w



**QG工作室项目报告**

学 院 物理与光电工程学院

专 业 光电信息科学与工程

班 级 2023级 1 班

组 别 图形组

姓 名 李文皓

学 号 3123006526

### 2024 年 4 月

### 目录

目录

[2024 年 4 月 1](#_Toc164896528)

[目录 2](#_Toc164896529)

[一．项目简介 3](#_Toc164896530)

[1.1 设计要求 3](#_Toc164896531)

[1.2 功能模块划分 3](#_Toc164896532)

[二．技术原理 6](#_Toc164896533)

[2.1 技术需求分析 6](#_Toc164896534)

[2.2 确定技术选型和外部库 6](#_Toc164896535)

[三．功能模块详细设计 7](#_Toc164896536)

[3.1 地形生成模块 7](#_Toc164896537)

[3.2 昼夜更替模块 7](#_Toc164896538)

[3.3 特效模块 7](#_Toc164896539)

[3.4 实体控制模块 7](#_Toc164896540)

[3.5 摄像机模块 7](#_Toc164896541)

[3.6 放置与破坏模块 7](#_Toc164896539)

[3.7 ImGui模块 7](#_Toc164896539)

[四．程序演示 14](#_Toc164896542)

[五．程序亮点与难点 20](#_Toc164896545)

[4.1 绘制优化 20](#_Toc164896539)

[4.1 相机与模型的变换 20](#_Toc164896539)

[4.1 地形生成 20](#_Toc164896539)

[4.1 特效的生成和管理 20](#_Toc164896539)

[六．程序不足之处 21](#_Toc164896549)

[4.1 优化 21](#_Toc164896539)

[4.1 区块的加载和卸载 21](#_Toc164896539)

[4.1 特效的生成和管理 21](#_Toc164896539)

[4.1 公告板技术 21](#_Toc164896539)

[4.1 物理效果 21](#_Toc164896539)

[4.1 渲染 21](#_Toc164896539)

[七．体会总结 22](#_Toc164896555)

# 一．项目简介

## 设计要求

**通过DirectX实现一个类我的世界游戏，实现地形场景生成和天空盒，第一人称和第三人称控制，昼夜更替，导入人物模型等功能。**

## 功能模块划分

**在具体的功能需求上分析的基础上，我将本软件的功能需求分为几大模块，如下图所示:**

#### 功能模块划分（大纲）

#### 

#### 具体的功能模块

##### 地形生成模块

* 1. 物块管理
  2. 区块设计
  3. 区块生成
  4. 区块绘制
  5. 地表物体生成

##### 昼夜更替模块

* 1. 天空盒
  2. 方向光

##### 特效模块

* 1. 雾效
  2. 淡入淡出
  3. 小地图

##### 实体控制模块

* 1. 玩家控制
  2. 敌人控制

##### 摄像机控制模块

* 1. 第一人称控制
  2. 第三人称控制
  3. 自由视角控制

##### 放置与破坏模块

* 1. 碰撞箱检测
  2. 射线检测

1. ImGui模块
   1. ImGui控制游戏参数
   2. ImGui控制特效选择

# 程序整体大致流程

# 二．技术原理

## 技术需求分析

**在技术原理方面， 由于游戏要追求流畅，因此需要使用实例化技术来高效的绘制网格模型相同的物体，从而减少因大量顶点和索引输入装配到渲染管线而造成大量不必要的开销。或是使用视锥体剔除技术来判断物体的碰撞箱是否在视锥体中，从而避免绘制可视范围外的物体，减少GPU的负担，但如此会增大CPU的负担，因此需要结合实际情况来使用，在我的项目里加入了视锥体剔除，但由于物块过多会给CPU造成过大的负担，因此默认关闭，可使用ImGui图形界面来打开。**

