and managing its child components;***
  - ***(optional)Defines an interface for accessing a component's parent in the recursive structure and implements it if that's appropriate;***
- ***Leaf***
  - ***Represents leaf objects in the composition. A leaf has no children;***
  - ***Defines behavior for primitive objects in the composition;***
- ***Composite***
  - ***Defines behavior for components having children;***
  - ***Stores child components;***
- ***Client***
  - ***Manipulates objects in the composition through the Component interface.***

二、***OSG*** 中组合 ***Composite*** 模式的应用

根据 ***OSG*** 的文档得到其场景节点的类图，如下图所示：

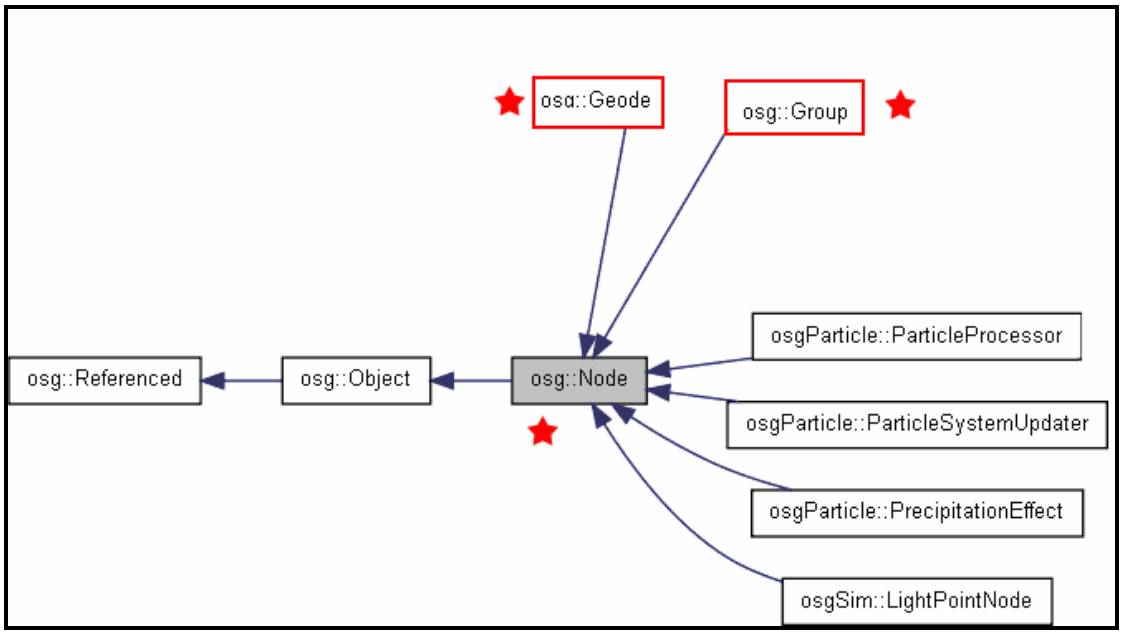

Figure 1.3 Inheritance Diagram for osg::Node

结合红星标示出的三个类：***osg::Node***、***osg::Geode***、***osg::Group*** 来讲述组合模式的具体应用。

以下为声明类 ***osg::Node*** 时给出的注释：

```
/** Base class for all internal nodes in the scene graph.
   Provides interface for most common node operations (Composite Pattern).
*/
```

即 ***osg::Node*** 类为所有场景图形的基类，为大多数能用节点操作提供接口，采用提组合模式。

以下为声明类 ***osg::Geode*** 时给出的注释：

```
/** A \c Geode is a "geometry node", that is, a leaf node on the scene graph
  * that can have "renderable things" attached to it. In OSG, renderable things
  * are represented by objects from the \c Drawable class, so a \c Geode is a
  * \c Node whose purpose is grouping <tt>Drawable</tt>s.
*/
```

即 ***osg::Geode*** 类是一个几何节点，即场景节点中的一个叶子节点，可以把可渲染的东西绑定在它上面。在 OSG 中，可渲染的东西表示为由类 Drawable 生成的对象。所以，一个 Geode 目的就是使 Drawable 成组。具体实现的程序代码为：

```
typedef std::vector< ref_ptr<Drawable> > DrawableList;
```

保护成员变量：

```
DrawableList                    _drawables;
```

以下为声明类 ***osg::Group*** 时给出的注释：

```
/** General group node which maintains a list of children.
  * Children are reference counted. This allows children to be shared
  * with memory management handled automatically via osg::Referenced.
*/
```

