计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(10):133−138 [doi: 10.15888/j.cnki.csa.006028]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 Unity3D 的多人在线网络游戏设计与开发①

吴晶晶, 戴智超

(泉州师范学院 数学与计算机科学学院, 泉州 362000)
(福建省大数据管理新技术与知识工程重点实验室, 泉州 362000)
(智能计算与信息处理福建省高校重点实验室, 泉州 362000)

摘　要: 本文基于 Unity3D 技术对多人在线角色扮演类游戏进行了设计及实现. 为了增强游戏的效果, 采用了角色控制状态机与交互取值算法、移动策略和人工智能交互等关键技术. 针对如何改善游戏运行时的工作效率问题, 提出了使用协同程序技术实现预先式同步数据读取. 将协同程序技术应用在地图预读取中, 使漫长的地图读取过程可以在运行期的“闲时”进行处理. 效能分析实验证明了该优化方法有效降低了程序时间复杂度, 有效解决了游戏运行时的实效性问题.

关键词: 多人在线网络游戏; Unity 3D; 协同程序技术; 地图预读取; 交互取值算法; 人工智能交互

引用格式: 吴晶晶,戴智超.基于 Unity3D 的多人在线网络游戏设计与开发.计算机系统应用,2017,26(10):133–138. http://www.c-s-a.org.cn/1003-3254/6028.html

Development of Massively Multiplayer Online Role-Playing Games Based on Unity 3D

WU Jing-Jing, DAI Zhi-Chao

(College of Mathematics and Computer Science, Quanzhou Normal University, Quanzhou 362000, China)
(Fujian Provincial Key Laboratory of Data Intensive Computing, Quanzhou 362000, China)
(Key Laboratory of Intelligent Computing and Information Processing, Fujian Province University, Quanzhou 362000, China)

Abstract: A MMORPG (massively multiplayer online role-playing games) is developed based on Unity 3D. The character controller, the mouse-value interactive algorithm, the motion strategy and the AI-interaction are used to enhance the game performance. In order to improve the working efficiency of game, this paper proposes client resource forecast and synchronous reading by using coroutine technology. The strategy that uses coroutine technology in forecast reading of maps can make it possible to have long-time reading running in the spare time of the system. Experimental performance analysis and experimental results both show that the optimization method proposed can effectively reduce the time complexity of the program and enhance the efficiency of game runtime.

Key words: MMORPG; Unity3D; coroutine; forcast map reading; mouse interacting algorithm; AI- interaction

随着虚拟现实技术的发展和第三代互联网技术的逐渐成熟, 用于开发网络游戏的技术很多, 近几年热门的 Unity3D 得到了业界的青睐. Unity3D 是一款跨平台性突出, 兼容性强的游戏引擎. 本文借助 Unity3D 平台开发了一款 MMORPG 游戏系统. MMORPG 游戏即多人在线角色扮演类游戏, 是现在最为流行的经典网络游戏形式, 多名玩家通过网络在虚拟的环境下对人物角色进行操作, 以达到多人在线游戏娱乐.

目前, 国内外许多学者在 Unity3D 平台游戏开发方面做了大量的研究工作. 如伍传敏等人基于 Unity3D

① 基金项目: 福建省科技厅自然科学基金面上项目(2017J01776); 福建省省属高校科研专项项目(JK2015037); 泉州师范学院青年博士预研基金项目(2015QBKJ02); 泉州师范学院博士科研启动项目 (G17003)
收稿时间: 2017-01-23; 采用时间: 2017-03-02

完成了第一人称射击游戏的设计与开发[1]. 张典华等人基于 Unity3D 实现了多平台兼容的三维空战游戏[2]. 刘晋钢等人则研究了 Unity3D 与 Kinect 整合数据技术在体感游戏中的应用价值[3].

