文章编号:1003—6199(2019)04—0126—07
DOI:10.16339/j.cnki.jsjsyzdh.201904024

多人在线版贪吃蛇3D游戏App设计与开发

沈旭[1†],孟巍[2],彭正超[1]
(1.岭南师范学院 信息工程学院,广东 湛江 524048;2.山东电力集团公司电力科学研究院,山东 济南 250001)

摘 要:为了丰富移动用户的业余生活,开发智力、锻炼思维,设计开发了本游戏App。游戏的设计开发采用OpenGL,它是具有跨平台优势的图像API,在移动端OpenGL ES占有主导地位。游戏的设计开发采用OpenGL ES 3.2进行渲染,在场景管理中使用了可视化剔除与碰撞检测,以及提高游戏性能的场景管理八叉树。最后开发实现了多人在线版3D贪吃蛇游戏App,具有界面美观、图像细腻、音效逼真、运行稳定的特点。

关键词:多人在线;贪吃蛇;3D;OpenGL ES
中图分类号:TP301 文献标识码:A

Design and Development of Multiplayer Online Version of Gluttonous Snake 3D Game App

SHEN Xu[1†],MENG Wei[2],PENG Zheng-chao[1]
(1. Information Engineering School,Lingnan Normal University,Zhanjiang,Guangdong 524048,China
2. Power Science Research Institute of Shandong Electric Power Group Co. LTD,Jinan,Shandong 250001,China)

Abstract:In order to enrich the amateur life of mobile users,develop intelligence and train thinking,the game App is designed and developed. The game is designed and developed with OpenGL,which is a cross-platform advantage of the image API,OpenGL ES in the mobile side of the dominant position. The design and development of the game uses OpenGL ES 3.2 for rendering. Visual culling and collision detection are used in scene management,and octree of scene management is used to improve the performance of the game. Finally,a multiplayer online version of 3D Snake game App is developed,which has the characteristics of fine image,beautiful interface,realistic sound effect and stable operation.

Key words:multiplayer online;gluttonous snake;3D;openGL ES

在黑白屏手机时代,贪吃蛇游戏广受欢迎,随着时代的发展,贪吃蛇游戏也不断改变,这得益于硬件与软件的不断发展。贪吃蛇从黑白屏时代的粗糙形象华丽升级,最大的改变在外观上,蛇身拥有多种皮肤,游戏具有多种特效,在玩法上贪吃蛇也从单机模式升级为多人同时在线模式。游戏玩法简单,只需不断吃掉食物使蛇身不断变长,抢夺对方资源,击败其他用户。

收稿日期:2018—10—23
基金项目:国家自然科学基金资助项目(61402399);广东省哲学社会科学"十三五"规划资助项目(GD17XGL33);湛江市财政资金科技专项资助项目(2014A01010);岭南师范学院教育教学改革资助项目(LSJGMS1811)
作者简介:沈旭(1979—),男,山东单县人,硕士,讲师,研究方向:数据挖掘,移动应用。
† 通讯联系人,E-mail:mr.shenxu@qq.com

1 开发环境

1.1 开发工具

系统开发使用 Visual Studio 2017 编写核心代码，VS2017 具有智能提示功能，不足是对 C++语法过于宽松，能在 VS 上编译通过，可能存在在其他 C++编译器编译不过的问题。Android 程序使用 Android Studio 进行开发，Android Studio 采用 Java 语言，也可以采用 C++语言编写代码。同时，这两个 IDE 都支持 Git 进行代码管理。为了管理需要，采用 SVN 管理核心代码，并在 VS 和 AS 上同步核心代码。

1.2 OpenGL ES 与 EGL

OpenGL 是跨平台的图像接口，提供各种 API 标准，厂商负责实现 OpenGL 的标准。过去的 OpenGL 为了向后兼容，导致 API 非常膨大，不便使用。Khronos 接手后，把冗余的 API 都去掉，剩下核心部分。同时，为了匹配移动端芯片，推出了 OpenGL ES。因为具有跨平台的优势，在移动端，OpenGL ES 占据主要市场地位 [4]。EGL 是 Khronos 渲染 APIs（例如 OpenGL ES、OpenVG）和本地窗口系统之间的接口。使用 EGL 获取 Surface，然后可以用 OpenGL ES 在 Surface 上进行渲染。因此，EGL 主要作用是在本地窗口与 OpenGL ES 之间充当联系纽带。