**模型方面使用了Assimp模型加载库来导入模型。还使用Render-To-Texture技术来将原本渲染后输出的纹理重新作为输入来实现全屏特效和小地图。地形方面还使用了柏林噪声算法来生成地表，该技术还可作用在X轴和Z轴上来生成洞穴。**

## 确定技术选型和外部库

* **编程语言：C++，HLSL Shader Model 5.0**
* **图形API：DirectX11**
* **图形界面库：ImGui图形界面库**
* **模型加载库：Assimp模型加载库**
* **项目框架：基于X\_Jun教程中的框架**
* **集成开发环境：Visual Studio 2022**

# 三．功能模块详细设计

## 地形生成模块

#### 方块设计

#### Block方块类中包含了方块的枚举Id以区分不同的方块种类；方块的数据使用了框架中的GameObject类，来存放方块的纹理、变换、网格等数据；设计了BlockModel类来专门从文件中获取方块的纹理，设置纹理模型和材质。Block方块类中还设计了相应的方法来获取和设置类中成员；

#### 区块设计

#### Chunk区块类中存放了区块的坐标；实例方块，所有的实例对象都是基于对应的实例方块绘制的；实例数据（InstancedData），用来存储每个实例对象的世界矩阵和对应的逆矩阵的转置，由此便可将实例对象放置在世界坐标中的对应位置；若开启了视锥体裁剪，还会将每个实例对象的碰撞箱变换到视锥体的局部坐标中，来判断该物体是否在可见范围内，进行进一步的筛选，但由于项目中方块的数量过于庞大，一个区块中就有近上万个方块，判断每个方块的碰撞箱是否和视锥体碰撞箱相交会给CPU带来巨大的负担，所以我设置了静态变量来控制所有区块的视锥体剔除；存放物块Id的三维数组，用于碰撞测试，使玩家不穿墙；世界种子，用于控制柏林噪声的生成。

#### 获取柏林噪声的方法将三个不同频率的柏林噪声进行叠加，返回合并后的噪声，该方法设为静态方法，可方便之后在地表生成随机物体；方块Id函数在不同的位置会返回不同的Id，在加载区块时会把每个方块的坐标传入以确定其类型；在绘制场景时会调用实例化绘制来绘制区块。

#### 

#### 地表物体生成

#### 这里使用了与区块生成相同的技术，就不过多赘述，创建物体时使用了柏林噪声和我设置的参数来确定地表位置。

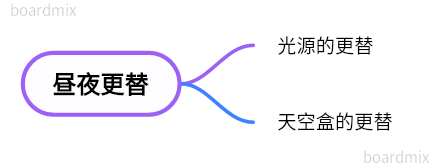
## 3.2昼夜更替模块

1**.光源的更替**

**这里我设置了四个方向光作为光源，并且将它们的漫反射系数设为可变参数，随着游戏时间的变化而减小或增大，以模拟日光的照射。**

2**.天空盒的更替**

**由于游戏开始时默认白日，所以使用了白天的天空盒，但天空盒并不受光源的影响，所以需要额外在天空盒的像素着色器中为采样的结果乘上一个系数，使着色器输出的像素相比原纹理像素更加暗淡，该系数由C++端输入并随游戏时间的变化而改变。**

**最后再调整一下两者变化的速度便可实现昼夜更替的效果。**

## 3.3特效模块

1**.雾效**

**雾效可以让远处的物体更加平滑的出现，且若是后面有时间实现区块的卸载和重**

**新加载，雾效可以让区块的加载显得不那么突兀，雾效的实现只需要设置雾效的颜色，并使用一个系数对雾效颜色和物体颜色进行线性插值即可，该系数由物体到摄像机的距离、雾效起点、雾效终点决定，也就是如下函数：**

s=(dist(**position**,**Eyes**)−fogStart​)/fogRange

**由此距离远的物体便会逐渐被雾效颜色替代。**

2**.全屏特效**

**玩游戏的时候看到在开始或退出游戏、或切换场景时会有淡入淡出的效果，这便**

**是使用了全屏特效，很多游戏也会自带小地图，这些效果都由Render-To-Texture技术实现，也就是不将后备缓冲区作为渲染目标，而是渲染到自己创建的2D纹理中，在之后的绘制中在选择性的将该纹理输出到屏幕，该技术也可以直接将渲染的效果进行输出，达到截屏的效果。**