本文通过 C++架设服务器以实现多人同时在线, 基于 Unity 引擎进行游戏客户端的设计与开发. 为了改善程序运行时效率, 提出了预先式读取策略, 该策略主要用于解决漫长的地图读取过程的问题. 通过在程序运行的空闲状态中, 预先读取接下来可能到达的地图数据, 从而降低了程序的时间复杂度, 以增强游戏体验中的实效性.

1 游戏设计

1.1 游戏背景

本游戏以中世纪欧洲的骑士文化为主背景. 游戏设计采取了任务制, 通过不断完成主线任务来推进剧情的一步步深入. 游戏中勇者职业一共有四种, 分别有不同的属性和技能. 游戏组队演示如图 1 所示.

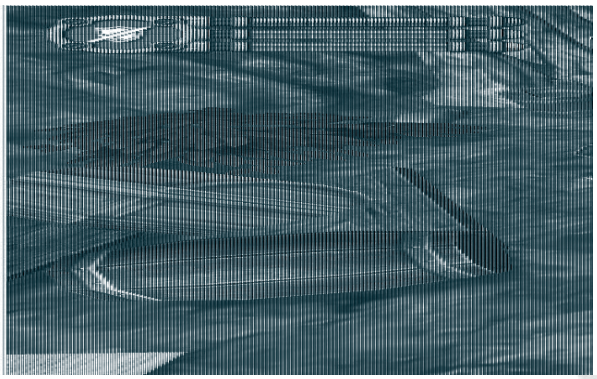

图 1 组队演示

1.2 设计策略

为了增强游戏的效果, 本文提出了以下几点设计策略:

① 游戏的客户端应具较高的实时率, 故客户端更新率设为 60 帧. 为降低游戏网络延迟对游戏影响, 服务器同步率设为 10 帧.

② 提供完整的可视辅助图形系统[4-6], 同时提供可自由选择的角色视角.

③ 设置怪物的属性、怪物的 AI, NPC 的对话内容、任务流程和 NPC 的 AI.

④ 设置音效、角色技能特效、角色死亡特效、传送特效, 传送特效如图 2 所示.

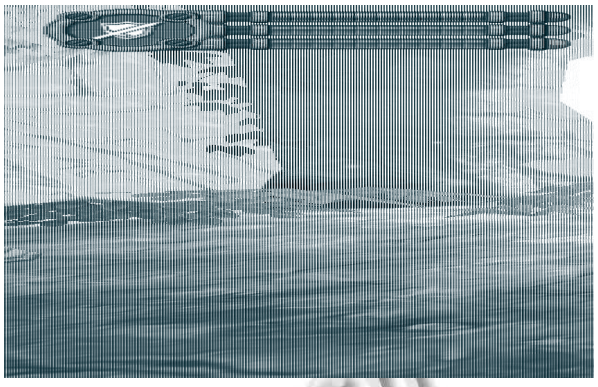

图 2 游戏传送特效

1.3 系统结构

本文提出的游戏系统结构如图 3 所示, 服务器采用了基于自主研发的核心架构, 通过核心连接“数据库”、“账号服务器”、“网络服务”、“副本”四个子程序构成; 客户端则基于 Unity 的脚本系统, 分成“游戏控制”、“美术逻辑”、“功能接口”三大模块.

