四川理工学院毕业设计（论文）

基于D3D动作类游戏的设计与实现

学　　生: 刘 祥 宽

学　　号: 13101020411

专　　业: 软 件 工 程

班　　级: 2013级4班

指 导 教 师: 华 才 健

四川理工学院计算机学院

二〇一七年五月

基于D3D动作类游戏的设计与实现

摘　要

随着21世纪信息时代的到来，计算机和网络成为每个人日常生活中必不可少的一部分，人们的娱乐方式也慢慢的偏向互联网娱乐，计算机游戏已经成为人们娱乐的重要组成部分，游戏是一个非常好的缓解人们工作压力的方式，游戏是幻想和现实之间的桥梁。本次设计主要讨论了Windows环境下游戏的设计开发和实现，主要利用了Window系统下的API函数和微软的Visual Studio 开发工具，本游戏简单的实现了游戏场景的搭建，游戏模型的载入和游戏人物的移动，攻击，游戏的碰撞检测等。

关键词： 3D游戏场景开发; 游戏场景搭建; WIN API;

Design and implementation of action game base on D3D

ABSTRACT

In twenty-first Century with the advent of the information era, computer and network become an indispensable part of everyone's daily life, people's entertainment also slowly toward Internet entertainment, computer games have become an important part of people's entertainment, the game is a very good way to relieve people's work pressure, the game is a bridge between fantasy and reality the.

This design mainly discussed the design and implementation of Windows development environment of the game, the main use of the API function under Window system and Microsoft's Visual Studio development tools, the simple realization of the game set up the game scene, mobile game model, load and game character attacks, game collision detection etc..

Keywords: 3D game scene development ,Game scene set up , WIN API

目　　录

第1章　前言

1.1　课题背景及意义

本次课题是根据个人对游戏开发的爱好开发的一个3D游戏Demo,从本次游戏开发中可以学到很多编程方面的知识，同时这次游戏开发可以学到很多用于游戏引擎制作的底层Windows API, 为以后使用游戏引擎开发游戏奠定了坚实的基础，同时这次的游戏开发相当于制作了一个小小的底层游戏引擎，为以后游戏的制作提供了大大的方便。

1.1.1　课题背景

游戏界技术发展日新月异，很多人一开始都直接使用游戏游戏引擎，忽视了底层具体游戏逻辑的实现，国外有很多的知名的游戏引擎，而国内寥寥无几，这次游戏使用Windows D3D API开发几乎是从最底层实现游戏逻辑，极大的丰富了游戏开发底层知识，让开发者了解了游戏引擎API实现的细节，说白了游戏引擎也知识组合封装了Windows D3D API而已。

1.1.2　课题意义

1.2　研究现状

在D3D游戏开发的时候中就是学习大量游戏开发知识的过程，我们在游戏开发之中可以学习到摄像机怎么组成的，地形是怎么构建的怎么改变的，怎么设计与地形的碰撞，这些都是游戏引擎无法带给你的，因为游戏引擎只会给与你封装好的相应的API函数，所有学习D3D游戏开发就有了具体的意义，在使用D3D游戏开发的时候你可以知道实现游戏构成的具体细节，从而使你以后的游戏开发更加的细腻。

1.3　主要研究内容

本课题使用Visual Studio + D3D API 实现，运用了面向对象的思想和模块化思想来进行设计和开发，分为主界面、.X文件加载模块，地形设计模块、天空盒设计模

块、摄像机模块、按键输入模块、碰撞检测模块、粒子系统模块。使用硬编码直接加载游戏模型，主界面实现加载上述模块。论文的主要章节安排如下：

第一章：介绍了本设计的背景和研究目的意义，以及3D游戏在国内外现状和游戏的测研究现状；

第二章：介绍了c++ DirectX开发技术，包括STL和DirectX API；

第三章：对3D游戏开发作了可行性分析，讨论了游戏开发的难点问题

第四章：利用DirectX API技术，对游戏设计进行了详细实现，并采用软件测试方法对游戏模型进行了测试；

第五章：总结本设计的主要工作和解决的问题，指出了研究中存在的缺陷与不足，对未来的发展与研究方向进行了展望。

第2章　相关开发技术

2.1　可行性分析前提

要求：实现地形设计和载入、实现第三人称摄像机的设计和载入、实现天空盒的设计和载入、实现人物的移动、实现粒子系统的设计和应用。目标：实现游戏的基本操作。条件：本次3D游戏设计是作为毕业设计，开发时间6个月，此游戏需要达成作为游戏的最基本的条件。可行性研究的方法：通过对Direct X API的研究，分析3D游戏的最基本组成。决定可行性的主要因素：计算机配置和相应的软件配置情况。

2.1.1　现有的3D游戏分析

目前，直接用Direct X 底层API编程3D游戏的列子在国内很少，所以在3D游戏开发时候很难得到相应的帮助，很多时候都靠自己解决，所以说开发出一个简单的3D游戏是必须去做的。

2.1.2　技术可行性分析

在时间限制下，游戏的目标条件基本达成，最大的影响因素是时间的紧迫，由于开发的时间限制，虽然游戏的基本要求达成，但是游戏的可玩性不高。1.3D人物模型的载入。2.地形的载入使用高度图实现地形的3D显示。3.摄像机的设计实现摄像机跟随人物移动。4.地形碰撞设计，实现了人物的碰撞检测。5.天空盒实现游戏的天空3D场景。

2.1.3　用户使用可行性分析

此次游戏的开发选择的是Windows平台是Visiual Studio,这是由于directX API是由微软公司开发提供的，该技术是现在游戏编程的主流技术。

2.2 开发技术

2.2.1 C++

20世纪70年代中期，Bjarne Stroustrup在剑桥大学计算机中心工作。他使用过Simula和ALGOL，接触过C。他对Simula的类体系感受颇深，对ALGOL的结构也很有研究，深知运行效率的意义。既要编程简单、正确可靠，又要运行高效、可移植，是Bjarne Stroustrup的初衷。以C为背景，以Simula思想为基础，正好符合他的设想。1979年，Bjame Sgoustrup到了Bell实验室，开始从事将C改良为带类的C（C with classes）的工作。1983年该语言被正式命名为C++。自从C++被发明以来，它经历了3次主要的修订，每一次修订都为C++增加了新的特征并作了一些修改。第一次修订是在1985年，第二次修订是在1990年，而第三次修订发生在c++的标准化过程中。在20世纪90年代早期，人们开始为C++建立一个标准，并成立了一个ANSI和ISO（Intemational Standards Organization）国际标准化组织的联合标准化委员会。该委员会在1994年1月25曰提出了第一个标准化草案。在这个草案中，委员会在保持Stroustrup最初定义的所有特征的同时，还增加了一些新的特征。在完成C++标准化的第一个草案后不久，发生了一件事情使得C++标准被极大地扩展了：Alexander stepanov创建了标准模板库（Standard Template Library，STL）。STL不仅功能强大，同时非常优雅，然而，它也是非常庞大的。在通过了第一个草案之后，委员会投票并通过了将STL包含到C++标准中的提议。STL对C++的扩展超出了C++的最初定义范围。虽然在标准中增加STL是个很重要的决定，但也因此延缓了C++标准化的进程。

委员会于1997年11月14日通过了该标准的最终草案，1998年，C++的ANSI/IS0标准被投入使用。通常，这个版本的C++被认为是标准C++。所有的主流C++编译器都支持这个版本的C++，包括微软的Visual C++和Borland公司的C++Builder。