2 系统需求分析

2.1 系统流程

2.1.1 开发流程

首先进行系统需求分析，抽象出系统需要的功能。接着进行框架结构分析，主要分为游戏场景管理、碰撞检测、OpenGL ES 渲染、网络模块、设计游戏框架，在游戏框架基础上实现 3D 贪吃蛇。最后进行游戏测试。开发流程如图 1 所示。

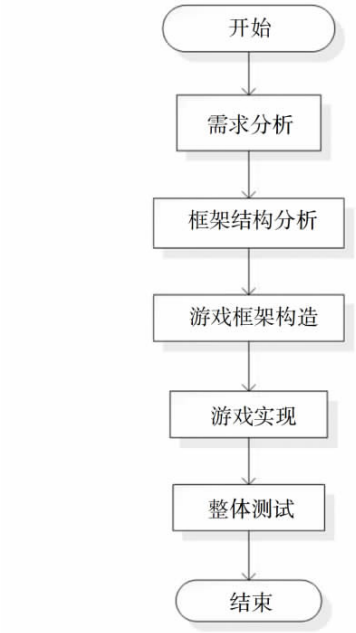

图 1　开发流程图

2.1.2 游戏流程

首先，场景相对于相机椎体进行可见性判断，剔除不可见的 Entities，这可以降低每一次运行的时间长度，有效提高游戏渲染速度。对于可见的 Entities，进行渲染，然后显示在屏幕上。对于场景每一节点的 Entities，在本节点向下进行碰撞检测，发生碰撞的 Entities 回调给场景进行管理。流程如图 2.

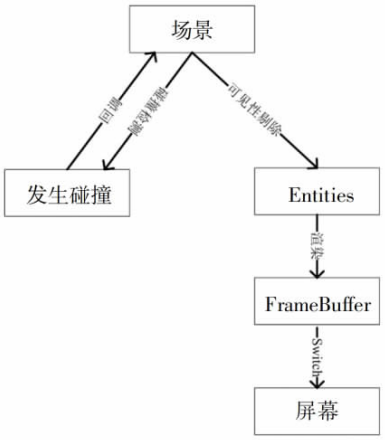

图 2　游戏流程

3 系统实现

3.1 包围体生成

包围体，对实体渲染和碰撞检测都是非常重要的，为每个实体构造一个包围体；可见性判断可以从无规则的多面体转换到有规则包围体，常见的包围体有轴对称包围盒，方向包围盒，球体等[1]。有了包围体就可以快速定位可见物体，加速渲染。

一般会把场景空间进行切割，分割成多个子空间，例如八叉树，每一块最小的空间对应树的一个叶子节点，一个父节点是所有其子节点的空间之和。通过包围盒，就可以把物体放到八叉树的节点上，碰撞检测时，只需要对物体对应的节点和节点的子节点进行搜索即可。

3.1.1 轴对称包围盒

通过遍历物体的每个坐标,找出每个轴上最大值和最小值,便可以构造出一个轴对称包围盒。

轴对称包围盒有多种表示方法:常见的是“最小值最大值储存法”:

```
mutable vector3<_Ty> mMinimum;
mutable vector3<_Ty> mMaximum;
```

还有,“中心点与长宽高”表示法:

```
mutable vector3<_Ty> mCenter;
mutable vector3<_Ty> mLens;
```

游戏系统采用第一种方法。下面给出生成轴对称包围体的伪代码:

```
void
setExtents(std::vector<vector3<_Ty
>> const& points){
    if(points.size() == 0)
        return;
    float32 r = points [0].x,l = points
    [0].x,t = points[0].y,b = points[0].
    y,n = points[0].z,f = points[0].z;
      for (int i = 0;i <
    points.size();i++) {
        auto &p = points[i];
        if (p.x > r)
          r = p.x;
        if (p.x < l)
          l = p.x;
        if (p.y > t)
          t = p.y;
        if (p.y < b)
          b = p.y;
        if (p.z > n)
          n = p.z;
        if (p.z < f)
          f = p.z;
    }
    mMaximum.x=r;
    mMinimum.x=l;
    mMaximum.y=t;
    mMinimum.y=b;
    mMaximum.z=n;
    mMinimum.z=f;
  this->setExtent(EXTENT_FINITE);
}
```