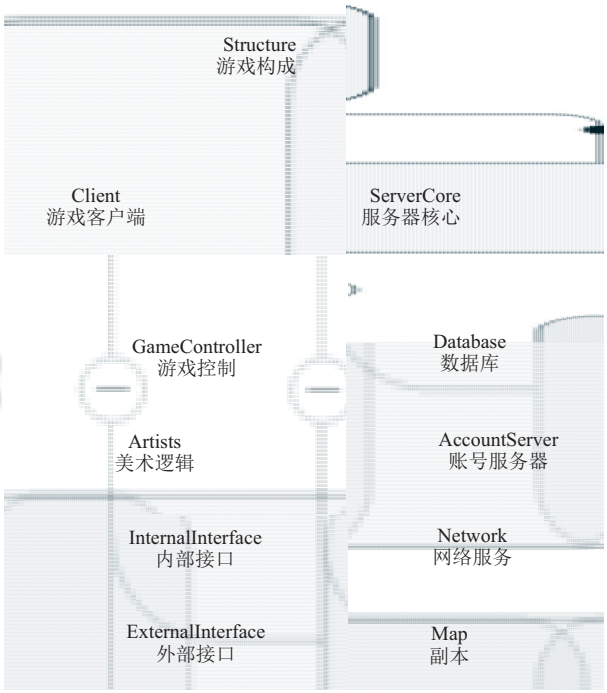

图 3 游戏系统结构

文中的服务器多子程序连接式结构, 提高了各功能模块间的独立性; 并且引入核心架构提高了运行效率和编译耦合度. 客户端使用 Unity 脚本系统, 降低了 3 大模块间的耦合度, 但同时也降低后期维护难度, 这样的设计基于帕累托法则[7]有助于提高软件效率.

2 关键技术与游戏系统实现

2.1 鼠标交互实现游戏控制

对于 MMORPG 游戏而言, 良好的鼠标操作手感是至关重要的. 为了实现鼠标交互代替控制器, 需要的是优秀的角色控制状态机与交互取值算法. 此外, 还需要移动策略的寻路算法[8] 来完成鼠标控制移动. 移动实现的效果图如图 4 所示, 该部分实现通过 Unity 脚本系统编写, 整体实现的伪代码如下:

① 创建角色控制状态机, 鼠标取值 (坐标) 并传给角色控制状态机.

② 状态机判断鼠标获得坐标在三维场景中是否存在.

③ 启动寻路算法, 角色按寻路策略走向该坐标.

④ 将移动结果发送到服务器中, 使每个客户端的移动结果在服务器中同步.

```
if( GetMouseButton( Left ) ){
    Vector3 pos = GetMousePosition( Left ) ;
    if( ThePositionOnTheMap( pos ) )
    {
        while( DistanceTo( pos ) != 0 )
        {
            Vector3 step =
                NextStepInNavigation(pos) ;
            CharacterMove ( step ) ;
            SendStepToServer( step ) ;
        }
    }
}
```

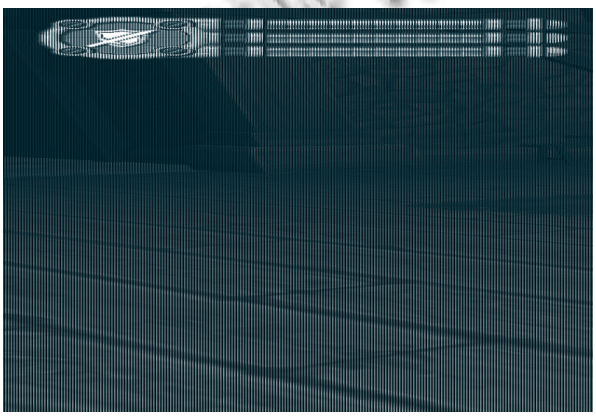

图 4 鼠标交互实现游戏控制

2.2 鼠标交互取值算法

不同于平面上的鼠标取值行为, 当鼠标在三维空间中取值时, 会出现歧义. 因为在平面上对三维空间取值, 返回值将是“一条离散的直线”. 故而需要对直线进行“照准”, 使直线的“击中点”落在场景中.

鼠标取值, 最终是为了控制角色移动. 故而单纯地让点“落在”场景中是不够的. 因为会出现击中点是“唯一点”, 但这个点却是角色无法到达的点, 例如该点是一个路障. 此时就要求系统计算出一个“更优秀的点”来代替该点, 因而本文引入了 Ray 照准技术. 在鼠标的点击时, 通过 Ray“发射一条射线”照准当前鼠标取值. 当取中的点不可取的点时, 则抛弃该点, 遍历选择最近的可取的点作为代替. 鼠标交互算法关键代码如下:

```
void Update()
{
    m_elapsedTime += Time.deltaTime;
    if ( Input.GetMouseButtonDown( 0 ) )
    {
        Ray ray = Camera.main.ScreenPoint-
ToRay(
            Input.mousePosition );
        RaycastHit hit;
        if ( Physics.Raycast( ray, out hit, 100 ) )
        {
            Entity entity = hit.collider.
                GetComponent<Entity>();
            if ( !entity )
            {
                m_target = 0;
                if ( !m_animator.IsInAction() )
                {
                    SendMove( hit.point );
                }
            }
        }
    }
}
```