2.2.2 语言特点

支持数据封装和数据隐藏

在C++中，类是支持数据封装的工具，对象则是数据封装的实现。C++通过建立用户定义类支持数据封装和数据隐藏。在面向对象的程序设计中，将数据和对该数据进行合法操作的函数封装在一起作为一个类的定义。对象被说明为具有一个给定类的变量。每个给定类的对象包含这个类所规定的若干私有成员、公有成员及保护

成员。完好定义的类一旦建立，就可看成完全封装的实体，可以作为一个整体单元使用。类的实际内部工作隐藏起来，使用完好定义的类的用户不需要知道类是如何工作的，只要知道如何使用它即可。

支持继承和重用

在C++现有类的基础上可以声明新类型，这就是继承和重用的思想。通过继承和重用可以更有效地组织程序结构，明确类间关系，并且充分利用已有的类来完成更复杂、深入的开发。新定义的类为子类，成为派生类。它可以从父类那里继承所有非私有的属性和方法，作为自己的成员。

支持多态性

采用多态性为每个类指定表现行为。多态性形成由父类和它们的子类组成的一个树型结构。在这个树中的每个子类可以接收一个或多个具有相同名字的消息。当一个消息被这个树中一个类的一个对象接收时，这个对象动态地决定给予子类对象的消息的某种用法。多态性的这一特性允许使用高级抽象。继承性和多态性的组合，可以轻易地生成一系列虽然类似但独一无二的对象。由于继承性，这些对象共享许多相似的特征。由于多态性，一个对象可有独特的表现方式，而另一个对象有另一种表现方式。[

2.2.3 工作原理

C++语言的程序因为要体现高性能，所以都是编译型的。但其开发环境，为了方便测试，将调试环境做成解释型的。即开发过程中，以解释型的逐条语句执行方式来进行调试，以编译型的脱离开发环境而启动运行的方式来生成程序最终的执行代码。

生成程序是指将源码（C++语句）转换成一个可以运行的应用程序的过程。如果程序的编写是正确的，那么通常只需按一个功能键，即可搞定这个过程。该过程实际上分成两个步骤。

第一步是对程序进行编译，这需要用到编译器（compiler）。编译器将C++语句转换成机器码(也称为目标码)；如果这个步骤成功，下一步就是对程序进行链接，这需要用到链接器（linker）。链接器将编译获得机器码与C++库中的代码进行合并。C++库包含了执行某些常见任务的函数（“函数”是子程序的另一种称呼）。例如，一个C++库中包含标准的平方根函数sqrt，所以不必亲自计算平方根。C++库中还包含一些子程序，它们把数据发送到显示器，并知道如何读写硬盘上的数据文件。

2.2.4 DirectX API

MicroSoft DirectX SDK 是DirectX 编译的软件。包含了开发尖端多媒体应用软件不可或缺的开发工具，以及runtime、headers及程序库、范例执行文件、DirectX工具、并且同时支持支援C++以及Visual Basic开发软件。

2.2.5 功能

DirectX可以开发出高性能实时的应用程序，可以直接访问计算机中的硬件和将来系统中新的硬件设备。DirectX在硬件和应用之间提供了一致的接口以减少安装和配置的复杂性，并且使硬件的利用达到最优。利用DirectX提供的接口，程序员能充分利用硬件的特性而不需要考虑其具体细节。

2.2.6 构成

DirectDraw 通过支持访问屏外显示内存中位图的软硬件加速技术，快速直接存取，利用硬件的位块传输和缓冲区翻转功能。

DirectSound 提供软硬件声音混合和录音再生功能。

DirectMusic 提供软硬件MID音乐的播放功能。

DirectPlay 使得游戏在调制解调器和网络之间的连接更加简单方便。

Direct3D 允许程序完成一个完全的三维图形系统和完全控制着色管道。

DirectInput 提供了基于Windows游戏的输入的API程序，包括键盘、鼠标和操纵杆，以及将来的基于Windows新的输入设备。

DirectSetup 提供了DirectX的一次性安装过程。

2.3 本章小结

本章主要介绍这次毕业设计程序开发需要用到的语言和技术。

第3章　系统总体设计

3.1　需求分析

游戏界技术发展日新月异，很多人一开始都直接使用游戏游戏引擎，忽视了底层具体游戏逻辑的实现，国外有很多的知名的游戏引擎，而国内寥寥无几，这次游戏使用Windows D3D API开发几乎是从最底层实现游戏逻辑，极大的丰富了游戏开发底层知识，让开发者了解了游戏引擎API实现的细节，说白了游戏引擎也知识组合封装了Windows D3D API而已。