由于轴对称包围盒简单易用,常用于实体可见性判断。

3.1.2 方向包围盒

首先介绍 PCA 主成分分析方法[2]。PCA 可以提取样本的主要成分,抛弃次要成分达到降维效果。我们的目的不是降维,而是提取主分量。通过分析主分量,把样本点投影到新的空间,这个空间的每个轴相互正交,主轴对应样本的主方向。

在 3 维空间中,通过 PCA 获取所有样本点的 3 个相互正交的方向轴,我们就可以构造出一个方向包围盒了,下面介绍 PCA 算法:

首先,计算由 $p_1,p_2,\cdots p_n$,点构成的点集合的平均数 m,其中

$$p_i=(x_i,y_i,z_i) \qquad m=\frac{1}{n}\sum_{i=1}^{n}p_i$$

接着构造一个 3X3 的协方差矩阵 C

$$C=\frac{1}{n}\sum_{i=1}^{n}(p_i-m)(p_i-m)^T$$

协方差表示两个变量的总体误差值。如果两个变量都大于或者都小于自身期望值,则协方差为正值。如果一个大于自身期望值,一个小于自身期望值,则协方差为负数。协方差有个特性,如果两个变量是相互独立的则其协方差值为 0. 而协方差矩阵表示两轴之间的协方差值,如果我们希望轴之间相互独立,则必须对协方差矩阵对角化。对角化后的每一轴与另外两轴可以是相互独立关系。这个对角化可以理解为把点集映射到另外的空间。

假设用矩阵 A 把点集变换到新的空间,新空间的点集协方差矩阵为对角化的,均匀分布在坐标轴上。设变换后的协方差矩阵为 C'.

$$C'=\frac{1}{n}\sum_{i=1}^{n}(Ap_i-Am)(Ap_i-Am)^T=$$

$$\frac{1}{n}\sum_{i=1}^{n}A(p_i-m)(A(p_i-m))^T=$$

$$\frac{1}{n}\sum_{i=1}^{n}A(p_i-m)(p_i-m)^TA^T=ACA^T$$

对角化定义如下,如果一个方块矩阵 A 相似于对角矩阵,也就是说,如果存在一个可逆矩阵 P 使得 P^(-1) AP 是对角矩阵,则它就被称为可对角化的。所以,A 是正交矩阵。并且,非奇异矩阵的特征向量是正交的。因此可以把求对角化矩阵转化为求矩阵特征向量。本游戏系统采用雅克比法求矩阵特征向量。求方向包围盒的伪代码如下:

```
void
setExtents(std::vector<vector3<_Ty
>> const& points){
        float32 rst_min [3] = {1E20,1E20,
        1E20},rst_max [3] = {1E -20,1E -
        20,1E-20};
        auto eigen =
        pzc::pca_points(points);
        rst[0] = eigen[0];
        rst[1] = eigen[1];
        rst[2] = eigen[2];
        for (int i = 0;i < points.size();i++) {
        for (int j = 0;j < 3;j++) {
            float32 dot_value = rst[J].dot(points
            [i]);
            if (rst_min[J] > dot_value)
            rst_min[J] = dot_value;
            if (rst_max[J] < dot_value) {
            rst_max[J] = dot_value;
                        }
                    }
                }
            float32 a,b,c;
            a = (rst_min[0] + rst_max[0]) / 2;
            b = (rst_min[1] + rst_max[1]) /2;
            c = (rst_min[2] + rst_max[2]) / 2;
            center = rst[0] * a + rst[1] * b + rst[2]
            * c;
            rst_len_half [0] = (rst_max [0] -
            rst_min[0]) / 2;
            rst_len_half [1] = (rst_max [1] -
            rst_min[1]) / 2;
            rst_len_half [2] = (rst_max [2] -
            rst_min[2]) / 2;
            this ->setExtent (EXTENT_FI-
            NITE);
            }
```

其中 pca_points 是求矩阵特征向量与特征值。所以，我们获取了 3 个相互正交的轴 R、S 和 T,然后每个点分别于 3 个轴点乘,得到每个轴上点积最大最小的点，分别是 Prmin,Prmax,Psmin,Psmax,Ptmin,Ptmax。Prmin 与 Prmax 的中点与 R 轴构成一个平面 Pr,Psmin,Psmax 的中点与 S 轴构成一个平面 Ps,Ptmin,Ptmax 的中点与 S 轴构成一个平面 Pt。通过求平面 Pr,Ps,Pt 的交点,就可以得到方向报文和中心点。

