

**《软件工程》项目报告**

**题目： MyMC——自制体素沙盒游戏**

**课程名称： 软件工程**

**专业班级： CS2202**

**学 号： U202214330**

**姓 名： 赵亦安**

**同组成员： 何志鑫**

**陈薛嘉**

**唐锐**

**指导教师： 范晔斌**

**报告日期： 2024.11.3**

**计算机科学与技术学院**

**任 务 书**

**一 总体要求**

1. 综合运用软件工程的思想，协同完成一个软件项目的开发，掌软件工程相关的技术和方法；

2. 组成小组进行选题，通过调研完成项目的需求分析，并详细说明小组成员的分工、项目的时间管理等方面。

3. 根据需求分析进行总体设计、详细设计、编码与测试等。

**二 基本内容**

根据给出的题目任选一题，自行组队，设计与开发中软件过程必须包括：

**1. 问题概述、需求分析：**正确使用相关工具和方法说明所开发软件的问题定义和需求分析，比如NABCD模型，Microsoft Visio，StarUML等工具 (20%)；

**2. 原型系统设计、概要设计、详细设计**：主要说明所开发软件的架构、数据结构及主要算法设计，比如墨刀等工具（35%）；

**3. 编码与测试**：编码规范，运用码云等平台进行版本管理，设计测试计划和测试用例（30%）；

**4．功能创新**：与众不同、特别吸引用户的创新（10%）；

**5. 用户反馈**：包括用户的使用记录，照片，视频等（5%）。

**目 录**

**任务书** I

**1问题定义** 4

1.1项目背景及意义 4

1.2项目基本目标 5

1.3可行性分析 6

1.4人员管理和项目进度管理 7

2需求分析 8

2.1原型系统设计 8

3 概要设计和详细设计 9

3.1 系统结构 11

3.1.1功能说明 11

3.1.2接口设计 12

3.2类图等 14

3.3关键数据结构设计 14

3.4关键算法设计 15

3.5数据管理说明 15

4 实现与测试 16

4.1 实现环境和代码管理 17

4.2 关键函数说明 17

4.3 测试计划和测试用例 18

4.4结果分析 19

5总结 20

5.1 用户反馈 21

5.2 总结 22

6 体会 23

**附录**  24

**1 问题定义**

**1.1项目背景与意义**

**1.1.1 现实需求（Need）**

MineCraft，中文名我的世界，是世界上销量最广的单机游戏，没有之一。它所开创的体素沙盒游戏玩法赋予了游戏极高的自由度，以方块的形式表现世界，长久以来为无数玩家所沉迷。然而我们也发现了目前我的世界游戏的一些痛点。第一，基础版我的世界只提供了非常基础的渲染画面效果，没有阴影、透明水面、反射的设计；第二，我的世界可以自定义添加材质包，但是由于添加材质包的社区人员水平参差，高表现力效果的画面会严重拖慢游戏性能；第三，MineCraft要求使用我的世界必须使用启动器来打开，非常的不方便。

针对这些痛点，我们决定根据这些需求打造一款相似，但是弥补了不足的模仿产品：

1. 提供更好的基础渲染效果；
2. 提供更为灵活的的方块创建交互方式；
3. 与微软启动器解耦，实现方便跨平台；
4. 针对低端机型对于高性能渲染效果和运算效果的优化；

**1.1.2 意义（Benefit）**

实现以上功能可以满足以下用户的需求：

1. 用户无需自己寻找复杂的材质包，无需懂得shader编写就可以看到较好的渲染画面效果；
2. 用户可以在不同的机型上畅玩；
3. 用户可以更方便的搭建大型地形；
4. 低端机型用户可以更好地体验游戏。