(1) 模块需求分析

本系统共包括7个模块，其中有：天空盒模块、摄像机模块、三维地形模块、设备信息接收模块、模型加载模块、碰撞检测模块、粒子模块。

(2) 栏目需求分析

本系统共包括7个目，中有：三维地形类、天空盒类、摄像机类、碰撞检测类、粒子类、文件加载类、控制检测类。

(3) 数据库需求分析

本次游戏开发采用硬编码类型开发实现简单的游戏逻辑，没有使用到数据库。

3.2　总体设计

3.2.1　功能

3.2.2　功能描述

人物流程图：

三维地形：利用载入高度图渲染绘制的地形网格，实现人物站立的平台。

人物加载类：该类用于加载磁盘上的.X文件，并实现设置人物的位置。

摄像机类：该类用于观察创建出的萨内立体世界。

天空盒：用于构造覆盖整个地形的三维天空。

游戏控制检测：用于接受键盘和鼠标设备的输入信息。

碰撞检测：用于检测人物与地形的碰撞使人物站立在地形上。

粒子类：用于实现粒子效果。

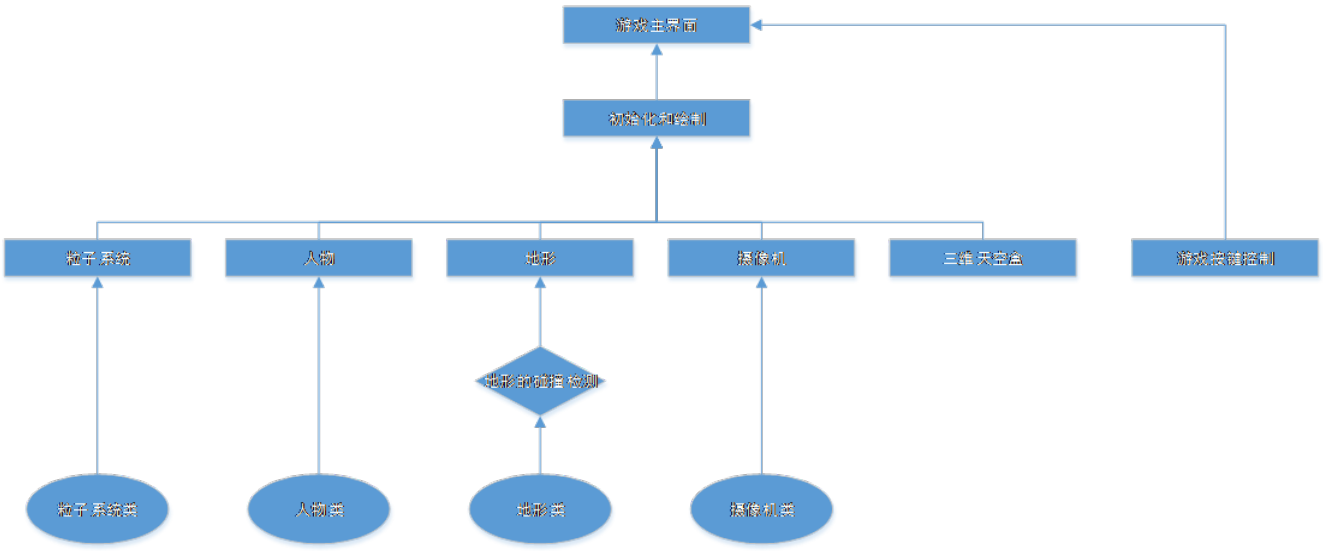

图 3-1 功能划分

3.2.3 功能描述

人物的移动：运行程序可以按键盘W、A、S、D实现前左下右的角色移动操作。

人物的攻击：采用粒子系统实现人物的攻击敌人。

人物移动中的碰撞检测：通过获取地形的高度图实现人物移动中与地形的碰撞检测。

敌人的碰撞检测：通过AABB包围盒实现让敌人对玩家的攻击进行检测，实现反击。

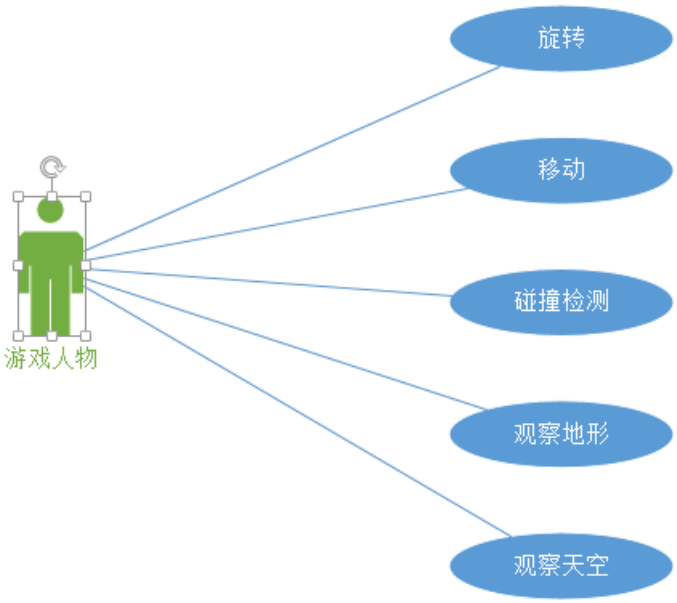

图 3-2 功能划分

第4章　系统详细设计与实现

详细设计的工作包括处理过程、代码设计两个部分。即根据总体设计的每个功能模块的要求和，利用相应的开发工具设计出应用系统的过程。由于篇幅的限制，这里不给出具体的程序编码，只给出主要模块的功能设计描述，三维地形类、天空盒类、摄像机类、碰撞检测类、粒子类、文件加载类、控制检测类的实现。

4.1　游戏的设计与实现

4.1.1　游戏项目文件截图

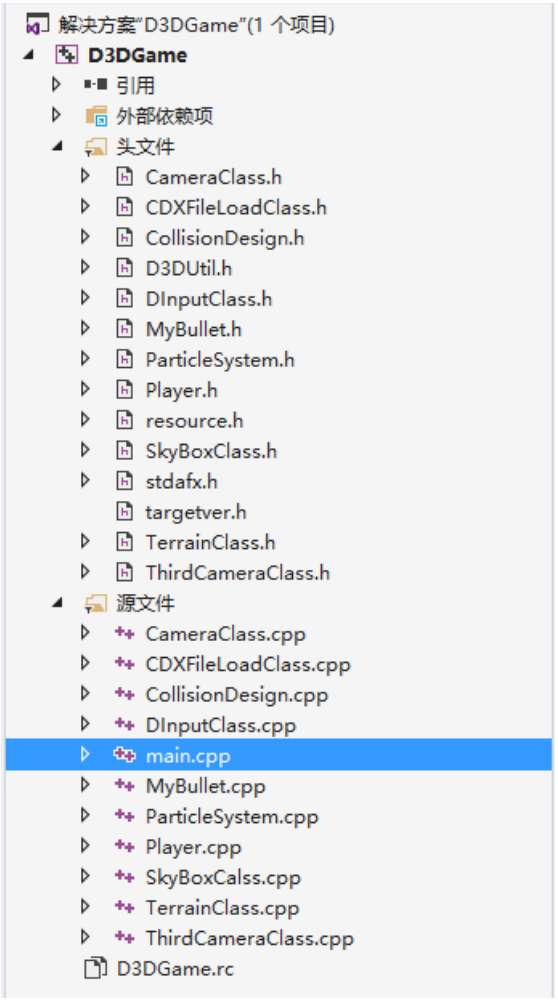

图 4-1 功能划分

4.2 摄像机类的设计与实现

4.2.1 摄像机成员函数的设计

摄像机位置和观察点设置成员函数

该成员函数主要用于摄像机位置的设定和观察点位置（玩家位置）的设定。

```
VOID CameraClass::SetTargetPosition(D3DXVECTOR3 *pLookat)
{
    //先看看pLookat是否为默认值NULL
    if (pLookat != NULL)  m_vTargetPosition = (*pLookat);
    else m_vTargetPosition = D3DXVECTOR3(0.0f, 0.0f, 1.0f);

    m_vLookVector =m_vTargetPosition - m_vCameraPosition;//观察点(
    D3DXVec3Normalize(&m_vLookVector, &m_vLookVector);//规范化m_vL

     //正交并规范化m_vUpVector和m_vRightVector
    D3DXVec3Cross(&m_vUpVector, &m_vLookVector, &m_vRightVector);
    D3DXVec3Normalize(&m_vUpVector, &m_vUpVector);
    D3DXVec3Cross(&m_vRightVector, &m_vUpVector, &m_vLookVector);
    D3DXVec3Normalize(&m_vRightVector, &m_vRightVector);
}
```

图 4-2 功能划分

摄像机取景变换函数该成员函数用于计算摄像机的取景变换，构造摄像机自身相对于世界坐标系的局部坐标系。

```
VOID CameraClass::CalculateViewMatrix(D3DXMATRIX *pMatrix)
{
    //1.先把3个向量都规范化并使其相互垂直，成为一组正交矩阵
    D3DXVec3Normalize(&m_vLookVector, &m_vLookVector);  //规范化观察分量
    D3DXVec3Cross(&m_vUpVector, &m_vLookVector, &m_vRightVector);    // 上向量与观
    D3DXVec3Normalize(&m_vUpVector, &m_vUpVector);                  // 规范化上向量
    D3DXVec3Cross(&m_vRightVector, &m_vUpVector, &m_vLookVector);    // 右向量与上
    D3DXVec3Normalize(&m_vRightVector, &m_vRightVector);            // 规范化右向量

    // 2.创建出取景变换矩阵
    //依次写出取景变换矩阵的第一行
    pMatrix->_11 = m_vRightVector.x;          // Rx
    pMatrix->_12 = m_vUpVector.x;             // Ux
    pMatrix->_13 = m_vLookVector.x;           // Lx
    pMatrix->_14 = 0.0f;
    //依次写出取景变换矩阵的第二行
    pMatrix->_21 = m_vRightVector.y;          // Ry
    pMatrix->_22 = m_vUpVector.y;             // Uy
    pMatrix->_23 = m_vLookVector.y;           // Ly
    pMatrix->_24 = 0.0f;
    //依次写出取景变换矩阵的第三行
    pMatrix->_31 = m_vRightVector.z;          // Rz
    pMatrix->_32 = m_vUpVector.z;             // Uz
    pMatrix->_33 = m_vLookVector.z;           // Lz
    pMatrix->_34 = 0.0f;
    //依次写出取景变换矩阵的第四行
    pMatrix->_41 = -D3DXVec3Dot(&m_vCameraPosition,&m_vRightVector);    // -P*R
    pMatrix->_42 = -D3DXVec3Dot(&m_vCameraPosition,&m_vUpVector);       // -P*U
    pMatrix->_43 = -D3DXVec3Dot(&m_vCameraPosition,&m_vLookVector);     // -P*L
    pMatrix->_44 = 1.0f;
}
```

