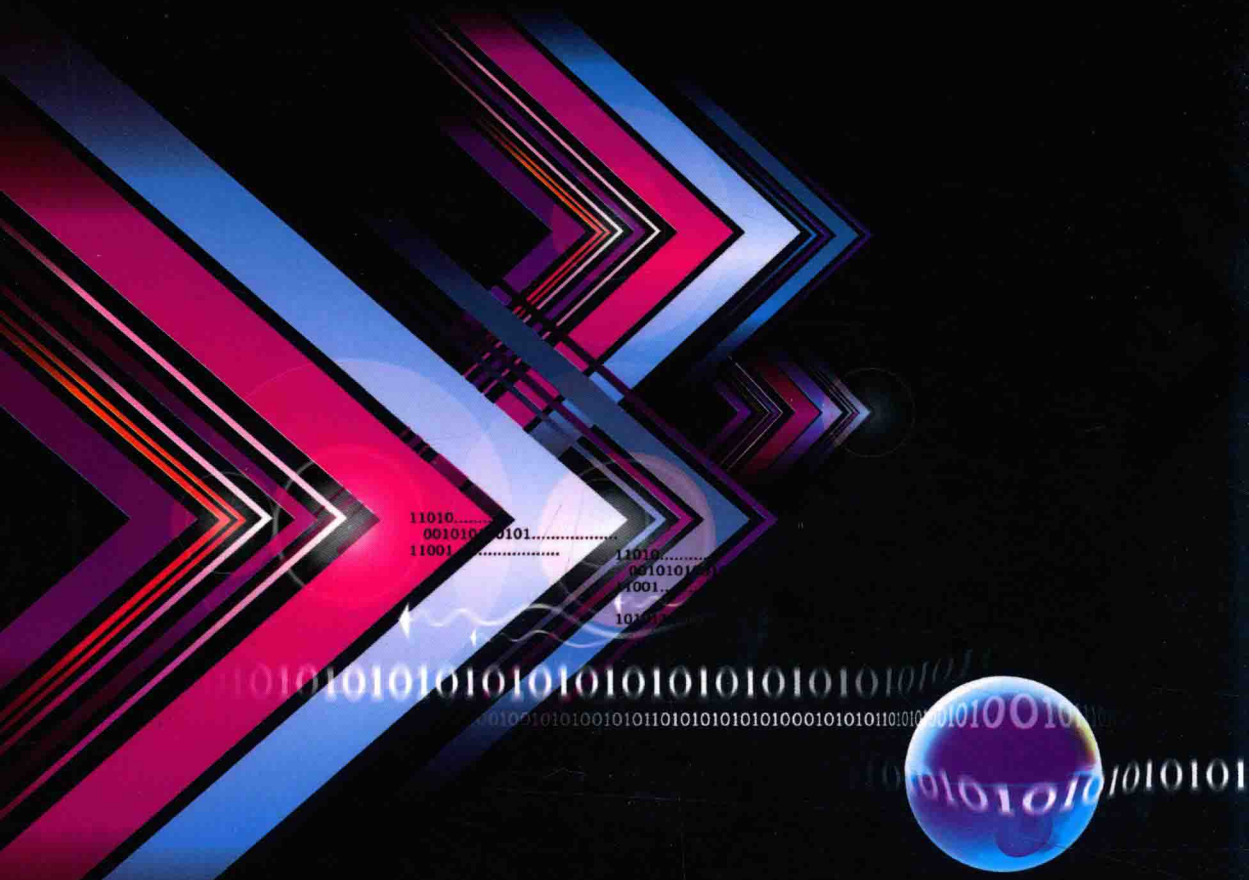
中国电子教育学会高教分会推荐

普通高等教育数字媒体专业“十三五”课改规划教材



DirectX 三维游戏编程

宋伟 著

西安电子科技大学出版社
http://www.xduph.com

中国电子教育学会高教分会推荐

普通高等教育数字媒体专业“十三五”课改规划教材

DirectX 三维游戏编程

宋 伟 著

西安电子科技大学出版社

内 容 简 介

本书严谨翔实地阐述了 DirectX 的工作机制，详尽地探讨了三维游戏的开发过程，并用图示的方法介绍了如何使用 D3D SDK。同时通过典型案例分析大部分三维编程的思路与问题，将 D3D 与 3dsMax 联系起来，让独立的美工与程序在游戏应用上得到完美的结合。