**1.2 项目基本目标**

实现渲染、地图、UI等三大部分的模拟制作。发布Release版本并构建发布网页供大家测试。

具体要求如下：

1. 提供更好的基础渲染效果
   1. 水面渲染

使用屏幕空间光线追踪（SSR）制作屏幕空间反射效果。

* 1. 阴影效果

使用屏幕空间环境光遮蔽（SSAO）制作AO效果；使用ShadowMap结合PCF和PCSS算法生成软阴影；使用CSM生成级联阴影，加强近处阴影分辨率。

* 1. 真实感方面

使用PBR构建部分材质，使用IBL来计算环境光照；增加天空盒和雾效。

1. 提供更为灵活和方便的方块创建交互模式
   1. 创新性提供类似方块工厂模式，玩家可以自由创建一个大物体并保存为一个物体，直接放置（如自己搭建房屋后存为一个模块，之后可以直接在指定位置放置一个房屋）；
   2. 提供飞行、走路、奔跑、摄像机等多种模式。
2. 与Microsoft启动器解耦，实现跨平台
   1. 使用OpenGL&C++。
3. 针对低端机型对于高性能渲染效果和运算效果的优化
   1. DrawCall动态合批
   2. 实例化的使用
   3. 多Chunk实时卸载加载更新
   4. 多种可调节的剔除效果。如视锥剔除、遮挡剔除、距离剔除。

**1.3 可行性分析**

我们针对各部分目标分别分析可行性。

1. 渲染部分：
   1. 采用OpenGL和C++进行软件构建。OpenGL是跨平台的图形Api，简单易学。本人有完整的使用OpenGL和C++的渲染引擎开发经验和架构经验。
   2. 需要实现SSR,SSAO,PCF,PCSS,CSM等现代主流实时渲染算法。这些算法思想朴素，实现简单。本人拥有完整的实现算法经验。
   3. 需要实现多种多样的剔除效果。本人拥有完整实现算法经验。
2. 地图部分
   1. 需要使用随机生成地形、河流、树木等部分。经过调研，可以使用成熟的柏林噪声和种子技术进行多层噪声叠加以生成高度图来生成地形。河流可以通过模拟随机侵蚀算法生成。
   2. 地图数据结构。采用传统体素方格Chunk定制。
3. GamePlay部分
   1. 需要实现碰撞检测，人物行走，物块放置和挖去。成熟架构已经将GamePlay完全解耦剥离，非常容易实现。
4. UI部分
   1. 使用成熟的UI库-ImGUI。组员何志鑫拥有成熟的前端书写经验。
5. 网络联机部分
   1. 计划采用状态同步进行多人联机开发。组内对联机技术经验较少，将此设计为拓展内容。
6. 宣传与发布部分
   1. 博客搭建。本人已经注册域名Zzzzzya.cn，并使用hexo搭建博客。
   2. 发布安装包。计划使用Github的Release发布。

