模拟训练视景系统、三维游戏开发工具 RGB Game Builder 的设计

赵 辉 余云宜 梁应宏 贾广威
(哈尔滨工业大学,哈尔滨 150001)

摘 要 介绍一种模拟仿真类实时三维图形应用开发工具 RGB Game Builder 的设计,它的组成和应用结构、实时引擎和使用的主要三维图形处理技术。

关键词 计算机图形学 引擎 模拟仿真 视景系统 三维游戏

0 引 言

目前,各种基于 PC 机平台的3D 加速卡性能越来越高,使用这些廉价的3D 加速卡开发模拟仿真的视景系统、逼真的三维游戏已经成为现实。但是到目前为止,3D 应用开发人员的队伍还非常弱小。其原因在于两个方面:首先,三维图形的处理是一个庞大的、不断发展的专业知识体系,不经过专门的学习和训练很难成为一个合格的开发者;其次,缺乏通用的、廉价的开发工具。

RGB Game Builder 产品的设计宗旨是采用三维图形、计算机软件的前沿技术,将三维图形复杂的知识框架和技术难点隐藏起来,提供一套方便、实用的开发工具,使用户不需要深入了解三维图形和软件开发的复杂技术就可以容易地开发自己的应用。

1 RGB Game Builder 软件组成和应用结构

RGB Game Builder 软件由两部分组成:交互建模工具和引擎。其应用结构如图1所示。

交互建模工具是 Windows 平台下的一个场景创建、浏览和编辑工具。它类似3DS 和 TrueSpace 一类的三维建模工具,所不同的是在景物上引入了控制形状、行为的参数,用户通过编程可以实时控制这些参数,从而实现与三维数据库的交互作用。用户通过交互建模工具产生一个表达场景的三维数据库和参考信息文件。

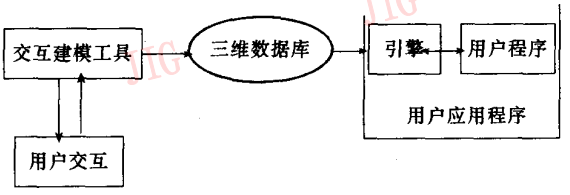


图1 RGB Game Builder 应用结构

引擎是用户开发应用程序的支持工具。引擎的基本功能是三维数据库的实时显示,提供控制三维数据库中的各种参数 API 接口,封装图形、声音的实现平台。此外引擎对复杂的应用,例如:碰撞检测、地形匹配、雷达、导弹追踪、智能目标、景物动态生成,提供了内部支持。

用户应用程序的主体就是引擎。用户程序所要关心的主要是玩法,将一个玩法分解为对三维数据库的控制,并通过引擎提供的各种 API 实现所需要的控制。RGB Game Builder 的这种应用结构为用户应用提供了最大的灵活性和简洁性。

收稿日期:1998-12-11

2 交互建模工具

交互建模工具由元实体创作模块和实体(景物或物体)创作模块组成。元实体代表在场景可直接观察到的几何形状。交互建模工具采用标准的 Windows 界面规范,简洁易用,用户非常容易掌握,同时对焦点、捕捉等进行了有效的智能处理,使用起来得心应手。

2.1 元实体(Atom)建模工具

元实体建模工具主要用于创建点组、面、体等几何对象,使用三维变换操作几何对象,对面和体进行着色。

点组是一组有序三维点集。可以通过折线形、圆弧、Bezier 曲线、Bspline 曲线等工具创建。对点组可以进行顶点的移动、插入、删除操作,打开、封闭、切分点组操作,反转点组的方向、删除共线点及由点组创建面等操作。

面由一个封闭的点组构成的空间多边形描述。面的法向量方向(可见方向)根据点组顺序由右手法则决定。对面可以设置双面可见属性,可以指派一个颜色。对面的操作可以通过对点组的操作实现,此外面可以进行规范化、凸剖分和反转可见方向等操作。