3.1.3 球体

简单构建一个球体,并不是构造最优球体。通过 PCA 提取点集主轴方向 R,所有点与 R 点乘,获得最小值点 P_1 和最大值点 P_2，则中心点 Q 和半径 R 为

$$Q = \frac{p_1 + p_2}{2}$$

$$R = \| p_1 - Q \|$$

另外,还可能要对 Q 和 R 进行校正,因为不是所有点都落在这个球体中。对于 P_i, $\| P_i - Q \|$,的话,P_i 落在球体外,校正 Q 和 R 为 Q'和 R'。

$$Q' = \frac{G + p_i}{2}$$

$$R' = \| p_i - Q' \|$$

$$\text{其中 } G = Q - R\frac{p_i - Q'}{\| p_i - Q' \|}$$

3.1.4 圆柱体

圆柱体可以通过两个圆的点和半径来表示,则可以转化为求圆的问题。第一步是把所有点集 p_1, $p_2,\cdots,p_n$ 投影到一个圆上,得到 $H_1,H_2,\cdots,H_n$,

$$H_i = p_i - R(p_i \cdot R)$$

向量 S 是特征值第二大的特征向量，假设 H_1 与 S 点积最小,H_2 与 S 点积最大,则圆的圆心 Q 和 R 为

$$Q = \frac{H_1 + H_2}{2}$$

$$R = \| H_1 - Q \|$$

对于 $H_1,H_2,\cdots,H_n$,如果存在 $\| H_i - Q \| > R$,如果有精度要求,则必须进行校正。校正步骤和球体校正差不多。假设校正的圆心为 Q',半径为 R'两个圆的圆心 Q_1 和 Q_2 分别为

$$Q_1 = Q' + \min\{p_i \cdot R\}R$$

$$Q_2 = Q' + \max\{p_i \cdot R\}R$$

3.2 包含测试

包含测试可以用做可见性测试或者碰撞检测。平面可以用 4 维向量表示。假设一个平面 $Ax + By + Cz + D = 0$,则 A,B,C 构成的向量 N 为平面的法

向量。平面上的一点有 $Q\cdot N=-D$。所以我们可以用 $L=<N,D>$来表示平面，其中 N 为平面的法向量。

假设有一点 P 表示为$<x,y,z,1>$，则 $P\cdot L=d$ 表示 P 点到平面的距离，如果 $d=0$，则点在平面上。

3.2.1 平面与包围盒

假设有一个方向包围盒，其三个主轴分量分别为 R,S,T，假设 R,S,T 的长度分别对应包围盒的各边的长度。某一平面 $L=<N,D>$，则包围盒中心 Q 到平面的距离，即有效半径 $Ref=(|R\cdot N|+|S\cdot N|+|T\cdot N|)/2$。设 $d=Q\cdot L$，如果 d 大于零 Ref，这包围盒位于平面正面，如果 d 小于 $-Ref$，则包围盒位于平面负面。否则，包围盒与平面相交。下面介绍求包围盒与平面相交时间。

假设包围盒速度为 $V=(a,b,c)$，起点为 Q，经过时间 t 其位置为

$$Q(t)=Q+tV$$

假设包围盒位于平面正面，经过 t_1 时间包围盒与平面刚好相交，则

$$(Q+t_1V)\cdot L-Ref=0$$

所以

$$(Q+t_1V)\cdot L-Ref=0$$

如果 $V\cdot L=0$，说明包围盒运行轨迹与平面平行，不会相交。

3.2.2 平面与球体

球体可以用于和任意复杂的多面体进行碰撞检测，计算量少。所以，很多可见性判断可以用球体做测试。下面主要介绍球体与平面的相交测试。已知球体球心为 Q，半径为 R，某一平面 $L=<N,D>$，则判断球心与平面的距离可以知道是否相交。

对于一些高速运动的球体，有可能发生这种情况。在 a 时刻检测球体位于平面正面，b 时刻球体位于平面负面。虽然在 a,b 时刻球体没有与平面相交，但确实在 a 与 b 时刻之间发生碰撞。如果仅仅只检测某一时刻是否发生相交，则会发生子弹穿过墙的情况。因此，还必须对球体经过的路径进行检测。可以将路径看作一条线段，计算线段与平面相交情况。