即 ***osg::Group*** 节点维护一个孩子表，孩子是引用计数的。这样就可以由内存管理机制来管理这些孩子。具体实现的程序代码为：

```
typedef std::vector< ref_ptr<Node> > NodeList;
```

保护成员变量：

```
NodeList _children;
```

综上所述，得出它们的类图：

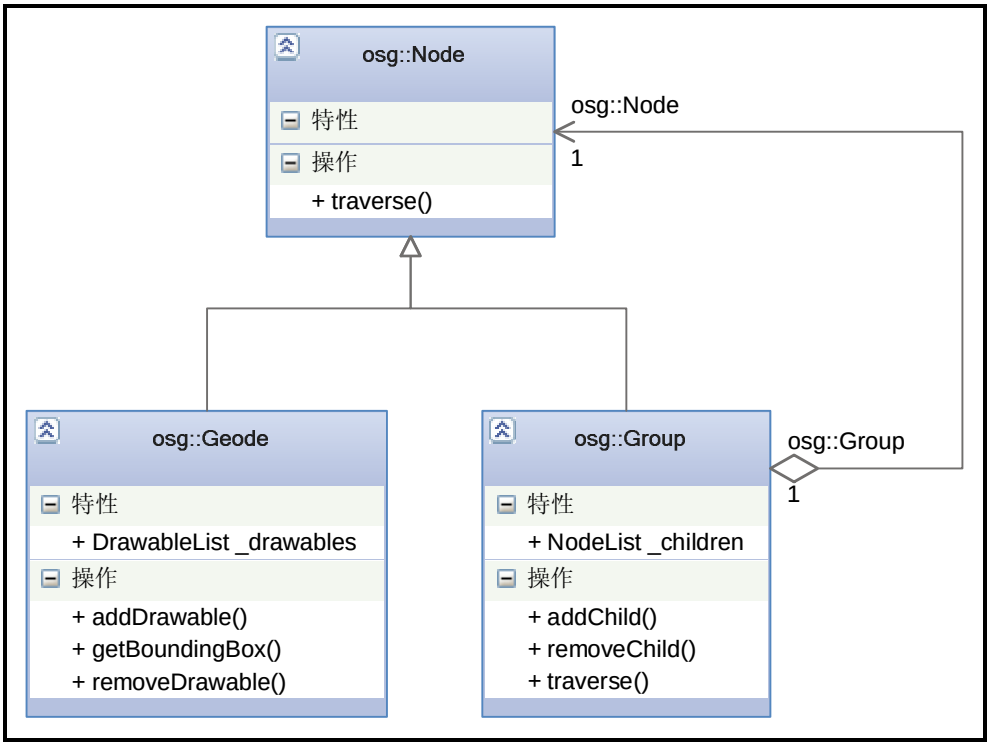

Figure 1.4 OSG Node Class Diagram

由类图可知，这个类图与图 1.2 所示的组合模式的类图相似。其中，类 ***osg::Node*** 可以看成是 ***Component*** 类，为所有的场景节点的通用操作声明接口；***osg::Geode*** 类可看作 ***Leaf*** 类，是一个具体的可渲染的场景节点；***osg::Group*** 类可看作 ***Composite*** 类，它可以包含叶节点或其它节点。