图 4-3 功能划分

摄像机位置Y轴高度设置函数和摄像机观察点位置设置函数

该成员函数用于设置摄像机Y轴高度和摄像机观察点Y轴高度，用于模拟摄像机和人物移动时和地面的碰撞效果。

```
void CameraClass::SetCameraHeight(float iY)
{
    m_vCameraPosition.y = iY;
}

void CameraClass::SetTargetHeight(float iY)
{
    m_vTargetPosition.y = iY;
}
```

图 4-4 功能划分

摄像机绕观察点Y轴旋转函数

该成员函数用于实现摄像机绕观察点（玩家）Y轴旋转。

```
//绕角色Y轴旋转
VOID CameraClass::ThirdPersionRotationY(float Angle)
{
    D3DXMATRIX matRotationY;
    D3DXVECTOR3 vTemp;
    D3DX                    XVECTOR3(0.0f, 1.0f, 0.0f);
    D3DX                    atRotationY, &m_matup, Angle);
    D3DXVec3TransformCoord(&vTemp, &(m_vCameraPosition - m_vTargetPosition), &matRotationY);
    D3DXVec3TransformCoord(&m_vRightVector, &m_vRightVector, &matRotationY);
    D3DXVec3TransformCoord(&m_vLookVector, &m_vLookVector, &matRotationY);

    m_vCameraPosition =m_vTargetPosition+vTemp;
}
```

struct D3DXVECTOR3

图 4-5 功能划分

摄像机绕观察点X轴旋转函数

该成员函数用于实现摄像机绕观察点（玩家）X轴旋转。

```
//绕角色X轴旋转
VOID CameraClass::ThirdPersionRotationX(float Angle)
{
    D3DXMATRIX matRotationX;
    D3DXVECTOR3 vNewEye, vTemp;
    D3DXVec3Cross(&m_vNewXAxis, &(m_vTargetPosition - m_vCameraPosition), &m_vUpVector);
    D3DXMatrixRotationAxis(&matRotationX, &m_vNewXAxis, Angle);
    D3DXVec3TransformCoord(&vNewEye, &(m_vCameraPosition - m_vTargetPosition), &matRotationX);
    D3DXVec3TransformCoord(&m_vLookVector, &m_vLookVector, &matRotationX);
    D3DXVec3TransformCoord(&m_vUpVector, &m_vUpVector, &matRotationX);
    vNewEye += m_vTargetPosition;
    D3DXVec3Normalize(&vTemp, &m_vUpVector);
    float fAngle = D3DXVec3Dot(&vTemp, &m_vUpVector);
    if (fAngle<(-0.95f)||fAngle>0.95f)
    {
        return;
    }
    m_vCameraPosition = vNewEye;
}
```

图 4-6 功能划分

```
//-----------------------------------------------------
// Name: CameraClass::MoveAlongRightVec( )
// Desc: 沿右向里平移fUnits个单位
//-----------------------------------------------------
VOID CameraClass::MoveAlongRightVec(FLOAT fUnits)
{
    //直接乘以fUnits的里来累加就行了
    m_vCameraPosition += m_vRightVector * fUnits;
    m_vTargetPosition += m_vRightVector * fUnits;

}

//-----------------------------------------------------
// Name: CameraClass::MoveAlongUpVec( )
// Desc:  沿上向里平移fUnits个单位
//-----------------------------------------------------
VOID CameraClass::MoveAlongUpVec(FLOAT fUnits)
{
    //直接乘以fUnits的里来累加就行了
    m_vCameraPosition += m_vUpVector * fUnits;
    m_vTargetPosition += m_vUpVector * fUnits;
}

//-----------------------------------------------------
// Name: CameraClass::MoveAlongLookVec( )
// Desc:  沿观察向里平移fUnits个单位
//-----------------------------------------------------
VOID CameraClass::MoveAlongLookVec(FLOAT fUnits)
{
    //直接乘以fUnits的里来累加就行了
    m_vCameraPosition += m_vLookVector * fUnits;
    m_vTargetPosition += m_vLookVector * fUnits;

}
```

图 4-7 功能划分

该成员函数用于实现摄像机的平移。

摄像机设置投影变换函数

该成员函数将三维物体投影到二维屏幕。

```
//-----------------------------------------------------------------------------
// Name: CameraClass::SetProjMatrix( )
// Desc: 设置投影变换矩阵
//-----------------------------------------------------------------------------
VOID CameraClass::SetProjMatrix(D3DXMATRIX *pMatrix)
{
    //判断值有没有，没有的话就计算一下
    if (pMatrix != NULL) m_matProj = *pMatrix;
    else D3DXMatrixPerspectiveFovLH(&m_matProj, D3DX_PI / 4.0f, (float)((double)WINDOW_WIDTH / WINDOW_HEIGHT), 1.0f, 3000
    m_pd3dDevice->SetTransform(D3DTS_PROJECTION, &m_matProj);//设置投影变换矩阵
}
```

图 4-8 功能划分

4.3 文件载入类的设计与实现

4.3.1 文件成员函数的设计

X文件载入成员函数

模型渲染函数是将3D模型绘制在屏幕当中，主要调用LPDIRECT3DDEVICE9 设备接口实现。

```
HRESULT FileLoadClass::RenderModel()
{
    for (DWORD i = 0; i<m_dwNumMaterials; i++)
    {

        m_pd3dDevice->SetMaterial(&m_pMaterials[i]);
        m_pd3dDevice->SetTexture(0, m_pTextures[i]);
        m_pMesh->DrawSubset(i);
    }
    return S_OK;
}
```

图 4-9 功能划分

创建纹理纹理优化：

```
if (d3dxMaterials[i].pTextureFilename != NULL &&
    strlen(d3dxMaterials[i].pTextureFilename) > 0)
{
    //创建纹理
    if (FAILED(D3DXCreateTextureFromFileA(m_pd3dDevice, d3dxMaterials[i].pTex
    {
        MessageBox(NULL, L"SORRY~!没有找到纹理文件!", L"FileLoadClass类读取文
    }
}
}
//优化网格模型
m_pMesh->OptimizeInplace(D3DXMESHOPT_COMPACT | D3DXMESHOPT_ATTRSORT | D3DXMESHOPT
    (DWORD*)pAdjacencyBuffer->GetBufferPointer(), NULL, NULL, NULL);

return S_OK;
```

