Project report for CG 100433 course

Team member

小组成员


农烨


朱泽凯


刘奥


胥军杰


刘毅


王奕栋

教师和助教


赵君峤


张海

Project Title

Portal .5

Abstract

我们希望参照游戏《传送门》，制作一个可交互的传送门小程序，有基本的物理引擎（碰撞检测、物理定律、移动），可以放置传送门，可以通过传送门。用到的技术有虚拟相机、深度、模板检测、斜视锥体等。

Motivation

Portal 系列由 Valve 公司制作，著名的 Counter Strike（CS）就起源于他家的作品半条命。2011 年发布的《传送门 2》，拿下了 E3 的三个主要奖项“最佳电脑游戏”、“最佳 XBOX360 游戏”和“最佳 PS3 游戏”。是一个深受玩家喜爱的游戏系列.

其创新玩法和剧情深度令人惊叹。

在玩法中包含的 CG 技术也与课程比较贴合。

The Goal of the project

实现一个弱小版的 Portal，有基本的物理引擎（碰撞检测、物理定律、移动），可以放置传送门，可以通过传送门。

The Scope of the project

实现一个基本的物理引擎、完成传送门的机制实现。

Involved CG techniques

Transform the portal camera

Oblique projection

虚拟相机、深度、模板检测

斜视锥体（当人穿过传送门时，不要渲染到墙后面的内容）

递归渲染、优化处理

各种渲染技术——光照模型、透明渲染、阴影等。

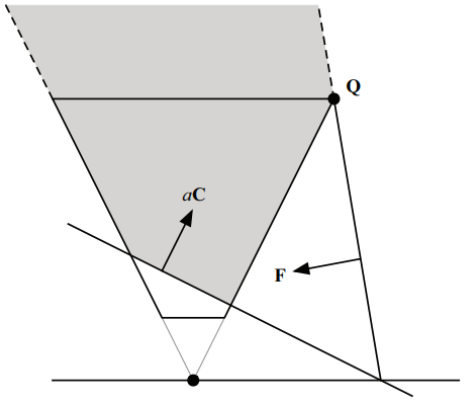
一些图片后效技术。

斜视锥：

原理：通过修改投影矩阵第三行的内容，将近平面设置为一个指定平面。

Table 1. Clip-space and camera-space view frustum planes. The matrix **M** represents the projection matrix that transforms points from camera space to clip space. The notation $\mathbf{M}_i$ represents the i-th row of the matrix **M**.

Frustum Plane	Clip-space Coordinates	Camera-space Coordinates
Near	$\langle 0,0,1,1\rangle$	$\mathbf{M}_4+\mathbf{M}_3$
Far	$\langle 0,0,-1,1\rangle$	$\mathbf{M}_4-\mathbf{M}_3$
Left	$\langle 1,0,0,1\rangle$	$\mathbf{M}_4+\mathbf{M}_1$
Right	$\langle -1,0,0,1\rangle$	$\mathbf{M}_4-\mathbf{M}_1$
Bottom	$\langle 0,1,0,1\rangle$	$\mathbf{M}_4+\mathbf{M}_2$
Top	$\langle 0,-1,0,1\rangle$	$\mathbf{M}_4-\mathbf{M}_2$



$$\mathbf{M}=\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix},$$

$$\mathbf{M}'_3=\frac{2}{1-d/f}\langle 0,0,-1,-d\rangle+\langle 0,0,1,0\rangle$$
$$=\left\langle 0,0,-\frac{f+d}{f-d},-\frac{2fd}{f-d}\right\rangle.$$

