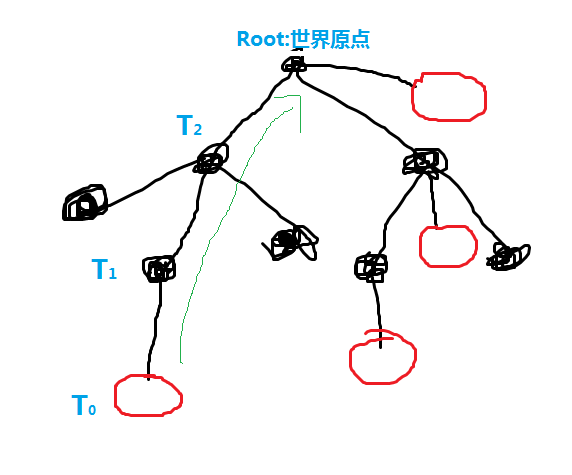
**Noise3D : Scene Graph**

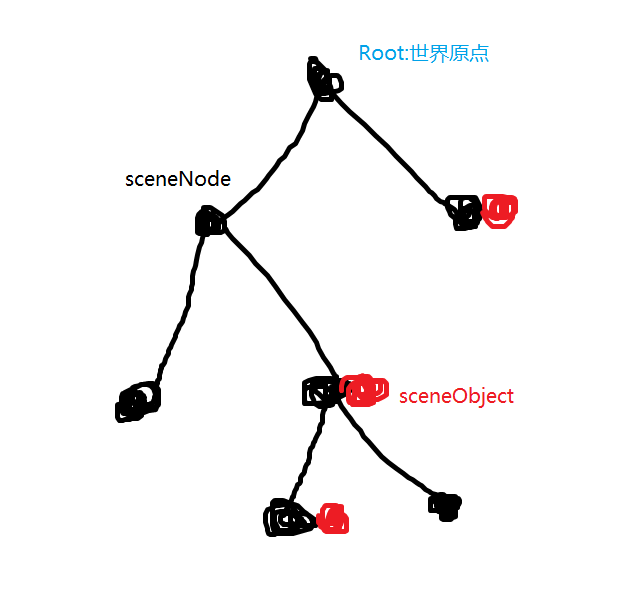
Scene Graph（场景图）是一种场景管理的方案，它一般是一颗树（起码在Noise3D里面是这样）。场景物体/场景对象会attach到scene node上面，于是每个scene node设置的一些属性就可叠加地apply给它的所有子节点（以当前节点为Root的话）。那么scene graph的一个很重要的特性就是hierarchical transformation，也就是子节点会跟着它的父节点们进行“连带式”的变换（沿着从根到叶节点的最短路径的所有transformation都concatenate起来）。

（2019.3.1-deprecated）现在打算即使继承了ISceneObject成为一种场景对象的类，也可以有自己的transform。



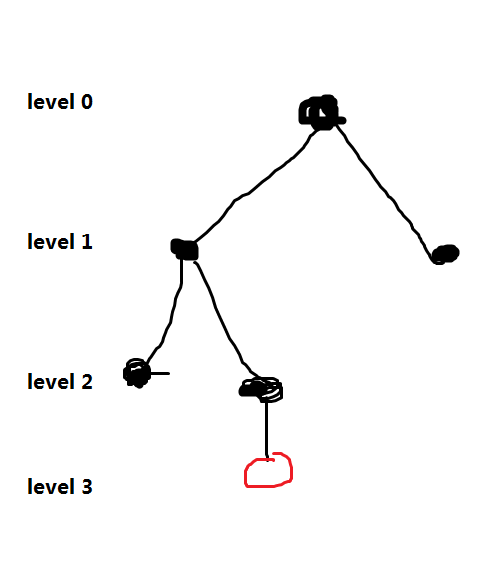
红色物体有自己的Local Transformation ，它这个是在它attach到的那个SceneNode的坐标系下的。于是Scene Node的变换又是相对于它的父节点而言的，同理就是相对于Scene Graph根节点而言的。之所以要这么做是我感觉，Camera、Light、Mesh等的变换的逻辑不太相同，那么他们最好还是要自己维护Transform信息以实现定制化。

（2019.3.2）现在打算ISceneObject继承了SceneNode，也就是说SceneObject已经用继承的方式强行和场景节点绑在一起了。也就是说：



然后为了方便起见，我想着要不现在**场景图就先只是实现hierarchical transform吧，addToRenderList的方式还是按回原来的手动添加**呗。

Hierarchical Transformation：规定使用列向量(column vector)。那么对于某一个挂在叶节点上的场景对象来说，它的每一个顶点相对于世界坐标系（scene graph根节点）来说的变换是：



遍历scene graph是从根节点开始，但是对于挂在叶节点上的场景对象而言，它的hierarchical transformation的apply顺序是从叶节点往上的。所以在遍历scene graph的时候，遍历的第层的某个节点时我们该把

传给它的第层的子节点。那样我们的子节点也很好求出连续变换的结果：

那么不管是用递归还是压栈实现，我们**每一个遍历到的scene node都需要储存好这个叠加变换的中间结果**。（有点动态规划的感觉）

[1] <https://en.wikipedia.org/wiki/Scene_graph>

[2] <https://www.cnblogs.com/java-source/archive/2012/07/07/2604408.html>