**1.4人员管理和项目进度管理**

按照组员分配任务如下：

* 赵亦安

1. 需求目标制定和任务分解
2. 程序总体架构设计
3. 开发环境搭建和教程书写
4. 光照与阴影实现
5. 水面反射SSR实现
6. 剔除优化实现

* 何志鑫

1. 甘特图制作实现
2. 游戏UI切换界面设计
3. UI基础交互实现
4. UI进阶美化实现
5. UI进阶定制实现

* 陈薛嘉

1. 项目原型设计实现
2. 随机地图分层噪声生成实现
3. 河流海洋模拟生成实现
4. 树木云朵随机生成实现
5. 地图存档与读取存档实现

* 唐锐

1. NABCD模型调研实现
2. 物体放置与破坏实现
3. 人物奔跑行走与碰撞检测实现

**2 需求分析**

**2.1 原型系统设计**

图形用户界面

描述已自动生成游戏方案并没有特别好的原型系统设计软件。这里直接使用ImGUI构建出我们想要的一些蓝图UI系统。

图 2.1 开始界面

电脑萤幕的截图

描述已自动生成开始游戏界面拥有三个按钮，分别代表开启新游戏，继续游戏，和加载游戏。

图2.2 设置界面

**3 概要设计和详细设计**

**3.1 系统结构**

图示

描述已自动生成根据我们的设计模块，总体架构应分为四大模块，分别是渲染模块，地图模块，GamePlay模块和UI模块。各个模块的通信应如图3.1所示。

图 3.1

各模块接口类如图3.2-图3.4所示

图形用户界面

描述已自动生成图形用户界面

描述已自动生成

图 3.3 地图模块接口设计

图3.2 UI模块接口设计

图形用户界面

描述已自动生成

图3.4 渲染模块接口

**3.2 类图介绍**

首先介绍最基本的轮子类，用于基础的渲染。

着色器类用于Shader的维护。Shader类提供构造函数直接从给定路径下读取并编译指定的顶点着色器和像素着色器，链接成程序后提供一个id为pro成员对象。Shader类提供一系列SetXXX方法用于glsl中的uniform变量的设置。

纹理类用于纹理的读取和维护。Texture基类提供构造函数直接从给定路径下读取并编译指定的纹理送入显存。Texture2D和CubeMap分别是两个特定的纹理子类。Texture2D负责管理2D纹理，CubeMap负责管理天空盒、Cube的六面纹理的同一保护。

FrameBuffer类用于帧缓冲的管理。帧缓冲提供渲染附件的获取方式和帧缓冲的id获取方式。帧缓冲的构造提供width和height以构造指定大小的帧缓冲。

三种类的UML图如图3.5所示。

图形用户界面, 文本, 应用程序

描述已自动生成接下来是整个程序的主体类图形用户界面, 文本

描述已自动生成Scene类。由于类成员和函数过多，这里只列出最主要的几个。

图3.6 Scene类

图 3.5 三基础类的UML图

Scene类构建整个程序的主体部分，一切全局参数全部在Scene中定义，包括但不限于程序窗口大小，玩家初始化，Chunk大小初始化，UI管理类App等。

图片包含 文本

描述已自动生成Scene有着最重要的方法，一个是InitScene负责窗口、OpenGL上下文等所有需要初始化的物体的初始化，另外一个是MainLoop()负责了整个渲染大循环。我们会在第4章中的具体函数介绍中详细介绍。

图3.7

接下来介绍最重要的数据结构类型，我们的地图结构类型。首先在OpenGL中顶点的inputLayout我们定义为Vertex的形式。分别是位置，纹理，面id，cubeId.其中面Id表示这个面是Cube中的哪个面，CubeID代表是哪个Cube。Mesh是各种类型的方块的基类，但是Mesh并不拥有Vertex，而是在Chunk需要在动态合批时去Mesh的子类中调用获取Vertex的vector的接口。基于Mesh类我们衍生出Cube类和CrossQuad类，其中Cube类代表所有方块，CrossQuad代表交叉四边形，如草地。

**3.3关键数据结构定义**

本工程最重要的数据流就是对于每一个物体的掌控。由3.2中的图3.7所示，整个场景由Mesh基类组织，在渲染线程中使用Chunk来组织。整个地图使用三维Vector存储，每一个对象是一个Mesh基类指针来维护。每一个Chunk固定维护16x200x16个Mesh的Vertex，在初始化时Chunk会遍历所管理的所有Mesh，根据坐标和Mesh存储的Exposed数组来生成所需顶点并保存。Exposed以一定顺序分别保存六个面是否可见的信息。

**3.4 关键算法设计**

本项目中采用了非常多经典的实时渲染算法。这里遴选经典PCF，CSM阐述。对于地图部分，选择柏林噪声生成地形描述。

* + 1. PCF

图片包含 文本

描述已自动生成图形用户界面

描述已自动生成PCF是对传统ShadowMap的改进。基本ShadowMap的设计为先以光源为视角相机渲染一遍场景，获取相机视角的场景深度；将光源矩阵保存并传入第二Pass中，在着色时手动计算着色点深度并与ShadowMap中保存的值对比，如果大于那么就认为在阴影中。PCF为了改进传统ShadowMap的走样问题，采用一个低通滤波对阴影采样进行滤波。PCF算法流程如图3.8所示。