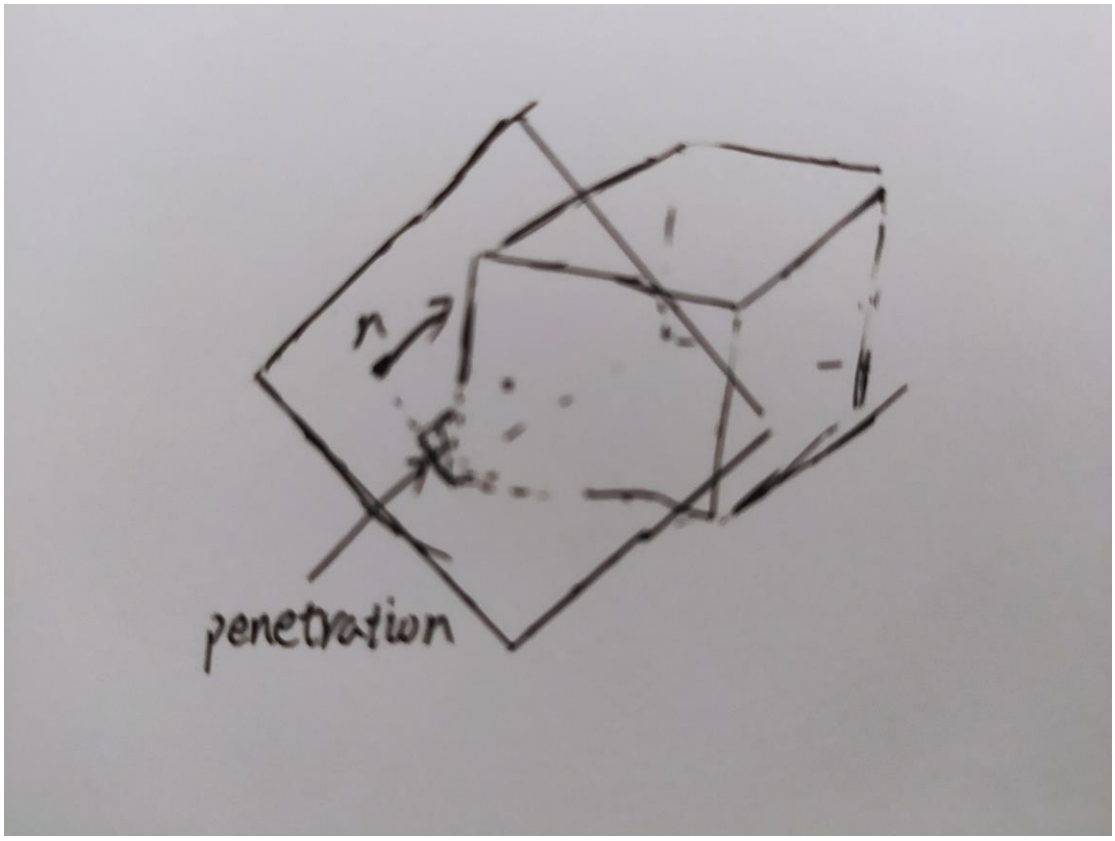
递归渲染：

原理：利用模板，不同深度的传送门用不同的模板数值。每一层都是先画一次模板，计算下一层的 view、projection，再开深度检测画其他场景，最后再画一次传送门模型填充深度与模板值。

碰撞检测：

原理：计算两个碰撞单元之间的相交深度判断是否存在相交，并根据相交深度的大小决定发生碰撞时先进行位置偏移量。

例如：以平面和立方体碰撞为例，对立方体的 8 个顶点投影到平面的法线方向，根据落点据平面的距离（含正负，即落在平面哪一边）判断是否相交与相交深度。



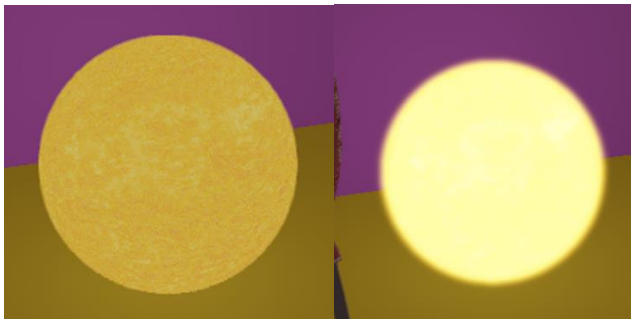
屏幕后处理部分：

1. HDR(High Dynamic Range，高动态范围)

在高动态范围中完成渲染，允许出现>1.0 的亮度，在后期处理时使用基于曝光(Exposure)参数的色调映射回低动态范围([0, 1.0] 区间内)，保留更多明暗信息。

