Project report for the CG 100433 course

Project Title：

The Legend of Block

Team member

姓名	学号
李晓龙	1854179
屈宁	1851985
朱月莹	1851978
陈恭傲	1852629
秦子航	1851848
张宇航	1851979

Abstract

我们的项目是构造一个类似我的世界的开放式沙盒游戏。首先我的世界是一个大家基本上都了解熟悉的游戏，对里面的玩法内容都很兴趣，并且我的世界对图形学课学到的技术都有很好的使用。我们的项目主要构建一个开放式世界，可以通过键盘使用控制移动，并且通过鼠标来增加删除方块，可以根据自己的想法来搭建一个新世界。使用的技术有碰撞检测、simplex 噪声、遮挡剔除等等。

Motivation

我的世界是一个热门游戏，许多人都玩过，对游戏内容和规则都很熟悉。搭建一个类似我的世界的游戏需要对图形学的知识有一个很好的理解和综合运用，我的世界方块数量很多，对实时渲染的要求很高，并且增添删除方块涉及的坐标转换也需要坐标更好的理解，因此我们决定搭建一个类似我的世界的沙盒游戏。

The Goal of the project

1. 多种方块（包括玻璃、发光方块等等）
2. 随机生成地图（包括树，花、云等），包括昼夜交替，以及太阳的移动
3. 方块增添删除
4. 移动以及碰撞检验
5. 渲染优化

The Scope of the project

1. 项目不包括除玩家外其他生物实体的生成和移动等。
2. 不包括多维度世界和传送门等机制。
3. 实现的方块种类数量有限。

Involved CG techniques

1. 遮挡剔除

   图像进行渲染的时候，如果将所有方块的所有面全部渲染，将会导致帧率急剧下降，所以需要对渲染的内容进行删除，只显示位于视线内的方块的面，借助于提前 cpu 上实现的视锥体剪裁和对面渲染的判定。
2. 碰撞检测

   本项目中实现的碰撞检测利用了 minecraft 游戏物体几何形态简单、数据结构单一的特点得以实现。在接受键盘输入时，根据 wasd 键的情况计算当前移动的方向，得到摄像机的移动情况。由于方块的储存结构是三维数组，就可以通过下一个即将到达的区域有没有方块的判断检测碰撞。另外的，y 轴方向的加速度始终是一个负值，计算摄像机位置时，只要脚下没有地面就会掉落，也就实现了重力。
3. Simplex 噪声

   Simplex 噪声根据 seed 对顶点生成伪随机数和梯度，然后通过插值的方法得到更加平滑的随机数 ，通过随机数来确定地图上每个点的高度，相比于之前的随机数可以让地形更加平滑。

Project contents

1. 多种方块，包括草、玻璃、发光方块等等，能过够通过 Q、E 来切换放置方块的时候使用的方块种类。
2. 能够根据随机生成的 SEED 来随机生成地图，包括天空云彩和树的随机生成
3. 鼠标左键来删除方块，右键增加方块
4. 简单的物理世界判定，实现跳跃和在地形中移动时被阻拦等操作。
5. 键盘 WSAD 来控制前后左右移动，空格键控制跳跃
6. 简单的优化渲染

Implementation

Results

已实现：

- 采用了一个简单的视角判定，如果当前方块在视角的背面，不渲染
  实现代码：

```
if (a.x * camera.Front.x + a.y * camera.Front.y + a.z * camera.Front.z < 0)
{
    continue;
}
```