图 3.9 CSM

图 3.8 PCF流程

* + 1. CSM

CSM是目前主流阴影实现架构，通过对于视锥的分层包围AABB盒的构建来充分利用每一级的ShadowMap的采样精度。首先每一帧需要获取到视锥的八点坐标，构建出视锥，按照Percent进行视锥分层，每个视锥进行AABB盒包围。包围后设定各级渲染包围数据进行渲染ShadowMap。

* + 1. 柏林噪声

地图PCG生成采用泊林噪声。由分层柏林噪声生成高度图，根据高度图生成地图。地图由最高处向下铺设10层dirt，然后是stone，最后是基岩。然后随机铺设草地和树木。以固定海平面高度铺设海洋。最后铺设云朵。这里流程图略。

**3.5 数据管理说明**

1. 内存和显存

绝大部分数据都在内存和显存中流动。

2. 磁盘

Chunk动态加载卸载时，使用SQLite对于磁盘进行流式通信。

**4 实现与测试**

**4.1实现环境与代码管理**

4.1.1 代码环境

同一采用C++17 + CMAKE + VCPKG + OpenGL3.0的环境。选择vscode + mingw作为工具链。采用Github作为代码托管平台。

4.1.2 代码签入记录

如图4-1所示。

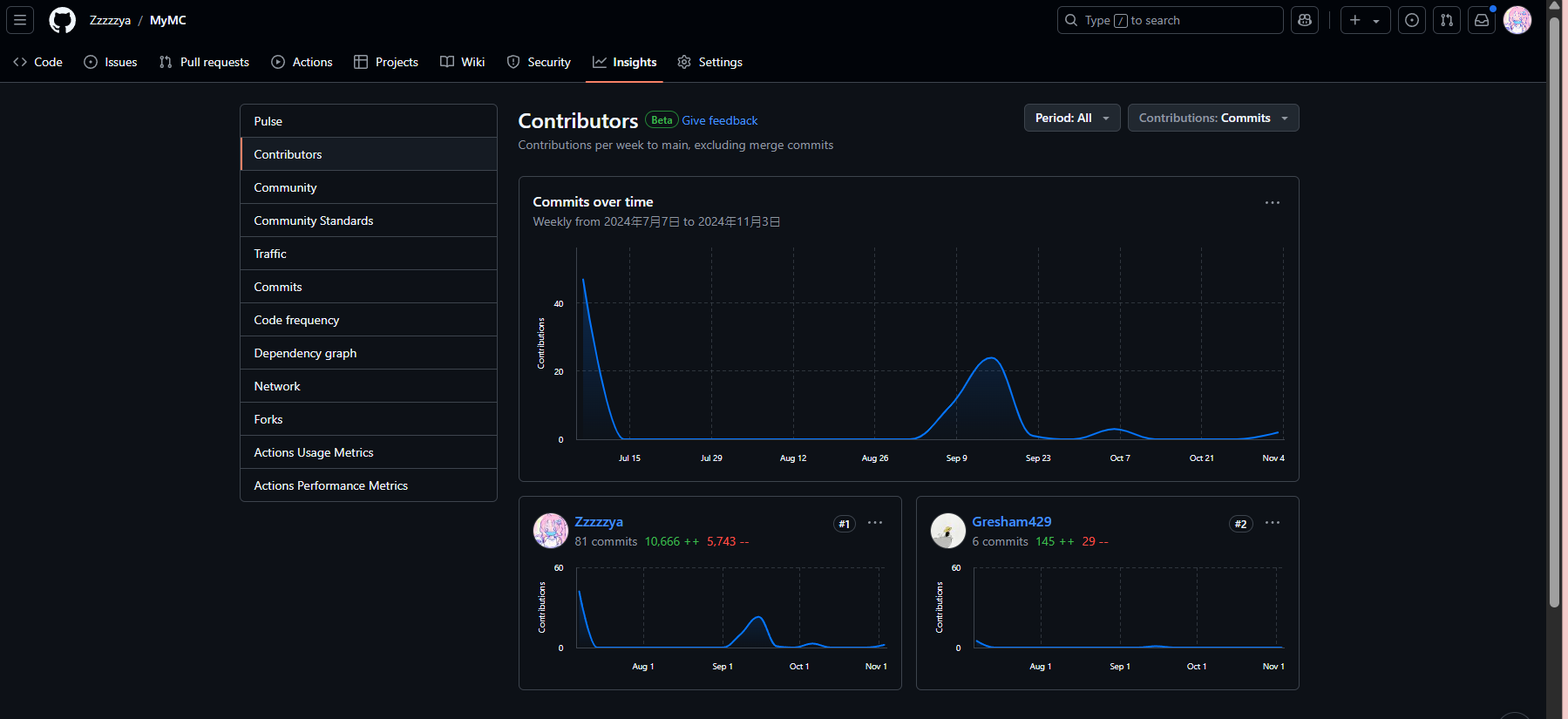


图4-1 Github签入

**4.2 关键函数说明**

这里说明主要Scene类中的两大函数InitScene和MainLoop。程序详见附录。

* + 1. InitScene()