3.2.3 球体之间

球体间距离 $\|Q_1-Q_2\|^2>(R_1+R_2)^2$，则没有发生碰撞。对于高速运动球体间，可以简化为求线段间最短距离问题。对于两条非相交直线 $P_1(t)=Q_1+tV_1$ 和 $P_2(t)=Q_2+tV_2$，假设直线 P_1 在 t_1，直线 P_2 在 t_2 使 $\|P_1(t_1)-P_2(t_2)\|$ 最小，则最短距离为 $\|P_1(t_1)-P_2(t_2)\|$，分别对 t_1 和 t_2 求偏导数为 0，可以求得 t_1 和 t_2

$$\begin{bmatrix}t_1\\t_2\end{bmatrix}=\frac{1}{(V_1\cdot V_2)-V_1^2V_2^2}\begin{bmatrix}-V_2^2 & V_1\cdot V_2\\ -V_1\cdot V_2 & V_1^2\end{bmatrix}\begin{bmatrix}(Q_2-Q_1)\cdot V_1\\(Q_2-Q_1)\cdot V_2\end{bmatrix}$$

如果 $(V_1\cdot V_2)^2-V_1^2V_2^2=0$ 则两直线平行。接着判断 P_1 和 P_2 是否都位于线段内。

情况 1，都位于线段内：如果 $\|P_1(t_1)-P_2(t_2)\|>R_1+R_2$ 则没有发生碰撞。情况 2，都位于线段外，两线段分别取一端点，使其离 P_1 和 P_2 最近，求两端点距离与 R_1+R_2 比较，可得出是否相交（这种情况只对两射线夹角小于 90 度成立）。其他情况也是使用类似方法比较。

3.2.4 圆柱之间

圆柱之间相交测试计算量很大，一般游戏不采用这种方式。但圆柱体对于蛇身拟合度高，有更高的真实感。

首先求出圆柱 A 的圆心 Q_1 离圆柱 Q_2 较近，圆柱 B 的圆心 Q_2 离圆柱 A 较近。相交测试步骤如下[3]：

（1）求 Q_1 所在圆是否与 Q_2 所在圆相交，相交则跳过以下步骤；

（2）求 Q_1 所在圆的平面与圆柱 B 是否相交，不相交则两圆柱不相交，跳过以下步骤；

（3）求 Q_2 所在圆的平面与圆柱 A 是否相交，不相交则两圆柱不相交，跳过以下步骤；

（4）求一平面法向量 N，法向量满足以下条件：1.通过法向量 N，Q_1 以最快速度接近圆柱 B。2.法向量 N 在 Q_1 所在圆的平面上。以法向量 N 方向 Q_1 移动圆的半径距离，得到点 P，以 P 和法向量 N 构成平面 L，如果圆柱 B 到平面 L 的距离大于 0 则不相交，否则相交。

3.2.5 圆柱与球体

圆柱体中心为 Q_1，球体中心为 Q_2，球体半径为 r，向量 $n=\frac{Q_1-Q_2}{\|Q_1-Q_2\|}$，设点 $P=Q_2+n*r$，以 n 和 p 构成以平面，求圆柱到平面的距离，大于 0 则不相交，否则，相交。

3.3 场景管理八叉树

八叉树是将场景空间分成 8 个子空间，然后在

子空间继续进行分割，直到节点物体个数为 1 或者达到某一深度时停止。利用八叉树，可以快速剔除不在相机椎体内或者不与相机椎体相交的物体。提高渲染效率，大幅提高游戏性能。相对于碰撞检测来说，只需要遍历物体所在节点和其所有子节点上的物体进行碰撞测试，所有可以大幅降低运算量。

3.4 网络库

3.4.1 Epoll 模型

Epoll 模型是 linux 网络编程非常重要的一个模型。如果 tcp 连接数量过大，如果使用 select 模型效率就变得低下[4]。epoll 是基于内存映射的，直接把数据复制到用户空间，省去了内核到用户空间数据 copy 的时间。向 epoll 注册某一文件描述符和对应事件，一旦这个文件描述符有对应事件发生，内核会快速激活这个文件描述符，当调用 epoll_wait()时，会得到相应通知。而 select 模型要自己轮询查询，并且文件描述符数量有限制，而 epoll 则不存在 select 模型的这个限制[5-7]。

3.4.2 Iocp 模型

Iocp 模型比 Epoll 模型更先进，采用异步机制。例如向完成端口注册读事件，当有事件发生时，内核会把数据复制到你提供的 buffer 里面，然后通知系统，不需要 recv 数据，只是通知系统数据已经就绪，可以使用。因此，Iocp 是一个更先进的模型[8]。