2.1**淡入淡出**

**裁剪出一个矩形区域覆盖整个NDC空间，在该空间中为每个像素乘上一个在零到一之间变化的系数即可，该系数由C++端控制，开始游戏时由零到一，退出时由一到零。**

2.2**小地图**

**小地图只需要绘制一次即可，在游戏初始化时将相机位置设置在地形的上方，向下俯览获取全景图。之后每次绘制完场景后将小地图绘制在指定位置即可，不过要注意需要将雾效等特效关闭，不然会把雾效也绘制出来。**

## 3.4实体控制模块

## 玩家和敌人类都继承自实体类，实体拥有自身的模型和变换数据，还有实体自身的速度。实体的模型都使用了Assimp模型加载库进行导入，导入时我也遇到了点麻烦。

1.**玩家**

**玩家的控制与摄像机的控制绑定在一起，于下文详细介绍。**

2.**敌人**

**由于敌人要追踪玩家，所以需要设置一个3D向量来确定玩家与敌人的相对位置；敌人还有自身的血量，当玩家从鼠标发出的射线与敌人的模型相交时，对应的血量值减少；血量还有对应的血条，敌人血量不同时血条会切换对应的模型，来显示剩余血量，当血量为0是停止绘制敌人和血条，同时血条要时刻朝向玩家，只需对血条模型进行相应的变换即可，但也要保证血条时刻在敌人头顶，这里本来想使用公告板技术来达到这个效果，但没有成功。**

**敌人类的主要方法是寻找玩家的方法，该方法会计算出每一帧由敌人指向玩家的向量，再将敌人往该方向移动，同时血条也进行类似操作。**

# 

## 3.5摄像机控制模块

5.1**自由视角控制**

**自由视角这里沿用了框架中的自由视角控制，也就是通过键盘往四周移动，和鼠标右键旋转视野。**

5.2**第一人称控制**

**由于第一人称不能随意移动，所以这里使用了先前区块中加载好的3D数组，通过3D数组来判断人物所在位置是否有方块，来限制人物移动；还判断人物是否在地面上，来控制人物的下坠和起跳。这里将玩家的模型时刻设置在相机位置，且Y轴的旋转量与相机始终保持一致，来达到控制模型的目的**