体为一组有序面的集合。可以通过长方体、球面、拉伸面、旋转面、连接面、曲面等工具创建。对一组面可以选择它的任意一个子集构成新的面组(体)进行独立操作。体之间可以进行联结和分离操作。在编辑过程中所有体之间自动产生 BSP 面,可以通过联结操作去掉 BSP 面。对一个体可以对它的面进行可见排序,可以进行人工 BSP 分割将其切分成两个体。可以对体指派颜色。

元实体建模工具还提供了一些集成工具,如由文本直接生成平面和3D 字,DXF 文件读入和自动 BSP 处理。

2.2 实体(Object)编辑工具

实体编辑工具用于对已经制作好的元实体、实体进行组合,产生新的实体。在编辑过程中对被调用的实体进行几何变换,指派运动、材料、纹理、开关等属性,对编辑后的实体预览。

实体编辑界面由两个窗口组成,一个图形窗口用于图形交互操作和浏览,一个组织结构窗口用于编辑和浏览实体的树状组织结构。

在图形编辑窗口中用户可以通过拖放、变换放置实体,设置显示环境和实体的各种显示方式,通过鼠标和键盘在实体中漫游。可以通过拖放方式制作运动路径并通过对话框设置各种属性。在图形编辑窗口还可以进行人工 BSP 操作。

在组织结构窗口用户可以选择实体,插入、删除实体,通过拖放安排实体在树中的位置,设置实体属性等。在组织结构窗口中设置了多种模板,用户可以方便地设置分支、LOD 等结构。

图形编辑窗口和组织结构窗口是紧密关联在一起的,通过二者的结合可以有效地完成实体的各种编辑操作。

3 引擎

引擎(Engine)是三维游戏数据库的执行外壳。通过引擎的 API 接口程序用户可以存取三维游戏数据库的数据,透明地完成对三维游戏数据库的各种操作。

3.1 引擎的组织结构

为了使用户控制程序开发容易进行,引擎被设计成宿主程序,用户所要编写的只是一些事件响应函数,用户的控制程序等待引擎在事件发生时回调(CALL BACK)。采用这种机制,用户在源码级不必关心引擎的实现平台,例如,引擎在 Windows 平台上实现,但用户不需要了解 Windows 编程的专门知识,只要简单地利用引擎提供的 API 编写事件响应函数即可。

如图2所示,引擎内部是一个外部事件处理及三维数据库的操作和显示的循环。引擎启动时,首先调用用户的 InitLevel()函数,将控制权交给用户控制程序。InitLevel()函数相当于用户的主函数,用户控制程序必须实现这个函数。在 InitLevel()函数中,用户通过引擎提供的 API 完成数据库加载、显示环境设定、操作设置、定义各种事件响应的回调函数等。

引擎的主循环体主要是完成数据库的操作和显示处理。在此之前要先进行帧前处理即处理外部事件并允许用户进行帧初始化及对整个进程进行控制。外部事件是指操纵杆、键盘、鼠标中断等的输入。如果外部事件引擎不能处理,则根据用户是否设置了外部事件回调函数,定是否将事件传递给用户控制程序处理。用户可以通过处理外部事件完成自己的交互操作过程。帧初始化主要是用户对显示环境

及目标驱动的控制。进行控制允许用户结束和重新加载新的数据库。在执行三维数据库的操作和显示处理时,当各种操作和传感器(时间、距离、接触等)产生事件时,如果用户对该事件设置了回调函数,则将事件传递给用户控制程序处理。