2.3 基于脚本系统搭载人工智能

对于 RPG 游戏而言, 人工智能是不可或缺的. 一套拥有良好“智能”的 AI 系统将会为玩家提供更好的游戏体验.

本文基于 Unity 的脚本系统, 并采用 MVC 模式[9,10] 设计人工智能. 为了保证人工智能各模块之间的相互独立, 本文基于策略设计模式. 使用组合式脚本提高 AI 状态机、AI 信息、AI 行为三大模块的耦合度. 组合式脚本逻辑思路如下:

① AI 信息获取“组合体”中的 AI 状态机和 AI 行为对应数据库中相应脚本文件, 并将这两个脚本读入为当前 AI 对象的成员. 数据库查询阶段如图 5 所示.

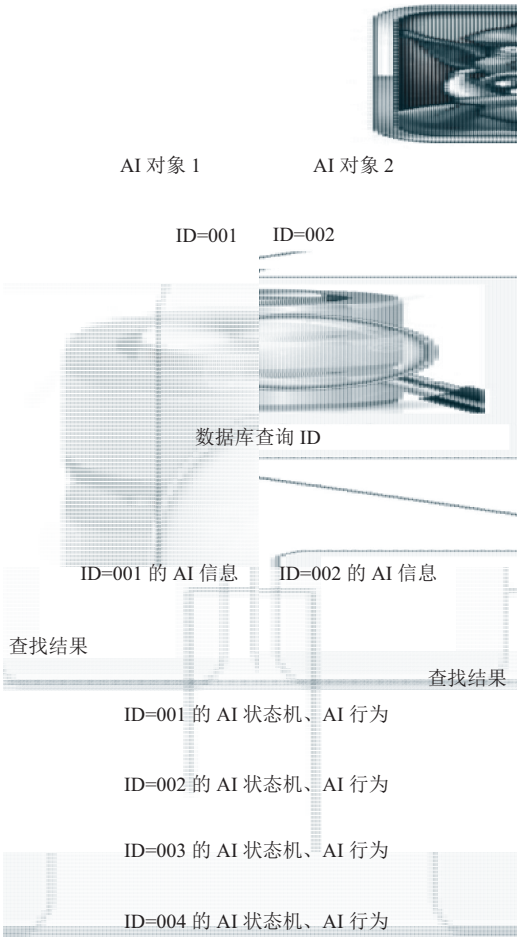

图 5 数据库中查询 AI 信息对应的 AI 状态机、行为

② AI 状态机判断状态变化, 并切换相应的模型动画, 使角色“动起来”, 并将取值结果做为参数调用 AI 行为.

③ 对象基于 AI 行为, 启动相应的行为策略, 例如追击行为和巡回行为有各自的移动策略及相应的寻路算法, 攻击行为有优先级攻击策略. 基于多元的行为策略, AI 做出相应的应对方式. 组合式脚本分析如图 6 所示.

