第 33 卷 第 1 期 计 算 机 工 程 2007 年 1 月
Vol.33 No.1 Computer Engineering January 2007
·软件技术与数据库· 文章编号：1000—3428(2007)01—0065—03 文献标识码：A 中图分类号：TP311

J2ME M3G 中 RayIntersection 的设计与实现

徐中礼，李冰峰，高传善
(复旦大学计算机科学与工程系，上海 200433)

摘 要：J2ME M3G(Mobile 3D Graphics API) 规范了移动设备上 Java 三维图形程序的 API 和框架，使得符合 M3G 标准的 Java 程序能在不同的设备和平台上运行。由于 M3G 中 RayIntersection 在设计和实现中的复杂性，该文结合在 XORP 上开发 M3G 类库经验的基础，总结并给出了其设计和实现的参考模型和理论基础，并在此基础上重构了具体实现。

关键词：J2ME；M3G；框架；Java；RayIntersection；XORP；重构

Design and Implementation of RayIntersection in J2ME M3G

XU Zhongli, LI Bingfeng, GAO Chuanshan
(Department of Computer Science and Engineering, Fudan University, Shanghai 200433)

【Abstract】 J2ME M3G(Mobile 3D Graphics API) specifies the Java 3D graphics API and framework on the mobile facilities. Applications programmed in M3G can run on different equipments and platforms. Because it is complicated in the design and implementation of RayIntersection in M3G, this paper summarizes the reference model and theory according to the M3G library that it have been developed on XORP. And it restructures the implementation based on them.

【Key words】 J2ME; M3G; Framework; Java; RayIntersection; XORP; Restructure

随着移动设备的普及和性能的不断提高，其运行的应用程序日趋多样化。同时 Java 作为跨平台、面向对象的语言而受到青睐。开放式运行平台 ORP 是一个高性能的可控制运行环境软件，是用来研究垃圾收集和动态编译技术的开放资源研究平台，它支持执行类型安全字节码，能够运行 Java 程序。经过我们的改进，它已可以运行 J2ME 程序。ORL 就是我们在改进的 ORP(也就是 XORP)基础上，遵循了 Clean Room 的原则，严格按照 Sun 公司的 J2ME 规范开发出的 J2ME 类库。目前我们在 ORL 的基础上开发 M3G 类库。

1 M3G 概述

为了使 J2ME 支持三维图形程序，多家公司和组织联合制定了 Mobile 3D Graphics API (M3G)，规范了三维图形程序的 API 和框架。

M3G 共支持 30 个类，其中以 Object3D 为所有类的基类，图 1 中给出了用于描述物体到主要类之间的继承关系。对于继承 Transformable 的类都有自己的变换矩阵，只有 Node 及其子类可被显示出来。Camera 用来设定观察视角和位置；Group 用来构建 Node 场景，可以包含一组 Node 对象，并且可以嵌套；Light 用来设定光照效果；Mesh 用来构造物体，包括物体的顶点信息，颜色，和纹理(Texture2D)信息等；Sprite3D 用来把二维的图片显示在三维空间中。如果要构建一个完整的场景，所有的信息都保存在类 World 中，在本文 3.1 节中有详细的描述。SkinnedMesh 和 MorphingMesh 用来刻画动画的物体，前者通过制定骨骼框架而后者通过起始和终止位置的线性插值来决定某一时刻的物体位置。其它的类还包括背景、雾、材料、坐标变换等辅助类。

这些类构成了完整的 M3G 类库，因此只要符合 M3G API 标准的 Java 程序就能在不同的设备和平台上运行。

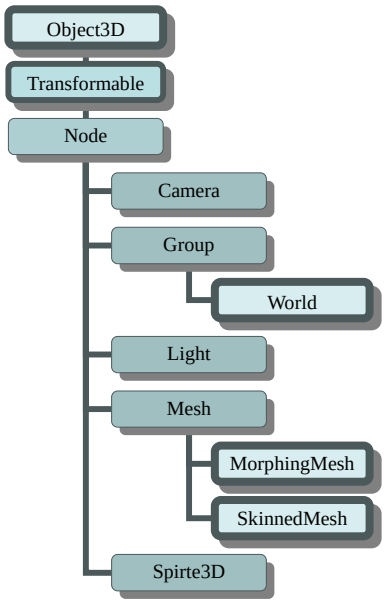

图 1 类的继承关系

2 RayIntersection 概述

试想一下有这样一个场景，在你的面前摆放着很多物体，只想移动视线来选取其中的一个。当视线汇集于某个物体时，此物体将变成与众不同的颜色。换言之，如果把人的视线转

基金项目：Intel 公司基金资助项目“ J2ME Class Libraries with Small Footprint, Low Power and High Performance on XScale Processor ”
作者简介：徐中礼(1982－)，男，硕士生，主研方向：计算机网络和信息工程；李冰峰，硕士生；高传善，教授、博导
收稿日期：2006-01-18 **E-mail：**cgao@fudan.edu.cn