图 4-10 功能划分

文件载入类成员函数

该成员函数用于X文件模型的载入，使用API 函数D3DXLoadMeshFromX 从磁盘中读取需要的X文件模型，使用API 函数D3DXCreateTextureFromFileA 创建纹理。

```
//
HRESULT FileLoadClass::LoadModelFromXFile(WCHAR* strFilename)
{

    LPD3DXBUFFER pAdjacencyBuffer = NULL;  //网格模型邻接信息
    LPD3DXBUFFER pD3DXMtrlBuffer = NULL;   //存储网格模型材质的缓存对象

                                           //从磁盘文件加载网格模型
    D3DXLoadMeshFromX(strFilename, D3DXMESH_MANAGED, m_pd3dDevice, &pAdjacencyBuffer,
        &pD3DXMtrlBuffer, NULL, &m_dwNumMaterials, &m_pMesh);

    // 读取材质和纹理数据
    D3DXMATERIAL* d3dxMaterials = (D3DXMATERIAL*)pD3DXMtrlBuffer->GetBufferPointer();
    m_pMaterials = new D3DMATERIAL9[m_dwNumMaterials];
    m_pTextures = new LPDIRECT3DTEXTURE9[m_dwNumMaterials];

    //逐子集提取材质属性和纹理文件名
    for (DWORD i = 0; i<m_dwNumMaterials; i++)
    {
        //获取材质，并设置一下环境光的颜色值
        m_pMaterials[i] = d3dxMaterials[i].MatD3D;
        m_pMaterials[i].Ambient = m_pMaterials[i].Diffuse;

        //创建一下纹理对象
        m_pTextures[i] = NULL;
```

图 4-11 功能划分

4.4 输入控制类的设计与实现

4.4.1 输入控制成员函数的设计

返回鼠标X轴坐标函数

该成员函数用于返回鼠标的坐标信息，用于操控摄像机的动作

```
//-----------------------------------
// Name: DInputClass::MouseDX
// Desc: 返回鼠标指针的X轴坐标值
//-----------------------------------
float DInputClass::MouseDX()
{
    return (float)m_MouseState.lX;
}
```

图 4-12 功能划分

获取输入信息成员函数

该成员函数用于获取键盘和鼠标的输入信息。

```
void DInputClass::GetInput()
{
    HRESULT hr = m_KeyboardDevice->GetDeviceState(sizeof(m_keyBuffer), (void**)&m_keyBuffer);
    //获取键盘输入消息
    if (hr)
    {
        m_KeyboardDevice->Acquire();
        m_KeyboardDevice->GetDeviceState(sizeof(m_keyBuffer), (LPVOID)m_keyBuffer);
    }

    hr = m_MouseDevice->GetDeviceState(sizeof(DIMOUSESTATE), (void**)&m_MouseState);
    //获取鼠标输入消息
    if (hr)
    {
        m_MouseDevice->Acquire();
        m_MouseDevice->GetDeviceState(sizeof(DIMOUSESTATE), (void**)&m_MouseState);
    }
}
```

图 4-13 功能划分

输入控制类默认构造函数

该函数用于初始化输入控制类的成员变量。

```
HRESULT DInputClass::Init(HWND hWnd, HINSTANCE hInstance, DWORD keyboardCoopFlags, DWORD mouseCoopFlag
{
    HRESULT hr;
    //初始化一个IDirectInput8接口对象
    HR(DirectInput8Create(hInstance, DIRECTINPUT_VERSION,
        IID_IDirectInput8, (void**)&m_pDirectInput, NULL));

    //进行键盘设备的初始化
    HR(m_pDirectInput->CreateDevice(GUID_SysKeyboard, &m_KeyboardDevice, NULL));
    HR(m_KeyboardDevice->SetCooperativeLevel(hWnd, keyboardCoopFlags));
    HR(m_KeyboardDevice->SetDataFormat(&c_dfDIKeyboard));
    HR(m_KeyboardDevice->Acquire());
    HR(m_KeyboardDevice->Poll());

    //进行鼠标设备的初始化
    HR(m_pDirectInput->CreateDevice(GUID_SysMouse, &m_MouseDevice, NULL));
    HR(m_MouseDevice->SetCooperativeLevel(hWnd, mouseCoopFlags));
    HR(m_MouseDevice->SetDataFormat(&c_dfDIMouse));
    HR(m_MouseDevice->Acquire());
    HR(m_KeyboardDevice->Poll());

    return S_OK;
}
```

图 4-14 功能划分

4.5 地形类的设计与实现

4.5.1 地形类成员函数的设计

地形FVF结构体

设置地形的结构体，用于填充相应的函数。

```
//定义一个地形的FVF顶点格式
struct TERRAINVERTEX
{
    FLOAT _x, _y, _z;
    FLOAT _u, _v;
    TERRAINVERTEX(FLOAT x, FLOAT y, FLOAT z, FLOAT u, FLOAT v)
        :_x(x), _y(y), _z(z), _u(u), _v(v) {}
    static const DWORD FVF = D3DFVF_XYZ | D3DFVF_TEX1;
};
```

图 4-15 功能划分

高度图加载函数

该成员函数用于加载存储磁盘中的高度图的高度信息，主要是用C++数据流形式，通过初始化一个容器来存储高度信息，载入高度信息可以改变地形的高度。

```
BOOL TerrainClass::LoadTerrainFromFile(wchar_t *pRawFileName, wchar_t *pTextureFile)
{
    // 从文件中读取高度信息
    std::ifstream inFile;
    inFile.open(pRawFileName, std::ios::binary);   //用二进制的方式打开文件

    inFile.seekg(0, std::ios::end);                          //把文件指针移动到文件末
    std::vector<BYTE> inData(inFile.tellg());          //用模板定义一个vector<BYTE>

    inFile.seekg(std::ios::beg);                                //将文件指针移动到文
    inFile.read((char*)&inData[0], inData.size());  //关键的一步，读取整个高度信息
    inFile.close();                                                  //操作结束，可以

    m_vHeightInfo.resize(inData.size());                    //将m_vHeightInfo尺寸取为
    //遍历整个缓冲区，将inData中的值赋给m_vHeightInfo
    for (unsigned int i = 0; i<inData.size(); i++)
        m_vHeightInfo[i] = inData[i];

    // 加载地形纹理
    if (FAILED(D3DXCreateTextureFromFile(m_pd3dDevice, pTextureFile, &m_pTexture)))
        return FALSE;

    return TRUE;
}
```

图 4-16 功能划分

地形初始化函数

该成员函数用于初始化地形的高度信息，主要用于初始化行顶点个数、列顶点个数、顶点间距、地形宽度、地形高度、每列单元格数、每行单元格数，这样地形渲染的时候就可以根据地形初始化的数据填充网格信息和纹理。

```
BOOL TerrainClass::InitTerrain(INT nRows, INT nCols, FLOAT fSpace, FLOAT fScale)
{
    m_nCellsPerRow = nRows;  //每行的单元格数目
    m_nCellsPerCol = nCols;  //每列的单元格数目
    m_fCellSpacing = fSpace;    //单元格间的间距
    m_fHeightScale = fScale; //高度缩放系数
    m_fTerrainWidth = nRows * fSpace;  //地形的宽度
    m_fTerrainDepth = nCols * fSpace;  //地形的深度
    m_nVertsPerRow = m_nCellsPerCol + 1;  //每行的顶点数
    m_nVertsPerCol = m_nCellsPerRow + 1; //每列的顶点数
    m_nNumVertices = m_nVertsPerRow * m_nVertsPerCol;  //顶点总数

    // 通过一个for循环，逐个把地形原始高度乘以缩放系数，得到缩放后的高度
    for (unsigned int i = 0; i<m_vHeightInfo.size(); i++)
        m_vHeightInfo[i] *= m_fHeightScale;
    //-----------------------------------------------------------
    // 处理地形的顶点
    //-----------------------------------------------------------
    //1，创建顶点缓存
    if (FAILED(m_pd3dDevice->CreateVertexBuffer(m_nNumVertices * sizeof(TERRAINVERTEX),
        D3DUSAGE_WRITEONLY, TERRAINVERTEX::FVF, D3DPOOL_MANAGED, &m_pVertexBuffer, 0)))
        return FALSE;
    //2，加锁
    TERRAINVERTEX *pVertices = NULL;
    m_pVertexBuffer->Lock(0, 0, (void**)&pVertices, 0);
    //3，访问，赋值
    FLOAT fStartX = -m_fTerrainWidth / 2.0f, fEndX = m_fTerrainWidth / 2.0f;        //指
    FLOAT fStartZ = m_f                                                             rainDepth / 2.0f;    //指定起
    FLOAT fCoordU = 10.                                                             指定纹理的横坐标值
    FLOAT fCoordV = 10.                                                             指定纹理的纵坐标值
```