InitScene()负责所有世界场景所需物品的初始化。从先到后分为初始化GLFW窗口，初始化OpenGL，初始化渲染模块，初始化ImGUI，初始化纹理，初始化Shader。每一个初始化进程都

* + 1. MainLoop()

MainLoop()负责整个场景的大循环。由前往后分别为更新系统时间，更新应用状态，根据状态选择渲染模式，处理windows事件，和处理GamePlay状态。详细MainLoop流程图见图4-2.

图4-2 MainLoop

**4.3 测试计划和测试用例**

4.3.2 测试计划

计划对以下模块进行测试。所有测试图依次见4.3.3中的测试结果。由于本项目是游戏项目，主要侧重画面效果，因此测试结果直接给出运行截图。

1. PCF阴影模块
2. SSR水面模块
3. GamePlay方块选择模块
4. GamePlay方块放置和挖除模块
5. UI界面切换模块

电脑游戏的截图

低可信度描述已自动生成4.3.3 测试结果

图4.4 水面倒影SSR

图4.3 PCF软阴影效果

F图4.4

图片包含 草, 蛋糕, 桌子, 游戏机

描述已自动生成电脑萤幕的截图

描述已自动生成

图4.6 Game玩家交互部分

图4.5 UI setting部分

**4.4 结果分析**

根据画面显示，完全符合设计预期。我们完成了原版MC未有的阴影效果和水面反射效果，并在此基础上做了低端机器优化配置。

**5 总结**

**5.1 用户反馈**

文本

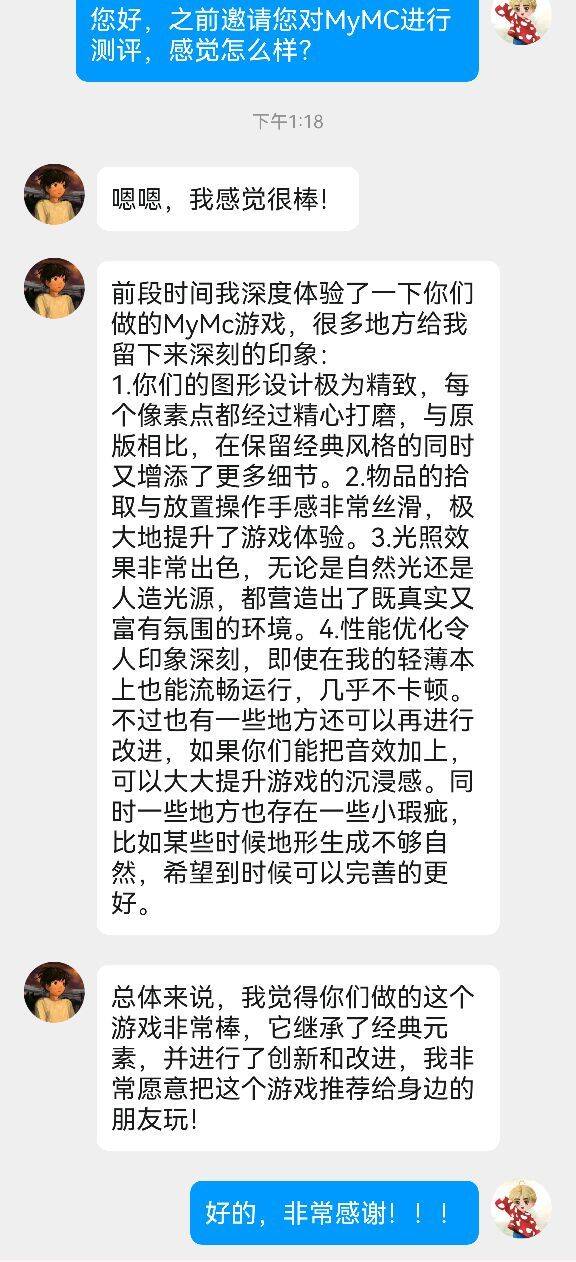
描述已自动生成我们搭建了软件发布网站，给很多同学发布并收集了测试。以下展现收集的部分反馈。