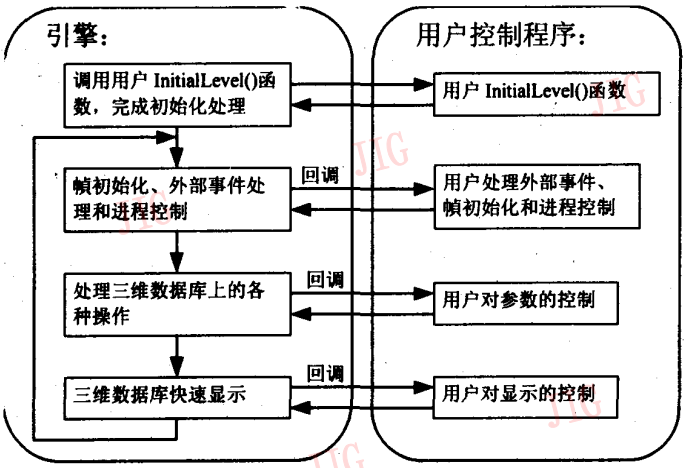


图2　引擎的组织结构

3.2　引擎的 API

引擎提供的 API 可以分为以下几类:数据库的加载和初始化、回调函数设置、显示环境设定、操作设置、数据库参数控制、用户界面显示支持以及交互设备和声音的支持。

三维数据库的加载和初始化是将三维游戏数据库加载到内存,在加载过程中对三维数据库进行扩展,产生各种信息表、便于快速处理的操作和显示的数据集。

引擎 API 接口 SetCallback(name, type, callback, data)用于设置回调函数。其中 name 为对象名,type 为响应的事件类型,callback 是用户程序对该对象相应事件的响应函数,data 为用户处理该对象的事件时附加的数据区指针。当与该对象连接的事件发生时,引擎回调 void callback(HANDLE handle, DWORD& data) 函数,其中 handle 为对象标识,data 为用户数据区指针。用户控制该对象时,将 handle 作为参数调用相应的参数控制 API。对于习惯面向对象编程的用户,引擎也提供了一套以虚函数机制实现的事件响应机制,用户只要具体实现对象事件处理虚函数即可。虚函数机制可以免去设置回调函数过程。

显示环境设定 API 主要提供对视窗、显示层、视点、光源等的控制。用户在 InitLevel()函数中可以设置初始的视窗、显示层、视点、光源,另外在主循环的操作和显示处理前调用 SetViewportCallBack 函数,用户通过编写这个函数实现每帧对显示环境的控制。

操作设置 API 用于定义和使用引擎提供的各种复杂操作,如碰撞检测、地形匹配、雷达、导弹寻地、目标智能等。这些操作由引擎内部特殊处理和结构支持,可以快速执行,同时减少用户程序开发的复杂性。

数据库参数控制 API 是用户存取三维数据库中各种控制参数的接口。用户的事件响应函数一般都是通过组合各种参数控制 API 实现的。

用户界面显示支持 API 提供一组用户向界面上输出图象和文字的接口。交互设备支持 API 便于用户实现界面的交互。

声音支持 API 实现了3D 声音、背景声等,使得声音的实现平台对用户透明。

4　主要技术

本节概要介绍 RGB Game Builder 所涉及和使用的主要的三维图形处理技术。

4.1　BSP 技术

BSP 技术的起源可以追溯到80年代,起初是作

为消隐的方法提出来的,后来由于 z-buffer 消隐算法大行其道,BSP 方法黯然失色,在一般的计算机图形学书里已难寻踪迹。究其根本原因在于应用 BSP 方法存在两大障碍:一个是运动块难于处理,另一个是数据库建模比较复杂。

近年来一些著名的游戏,如 DOOM、Quake,都宣称它们使用了 BSP 方法,这表明 BSP 方法是有其自身优越性的,尤其是在数据库的高层组织上,BSP 方法将数据库在空间上组织成二元树结构,这使得游戏中涉及的许多数据库搜索操作得以高速处理。从硬件的角度来看,理论上讲,BSP 消隐算法的象素填充速度要二倍于 Zbuffer 消隐算法,因为在 BSP 消隐方法中,数据只通过存储器管脚一次,而在 Zbuffer 消隐算法中要两次。尤其是当今图形存储器(如 VRAM、SGRAM)都支持块填充,即每个周期可以填充8或16个象素,使用 BSP 技术因为不需要逐个象素处理消隐问题,可以非常容易地实现块填充。

几年来,我们在研究和使用 BSP 方法中有许多心得体会和重要进展,在可见性处理、运动块处理、多边形面排序、自动产生 BSP 面等方面我们都发展了 BSP 方法,在我们的软件中实现了一些行之有效的工具。可以说我们已经基本上解决了应用 BSP 方法的两大障碍。我们坚信,BSP 技术将逐渐扩大它的应用领地,成为今后计算机三维图形处理的主流技术。