本书可作为高等院校数字媒体技术专业课教材，也可作为应用型本科院校计算机、信息管理与信息系统、电子商务等相关专业的实验教学用书，还可作为高职院校及企业工程人员的参考用书和对计算机图形学感兴趣的读者的参考资料。

图书在版编目(CIP)数据

DirectX 三维游戏编程/宋伟著. —西安：西安电子科技大学出版社，2016.8
普通高等教育数字媒体专业“十三五”课改规划教材
ISBN 978-7-5606-4185-0

Ⅰ. ① D… Ⅱ. ① 宋… Ⅲ. ① DirectX 软件—程序设计—高等学校—教材 ② 游戏程序—程序设计—高等学校—教材 Ⅳ. ① TP317

中国版本图书馆 CIP 数据核字(2016)第 187129 号

策　　划　刘小莉
责任编辑　阎　彬　张　欣
出版发行　西安电子科技大学出版社(西安市太白南路 2 号)
电　　话　(029)88242885　88201467　　邮　　编　710071
网　　址　www.xduph.com　　电子邮箱　xdupfxb001@163.com
经　　销　新华书店
印刷单位　陕西华沐印刷科技有限责任公司
版　　次　2016 年 8 月第 1 版　2016 年 8 月第 1 次印刷
开　　本　787 毫米×1092 毫米　1/16　印　张　13.5
印　　数　1～2000 册
字　　数　313 千字
定　　价　29.00 元
ISBN 978-7-5606-4185-0/TP

XDUP 4477001-1

前　言

本书为北方工业大学2015年教育教学改革课题成果，书中涉及的三维游戏设计内容，可以作为数字媒体技术专业游戏开发方向的必修课教材内容。本书在教授图形学理论的同时，强化三维编程环节，并为爱好网络游戏开发的IT工作者提供网络层面的技术指导。本书是将图形图像学与三维美工制作课程衔接起来的桥梁，让三维编程更实用，使读者在学习三维图形API并了解计算机图形学知识的基础上，还间接学习了以面向对象为思想编写程序的方法和相对复杂的程序的编写方法。

本书第1章为面向对象的游戏编程原理，介绍了游戏开发中常用的面向对象设计思想和简单的游戏过程原理；第2章为Windows编程基础，介绍了Win32的消息机制和窗口应用程序；第3章为DirectX简介，介绍了如何配置Direct3D开发环境和三维场景绘制的实现过程，并通过案例讲述基于面向对象思想的Direct3D开发过程模块封装方法；第4章为基本空间变换，介绍了计算机图形学的空间变换原理和算法；第5章为Direct3D的绘制方法，介绍了利用三维顶点、颜色、纹理等元素创建三维模型并绘制的方法；第6章为Alpha融合，介绍了利用Alpha通道的透明渲染方法；第7章为光照与材质，介绍了Direct3D的光照原理和物体材质创建和使用的方法；第8章为三维网格模型，介绍了从3dsMax的模型导出XFile文件的方法，以及如何通过程序将其载入Direct3D环境并进行渲染，还讲解了通过三维模型的边界检测原理实现物体碰撞的过程；第9章为动画网格模型，介绍了蒙皮动画的原理，并讲解如何将骨骼动画数据从3dsMax导出为XFile文件，并实现其在Direct3D环境中的加载和渲染。

本书由北方工业大学计算机学院宋伟编写而成，北方工业大学计算机学院

的本科生和研究生，包括于少波、武冬、郝书嘉、文明云、张伟强、黎佳雪、黄凯斯、郭海涛、廖金巧、田逸非等同学也参与了本书的编写工作。

限于编者水平，书中难免存在不足之处，殷切希望广大读者批评指正。联系邮箱：swneu@126.com。

宋　伟

北方工业大学计算机学院

2016年5月

目　录

第 1 章　面向对象的游戏编程原理

网络游戏开发分为以下几大过程：服务器编程、客户端编程、人工智能、数据库管理、游戏策划、美工设计、音乐特效等。大型游戏往往需要团队合作开发，因此面向对象的编程思想在网络游戏中得到了广泛应用。本章将阐述如何将面向对象的编程原理应用在电脑游戏开发中。