2. Bloom(泛光)

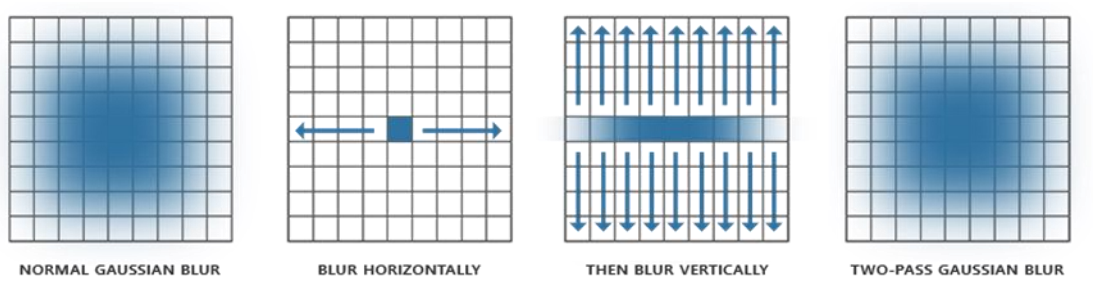
在 HDR 技术的基础上，在主渲染阶段提取渲染结果中高亮度的部分(使用了 HDR 后，可以将高亮度范围设置为>1.0f)，经过多次两步高斯模糊后，在后期处理阶段中与原渲染结果混合。可以通过调整高斯权重，来获得不一样的泛光效果。



泛光的基本要点只有两个，高斯模糊和混合模糊结果到原渲染结果中。

要实现高斯模糊过滤，我们需要一个二维 n*n 四方形作为权重，从这个二维高斯曲线方程中去获取它。但是，二维高斯模糊需要 n*n 次采样，效率很低。幸运的是，高斯模糊允许我们把二维方程分解为两个更小的方程：一个描述水平权重，另一个描述垂直权重。然后先水平

模糊，再垂直模糊，这样只需要 2*n 此采样。这叫做两步高斯模糊。



如此，对一个图片进行两步高斯模糊，我们最好需要两个缓冲对象。先在场景纹理的第一个缓冲中进行模糊，然后在把第一个帧缓冲的颜色缓冲放进第二个帧缓冲进行模糊，接着，将第二个帧缓冲的颜色缓冲放进第一个，循环往复。像一对乒乓球，称之为 pingpongBuffer / pingpongFBO。这样我们可以对任意图像进行任意次模糊处理；高斯模糊循环次数越多，模糊的强度越大。

3. FXAA(快速近似抗锯齿)

通过 NVIDIA 提供的公式工作。原理是在渲染完成后的屏幕图像中，通过像素颜色检测边缘，并由此完成不同程度的模糊完成抗锯齿。

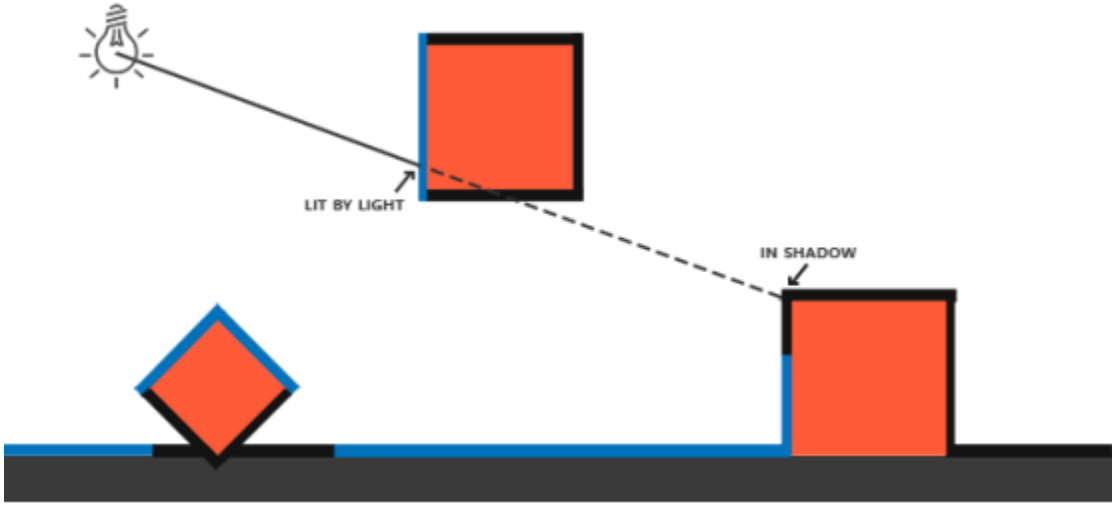
FXAA 主要思路是认为每个锯齿都是由连续的水平或竖直的锯齿边缘组成的，可以通过着色片段在锯齿边缘中的位置，近似求出其落在原始边界内的比例。