图 5-2

图 5-1

手机屏幕的截图

描述已自动生成

图5-3

**5.1 全文总结**

赵亦安：

（1）对项目总体设计、架构负责。统筹安排管理项目进行。

（2）完成总体代码架构，拆解任务下分。

（3）实现渲染部分代码，实现渲染算法。

陈薛嘉：

1. 实现地图部分分层噪声生成，河流海洋模拟生成。
2. 实现地图存档与读取存档。

何志鑫：

1. 负责UI设计
2. 负责页面切换，UI渲染实现
3. 负责文档整理

唐锐：

1. 负责GamePlay物体放置和破坏。
2. 负责人物奔跑与碰撞检测。

**6 体会**

本小组提前得知学期的软件工程项目，四人对于实现过的前后端项目感到厌倦，又由于组长拥有图形引擎开发的经验，遂打算构建一个游戏。游戏的开发囊括非常多的方面，渲染、物理、动画、UI、GamePlay，我们深知自己无法模拟所有的成熟引擎效果，因此遴选了渲染部分进行深入，其余部分进行简略涉及。由于要保证代码量，我们并未借助现有的商业引擎，而是模仿着成熟商业引擎O3DE的渲染管线，自己从0构建渲染引擎，并加入了物理、优化等部分。

引擎的开发设计难度我们有所预料，但是实践过程中很多的细节点是远超我们的想象的。我们计划了四种渲染主流算法的应用。虽然渲染算法的思想非常的朴素，但是工程细节过多。比如说SSR屏幕空间反射算法，完全只需要对于Shader进行一次自定义定制。但是，在此之前如何构成自定义配置模板Pass，Stencil的设置，最后的反射颜色与原颜色如何混合，都是十分令人头疼的问题。我们调研了很多技术文章资料，查阅了算法的原论文的过程，还翻阅了很多工程源码，最终终于做出了较为令人满意的效果。虽然不能说完美，但是我们胜在量多，每一个部分我们都保质保量地完成了90%的效果，我们认为这是非常不容易的。

团队协作是另一大难点。Git对于代码的管理非常的简略，但是工程量一大起来，分支、tag、Release的发布都是需要学习和统筹的事情。我们花费了一些时间进行初期的代码磨合和协作测试才得以继续。

总之，本学期的软件工程是非常令人难忘的项目，真正意义上使得我们的团队协作代码能力有了质的飞跃。在此鸣谢老师对于软件工程项目的指导，也在此感谢每一位成员对于此项目的付出。以上。

**附录**

附录呈现主要的三部分代码。

1. main.cc

#include "Scene.hpp"

#include "Log/Logger.hpp"

/\* 主世界 \*/

auto scene = make\_shared<Scene>();