4.1.1 基于 BSP 的可见性处理

4.1.1.1 视锥 BSP 面的检测

传统的 BSP 消隐算法的处理思想是根据视点与 BSP 面的位置关系,决定景物的遍历处理顺序,即先显示与视点异侧区域内的景物,再显示与视点同侧区域的景物。在决定视点与 BSP 面的位置关系时,可以进一步引入视锥信息,当视锥与 BSP 面不相交时,仅需要遍历与视点同侧的区域。这样在 BSP 面处可能将遍历问题转化为搜索问题,提高场景的显示处理速度。

如图3所示,视点 $\boldsymbol{E}$,视方向 $\boldsymbol{V}$,视锥角 α,*BSP* 面的法向量 $\boldsymbol{N}$,视方向和 *BSP* 面的夹角 θ,其中 $\boldsymbol{V}$ 和 $\boldsymbol{N}$ 是单位向量。视轴线方程:$\boldsymbol{P}=\boldsymbol{E}+t\cdot\boldsymbol{V}$,*BSP* 面方程:$\boldsymbol{N}\cdot\boldsymbol{P}+D=0$。视轴线与 *BSP* 面交点 P_C 对应的 $t=-(\boldsymbol{N}\cdot\boldsymbol{E}+D)/(\boldsymbol{N}\cdot\boldsymbol{V})$。判断 *BSP* 面与视锥是否相交:如果 $\boldsymbol{N}\cdot\boldsymbol{V}=0$,视方向平行 *BSP* 平面,如果 $t>0$,视方向朝向 *BSP* 面,这两种情况 *BSP* 面与视锥相交;如果 $t\leqslant 0$,视方向背向 *BSP* 面,要进一步检查 θ 与 α 之间的关系。如果 $\theta\geqslant\alpha$ 视锥与 *BSP* 面不相交,此时若 $t=0$,视点在 *BSP* 面上,由 $\boldsymbol{N}\cdot\boldsymbol{V}$ 的符号判断遍历 *BSP* 面的那一个分支。

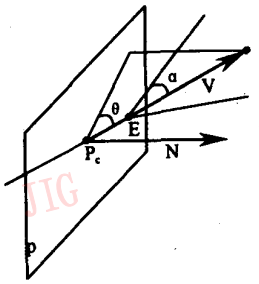


图3 视锥与 BSP 面的关系

4.1.1.2 使用单侧 BSP 面

在场景中一些景物,例如一面墙,当视点在一侧时,看不到另一侧,我们可以设置一个特殊的 BSP 面代表墙的性质,对该 BSP 面只遍历与视点同侧的区域。我们称这种 BSP 面为单侧 BSP 面。使用单侧 BSP 面,可以解决景物之间显示覆盖这类复杂问题,显著提高处理速度。对于室内场景和复杂景物的建模,可以巧妙地使用单侧 BSP 面减少可见面的数量。

4.1.2 运动物体处理

BSP 方法是对静态场景的组织方法,当有运动物体存在时,要将运动物体实时插入到场景的 BSP 树中去,如果运动物体和 BSP 面相交,则问题变得非常棘手,残忍的作法是将运动物体切开[1],这种处理的计算量是巨大的。人们将 BSP 下运动块称为“ghost”,可见这个问题的难度。在 Quake 游戏中,采用 BSP 与 z-buffer 结合的处理方法,先用 BSP 方法显示静态场景,保留屏幕上象素点的 Z 值,然后用 z-buffer 算法显示运动物体。这种处理方法在显示静态场景时虽然省略了 Z 值的比较,但计算每个象素点 Z 值所引入的计算量也要比单纯的 BSP 方法大很多,如果没有硬件支持 Z 值计算,其处理速度可能比运动物体切开方法慢。