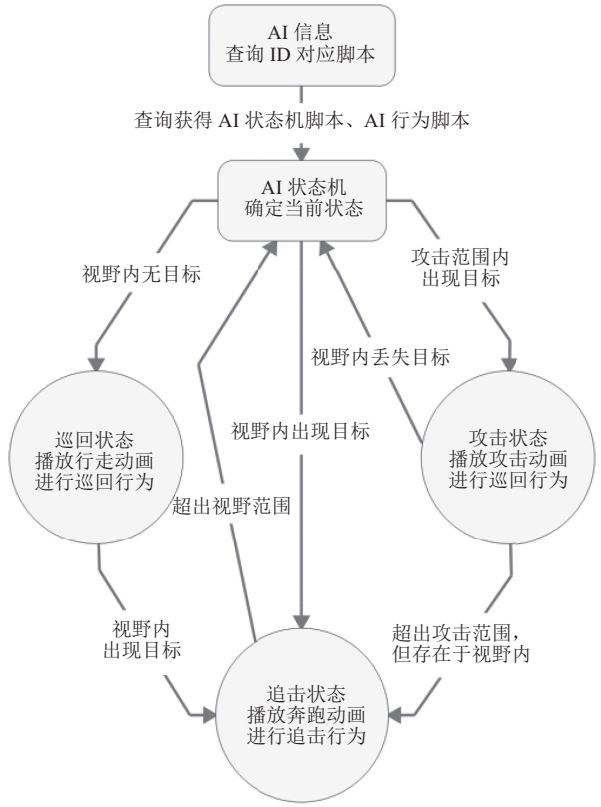

图 6 组合式 AI 脚本系统

3 改善效率的特殊策略——基于协同程序实现地图预读取

3.1 协同程序的设计

对于玩家而言, 漫长的地图读取往往是一件令人厌烦事. 若可以在当前地图中预读取之后可能进入的地图, 将大幅减少等待读取时间, 改善游戏体验. 因而本文提出了使用协同程序实现地图预读取的策略.

Unity 引擎提供了 IEnumerator 接口, 这个接口的特点就在于: 当所有类中 Start 函数运行完后, 每次运行类中的 Update 函数之前, 都会先运行返回类型为 IEnumerator 的函数. 在这个函数中写入与 CPU 线程需求相关的判断句式, 就会形成每一更新帧读取到 Update 函数之前, 都会访问一次 CPU 线程需求的状态. 若前一更新帧的计算结果中产生了 CPU 线程需求时, 在下一更新帧前, 就会进入协同程序. 然后协同程序发送信息给 CPU, CPU 依照这条消息开设新的线程, 运行信息的需求. 本文协同程序设计如图 7 所示.

3.2 协同程序实现预读取的策略特性与效能分析

本文协同程序在设计上具有两个特点:

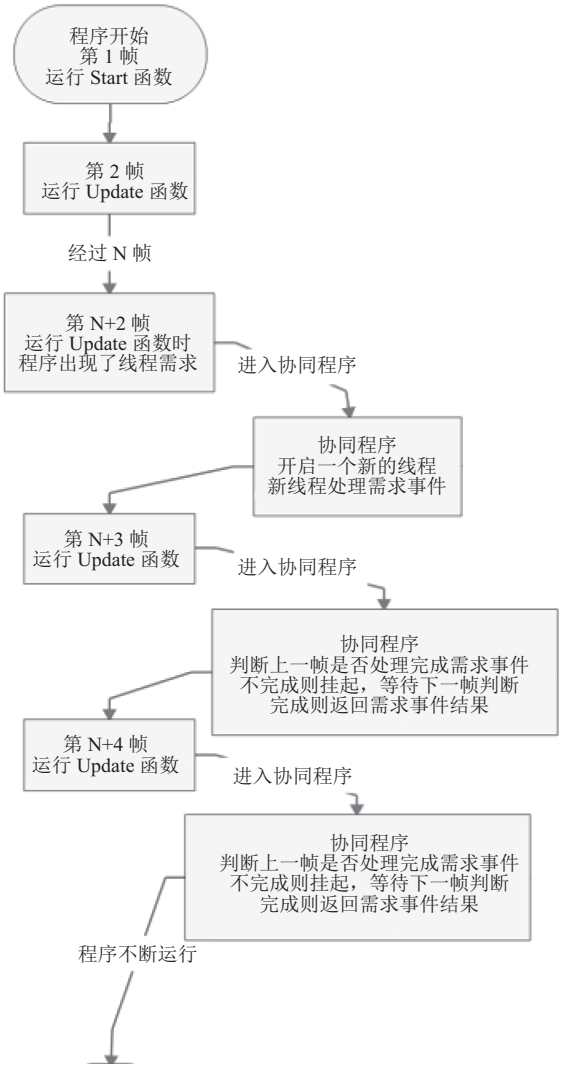

图 7 协同程序设计

① 本文的协同程序接收器是一个生命周期“比较长”的局部变量[11]. 因为该协同程序读取的文件是地图数据, 而高质量模型的地图场景数据量往往是巨大的. 相较于这些地图数据, 每一次构造一个监听器的成本几乎可以忽略不计.

② 本文在协同程序的设计中包含 IEnumerator, 因而可以“挂起”. 创建完监听器后, 如果数据读取完成, 就将数据包返回给 IEnumerator; 如果读取未完成就挂起, 暂停协同程序, 等待下一帧进入协同程序再监听. 因为它始终保持“挂起”, 几乎避免了管理这块内存[12,13]. 由于它始终被占用着, 并且析构时会将 IEnumerator 一起析构, 避免了内存泄露.

从效能角度看, 协同程序增加了系统的空间占用, 但它大幅降低了数据读取的时间复杂度. 如表 1 实践结果所示, 由于协同程序的引入, 数据可在闲时进行预读取, 理想状态下, 可将忙时读取时长缩短至可忽略范围. 即使玩家频繁的切换地图, 依然可以将平均忙时读取时长压缩到 0.8 s 内. 同时从表中数据可知, 协同程序的引入对于程序的空间占用并未造成太大的影响.

表 1 协同程序优化前后的效能分析

		优化前	优化后
忙时读取时长(s)	平均	4.9	0.8
	最小	2.7	0
切换等待时长(s)		2.0	2.0
平均切换时长(s)		6.9	2.8
占用RAM(MB)	最大占用RAM	197	227
	平均占用RAM	176	179
ROM大小(MB)		189	189

4 网络游戏系统的实现

本文采用上述的设计策略和关键技术, 实现了基于 Unity3D 的 MMORPG 游戏的开发. 其中图 8 游戏登录后的新建角色界面. 值得一提的是在创建角色时, 角色框中的“骑士”是可动的. 图 9 个人游戏账号存档界面. 图 10 游戏测试界面, 图中红色方框内是对角色生命槽的动态测试结果. 从图中可以看出游戏画质精度高、同步率良好. 同时本游戏具有 270°视角转换及良好的操作手感, 如动态可视化工具栏, 能够很好满足用户不同的需求.