/\* 鼠标绑定函数 \*/

void ProcessCursorPos(GLFWwindow \*window, double xpos, double ypos);

void ProcessScroll(GLFWwindow \*window, double xoffset, double yoffset);

int main(int argc, char \*\*argv) {

Loggers::init();

/\* 世界初始化准备 \*/

scene->InitScene(ProcessCursorPos, ProcessScroll);

/\* 世界大循环准备 - 包括游戏循环 & 渲染循环 \*/

scene->MainLoop();

scene->DestroyScene();

return 0;

}

/\* GLFW 鼠标绑定函数 \*/

void ProcessCursorPos(GLFWwindow \*window, double xpos, double ypos) {

if (!scene->cursorInWindow) {

return;

}

if (scene->firstMouse) {

scene->LastCursorX = static\_cast<float>(xpos);

scene->LastCursorY = static\_cast<float>(ypos);

scene->firstMouse = false;

}

float offsetx = static\_cast<float>(xpos - scene->LastCursorX);

float offsety = static\_cast<float>(scene->LastCursorY - ypos);

scene->LastCursorX = static\_cast<float>(xpos);

scene->LastCursorY = static\_cast<float>(ypos);

scene->player->camera.ProcessCursorPos(offsetx, offsety);

};

void ProcessScroll(GLFWwindow \*window, double xoffset, double yoffset) {

if (!scene->cursorInWindow)

return;

scene->player->camera.ProcessScroll(yoffset);

if (scene->app.state == App::State::RUN) {

static float off = 0.0f;

static float noff = 0.0f;

if (yoffset > 0) {

off += yoffset;

noff = 0.0f;

}

else {

noff += yoffset;

off = 0.0f;

}

if (off > 1.0f) {

scene->player->CurBlockID++;

scene->player->CurBlockID %= CB\_NUM;

off = 0.0f;

}

if (noff < -1.0f) {

scene->player->CurBlockID--;

scene->player->CurBlockID = (scene->player->CurBlockID + CB\_NUM) % CB\_NUM;

noff = 0.0f;

}

}

};

2.Scene.hpp

#pragma *once*

#ifndef SCENE\_HPP

#define SCENE\_HPP

#include "Header.hpp"

#include "Shader.hpp"

#include "mTexture.hpp"

#include "Mesh.hpp"

#include "Player.hpp"

#include "Map.hpp"

#include "Frustum.hpp"

#include "FrameBuffers.hpp"

#include <GLFW/glfw3.h>

#include "Application.hpp"

using Camera::BACKWARD;

using Camera::FORWARD;

using Camera::LEFT;

using Camera::RIGHT;

*/\* 场景类 \*/*

*/\* 世界类 一切参数在此设定 \*/*

class Scene : public std::enable\_shared\_from\_this<Scene> {

  public:

*/\* 视口大小 \*/*

    int **imageWidth** = 1600;

    int **imageHeight** = 900;

    int **display\_w** = 0, **display\_h** = 0;

*/\* 时间 \*/*

    float **deltaTime** = 0.0f;

    float **lastTime** = 0.0f;

    float **curTime** = 0.0f;

    float **fps** = 0.0f;

    bool **showFPS** = true;

*/\* 光标位置 \*/*

    float **LastCursorX** = 0;

    float **LastCursorY** = 0;

*/\* 鼠标是否第一次进入窗口 \*/*

    bool **firstMouse** = true;

*/\* 鼠标是否在窗口内 \*/*

    bool **cursorInWindow** = false;

*/\* 窗口 \*/*

    GLFWwindow \***window** = nullptr;

*/\* 玩家 \*/*

    shared\_ptr<Player> **player** = make\_shared<Player>(vec3(20.0f, 100.0f, -20.0f));