3.5 C++11 线程与条件变量

C++11 提供了多线程 std::thread，可以用更少代码实现所需功能，而且不用考虑平台问题。有了多线程，线程间需要通信，就必须进行同步，同步问题可以通过 C++11 提供的条件变量解决(std::condition_variable)。使用 C++11 的线程和条件变量更加有利于代码维护[9]。

3.6 游戏运行效果

首先进入引导界面，如图 3：

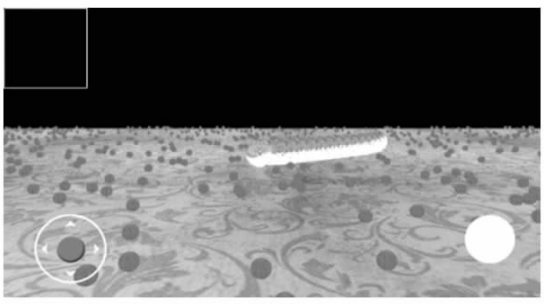

图 3 游戏引导界面

游戏运行效果图(3D 视角)如图 4：

图 4 游戏运行效果图

4 系统测试

利用宏可以实现打印 Log 的功能，不仅可以输出到屏幕，还可以保存到文件：

```
#ifdef ANDROID
#include <android/log.h>
#include <jni.h>
#define log_print(tag,format,...)
__android_log_print(ANDROID_L
OG_INFO,    tag,format,
##__VA_ARGS__)
#else
#define log_print(tag,format,...)
  do{
  printf("%s:",tag);
  printf(format,
##__VA_ARGS__);
  putchar('\n');
  } while (0)
#endif
```

这段打印 Log 的宏可以在 windows 上运行，也可以在 linux 运行，还可以在 Android 上运行[10]。再结合 C++不定参数模板，可以实现具有更多功能的 Log：

```
template<typename...Args>
static void log_ex(char const *tag,
const char *const format,
        Args const &...args) noexcept{
  std::lock_guard<std::mutex>
lock(mtx);
  if (flag&SCREEN)
```

```
    log_print(tag,format,
Argument(args)...);
  if (flag&FILE) {
    auto fp
 =fopen((std::string(ProgramName)+
".log").c_str(),"ab+");
    if(fp! = nullptr) {
      time_t timer;
      struct tm *tblock;
      timer = time(NULL);
      tblock =
localtime(&timer);
      fprintf(fp,"%s:",
asctime(tblock));
      fprintf(fp,format,
Argument(args)...);
      fputc('\n',fp);
      fclose(fp);
    }
  }
}
```

5 结 论

3D 贪吃蛇游戏 App 的设计开发采用 OpenGL,它是具有跨平台优势的图像 API,在移动端 OpenGL ES 占有主导地位。OpenGL ES 3.2 所支持的 GLSL 语言采用 in、out 格式,方便使用。游戏的设计开发使用 OpenGL ES 3.2 进行渲染,在场景管理中使用了可视化剔除与碰撞检测,以及提高游戏性能的场景管理八叉树。最后实现了安卓环境下多人在线版 3D 贪吃蛇游戏 App,可供多人多终端同步游戏,系统具有界面美观、图像细腻、音效逼真、运行稳定的特点。

参考文献

[1] LENGYEL E. 3D 游戏与计算机图形学中的数学方法（第三版)[M]. 北京:清华大学出版社,2016:1—101.

[2] EBERLY D H. 3D 游戏引擎设计(实时计算机图形学的应用方法)[M]. 北京:人民邮电出版社,2009:100—201.

[3] 埃里克森. 实时碰撞检测算法技术 [M]. 北京:清华大学出版社,2010:70—119.

[4] SHREINER D. OpenGL 编程指南 [M]. 北京: 机械工业出版社,2010:20—80.

[5] STANLEY B,LIPPMAN B E,Josée L M. C++ Primer [M].北京:人民邮电出版社,2006:54—230.

[6] ECKEL B. Thinking in C++[M]. 美国:Prentice Hall,2000:10—100.

[7] 洛夫. Linux 系统编程 [M]. 南京:东南大学出版社,2009:90—100.

[8] 任泰明. TCP/IP 网络编程 [M]. 北京:人民邮电出版社,2009:130—140.

[9] NUDELMAN G. Android 应用 UI 设计模式 [M]. 北京:人民邮电出版社,2013:111—119.

[10] 熊力. Windows 用户态高效排错[M]. 北京:电子工业出版社,2007:112—119.