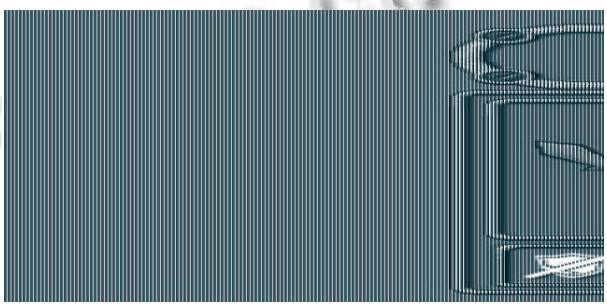

图 8 创建新角色

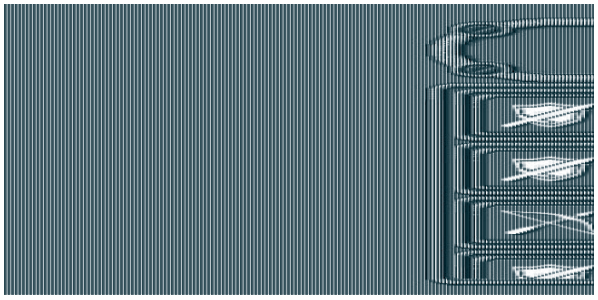

图 9 个人账号存档

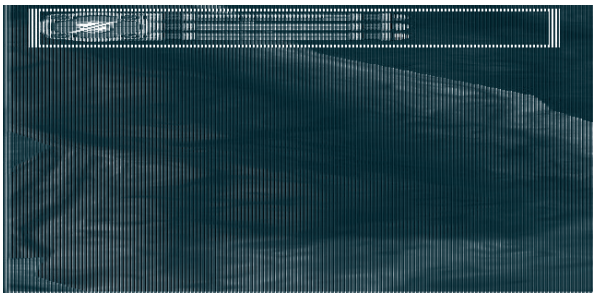

图 10 游戏动态测试

本文设计的游戏内存和内核使用率低, 但对于 GPU 硬件基础要求较高. 基本运行时和最高特效运行时各自的硬件需求如表 2 所示.

表 2 游戏硬件要求

	游戏基本运行	最高特效运行
CPU	单核0.8 GHz以上	单核1.0 GHz以上
RAM	>512 MB	>1 GB
GPU	无须独立显卡	Nvidia GeForce GT630以上 (显存1 GB, 核频810 MHz)

5 结束语

本文基于 Unity3D 进行了 MMORPG 游戏的设计与开发. 在自主研发架设游戏服务器并设计实现了游戏客户端的基础上, 采用了角色控制状态机与交互取值算法、移动策略和人工智能交互等关键技术. 通过“协同程序实现地图预读取” 策略改良了游戏客户端运行时的效率, 效能分析实验证明了该优化方法有效降低了程序时间复杂度. 最后, 通过开发的网络游戏系统验证了所提方法的可行性.

参考文献

1 伍传敏, 张帅, 邱锦明. 基于 Unity3D 的 FPS 游戏设计与开发. 三明学院学报, 2012, 29(2): 35–40.

2 张典华, 陈一民, 李磊. 基于 Unity3 D 的多平台三维空战游戏的开发. 计算机技术与发展, 2014, 24(1): 192–195.

3 刘晋钢, 刘卫斌, 刘晋霞. Kinect 与 Unity3D 数据整合技术在体感游戏中的应用研究. 电脑开发与应用, 2014, 27(11): 7–11, 14. [doi: 10.3969/j.issn.1003-5850.2014.11.003]

4 吕伟伟, 孟维亮, 薛盖超, 等. 基于 GPU 的近似软影实时绘制. 计算机辅助设计与图形学学报, 2009, 21(3): 275–281, 288.

5 方志力, 温维亮, 郭新宇, 等. Kinect 体感控制性能优化方法. 计算机工程与设计, 2014, 35(12): 4350–4355. [doi: 10.3969/j.issn.1000-7024.2014.12.057]

6 罗仕鉴, 龚蓉蓉, 朱上上. 面向用户体验的手持移动设备软件界面设计. 计算机辅助设计与图形学学报, 2010, 22(6): 1033–1041.

7 黄茂生, 杨春晖. 基于帕累托法则的软件测试策略优化. 现代电子技术, 2008, 31(24): 89–92. [doi: 10.3969/j.issn.1004-373X.2008.24.028]

8 梁毅, 周刚. 基于定位点和路径复用的大型多人在线游戏寻路算法. 计算机应用, 2010, 30(12): 3215–3217.

9 李园, 陈世平. MVC 设计模式在 ASP.NET 平台中的应用. 计算机工程与设计, 2009, 30(13): 3180–3184.

10 李春红, 高建华. 使用分层模型改进 MVC 设计架构. 计算机工程与设计, 2007, 28(4): 766–769.

11 伍鸣, 齐骥, 邹琼. 基于长生命周期对象的混合垃圾收集. 小型微型计算机系统, 2008, 29(7): 1190–1195.

12 魏栋, 谭功全, 叶建平. Android 系统的内存管理研究. 单片机与嵌入式系统应用, 2012, 12(4): 9–12.

13 肖康, 刘福岩. 大型 3D 场景漫游系统内存管理. 计算机工程与设计, 2010, 31(10): 2320–2322, 2326.