BSP 方法是一种以从后向前显示景物的方法,但它并不是从后向前显示景物的唯一排序方法。在有运动物体时,能否修改 BSP 从后向前显示景物的机制,采用另一种从后向前显示景物的机制为运动物体找到正确显示次序呢?我们本着这个思路,成功地给出了一种不用切割的运动物体处理方法。

如图4所示,BSP 面 S1,S2,S3,S4将空间分成 A,B,C,D,E 五个区域,运动物体 M1与 S1相交,视点位于区域 E 中。如果没有 M1,正常的显示次序为

A,B,C,D,E,当 M1加入后,因 M1与有 BSP 面 S1相交,按正常的显示次序将会出现错误。此时正确的显示次序为 A,C,M1,B,D,E。以下我们简介获得正确显示次序的方法,更详细的算法已另文介绍。

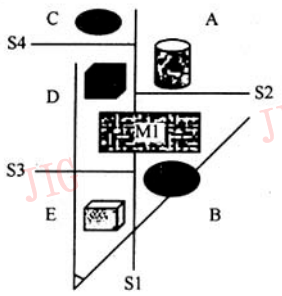


图4　运动块处理

(1)查找与 M1的 BSP 区域,标记与 M1相交的 BSP 面。此例中,M1与 B,D 区域相交,与 BSP 面 S1相交。

(2)遍历 BSP 树,显示景物。先将区域 A 中景物画出,然后遍历到 B,因 B 中有运动物体,且运动物体位于景物之后,先不能画 B 中景物。保留现场,跳转到与运动物体相交的 BSP 面 S1的另一侧,继续遍历显示景物,先碰到区域 C,将 C 中景物画出,然后遍历到区域 D,因 D 中有运动物体,并且 D 是与运动物体相交的最后区域,此时将运动物体画出,因 D 中景物位于 M1之后,先将 D 中景物,接着将 M1画出,将 D 区的后继 E 标记为 S1的另一侧,跳回与 M1相交的的第一个区域 B,将 B 中景物画出,继续遍历。接下来会处理 S1的另一侧,因为此时的后继为区域 E,将 E 中画出后结束了整个场景的处理。

从这个例子给出的处理过程我们可以看出,运动物体处理的实质是以运动物体所在区域和与运动物体相交的 BSP 面为线索,打破 BSP 给出的处理次序,找到一种合理的显示次序。我们隐去了很多细节尤其是数据结构的组织,对运动物体和多个 BSP 面相交及多个运动物体的情况,处理起来更为复杂,在此就不赘述了。

4.1.3 多边形面排序

采用 BSP 方法处理遮挡问题,物体如果存在自身遮挡情况,要使用 BSP 面将物体切开,实际上要保证每个 BSP 子空间的景物都是一个凸体。事实上,有相当多的凹体,只要合理的安排多边形面的显示次序,不使用 BSP 面也不会出现自身遮挡问题。

如图5所示的两个物体,它们都是凹体,但我们只要先显示有阴影部分的面,再显示其它面,视点在任何位置观查时,都不会产生自身遮挡问题。这主要是因为有阴影的面,不遮挡其它的面。对给定的一组面,只要我们能给出面的一种显示次序,使先显示的面不遮挡后显示的面,则不会产生自身遮挡问题。问题的关键在于对给定的两个面,要决定它们是否存在固定的遮挡关系。

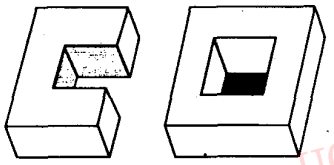


图5　自身遮挡的物体

如图6所示,p,q 的法向为 Np,Nq,面 p 位于面 q 的前向。如果存在一条视线 R,从面 p 的前向穿入,并且可以看到面 q,则面 p 遮挡面 q。可以证明面 p 遮挡面 q 的充要条件是在多边形面 p 上存在多边形的一个顶点 P,在面 q 上存在多边形的一个顶点 Q,通过 P,Q 的视线从面 p 的前向可以看到面 q。根据这个性质,我们可以快速判断两个面的遮挡关系,并根据此遮挡关系对面进行排序。对给定的一组面,如果排序成功,则不用进行 BSP 处理。