阴影渲染：

1. 阴影贴图

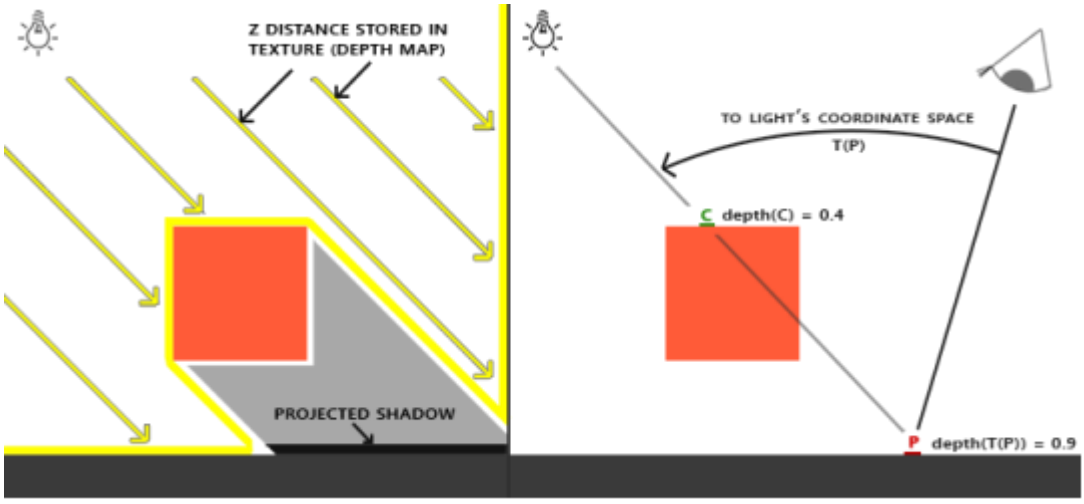
从计算机图形学发展以来，发展出多种算法来渲染出阴影。这些方法包括：ground shadow（地面阴影）、shadow texture（阴影贴图）、shadow map（阴影映射）、shadow volume（阴影体）以及光线追踪的方法。

目前这些方法中，ground shadow、shadow texture 都已经是过时的方法，且我们的程序是通过光栅化来完成渲染，所以光线追踪算法也不在考虑范围内。考虑 shadow map 和 shadow volume，前者计算速度比后者快很多，实现起来前者简单，且现在能考察到的资料中按前者实现阴影渲染的要远多于后者，因此便于查找相关资料。综上原因，选择用阴影映射（shadow map）的方法来渲染阴影。



我们考察了 learnopengl 等资料中关于阴影映射原理的介绍，下面就先介绍阴影映射算法的工作原理。

总体上，渲染思路是：以光的视角进行一次渲染，光能看到的东西都能被点亮，光看不到的地方都点不亮。如上图所示，光线照到第一个箱子上，箱子上相关区域被点亮，后面箱子上被遮挡的区域，光看不到，就在阴影中。具体而言，我们希望得到光线第一次击中的那个物体，然后用这个最近点和光线上其他点进行对比。如果光线上某个点在这个最近点后面（也就是相对于光源的深度大于最近点相对于光源的深度），那么这个点就在阴影中；反之，则被光照到。然后，如果对光源射出的所有光线上的点全部做这个判断，那性能消耗就不能接受。阴影映射算法的做法就是用一个深度纹理贴图（或者叫阴影贴图）将从光源的透视图来渲染场景的最近点的深度保存在这个贴图中。



如图就是阴影贴图原理图。左侧的图显示了一个定向光源在立方体下的表面投射的阴影。通过储存到深度贴图中的深度值，我们在进行像素渲染的时候就能找到每个像素位置对应的深度贴图中的最近点到光源距离。在像素着色器渲染每个像素的时候，根据该最近点到光源的距离和像素（这里的意思是该像素对应的物体）到光源的距离用以决定像素是否在阴影中。

2. 阴影贴图的改进

原始的阴影映射会造成一些失真。比如毛刺现象 Shadow Acne，例如地面上渲染出很多交替的黑线。因为阴影贴图受限于分辨率，在距离光源时，多个片段可能从深度贴图的同一个值中去采样。当光源以一个角度朝向表面时，这种情况下深度贴图也是从一个角度下进行渲染的。多个片段就会从同一个斜坡的深度纹理像素中采样，有些在地板上面，有些在地板下面，