FLOAT TerrainClass::m_fTerrainWidth
地形的宽度

图 4-17 功能划分

```
int nIndex = 0, i = 0, j = 0;
for (float z = fStartZ; z >= fEndZ; z -= m_fCellSpacing, i++)       //Z坐标方向上起始顶点到结
{
    j = 0;
    for (float x = fStartX; x <= fEndX; x += m_fCellSpacing, j++)   //X坐标方向上起始顶点到结
    {
        nIndex = i * m_nCellsPerRow + j;        //指定当前顶点在顶点缓存中的位置
        pVertices[nIndex] = TERRAINVERTEX(x, m_vHeightInfo[nIndex], z, j*fCoordU, i*fCoordV);
        nIndex++;                                           //索引数自加1
    }
}
//4，解锁
m_pVertexBuffer->Unlock();
```

图 4-18 功能划分

地形渲染函数

该函数用于渲染地形，绘制地形，可以根据要求渲染出大小合适的地形环境使人物可以在地形上自由移动。

```
BOOL TerrainClass::RenderTerrain(D3DXMATRIX *pMatWorld, BOOL bRenderFrame)
{
    m_pd3dDevice->SetStreamSource(0, m_pVertexBuffer, 0, sizeof(TERRAINVERTEX));
    m_pd3dDevice->SetFVF(TERRAINVERTEX::FVF);//指定我们使用的灵活顶点格式的宏名称
    m_pd3dDevice->SetIndices(m_pIndexBuffer);//设置索引缓存
    m_pd3dDevice->SetTexture(0, m_pTexture);//设置纹理

    m_pd3dDevice->SetRenderState(D3DRS_LIGHTING, FALSE);    //关闭光照
    m_pd3dDevice->SetTransform(D3DTS_WORLD, pMatWorld); //设置世界矩阵
    m_pd3dDevice->DrawIndexedPrimitive(D3DPT_TRIANGLELIST, 0, 0,
        m_nNumVertices, 0, m_nNumVertices * 2);     //绘制顶点

    m_pd3dDevice->SetRenderState(D3DRS_LIGHTING, TRUE);  //打开光照
    m_pd3dDevice->SetTexture(0, 0); //纹理置空

    //对是否渲染线框的处理代码
    if (bRenderFrame)  //如果要渲染出线框的话
    {
        m_pd3dDevice->SetRenderState(D3DRS_FILLMODE, D3DFILL_WIREFRAME); //把填充
        m_pd3dDevice->DrawIndexedPrimitive(D3DPT_TRIANGLELIST, 0, 0,
            m_nNumVertices, 0, m_nNumVertices * 2); //绘制顶点
        m_pd3dDevice->SetRenderState(D3DRS_FILLMODE, D3DFILL_SOLID);    //把填充
    }
    return TRUE;
}
```

图 4-19 功能划分

地形默认构造函数

该成员函数用于对地形成员变量的初始化。

```
TerrainClass::TerrainClass(IDirect3DDevice9* pd3dDevice)
{
    //给各个成员变量赋初值
    m_pd3dDevice = pd3dDevice;
    m_pTexture = NULL;
    m_pIndexBuffer = NULL;
    m_pVertexBuffer = NULL;
    m_nCellsPerRow = 0;
    m_nCellsPerCol = 0;
    m_nVertsPerRow = 0;
    m_nVertsPerCol = 0;
    m_nNumVertices = 0;
    m_fTerrainWidth = 0.0f;
    m_fTerrainDepth = 0.0f;
    m_fCellSpacing = 0.0f;
    m_fHeightScale = 0.0f;
}
```

图 4-20 功能划分

返回地形高度函数

该成员函数通过网格点的计算返回地形的高度，该函数用于摄像机和人物与地形的碰撞。

```
float TerrainClass::GetHeight(float x, float z)
{
    x = x + m_fTerrainWidth / 2.0f;//x=x-m_fstartX
    z = m_fTerrainDepth / 2.0f - z;//z=m_fstartZ-z
    //将坐标转换成行和列
    int row = (int)floor(z / m_fCellSpacing);//向下取整
    int col = (int)floor(x / m_fCellSpacing);
    //获取一个小格子四个顶点的高度信息
    float A = GetHeightFromXZ(row, col);
    float B = GetHeightFromXZ(row, col + 1);
    float C = GetHeightFromXZ(row + 1, col);
    float D = GetHeightFromXZ(row + 1, col + 1);
    //找出点在哪个三角形上
    float dx = x - col*m_fCellSpacing;
    float dz = z - row*m_fCellSpacing;
    float fHeight;
    //利用斜率计算点的位置是在上三角形或者下三角形,如果在三角形ABC上
    if (dx + dz<(float)m_fCellSpacing)
    {
        float fABHeight = B - A;
        float fACHeight = C - A;
        fHeight = A + dx / m_fCellSpacing*fABHeight + dz / m_fCellSpacing*fACHeight;
    }//或者在三角形BCD上
    else
    {
        float fBDHeight = B - D;
        float fCDHeight = C - D;
        fHeight = D + (m_fCellSpacing - dx) / m_fCellSpacing*fCDHeight + (m_fCellSpacing - dz) / m_fCellSpac
    }
    return fHeight;
}
```

图 4-21 功能划分

4.6 碰撞帮助类的设计与实现

4.6.1 碰撞帮助成员函数的设计

包围球结构体

包围球的结构分为球心和半径。

```
//包围球结构体
struct sBoundingSphere
{
    D3DXVECTOR3 pos;//球的球心
    float radius;  //半径
};
```

图 4-22 功能划分

AABB立方包围体

AABB立方包围体分为四个点的位置

```
//AABB立方包围体
struct sRect
{
    float left, top, right, bottom;
};
```

图 4-23 功能划分

盒形碰撞检测函数

该成员函数用于盒形碰撞的检测。

```
//碰撞检测
bool Helper::CheckBoxCollide(D3DXVECTOR3 min1, D3DXVECTOR3 max1, D3DXVECTOR3 min2, D3DXVECTOR3 max2)
{
    if (min1.y>max2.y || max1.y<min2.y)
    {
        return false;
    }
    sRect rect1 = { min1.x,min1.z,max1.x,max1.z }, rect2 = { min2.x,min2.z,max2.x,max2.z };
    if (rect1.right < rect2.left) return FALSE;
    if (rect1.left > rect2.right) return FALSE;

    if (rect1.bottom < rect2.top) return FALSE;
    if (rect1.top > rect2.bottom) return FALSE;

    return TRUE;

}
```