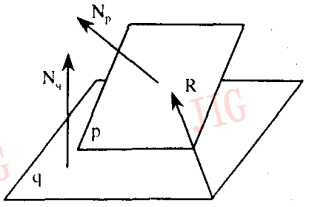


图6　面的遮挡

4.1.4　自动生成 BSP 面

自动生成 BSP 面的处理包括两个方面的问题:一是分隔多个物体,二是分隔物体内的多边形。场景中的所有物体之间都要进行 BSP 处理,使每个 BSP 区域内只包含一个物体。对每个物体,只有当它的多边形面不能排序开时,才进行 BSP 面的处理。

4.1.4.1 物体间 BSP 面的生成

物体间的 BSP 面一般考虑根据物体的包围盒或物体的凸壳生成。生成物体间 BSP 面的规则是不允许 BSP 面穿过物体。物体间 BSP 面的生成对一个子空间递归进行处理,我们使用以下步骤寻找一个

物体间的 BSP 面：

·物体包围盒平面作为 BSP 面

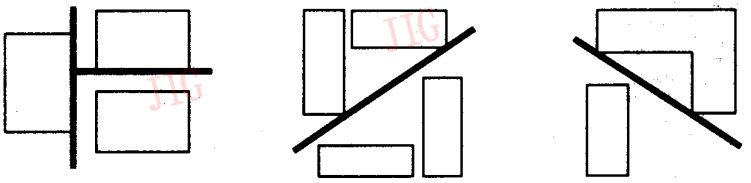
如图7(a)所示，物体可以通过包围盒平面分隔开。

·物体间包围盒的顶点构成的平面作为 BSP 面

如图7(b)所示，所有包围盒平面都不能将物体分隔开，物体间包围盒的顶点构成的平面将物体分隔开。

·物体凸壳上的面作为 BSP 面

如图7(c)所示，所有包围盒平面都不能将物体分隔开，物体间包围盒的顶点构成的平面也不能将物体分隔开，物体凸壳上的面将物体分隔开。



(a) (b) (c)

图7 物体间的 BSP 面

·人工定义 BSP 面

当上述步骤均失败时，提示用户要如果人工定义 BSP 面。我们在交互界面提供了方便的定义 BSP 面工具。

4.1.4.2 物体内 BSP 面的生成

物体内 BSP 面的生成具有相当的复杂性，我们在此仅给出大致的处理步骤，详细算法已另文处理。算法的目标是在寻找 BSP 面的过程中尽量减少切开的多边形的数量。算法处理步骤如下：

·对多边形面进行排序，建立遮挡图，如果遮挡图存在环路，要进行 BSP 切割。

·统计所有最短环路中面的出现次数，选取次数最高的面。

·以该面、该面的边与其它多边形构成的面 BSP 面作为候选 BSP 面，寻找破坏最短环路最多，多边形切开最少的面作为 BSP 面。

在创作物体时，一般很少需要使用物体内 BSP 面的生成功能。在交互界面上提供了自动排序和人工分割的工具，可以方便的产生物体内 BSP 面。在其它数据库读入时，我们采用物体内 BSP 面自动生成方法。

4.2 可能可见集技术

随着3D 加速硬件的进步，三维图形显示过程中的显示覆盖问题现在已成为提高显示速度最大的障碍，目前人们普遍采用可能可见集(PVS)技术来克服显示覆盖问题。可能可见集技术是对视点所在的区域，预先计算出当视点遍历视点区域时，场景中那些面是可见的，那些面因被其它景物遮挡不可见，表示这些信息的集合称为可能可见集。每个视点区域，有其自己的 PVS，显示过程中，根据视点所在的区域的 PVS，决定每个景物及其上的每一个面是否要显示。

可能可见集是与场景层次 BSP 树对应的集合，在 BSP 面检测、包围盒检测及多边形面处设置位信息记录可见性。当 BSP 面分支或包围盒不可见时，则它们作为树的叶结点。