5.3**第三人称控制**

**第三人称的人物控制与第一人称相同，只不过摄像机的准星始终朝向模型的头顶。**

## 3.6放置与破坏模块

6.1**放置方块**

**放置方块使用了碰撞箱检测，在按下按键和鼠标后会在对应位置生成一个方块模型，将该方块与周围的方块进行碰撞箱检测，若不相交则在绘制方块，否则取消放置，清除模型。**

6.2**破坏方块**

**破坏方块使用了射线检测，在鼠标处发出射线，并设置射线的范围，若与射线相交则销毁方块模型。**

## 3.7ImGui模块

6.1**控制游戏参数**

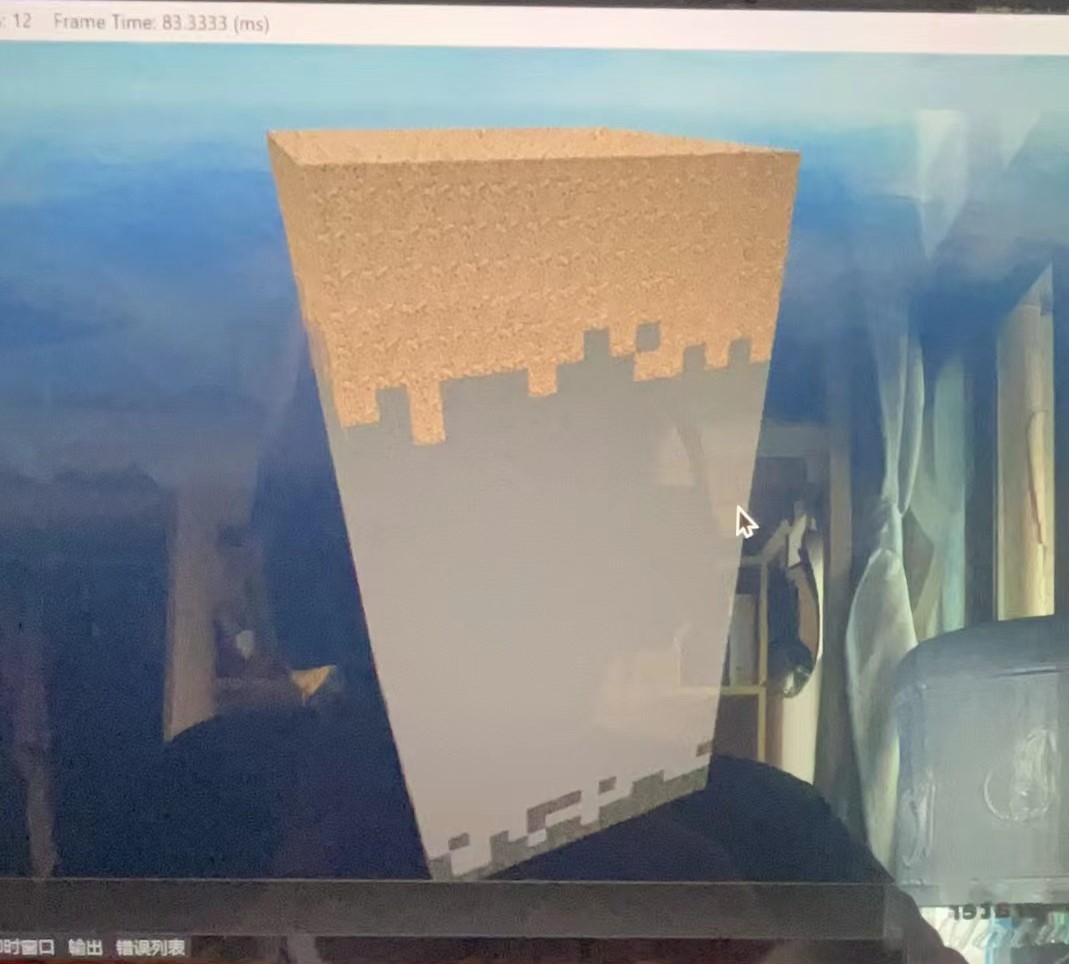
**玩家可通过ImGui控制游戏中的相机模式切换、玩家速度、第三人称观察距离等参数。**

6.2**控制特效选择**

**区块和地表物体的视锥体剔除可以使用ImGui选择性开启，雾效也可由玩家选择性开启，雾效的范围可以实时改变。**

# 四．程序演示

# 基础地形生成

**初次生成地形：**

**这便是未使用柏林噪声和实例化绘制生成的地形，可以看到仅仅生成16\*16\*32的区块帧数就掉到了12帧，这是因为所有物体的顶点都是作为顶点元素输入的，造成大量的输入装配操作，严重影响性能**

# 随后我的收到了学长给的建议，使用了硬件实例化技术进行绘制： 可以明显的看到游戏性能的到了巨大的提升，帧数直接涨到了两千多。 紧接着我又尝试使用视锥体剔除： 但帧数反而大幅下降，理由前面以进行阐明，这里不再赘述。 最后使用柏林噪声生成的地形如下图：

# 人物模型导入 导入模型并不是很顺利，一开始导入的时候人物模型的时候不知道什么原因人物的纹理并没有正常导入，导致整个模型都是白板。 随后看了学长推荐的文章后才知道OBJ模型文件都有对应的MTL文件，OBJ文件用来存储顶点索引等几何信息，MTL文件用来存储光照、透明度、折射等信息，包括使用的纹理是哪个，我使用的模型的MTL文件就是没有存放使用纹理的文件名。 最后更改后问题解决，效果如下： 相机控制 实现第一人称下模型与相机进行绑定花了挺多时间第三人称则沿用第一人称的移动方式，第三人称效果图如下：