化成一条(射线)光线，起点是眼睛所在位置，方向是人的视线所指的方向，而那些物体则转化为各种对象实例(Node，Group，World，Mesh，SkinnedMesh，MorphingMesh，Sprite3D等)，那么那个被选中而变色的物体就是与这条光线相交的对象，而这就是 RayIntersection 的作用。

3 RayIntersection 的设计与实现

在设计中 RayIntersection 只是用来保存相交对象的信息，例如相交的具体对象，距离，相交平面以及纹理坐标等。而如何拾取相交对象，则在 Group 类中定义了 2 个接口：

pick(int scope,float ox,float oy,float oz, float dx,float dy, float dz, RayIntersection ri)；

pick(int scope,float x,float y, Camera camera, RayIntersection ri)

scope 参数用来指明此 Group 中包含的哪些对象可以被拾取；最后 1 个参数 ri 用来保存拾取之后的结果。在第 1 个 pick 中直接指定了光线(以下简称 Ray)的起点 (ox,oy,oz) 和方向向量 (dx,dy,dz)，而在第 2 个 pick 中指定了一个照相机(Camera)用来提供观察点和此 Camera 的投影平面上的坐标 (x,y)，由此可以转化为第 1 个 pick 中 Ray 的起点和方向，因此在实现时第 2 个 pick 可以先把 Camera 转化为 Ray 的起点和方向，然后就可以调用第 1 个 pick 方法，这样可以优化代码结构，易于调试。唯一的区别是第 1 个 pick 只能对 Mesh 进行拾取，而第 2 个 pick 可以对 Mesh 和 Sprite3D 进行拾取。

3.1 坐标系的变换

由于在建模时各个物体有自己的坐标系，因此如何把它们统一到同一个坐标系中是我们所关心的。假设有一辆汽车，主要部件包括车身、车轮和用于固定车轮的螺丝等。可以用 World 对象来构造 1 个运行中的汽车的场景。World 包括一个 Node 来表示车身和一个 Group 对象来表示其它部分；此 Group 包含 4 个 Group 子对象表示 4 个车轮，而每个 Group 子对象又包含若干个 Node 对象来表示相应的螺丝。其结构类似图 2 所示。

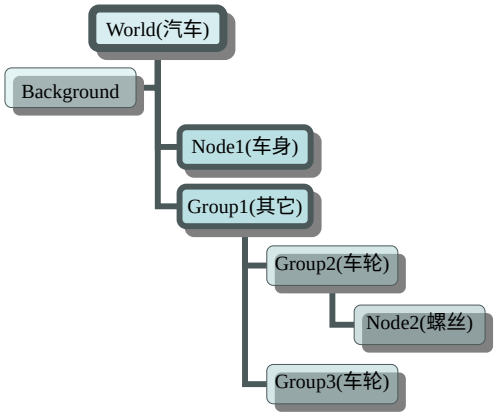

图 2 World 的场景

图 2 是一个 World 的场景，World 可以设置背景，由于 World 本身是一个 Group，可以添加 Node 对象，这样就构成了一个多层的结构，也就是一个完整的场景。但需要注意以下 2 点：(1)一个 Node 只能属于一个 Group，即一个 Node 只能有一个父亲。World 例外，它是根节点；(2)结构图中不能有循环。

当汽车前进时，汽车(即 World)就做平移，轮子(即 Group 子对象)随着汽车平移的同时在做旋转，而螺丝(即 Node)则相对于轮子没有运动。因此要把螺丝的坐标 p 变换到汽车所在坐标系，则只需要作如下变换：

$$p' = M_{World} \bullet M_{Group1} \bullet M_{Group2} \bullet p$$

其中 M_A 表示变换 $Node_A$ 的变换矩阵，而在汽车的场景中因为 Group1 相对World来说是静止的，所以是一个单位矩阵。

因此在结构图中要从 $Node_A$ 变化到 $Node_B$，则只需要作如下变换：

$$P_A' = M_{A\to root} \bullet M_{B\to root}^{-1} \bullet P_A$$

其中 $M_{A\to root}$ 和 $M_{B\to root}$ 分别表示从 $Node_A$ 和 $Node_B$ 到根节点的矩阵连乘。

3.2 RayIntersection 的拾取

空间中 3 点决定一个平面，因此 Mesh 对物体的描述采用的是三角片带的形式。对于某个三角形 ABC，如果点 R_o 是 Ray 的起点，$\overrightarrow{R_d}$ 是 Ray 的方向，点 V 是 Ray 和三角形 ABC 所在平面的交点，则如图 3 所示。