视点区域要求是长方体或一矩形。引擎支持两种 PVS 集合生成算法：以视点区域整体作为整体视点产生 PVS；将视点区域离散成网格，以每个网格点作为视点计算 PVS，把视点区域内所有网格点处的 PVS 并起来作为视点区域的 PVS。第一种方法生成的 PVS 一般大于理论 PVS，但在显示时不会出现遮挡错误；第二种方法生成的 PVS 小于等于理论 PVS，在显示时可能会出现遮挡错误，第二种方法另外比较耗时。生成 PVS 的具体算法比较复杂，我们另文处理，在此不赘述了。

4.3 动态可见性处理

PVS 技术只适用于静态场景，不能处理运动物体。在模拟仿真和三维游戏中，运动物体占有特殊重要的地位，一般要用大量的面将其描述的逼真、细腻。处理运动物体的可见性问题意义更大。运动物体的可见性必须在显示时进行处理，我们称为动态可见性处理。

Beam 树是一种从前向后的显示方法[2]，可以动

态处理可见性问题,但该方法在景物复杂时处理速度太慢,难以实用。我们在 Satyan Coorg 和 Seth Teller 思想[3]基础上,提出了一种动态可见性处理算法,很好地解决了运动物体和复杂景物的可见性处理问题。

算法的基本思想是在场景中选择一些主要遮挡物,定义长方体代表这些遮挡物,称为遮挡盒。选取场景中 BSP 面将遮挡盒组织成一株 BSP 树。在实时显示时,根据视点和遮挡盒的距离选择与视点较近的一些遮挡盒计算遮挡区域;场景按 BSP 树从后向前显示,对标记要作可见性处理的景物,搜索遮挡盒 BSP 树,计算景物的包围盒是否完全位于遮挡区域中,如果搜索成功,不显示该景物。

5　结束语

三维图形工具软件尤其是实时引擎的开发是现代计算机三维图形技术和编程技巧的有机组合,具有相当的复杂性,由于处理速度的制约,很难给出一种支持所有应用的实时引擎。我们在模拟仿真和三维游戏领域里探索了很多年,RGB Game Builder 是我们经验和心血的结晶,它的推出正值3D 加速卡大规模普及的时刻,我们期望它能给三维图形应用开发者们一双有力的翅膀,在虚拟世界里自由翱翔!RGB Game Builder 也正处在不断地发展完善过程中,我们在技术上本着开放的原则,恳切希望同仁们提出宝贵意见!

参考文献

1　Bruce Naylor, John Amanatides, William Thibault. Merging BSP Trees Yields Polyhedral Set Operations. Computer Graphics, 1990, 24(4):115～124.

2　Michael Abrash. 图形程序开发人员指南. 前导工作室译, 1998. 3.

3　Satyan Coorg, Seth Teller. Real-Time Occlusion Culling for Models with Large Occluders. Proc. 1997 ACM Symposium on Interactive 3D Graphics, 83～90.

赵　辉　副教授,1988年毕业于哈尔滨工业大学数学系,获硕士学位。主要研究方向为计算机图形学、虚拟现实软硬技术。

余云宜　高级工程师,1988年研究生毕业于北京航空航天大学自动控制专业,获硕士学位。主要从事应用软件开发和计算机三维图形的研究。

贾广威　1997年毕业于哈尔滨工业大学数学系,获硕士学位。主要从事应用软件开发和计算机三维图形的研究。

梁应宏　1996年毕业于北京邮电大学,获硕士学位,主要从事应用软件开发和计算机三维图形的研究。

The Design of RGB Game Builder
A Tool for Developing Real-time 3D Graphics Application

Zhao Hui, Yu Yunyi, Liang Yinghong, Jia GuangWei

(*Herbin Institute of Technology, Harbin* 150001)

Abstract　In this paper, we summarily introduce the design of RGB Game Builder, a tool for developing real-time 3D graphics application, such as the scene of simulation and training, 3D game, VR game etc. We have explained the framework of application, the structure of 3D database , the real-time engine and the main graphics technique that we have used.

Keywords　Computer graphics, Engine, Simulation, Scene, Game