图 4-24 功能划分

4.7 网格模型抽象基类

网格模型基类用于子类的继承。

```
class BaseMesh
{
public:
    virtual void Render() = 0; //渲染
    virtual ID3DXMesh* GetMeshObj() = 0;
    virtual void Release() = 0;
    virtual LPDIRECT3DDEVICE9 GetDevice() = 0;
    virtual ~BaseMesh() {}
};
```

图 4-25 功能划分

4.8 天空盒类的设计与实现

4.8.1 天空盒成员函数的设计

该函数用于初始化天空盒的结构，设计天空盒的形状大小等数值，为顶点缓冲区赋值。

```
//1.创建。创建顶点缓存
m_pd3dDevice->CreateVertexBuffer(20 * sizeof(SKYBOXVERTEX), 0,
    D3DFVF_SKYBOX, D3DPOOL_MANAGED, &m_pVertexBuffer, 0);

//用一个结构体把顶点数据先准备好
SKYBOXVERTEX vertices[] =
{
    //前面的四个顶点
    { -m_Length / 2, 0.0f, m_Length / 2, 0.0f, 1.0f, },
    { -m_Length / 2, m_Length / 2, m_Length / 2, 0.0f, 0.0f, },
    { m_Length / 2, 0.0f, m_Length / 2, 1.0f, 1.0f, },
    { m_Length / 2, m_Length / 2, m_Length / 2, 1.0f, 0.0f, },

    //背面的四个顶点
    { m_Length / 2, 0.0f, -m_Length / 2, 0.0f, 1.0f, },
    { m_Length / 2, m_Length
    { -m_Length / 2, 0.0f,
    { -m_Length / 2, m_Leng

    //左面的四个顶点
    { -m_Length / 2, 0.0f, -m_Length / 2, 0.0f, 1.0f, },
    { -m_Length / 2, m_Length / 2, -m_Length / 2, 0.0f, 0.0f, },
    { -m_Length / 2, 0.0f, m_Length / 2, 1.0f, 1.0f, },
    { -m_Length / 2, m_Length / 2, m_Length / 2, 1.0f, 0.0f, },

    //右面的四个顶点
    { m_Length / 2, 0.0f, m_Length / 2, 0.0f, 1.0f, },
    { m_Length / 2, m_Length / 2, m_Length / 2, 0.0f, 0.0f, },
    { m_Length / 2, 0.0f, -m_Length / 2, 1.0f, 1.0f, },
    { m_Length / 2, m_Length / 2, -m_Length / 2, 1.0f, 0.0f, },

    //上面的四个顶点
    { m_Length / 2, m_Length / 2, -m_Length / 2, 1.0f, 0.0f, },
    { m_Length / 2, m_Length / 2, m_Length / 2, 1.0f, 1.0f, },
    { -m_Length / 2, m_Length / 2, -m_Length / 2, 0.0f, 0.0f, },
    { -m_Length / 2, m_Length / 2, m_Length / 2, 0.0f, 1.0f, },
```

float SkyBoxClass::m_Length
天空盒边长

图 4-26 功能划分

天空盒的结构体

该结构体用于设计天空盒的结构。

```
struct SKYBOXVERTEX
{
    float   x, y, z;
    float   u, v;
};
```

图 4-27 功能划分

天空盒渲染函数

该函数用于天空盒在绘制，设置灵活顶点格式和选择是否要渲染线框然后将其放入三维空间。

天空盒贴图载入函数

```
void SkyBoxClass::RenderSkyBox(D3DXMATRIX *pMatWorld, BOOL bRenderFrame)
{
    m_pd3dDevice->SetTextureStageState(0, D3DTSS_COLOROP, D3DTOP_SELECTARG1);  /
    m_pd3dDevice->SetTextureStageState(0, D3DTSS_COLORARG1, D3DTA_TEXTURE);    //
    m_pd3dDevice->SetSamplerState(0, D3DSAMP_ADDRESSU, D3DTADDRESS_MIRROR);
    m_pd3dDevice->SetSamplerState(0, D3DSAMP_ADDRESSV, D3DTADDRESS_MIRROR);

    m_pd3dDevice->SetTransform(D3DTS_WORLD, pMatWorld);  //设置世界矩阵
    m_pd3dDevice->SetStreamSource(0, m_pVertexBuffer, 0, sizeof(SKYBOXVERTEX));
    m_pd3dDevice->SetFVF(D3DFVF_SKYBOX);  //设置FVF灵活顶点格式

    //一个for循环，将5个面绘制出来
    for (int i = 0; i<5; i++)
    {
        m_pd3dDevice->SetTexture(0, m_pTexture[i]);
        m_pd3dDevice->DrawPrimitive(D3DPT_TRIANGLESTRIP, i * 4, 2);
    }

    //对是否渲染线框的处理代码
    if (bRenderFrame)  //如果要渲染出线框的话
    {
        m_pd3dDevice->SetRenderState(D3DRS_FILLMODE, D3DFILL_WIREFRAME); //把填充
        //一个for循环，将5个面的线框绘制出来
        for (int i = 0; i<5; i++)
        {
            m_pd3dDevice->DrawPrimitive(D3DPT_TRIANGLESTRIP, i * 4, 2); //绘制顶
        }

        m_pd3dDevice->SetRenderState(D3DRS_FILLMODE, D3DFILL_SOLID);    //把填充
    }
}
```

图 4-28 功能划分

该函数用于载入天空盒的纹理贴图，使用API 函数D3DXCreateTextureFromFile从

磁盘加载文件，使天空盒看起来更加真实。

```
BOOL SkyBoxClass::LoadSkyTextureFromFile(
    wchar_t *pFrontTextureFile,
    wchar_t *pBackTextureFile,
    wchar_t *pLeftTextureFile,
    wchar_t *pRightTextureFile,
    wchar_t *pTopTextureFile)
{
    //从文件加载五张纹理
    D3DXCreateTextureFromFile(m_pd3dDevice, pFrontTextureFile, &m_pTexture[0]);  //前面
    D3DXCreateTextureFromFile(m_pd3dDevice, pBackTextureFile, &m_pTexture[1]);  //后面
    D3DXCreateTextureFromFile(m_pd3dDevice, pLeftTextureFile, &m_pTexture[2]);  //左面
    D3DXCreateTextureFromFile(m_pd3dDevice, pRightTextureFile, &m_pTexture[3]);  //右面
    D3DXCreateTextureFromFile(m_pd3dDevice, pTopTextureFile, &m_pTexture[4]);  //上面
    return TRUE;
}
```

图 4-29 功能划分

4.9 粒子系统的设计与实现

4.9.1 粒子系统成员函数的设计

粒子系统结构体

该结构体用于包含粒子的基本特征。

```
struct sParticle
{
    D3DXVECTOR3 pos;
    D3DXVECTOR3 normal;
    D3DCOLOR col;
    float x, y;
    static const DWORD FVF = D3DFVF_XYZ | D3DFVF_NORMAL | D3DFVF_DIFFUSE | D3DFVF_TEX1;
};
```

图 4-30 功能划分

粒子渲染函数

该成员函数将粒子绘制在三维空间中。

```
void Particle::Render()
{
    if (m_ParVector.size()>0)
    {
        m_pdevice->DrawPrimitiveUP(D3DPT_POINTLIST, m_ParVector.size(), &m_ParVector[0], sizeof(sParticle));
    }
}
```