就出现了毛刺现象。
为了解决这种问题，采用阴影偏移（shadow bias）的方法，即对深度贴图中的深度应用一个偏移量，这样片段就不会被认为在表面之下了。
这样渲染出来的阴影是硬阴影，显得很死板，为了使阴影显得更加柔和，采用了 PCF（percentage-closer filtering）技术。它的实现思想非常简单，就是在像素渲染时，本来只在深度贴图中一个对应位置采样，采用 PCF 技术后则在对应位置周围一个局部窗口内采用，分别独立计算是否发生阴影，最后将计算结果合起来做平均，就可以得到柔和的阴影。
当然生成阴影中也有一些小问题，这里就简单介绍如下。
有时会出现一种阴影失真叫做悬浮(Peter Panning)，即阴影出现了夸张的偏移值，显得影子浮在物体上。解决办法就是在渲染阴影的时候设定正面剔除（front face culling）。
有时会出现一种采样过多的问题。现象上就是超出光的视锥体的区域全部被视为是在阴影中（即被认为光看不到），而不管那些区域是不是真的在阴影中。出现这种问题的原因就是超过光视椎体的投影坐标比 1.0 大，在深度贴图上的坐标超过 0 到 1 的范围。一种简单的方法就是把超过深度贴图范围的采样点采样到的值全部设为 1.0，这样它们就都不在阴影中了。同时，当一个点投影坐标的 z 坐标大于 1.0（比光的远平面还远），就强制把该位置设定为不在阴影中。

Project contents

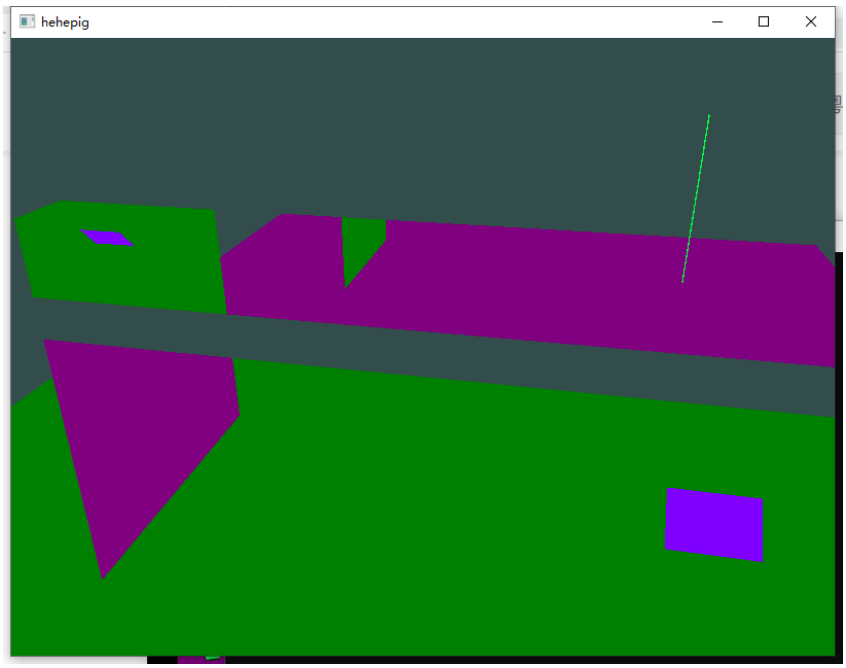
实现一个弱小版的 Portal，有基本的物理引擎（碰撞检测、重力加速度、移动），可以放置传送门，可以通过传送门。