*// vec3 SunLightVec = vec3(1.0f, 0.0f, -1.0f);*

*/\* 地图 \*/*

    int **mapX** = 16 \* 25;

    int **mapY** = 100 \* 1;

    int **mapZ** = 16 \* 25;

    vec3 **ChunkSize** = vec3(16, 100, 16);

    int **CubeSize** = 2.0f;

    int **MAX\_CHUNK\_X** = 25;

    int **MAX\_CHUNK\_Y** = 1;

    int **MAX\_CHUNK\_Z** = 25;

    shared\_ptr<Map> **map**;

    vector<vector<vector<shared\_ptr<Chunk>>>> **Chunks**;

*/\* UI \*/*

    App **app** = App(this);

    bool **bBeginCreatingNewGame** = false;

    bool **bMapReady**{false};

    shared\_ptr<Texture> **spark**;

    unsigned int **sparkVAO**, **sparkVBO**;

*/\* GamePlay相关 \*/*

    float **viewRayTraceDistance** = 10.0f;

    float **viewRayTraceStep** = 0.1f;

    bool **SelectedAnyBlock** = false;

    vec3 **SelectedBlockToDo** = vec3(0.0f, 0.0f, 0.0f);

    vec3 **SelectedBlockToAdd** = vec3(0.0f, 0.0f, 0.0f);

    unsigned int **SelectedBlockVAO**, **SelectedBlockVBO**;

    vector<Vertex> **SelectedBlockVertices**;

    float **LastMouseCheckTime** = 0.0f;

    float **mouseCheckInterval** = 0.3f;

*/\* Render相关 \*/*

    mat4 **view**;

    mat4 **projection**;

    float **maxCullingDistance** = 300.0f;

    Frustum **frustum**;

    ScreenQuad **screenQuad**;

    ScreenMesh **screenMesh**;

    shared\_ptr<FrameBuffer> **ScreenBuffer**;

    shared\_ptr<FrameBufferOnlyRBO> **ScreenBufferForSSR**;

    shared\_ptr<FrameBufferDepthMap> **shadowMap**;

    SunChunk **sunChunk**;

    vec3 **SunPosition** = vec3(0.0f, 100.0f, 0.0f);

    vec3 **SunLightVec** = vec3(1.0f, -1.0f, -1.0f);

    float **shadowBias** = 0.088f;

    mat4 **lightMatrix**;

    bool **ssrOn** = false;

    SelectedBlockChunk **selectedBlockChunk**;

*/\* 可控类Setting \*/*

    bool **bVSync** = false;

    float **fogDensity** = 0.003f;

    Scene();

*/\*\**

*\* @brief 初始化场景 - 依次调度所有初始化函数*

*\**

*\* @param cursorPosCallback*

*\* @param scrollCallback*

*\*/*

    void InitScene(void (\*cursorPosCallback)(GLFWwindow \*, double, double) = nullptr,

                   void (\*scrollCallback)(GLFWwindow \*, double, double) = nullptr);

    void MainLoop();

    void DestroyScene();

  private:

    int InitWindow(void (\*cursorPosCallback)(GLFWwindow \*, double, double) = nullptr,

                   void (\*scrollCallback)(GLFWwindow \*, double, double) = nullptr);

    int InitRender();

    int InitShaders();

    int InitTextures();

    int InitMap();

    int InitGUI();

    void UpdateTimeAndFPS();

    void ProcessWindow();

    void CreatingNewGame();

    void MainRender();

*/\*\**

*\* @brief MainRender子函数集合*

*\**

*\*/*

    void SceneCulling(); *// 场景剔除*

    void UpdateVP();

    void ShadowMapDraw();

    void CubeShaderDraw();

    void WaterDraw();

    void WaterSSRDraw();

    void CloudDraw();

    void SkyDraw();

    void SunDraw();

    void SelectedBlockShaderDraw();

    void SparkShaderDraw();

    void PostProcessingDraw();

    void SelectedBlockPreviewDraw();

*/\*\**

*\* @brief 场景剔除子模块*

*\**

*\*/*

    bool DistanceCulling(Chunk &); *// 距离剔除*

    bool FrustumCulling(Chunk &); *// 视锥剔除*

    void ProcessKeyInput();

    void ProcessMovement();

};

#endif