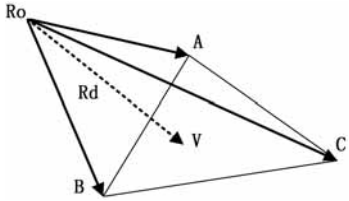

图 3 Ray 和三角形 ABC

如果交点在三角形 ABC 的内部(包括边界处)，则应该满足以下 3 个条件：

(1) $\overrightarrow{R_d} \bullet (\overrightarrow{R_oA} \times \overrightarrow{R_oB})$ 和 $\overrightarrow{R_oC} \bullet (\overrightarrow{R_oA} \times \overrightarrow{R_oB})$ 同时大于等于零或者同时小于等于零，即 $\overrightarrow{R_d}$ 和 $\overrightarrow{R_oC}$ 应在平面 R_oAB 的同一侧。

(2) $\overrightarrow{R_d} \bullet (\overrightarrow{R_oB} \times \overrightarrow{R_oC})$ 和 $\overrightarrow{R_oA} \bullet (\overrightarrow{R_oB} \times \overrightarrow{R_oC})$ 同时大于等于零或者同时小于等于零，即 $\overrightarrow{R_d}$ 和 $\overrightarrow{R_oA}$ 应在平面 R_oBC 的同一侧。

(3) $\overrightarrow{R_d} \bullet (\overrightarrow{R_oC} \times \overrightarrow{R_oA})$ 和 $\overrightarrow{R_oB} \bullet (\overrightarrow{R_oC} \times \overrightarrow{R_oA})$ 同时大于等于零或者同时小于等于零，即 $\overrightarrow{R_d}$ 和 $\overrightarrow{R_oB}$ 应在平面 R_oAC 的同一侧。

对于特殊情况的处理，主要有以下几点：

(1)如果交点 V 在三角形的边界处，那么在上述 3 个条件中已经包含了此情况，即 $\overrightarrow{R_d} \bullet (\overrightarrow{R_oA} \times \overrightarrow{R_oB})$，$\overrightarrow{R_d} \bullet (\overrightarrow{R_oB} \times \overrightarrow{R_oC})$ 和 $\overrightarrow{R_d} \bullet (\overrightarrow{R_oC} \times \overrightarrow{R_oA})$ 为零时。

(2)如果 Ray 的起点 R_o 在平面 ABC 上，此时交点 V 就是 R_o，那么主要就是判断点 V 是否在三角形内：

1) $(\overrightarrow{AB} \times \overrightarrow{AC}) \bullet (\overrightarrow{AB} \times \overrightarrow{AV}) \geq 0$，即点 C 和点 V 在直线 AB 的同侧。

2) $(\overrightarrow{BC} \times \overrightarrow{BA}) \bullet (\overrightarrow{BC} \times \overrightarrow{BV}) \geq 0$，即点 A 和点 V 在直线 BC 的同侧。

3) $(\overrightarrow{CA} \times \overrightarrow{CB}) \bullet (\overrightarrow{CA} \times \overrightarrow{CV}) \geq 0$，即点 B 和点 V 在直线 CA 的同侧。

(3)并不是每个三角片都能被拾取的。当平面法向量的方向 $\overrightarrow{N}$ 与 Ray 的方向向量 $\overrightarrow{R_d}$ 的点积为负，且此平面的正面可显示时，才是可被拾取的。同理，当平面被设置为反面可显示，$\overrightarrow{R_d}$ 和 $\overrightarrow{N}$ 的点积为正时，此平面才是可被拾取的。当然当正反面都可显示时，无须考虑 $\overrightarrow{R_d}$ 和 $\overrightarrow{N}$ 的点积，平面均可被拾取。

3.3 Camera 的转化

为了使两个 pick 之间可以相互利用，需要转化二者的参数。Camera 的作用就好比照相机一样，把处于近平面和远平

面的物体做投影变换，在规格化的坐标系中，近平面处的 Z 轴坐标为－1，而远平面处的坐标为 1。如图 4 所示。

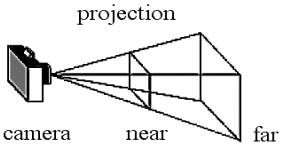

图 4 Camera 透视投影

在 Camera 中，Ray 被演化为起点 $p(x,y,z,w)$ 在 Camera 近平面，方向指向远平面上点 $p'(x',y',z',w')$ 的射线。同时也需要 (x,y) 坐标来确定经过透视投影并规格化之后 p 和 p’的坐标：