# 添加雾效 夜晚和白天的效果如下：

# 小地图和随机生成树 效果图如下： 添加敌人

# 敌人的模型也从外部导入，也是遇到一样的问题不过事先就解决了。下面是演示敌人追踪和敌人与血条一直朝向玩家，当血量为零时敌人会被消除： 放置与破坏方块

# 这一方面我认为还不是很完善，因为我是想模拟我的世界放置方块时方块时附着在另一个方块上，效果如下：

# 五．程序亮点与难点

**1.绘制优化**

**区块绘制时使用了实例化绘制，极大减轻了GPU的负担，使帧数大幅提高，绘制地表物体时还使用了视锥体剔除，提前抛弃了无需绘制的顶点，减轻GPU的负担，但轻微加重了CPU的负担。**

**2.相机与模型的变换  
将第三人称的视角与第一人称的人物移动相结合，产生了类似我的世界第三人**

**称的效果。同时将玩家模型进行合适的变换，使玩家有操纵模型而不是相机的感觉，敌人追踪时的也要对模型进行相应变换，使其一直朝向玩家，寻找到合适的变换也算一个难点。**

**3.地形的生成**

**地形的生成使用了柏林噪声并对不同频率的噪声进行叠加，使其不是单一的平面，产生小山丘和洼地。**

**4.特效的生成和管理**

**由于我对渲染不是很熟悉，由此使用Shader产生不同的特效和对特效进**

**行管理我认为是一个难点。**

# 

# 

# 六．程序的不足之处

**1．优化**  
**优化方面还可以继续提升，一个区块的所有方块都可以只使用一个实例绘制，**

**只需在C++端确认一个表示符，在Shader中匹配对应的纹理即可。还可使用表面剔裁剪掉不与空气方块接触的方块，提高渲染效率。**

**2．区块加载与卸载**

**我的世界最大的特点无限大的世界可通过加载和卸载区块实现，在玩家附近的区块才需加载，这一点我还并没有实现。**

**3．公告板技术**

**在制作敌人血条时想过使用公告板技术，但最后并没有成功实现，该技术同样可以应用在优化方面，将远处的物体都换成贴图以减少顶点和索引的输入，提高效率。**

**4．物理效果**

**项目中的下坠起跳等效果并没有使用物理公式来模拟现实，同时人物与方块的碰撞没有做好，人物经常会穿墙**

**5．渲染**

**在项目中并没有使用什么渲染技术，人物的阴影等效果没有实现，虽然我的世界中的渲染也没有很精细。**

# 

# 七．体会总结

**跟着训练营学习的这段时间也是收获良多，从一开始的不知道计算机图形学和图形API是什么，到现在能写一些粗糙的东西出来也算小有进步。**

**在寒假的时候就跟着训练营学习，但当时并没有跟上进度，也就稍微学习了一点C++，图形学的部分只看了一点Games101。**

**开学后跟着训练营学习数据结构和图形学，也算勉强跟的上，但感觉还是有很多东西没有吃透。训练营的这几周算是入学以来最忙的几周了，平时周末和朋友打游戏的时间都只能砍掉来完成训练营的作业，虽然十分辛苦但自己的收获和进步都是颇多的。**

**学习这些知识都是我兴趣使然，在上学期学习C语言时便觉得挺有意思的，也尝试用C语言写一些小游戏，虽然都做的不怎么样。可以说我学习计算机相关知识大部分是源于对游戏热爱，特别是我的世界，在选题的时候我也是毫不犹豫的选择了做一个类MC游戏，在做的途中也是想往MC靠近，比如地形生成我就是很想做出像MC一样的地形，不过能力有限，暂时只能做到这种程度。**

**在实现项目的时候发现项目中要用到的知识很多都是前几周没学到的，都是在期间自学，且项目的框架我也没有非常熟悉，框架中很多封装好的类也都是前几周基本没见过的。最后也是磕磕绊绊把项目做了出来，在做的时候遇到自己满意的地方，比如生成的地形，也是马上截图给和我一样喜欢MC的朋友看，老有成就感了。**

**最后感谢一下这段时间一直为我解答问题、帮助我的学长，在我没有思路的时候给我指引。**