- 对于 16*16*16 的方块当成整体进行视锥判定，采用的原理是求出视锥体六个面的方程，然后判断大的方块的六个顶点，只要有一个在其中，就可以整体进行渲染
  实现代码：

```cpp
void ViewingFrustumPlanes(glm::vec4* result, const glm::mat4& vp) {
    //左侧
    result[0].x = vp[0][3] + vp[0][0];
    result[0].y = vp[1][3] + vp[1][0];
    result[0].z = vp[2][3] + vp[2][0];
    result[0].w = vp[3][3] + vp[3][0];
    //右侧
    result[1].x = vp[0][3] - vp[0][0];
    result[1].y = vp[1][3] - vp[1][0];
    result[1].z = vp[2][3] - vp[2][0];
    result[1].w = vp[3][3] - vp[3][0];
    //上侧
    result[2].x = vp[0][3] - vp[0][1];
    result[2].y = vp[1][3] - vp[1][1];
    result[2].z = vp[2][3] - vp[2][1];
    result[2].w = vp[3][3] - vp[3][1];
    //下侧
    result[3].x = vp[0][3] + vp[0][1];
    result[3].y = vp[1][3] + vp[1][1];
    result[3].z = vp[2][3] + vp[2][1];
    result[3].w = vp[3][3] + vp[3][1];
    //Near
    result[4].x = vp[0][3] + vp[0][2];
    result[4].y = vp[1][3] + vp[1][2];
    result[4].z = vp[2][3] + vp[2][2];
    result[4].w = vp[3][3] + vp[3][2];
    //Far
    result[5].x = vp[0][3] - vp[0][2];
    result[5].y = vp[1][3] - vp[1][2];
    result[5].z = vp[2][3] - vp[2][2];
    result[5].w = vp[3][3] - vp[3][2];
}
```

```cpp
bool Point2Plane(const glm::vec3& v, const glm::vec4* p) {
    for (int i = 0; i < 6; ++i)
    {
        if (p[i].x * v.x + p[i].y * v.y + p[i].z * v.z + p[i].w >= 0)
        {
            return true;
        }
    }
    return false;
}
```

```cpp
    {
        vertex[0] = glm::vec3((nowX + i) * MAX_X, k * 16, (nowZ + j) * MAX_Z);
        vertex[1] = glm::vec3((nowX + i + 1) * MAX_X - 1, k * 16, (nowZ + j) * MAX_Z);
        vertex[2] = glm::vec3((nowX + i) * MAX_X, (k + 1) * 16 - 1, (nowZ + j) * MAX_Z);
        vertex[3] = glm::vec3((nowX + i + 1) * MAX_X - 1, (k + 1) * 16 - 1, (nowZ + j) * MAX_Z);
        vertex[4] = glm::vec3((nowX + i) * MAX_X, k * 16, (nowZ + j + 1) * MAX_Z - 1);
        vertex[5] = glm::vec3((nowX + i + 1) * MAX_X - 1, k * 16, (nowZ + j + 1) * MAX_Z - 1);
        vertex[6] = glm::vec3((nowX + i) * MAX_X, (k + 1) * 16 - 1, (nowZ + j + 1) * MAX_Z - 1);
        vertex[7] = glm::vec3((nowX + i + 1) * MAX_X - 1, (k + 1) * 16 - 1, (nowZ + j + 1) * MAX_Z - 1);
        bool flag = false;
        for (int p = 0; p < 8; ++p)
        {
            if (Point2Plane(vertex[i], frustumPlanes))
            {
                flag = true;
            }
        }
        if(flag)
            Chunks[glm::vec2(nowX + i, nowZ + j)]->show(nowX + i, nowZ + j, shader, tex, camera, k);
    }
```

- 对六个面进行分别判断和渲染
  实现代码：

```
flag[0] = !(z - 1 > 0 && block[x][y][z - 1] != NULL && block[x][y][z - 1]->getName() != Block_Name::glass);
flag[1] = !(z + 1 < MAX_Z && block[x][y][z + 1] != NULL && block[x][y][z + 1]->getName() != Block_Name::glass);
flag[2] = !(x - 1 > 0 && block[x - 1][y][z] != NULL && block[x - 1][y][z]->getName() != Block_Name::glass);
flag[3] = !(x + 1 < MAX_X && block[x + 1][y][z] != NULL && block[x + 1][y][z]->getName() != Block_Name::glass);
flag[4] = !(y - 1 > 0 && block[x][y - 1][z] != NULL && block[x][y - 1][z]->getName() != Block_Name::glass);
flag[5] = !(y + 1 < MAX_Y && block[x][y + 1][z] != NULL && block[x][y + 1][z]->getName() != Block_Name::glass);

for (int i = 0; i < 6; ++i)
{
    if(flag[i])
        block[x][y][z]->show(show_pos, shader, tex, i);
}
```

可以提升的地方：

- 对于每一个 cube 采用的分别渲染调用 gldrawarrays，没有利用 GPU 的并发性，可以采用 batch render 进行批量渲染，这应该是我们和另一个小组渲染出来帧率很大差别的因素。
- 水的实现
- 光线的反射和阴影没有体现出来

Demonstrate your project

演示视频.mp4

Roles in group

张宇航：负责方块显示和光照效果、透明效果等的实现，以及方块、区块和世界储存逻辑和 render 方法的设计和实现。

李晓龙：主要通过编写 simplex 噪声，来实现随机生成高度更加平滑的地图，以及树、花、云等在地图长的随机显示 。

屈宁：负责帧率的优化，采用视锥体提前优化以及每个面分别优化渲染。帮助实现了人物飞行、方块增添等一些操作和简单的背包系统。上台讲述中期和最终 ppt。

朱月莹：主要负责实现对方块的更改种类的增加操作和简单的背包系统展示的实现和整合。

秦子航：负责方块的消除与生成、准星、手持方块的实现，协助设计光照效果和世界逻辑。

陈恭傲：实现碰撞检测机制和重力，对性能进行优化，研究、实现一个粗糙的视锥体裁剪。

References

[1] https://learnopengl-cn.github.io
[2] https://en.wikipedia.org/wiki/Simplex_noise
[3] https://blog.csdn.net/qq_31709249/article/details/80175119