1.1　面向对象的游戏设计思想

本节主要介绍了简单的游戏开发基本流程和游戏中常用的人工智能方法，并通过程序简要地模拟了游戏过程。

1.1.1　游戏开发基本流程

游戏开发基本流程如图 1.1 所示。游戏初始化过程主要加载游戏内容的缓存，如游戏人物、背景模型、音乐等素材，并初始化相关参数。游戏实现过程主要是通过用户操作、游戏逻辑过程、游戏内容渲染三个过程交替循环实现的。在游戏执行过程中，当符合游戏结束条件时，将执行游戏结束程序，其主要工作为释放在游戏初始化时加载的缓存。若在程序结束时不释放动态分配的缓存所用的内存空间，系统通常会弹出内存泄露警告对话框。

游戏过程通常可以用有限状态机(Finite State Machine，FSM)来描述，如图 1.2 所示。当某一动作执行或事件发生后，根据 FSM 所定义的逻辑关系实现游戏中状态的转化。例如，当游戏中的人物被攻击后，将由健康状态跳转到掉血状态。

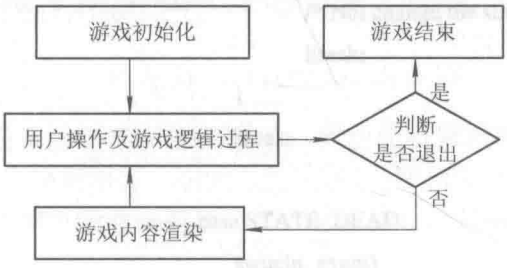


图 1.1　游戏基本流程图

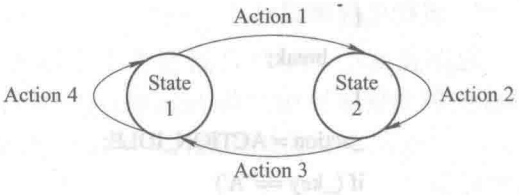


图 1.2　有限状态机原理

1.1.2　简单的游戏过程模拟程序

本小节中的模拟程序利用 C 语言实现了简单的游戏控制的模拟过程，游戏中人物具有

位置、状态、动作等属性，当按下 A、D 键时，人物分别向左和右移动，按下 E 键时，程序退出。

本程序利用枚举宏定义方法来定义游戏必要的状态(CHARACTER_ STATE)、动作(CHARACTER_ACTION)、事件(CHARACTER_EVENT)常量。这些常量在程序过程中常用于条件比较、转换赋值、触发事件等情况，具体代码如下：

```
enum   CHARACTER_STATE{STATE_LIVE = 0, STATE_DEAD};
enum   CHARACTER_ACTION{ACTION_IDLE = 0, ACTION_MOVE};
enum   CHARACTER_EVENT{EVENT_EMPTY = 0,
                       EVENT_ATTACKED, EVENT_RESTART};
```

在主程序开头为位置、状态、动作、事件等信息赋初值，实现初始化过程。然后，程序进入游戏的循环过程，直到按下 E 键退出游戏。当按下 A、D 键时，人物执行行走动作。游戏根据 _action 的值显示对应的人物动作，再根据 _state 和 _event 的值，完成有限状态机所定义的游戏逻辑。相关代码如下：

```
#include <conio.h>
#include<stdio.h>

void main()
{
    int _pos = 0;
    int _state = STATE_LIVE;
    int _action = ACTION_IDLE;
    int _event = EVENT_EMPTY;

    int _key;
    while (1)
    {
        _key = _getch();

        if (_key == 'E')
        {
            break;
        }
        _action = ACTION_IDLE;
        if (_key == 'A')
        {
            _action = ACTION_MOVE;
            _pos--;
        }
        if (_key == 'D')
```

```
    {
        _action = ACTION_MOVE;
        _pos++;
    }

    switch(_action)
    {
        case ACTION_IDLE:
            /* Render idle animation*/
            printf("Idel at position: %d\n", _pos);
            break;
        case ACTION_MOVE:
            /* Render walk animation */
            printf("Walk at position: %d\n", _pos);
            break;
    }

    switch(_state)
    {
        case STATE_LIVE:
            switch(_event)
            {
                case EVENT_ATTACKED:
                    /* Change the state */
                    _state = STATE_DEAD;
                    break;
                case EVENT_RESTART:
                    /* Not change the state */
                    break;
            }
            break;
        case STATE_DEAD:
            switch(_event)
            {
                case EVENT_ATTACKED:
                    /* Not change the state */
                    break;
                case EVENT_RESTART:
```

```
                    /* To initialize the game */
                    _state = STATE_LIVE;
                    break;
                }
            break;
        }
    }
}
```