图 4-31 功能划分

粒子初始化函数

该函数用于粒子的初始化。

```
void Particle::Reset(LPDIRECT3DDEVICE9 device, bool bSpriteEnable/*=true*/, bool bScaleEnable/*=true*/)
{
    m_pdevice = device;
    m_ParVector.clear();
    m_pdevice->SetRenderState(D3DRS_POINTSPRITEENABLE, bSpriteEnable);
    m_pdevice->SetRenderState(D3DRS_POINTSCALEENABLE, bScaleEnable);
    m_pdevice->SetRenderState(D3DRS_POINTSCALE_A, FToD(0));
    m_pdevice->SetRenderState(D3DRS_POINTSCALE_B, FToD(0));
    m_pdevice->SetRenderState(D3DRS_POINTSCALE_C, FToD(1));
}
```

图 4-32 功能划分

粒子更新函数

该函数用于粒子状态的更新。

```
void Particle::Render()
{
    if (m_ParVector.size()>0)
    {
        m_pdevice->DrawPrimitiveUP(D3DPT_POINTLIST, m_ParVector.size(), &m_ParVector[0], sizeof(sParticle));
    }
}
```

图 4-33 功能划分

第 5 章　系统测试

5.1　系统测试目的

在软件的生命周期的每个阶段都不可避免的引入新的错误。如果在软件投入运行之前没有，没有发现并纠正软件中的大部分错误，那么这些错误就会在软件运行中慢慢的暴露出来，那个时候不仅纠正错误的代价更高，而且往往会造成极为恶劣的影响。因此必须对软件进行测试。测试的目的就是为了在软件投入运行之前，尽可能多的发现其中的错误。

5.2　功能测试与分析

(1)控制类的测试和摄像机的测试

按键接受测试	测试键位	输入结果	测试结果	结果分析
1	A	A	正常	
2	W	W	正常	
3	S	S	正常	
4	D	D	正常	

摄像机测试名称	测试输入	输入结果	测试结果	结果分析
摄像机观察三维物体	运行程序	能观察到载入的三维地形和天空	正常	
摄像机的移动	W	向前移动	正常	
摄像机的移动	A	向左移动		
摄像机的移动	S	向后移动		
摄像机的移动	D	向右移动		
摄像机绕人物 Y 轴的旋转	鼠标 Y 轴移动	绕任务旋转	正常	
摄像机绕人物 X 轴的旋转	鼠标 X 轴移动	摄像机抛弃观察点，绕自身 X 轴旋转	不正常	由于摄像机 Y 位置被动态锁定在地面以上，导致摄像机不能绕人物 X 轴旋转

地形和天空盒载入测试和人物的载入测试

地形和天空盒的载入	测试输入	输入结果	测试结果	结果分析
地形的载入	改变地形的高度	地形高度在摄像机的观察中改变	正常	
天空盒的载入	载入天空盒	摄像机看到了三维天空	正常	
地形碰撞测试	鼠标控制观察方向朝地面，观察是否穿透地面	不能穿透地面	正常	
超过地面范围测试	移动人物到地形边缘，观察人物是否冲出地形	不能冲出地形	正常	

人物的载入	测试输入	输入结果	测试结果	结果分析
人物载入测试	运行程序，摄像机观察人物模型	摄像机观察到人物模型	正常	
人物的移动	W	摄像机和人物一起向前移动	正常	
人物的移动	A	摄像机和人物一起向左移动	正常	
人物的移动	S	摄像机和人物一起向后移动	正常	
人物的移动	D	摄像机和人物一起向右移动	正常	
人物的移动	X	没有粒子出现	点击 X 没有粒子出现	

5.3 测试结果分析

通过黑盒测试，几类错误如下:

(1)系统行为错误、基于规格说明的错误;

(2)基于规格说明的功能错误;

(3)面向用户的使用错误;

(4)基于规格说明的性能错误;

(5)黑盒接口错误。

通过对游戏模块各功能进行黑盒测试，基本满足了以下要求:

(1)在输入数据为指定测试方案中设计的输入数据时，系统的响应符合需求分析和系统性能要求。

(2)数据信息项符合需求说明，并能够正确输入默认值。

(3)能够对输入数据进行合理的检验，数据格式与测试方案相符。

(4)能够对数据信息描述所指定的数据相互关系输入响应。

综上所述，游戏系统运用基线测试方法，必需数据信息是不是空校验测试、数据项约束关系测试和数据格式有效性校验测试。

第6章　结论

在游戏的开发设计中，我主要完成了系统的总体设计和系统的详细设计。

(1) 系统总体设计包括：需求分析、模块总体设计；

(2) 系统详细设计包括：三维地形类的设计与实现、天空盒类的设计与实现、摄像机类设计与实现、碰撞检测类设计与实现、粒子类的设计与实现、文件加载类的设计与实现、控制检测类设计与实现。

本科毕业设计经历了4个月，在这四个月中，我能按照预期进度安排，按时按量的完成了毕业设计。在需求分析准备阶段我根据需要阅读了大量的D3D方面的书籍，其中特别关注了DIREECTX 9.0.3D游戏开发编程设计基础和逐梦旅程WINDOWS游戏编程之从零开始。在设计中期，利用所学的知识进行游戏的方案设计。在设计的后期，主要进行游戏的代码实现与测试。设计期间得到了指导老师和软件工程的同学的诸多指导和帮助，在这里，我想对他们表示感谢。毕业设计是大学四年对所学知识的综合运用，是理论到实践的过程。在毕业设计期间，我复习了很多以前老是课堂上讲过的专业知识，学习和体会到了许多编程上的重要思想。作为一个软件工程师，我深刻的理解到了在开发软件的过程中必须秉着严谨、好学、坚持的态度才能成功的做出一个像样的软件作品出来。在做毕业设计期间，我个人觉得四年的学习没有白费，同时感叹基础的重要性，在学习的过程中感觉知识和实际是存在差别的，在这种心理的支配下，我个人觉得我对知识的学习是有所松懈的，不过在这次毕业设计当中，我又重新拾起了以前的基础知识，同时我也深刻的感受到了基础对于提高来说是必不可少的。

毕业设计是我在大学四年中最为重要的阶段，我初步了解并掌握了3D游戏制作的相关知识，我相信在我以后的学习生活中，我会更加的加强这方面的知识储备，争取成为一名成功的软件工程师。

付出了努力后，最终完成了游戏设计的基本功能，但遗憾的是，有些功能模块由于时间关系无法实现，且模型的设计还不够美观，在游戏的性能上还有很多的不足。

因此，在今后的学习中，我还将会对这方面的知识进行深入的学习和研究。

致　谢

首先，本次设计的顺利进行需要感谢我的辅导老师和华才建指导老师老师。他们学识渊博、指导认真，在他们的悉心指导中，我不仅学习到了扎实的基础专业知识，也为我以后的发展奠定了良好的基础。在他们孜孜不倦的教导中我顺利的完成了我的毕业设计，在此，我对他们献上我衷心的感谢。

感谢四川理工学院计算机学院所有帮助过我的老师和同学，正是因为他们的帮助，我才吸收到了大量的知识，圆满的完成了我的学业，对他们我献上我由衷的感激。最后感谢同学对我的帮助和评委老师对此论文的审阅。

参考文献