$p_{ndc}=(2x-1,1-2y,-1,1)^T$

$p'_{ndc}=(2x-1,1-2y,1,1)^T$

因此有以下等式成立：

$p_c=P^{-1}p_{ndc}$ 和 $p'_c=P^{-1}p'_{ndc}$

其中 P^{-1} 为 Camera 的透视投影矩阵逆矩阵。

接着需要做规格化：

$p=p_c/w$ 和 $p'=p'_c/w'$

因此该 Camera 所对应的 Ray 的起点为 p，方向为 p' – p。

3.4 纹理坐标

三维世界是多姿多彩的，需要为构造出的对象贴上生动的纹理。通过指定每个三角形 3 个顶点的纹理坐标，Texture2D 中相应的区域内的内容就会被复制到此三角形内，如图 5 所示。

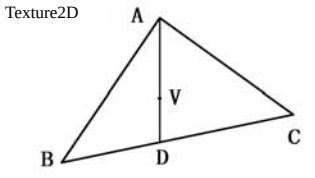

图 5 内容复制

在图 5 中 V 点为 Ray 和三角形的交点，D 点为直线 AV 和线段 BC 的交点，假设已知 $A(X_A,Y_A,Z_A)$，$B(X_B,Y_B,Z_B)$，$C(X_C,Y_C,Z_C)$，$V(X_V,Y_V,Z_V)$ 和它们的纹理坐标 $A'(S_A,T_A),B'(S_B,T_B),C'(S_C,T_C),V'(S_V,T_V)$，通过计算可以得到 $D(X_D,Y_D,Z_D)$，另外直线 VA 和直线 BC 的表示方式分别为：

直线 VA：

$X=X_V+A_0\times T_0$,其中$A_0=X_V-X_A$

$Y=Y_V+B_0\times T_0$,其中$B_0=Y_V-Y_A$

$Z=Z_V+C_0\times T_0$,其中$C_0=Z_V-Z_A$

直线 BC：

$X=X_B+A_1\times T_1$,其中$A_1=X_C-X_B$

$Y=Y_B+B_1\times T_1$,其中$B_1=Y_C-Y_B$

$Z=Z_B+C_1\times T_1$,其中$C_1=Z_C-Z_B$

通过计算，D 点处相对应的 T_1 和 T_0 的值分别为

$T_{1d}=(B_0\times(X_V-X_B)-A_0\times(Y_V-Y_B))/TT$

$T_{0d}=(A_1\times T_{1d}+X_B-X_V)/A_0$

其中$TT=A_1\times B_0-A_0\times B_1$

则 D 点纹理坐标为

$S_D=T_{1d}\times(S_C-S_B)+S_B$

$T_D=T_{1d}\times(T_C-T_B)+T_B$

所以最后交点 V 点的纹理坐标为

$S_V=(S_D-S_A\times T_{0d})/(1-T_{0d})$

$T_V=(T_D-T_A\times T_{0d})/(1-T_{0d})$

4 RayIntersection 重构

由于SkinnedMesh和MorphingMesh都继承了Mesh，并且对于拾取的算法都很类似，唯一差异是对三角形顶点信息的获取途径不同，原先我们处理二者的方法是在pick函数中对它们进行特殊处理，即通过instanceof语句。然而我们开发的是类库，对于性能、可读性、易维护和代码大小都有很高的要求，并且instanceof是一个很昂贵的操作。所以，基于Mesh，SkinnedMesh和MorphingMesh对于拾取的共性，可以运用Java语言面向对象的特性来进行重构[1~3]。

对于三者，具体拾取的过程和算法都是大同小异，因此我们把拾取的算法过程放到 Mesh 基类中，而它们不同的取顶点信息的方式则放在各自对基类 Mesh 抽象方法的具体实现中。具体如图 6 所示。

```
Mesh {
 checkPicked() {
        //implementing here
 }
 Abstract getInfo();
}
SkinnedMesh extends Mesh {
 getInfo() {
        //implementing here
}
}
SkinnedMesh extends Mesh {
 getInfo() {
        //implementing here
}
}
```

图 6 RayIntersection 重构

那么在 pick 方法中就无需区分 SkinnedMesh 和 MorphingMesh，只要把它们都看作为 Mesh，调用 Mesh 的 checkPicked()方法即可，这样代码清晰明了，不仅性能、可读性提高，代码也精简了很多。

5 总结

本文提出了如何设计和实现 J2ME M3G 类库中的 RayIntersection 类。在设计和实现的过程中必须遵守正确性、可扩展性、统一性和简洁性。本文列举的设计和实现方法都是从实际开发中总结出来的，有很好的借鉴作用。

以上的一些设计和实现方法不仅仅适用于 J2ME M3G 类库的开发，它们更具有普遍的意义，也适合其它类库的设计和开发。

参考文献

1 Bloch J. Effective Java Programming Language Guide[Z]. Sun Microsystems, Inc., 2001.

2 Shirazi J. Java Performance Tuning[Z]. O'Reilly & Associates, Inc., 2002.

3 Gosling, James, Joy B, et al. The Java Language Specification[M]. Second Edition. Addison-Wesley, Boston, 2000.