Implementation plan

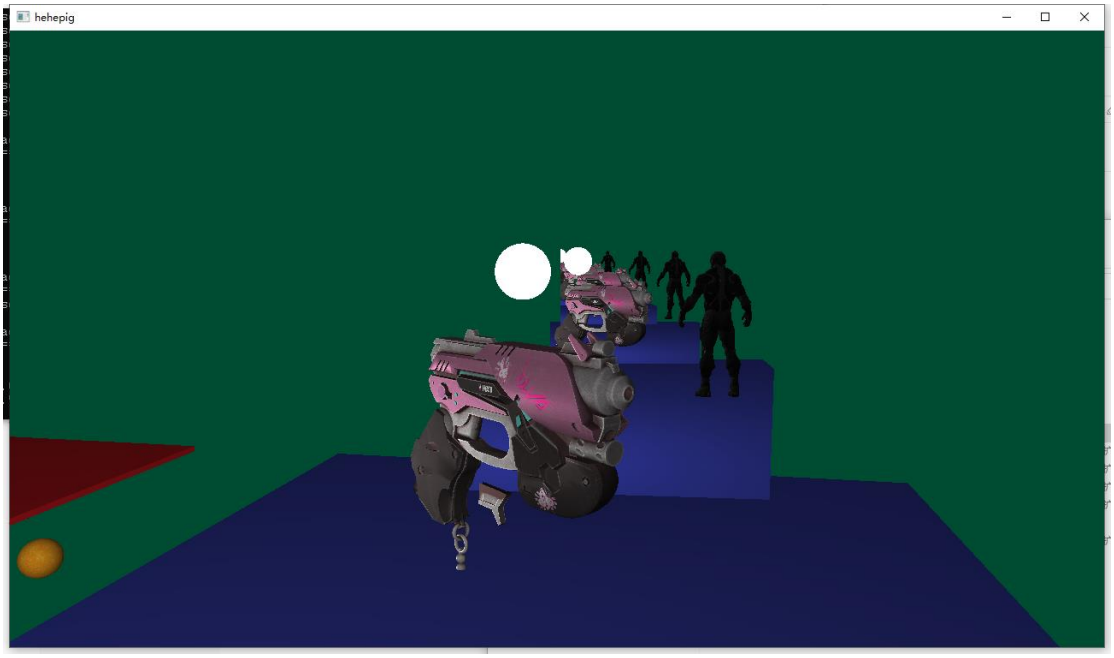
Week9	试着做了个传送门小 demo
Week 10-11	渲染：试验性地实现传送门效果、学习光照模型、阴影渲染、框架确定
	物理：学习物理引擎，在框架中确定好接口
Week 12-13	渲染：进一步细化各个渲染部分
	物理：实现物理引擎
Week 14-15	整合