三、程序示例

编程实现由多个模型来构成一个场景，为了简便起见，模型由文件得到。场景的树形层次结构如下图所示：

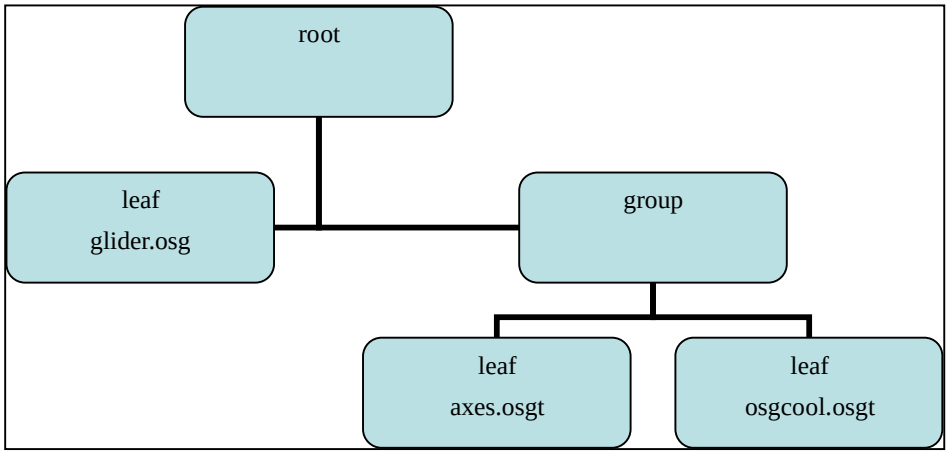

Figure 1.5 Add More Models to Scene Graph

程序代码如下：

```
//-----------------------------------------------------------------------------
//  Copyright (c) 2012 eryar All Rights Reserved.
//
//      File    : Main.cpp
//      Author  : eryar@163.com
//      Date    : 2012-1-3 20:58
//      Version : 1.0v
//
//  Description : Add more models to the Secne.
//
//=============================================================================

#include <osg/Node>
#include <osgDB/ReadFile>
#include <osgViewer/Viewer>
#include <osgViewer/ViewerEventHandlers>

int main(int argc, char* argv[])
{
    osgViewer::Viewer          viewer;
    osg::ref_ptr<osg::Group>    root    = new osg::Group;
    osg::ref_ptr<osg::Group>    group   = new osg::Group;

    root->addChild(osgDB::readNodeFile("glider.osg"));

    group->addChild(osgDB::readNodeFile("osgcool.osgt"));
    group->addChild(osgDB::readNodeFile("axes.osgt"));
```

```
    root->addChild(group);

    viewer.setSceneData(root);
    viewer.realize();

    viewer.addEventHandler(new osgViewer::WindowSizeHandler);
    viewer.addEventHandler(new osgViewer::StatsHandler);

    return viewer.run();
}
```

程序效果图如下图所示：

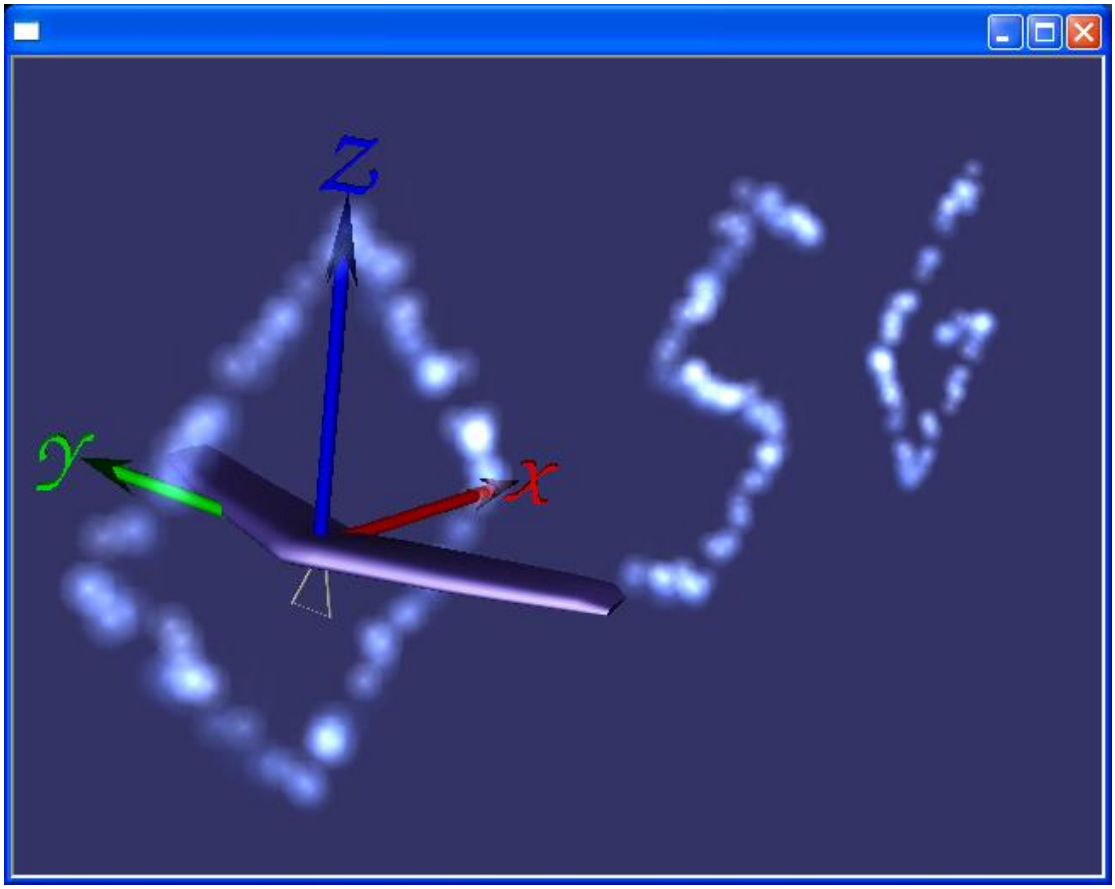

Figure 1.6 Render OSG Node