经验：函数名和变量命名是需达意的，要使程序即便在没有注释的情况下也是可读的。命名的方法在同一程序、同一项目、同一企业要一致，保证程序易记易懂。匈牙利命名法是一种常用的命名规范，变量名要体现属性、类型、对象等描述。此外，运算符与变量、函数等对象需要用空格隔开，以使代码更美观，增强程序的可读性。软件工程强调程序中要写详细的注释，但在实际程序开发中，优秀的程序开发者所编写的代码在命名上能够直观达意，并可以充分利用面向对象编程思想，不需要过多注释就能达到易被他人看懂并且再次使用的效果。

1.2　面向对象程序设计

面向对象程序设计是针对开发较大规模的程序而提出来的，目的是提高软件开发的效率。本节将简要介绍基于 C++ 的面向对象程序开发的基础知识，并通过游戏模拟程序介绍如何将面向对象思想应用在游戏开发过程中。

1.2.1　从 C 语言到 C++

C 语言是面向过程的、结构化和模块化的语言。1.1.2 节的程序模拟例子只是面向过程的，代码逻辑复杂且不易理解。为提升代码的可读性，在 C 语言程序中，经常采用函数和结构体将代码按照功能和内容封装成不同的结构和模块。在此，我们将 1.1.2 节的简单游戏模拟题目用结构化和模块化的编程风格进行一次优化。

1. 结构化程序设计

程序设计中，往往会遇到多个个体具有相同属性的问题，如游戏中玩家控制角色(Player)和非玩家控制角色(Non-Player Character，NPC)均具有位置、状态、动作等属性。因此，对这些个体的相同属性归纳后，抽象出结构体，可以简化复杂对象内存的声明和属性控制。

根据 1.1.2 节程序中提到的游戏人物具有的位置、状态、动作等属性，我们创建一个 Character 结构体，代码如下：

```
struct Character
{
    int _pos;
```

```
    int _state;
    int _action;
    int _event;
};
```

在主函数中，只要声明一个 Character 对象 _Player 即可，通过更改其属性，可完成其初始化，代码如下：

```
Character  _Player;
_Player._pos = 0;
_Player._state = STATE_LIVE;
_Player._action = ACTION_IDLE;
_Player._event = ENENT_EMPTY;
```

2. 模块化结构设计

由于游戏中人物间的交互过程比较复杂，面向过程的编程方法使得代码量大且复杂。为此，常通过编写与主程序相独立的基本功能函数，再通过基本功能函数组合成更复杂的功能函数，将代码模块化。

游戏循环过程可以分为游戏逻辑和游戏渲染两大部分。在简单的游戏中，游戏逻辑部分主要包括游戏人物行为计划、人物间交互、人工智能(Artificial Intelligence，AI)等功能。游戏中人物的基本动作功能相对独立，这些动作常被封装为一些常用的函数，以供游戏逻辑部分直接调用。在减少了代码量的同时，模块化的代码也易被理解和使用。

我们将 1.1.2 节中的人物移动、静止以及人工智能封装为 Character_Move()、Character_Idle()、Character_AI()函数，函数封装的代码如下：

```
void Character_Move(Character* _agent, int _step)
{
    (*_agent)._action = ACTION_MOVE;
    (*_agent)._pos += _step;
}
void Character_Idle(Character* _agent)
{
    (*_agent)._action = ACTION_IDLE;
}
void Character_AI(Character* _agent)
{
    switch((*_agent)._state)
    {
        case STATE_LIVE:
            switch((*_agent)._event)
            {
                case EVENT_ATTACKED:
```

```
                    /* Change the state */
                    (*_agent)._state = STATE_DEAD;
                    break;
                case EVENT_RESTART:
                    /* Not change the state */
                    break;
            }
            break;

        case STATE_DEAD:
        switch((*_agent)._event)
        {
            case EVENT_ATTACKED:
                /* Not change the state */
                break;
            case EVENT_RESTART:
                /* To initialize the game */
                (*_agent)._state = STATE_LIVE;
                break;
        }
        break;
    }
}
```