Results

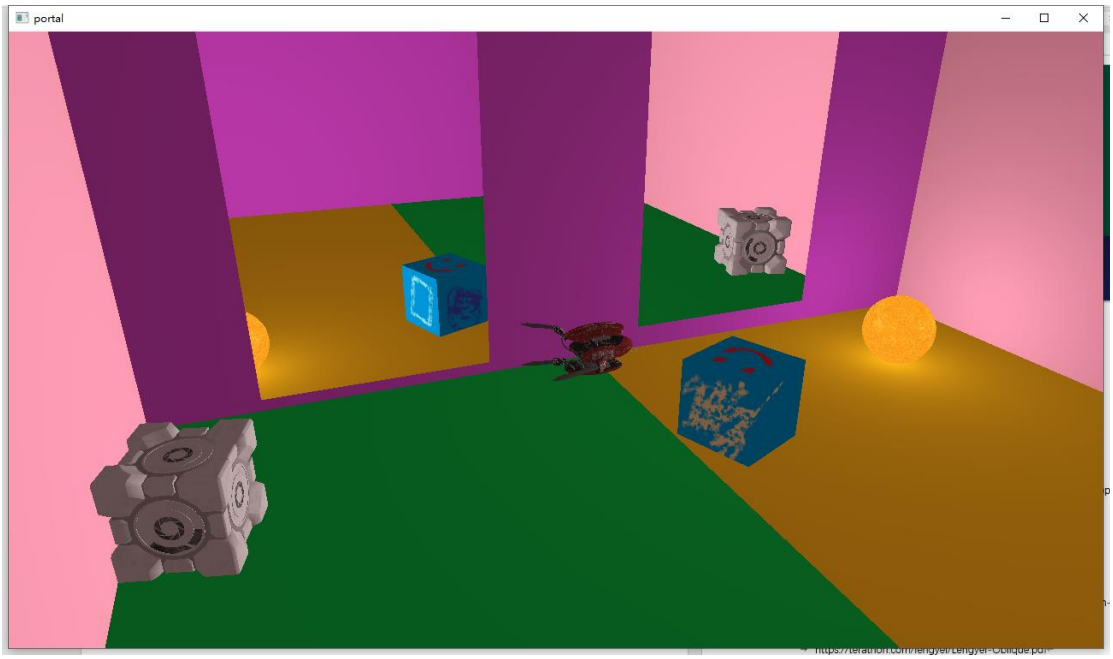
2022/11/15，实现色块验证的单层传送门渲染



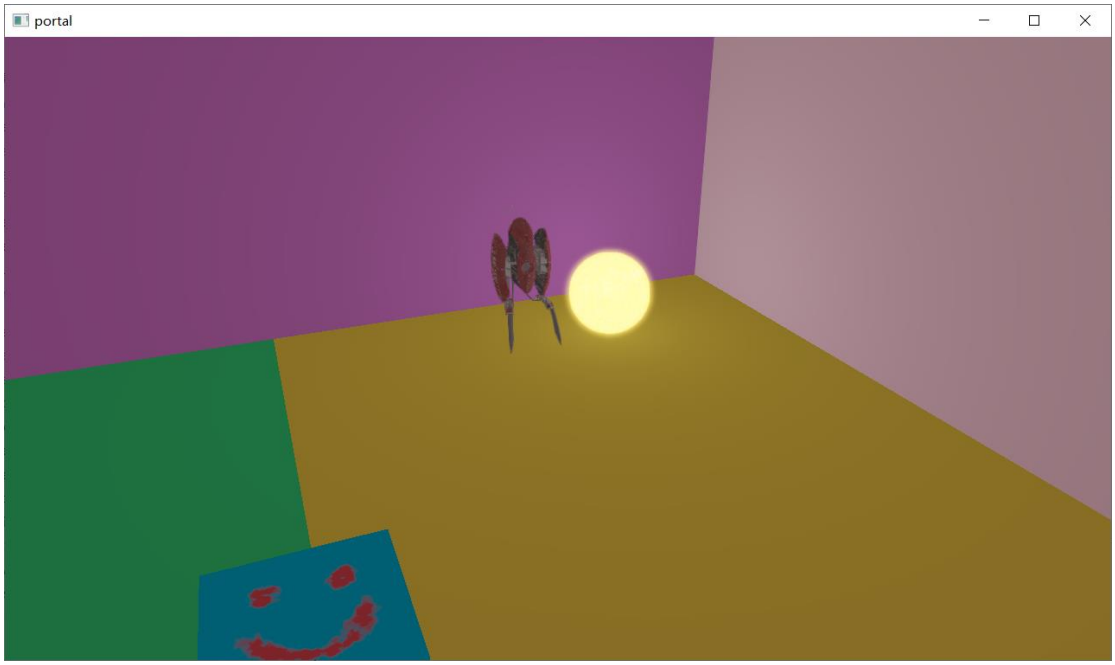
2022/11/20、实现导入模型、递归渲染传送门、斜视锥裁剪



2022/12/10、与物理引擎部分合并，实现鼠标按键交互、添加配置文件读取场景信息



后效



Roles in group

朱泽凯	传送门机制、裁剪矩阵、合并代码
胥军杰	物理引擎、合并代码
刘毅	阴影
农烨	后效
刘奥	配置文件
王奕栋	展示

百分比

朱泽凯	胥军杰	刘毅	农烨	刘奥	王奕栋
100%	100%	100%	100%	100%	100%

References

Coding Adventure: Portals

https://www.youtube.com/watch?v=cWpFZbjtSQg

Portal Problems - Lecture 11 - CS50's Introduction to Game Development 2018

https://www.youtube.com/watch?v=ivyseNMVt-4&t=3642s

How were the portals in Portal created? | Bitwise

https://www.youtube.com/watch?v=_SmPR5mvH7w

Portal Rendering with Offscreen Render Targets

http://tomhulton.blogspot.com/2015/08/portal-rendering-with-offscreen-render.html

Oblique View Frustum Depth Projection and Clipping

https://terathon.com/lengyel/Lengyel-Oblique.pdf

物理引擎学习

《游戏物理引擎开发》

光照模型

learnopengl

OpenGL/fxaa.frag.glslat master · McNopper/OpenGL (github.com)

HDR- LearnOpenGL CN (learnopengl-cn.github.io)

泛光 - LearnOpenGL CN (learnopengl-cn.github.io)