在游戏过程中只要调用这 3 个函数即可化繁为简，函数使用的代码如下所示：

```
Character_Idle(&_Player);
Character_Move(&_Player, -1);
Character_AI(&_Player);
```

此外，游戏的渲染是最重要的过程，我们在人物渲染函数 Character _Render()中先利用语句输出方式对其模拟，代码如下所示：

```
void Character_Render(Character* _agent)
{
    switch((*_agent)._action)
    {
        case ACTION_IDLE:
            /* Render idle animation*/
            printf("Idel at position: %d\n", (*_agent)._pos);
            break;
        case ACTION_MOVE:
            /* Render walk animation */
```

```
            printf("Walk at position: %d\n", (*_agent)._pos);
            break;
        }
    }
```

调用渲染函数只用以下一条语句即可：

```
Character_Render(&_Player);
```

利用以上结构化和模块化的程序设计方法，程序的主函数将变得简单易懂，如下所示：

```
void main()
{
    Character   _Player;
    _Player._pos = 0;
    _Player._state = STATE_LIVE;
    _Player._action = ACTION_IDLE;
    _Player._event = ENENT_EMPTY;

    int _key;

    while (1)
    {
        _key = _getch();

        if (_key == 'E')
        {
            break;
        }
        Character_Idle(&_Player);
        if (_key == 'A')
        {
            Character_Move(&_Player, -1);
        }
        if (_key == 'D')
        {
            Character_Move(&_Player, 1);
        }
        Character_AI(&_Player);
        Character_Render(&_Player);
    }
}
```

3. 面向对象的 C++ 程序设计

C 语言在大规模程序合作开发的过程中，代码量较大，可读性差，软件开发接口的性质均为公有，多方程序开发者的软件算法也不易受到保护，因此其在合作研发过程中效率有限。

为方便大规模程序的研发，C++ 程序开发语言被广泛采用。C++ 是以面向对象语言，它由 C 发展而来，与 C 语言兼容。在 C 语言面向过程机制的基础上，C++ 对 C 语言的功能做了不少扩充，增加了面向对象的程序设计机制。

基于面向对象的程序开发思想，1.1.2 节中的代码可分解为三个文件：Character.h、Character.cpp、GameMain.cpp。在 Character.h 文件中声明人物类 Character，在 Character.cpp 文件中封装 Character 类的成员函数，在 GameMain.cpp 中编写整体游戏逻辑。作为本程序的开发接口，合作程序员只要看懂和使用 Character.h 文件即可，尽量不更改 Character.cpp 的函数封装部分。为减少所开发的程序被其他人更改后导致程序错误的风险，程序员常开发 dll 文件。即程序员对 Character.h 和 Character.cpp 进行编写并编译，交付给合作方 Character.h、Character.lib、Character.dll 以及 GameMain.cpp 等文件，从而使程序更加容易理解。本书主要讲解面向 .h 和 .cpp 的程序开发，感兴趣的读者可以学习 dll 文件的开发方法。

在 Character.h 文件中，我们将对 Character 类进行声明。一般而言，类的属性大多被声明为受保护成员或私有成员，而可供其他程序调用的函数声明为公有成员。代码如下所示：

```
class Character
{
    public:
        Character();
        ~Character();
        void Move(int _step);
        void Idle();
        void AI_Planning();
        void Render();

    protected:
        int _pos ;
        int _state ;
        int _action ;
        int _event ;
};
```

在 Character.cpp 中分别对类成员函数进行定义，先利用构造函数对类属性进行初始化，再封装移动动作、静止动作、行为计划、渲染等函数，代码如下所示：

```
#include "Character.h"
Character::Character()
{
```