编程环境

硬件

CPU: AMD R9-7945HX
内存:16G

软件

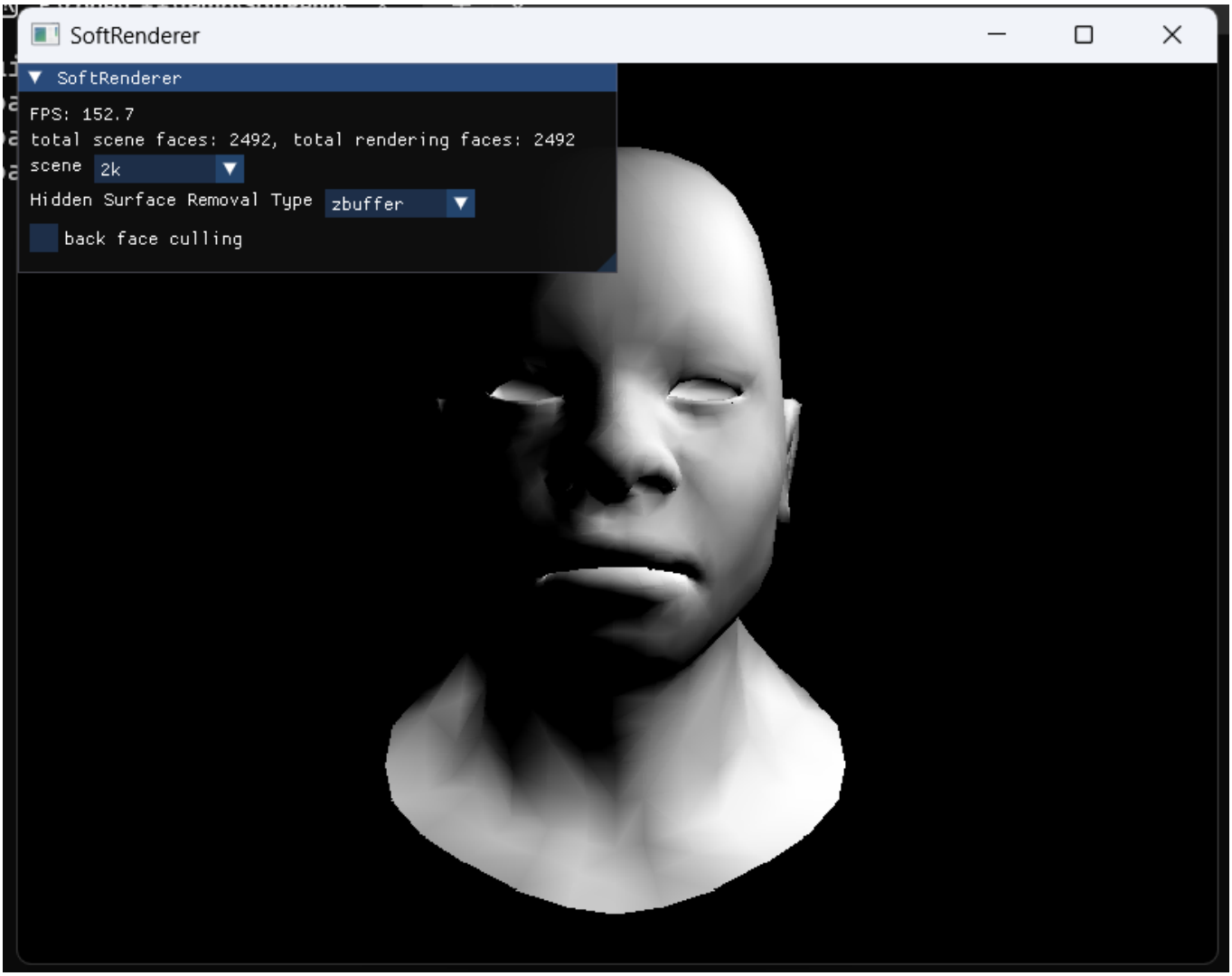
win11 + cmake + vscode + mingw

用户界面使用说明

编译方式

```
mkdir build
cd build
cmake .. -G "MinGW Makefiles"
make
```

界面说明



本项目实现的是层次zbuffer，本文一种提供了五个场景，可以通过scene下拉框进行切换，其中“2k"、”15k“、”144k“分别代表了相应面片数量的场景，而scene4和scene5是为了体现bvh-hizbuffer效果而绘制的遮挡场景（详细情况参考实验结果报告）。

本项目实现了zbuffer、hizbuffer、bvh加速的hizbuffer以及scanline zbuffer，四种效果可以在左上角窗口的Hidden Surface Removal Type选项进行切换。

左上角窗口的第一行为渲染帧率，为了体现层次zbuffer的快速拒绝效果，第二行显示了场景中的面片以及实际进行光栅化的面片数量。除此之外，第五行为背面剔除开关。**背面剔除只针对zbuffer、hizbuffer、bvh-hizbuffer有效。由于本项目没有实现视锥剔除，**

对于超出视锥范围而应被剔除的三角形此处也被计入了实际渲染的面片数量。

数据结构

本项目实现了简单的渲染管线，下面简单介绍一下各个类的作用。

首先说明使用到的第三方库：

- imgui + glfw：用于图形界面于交互
- eigen:：矩阵库，用于向量与矩阵运算
- assimp：用于读取obj文件
- stb_image：用于读取图片（实现了纹理功能但场景中未使用纹理）

核心类功能：

Triangle、TriangleMesh、Model、Scene类：用于表示读取的obj文件，其中TriangleMesh为若干三角形面片的集合，一个Model类里面包含若干的TriangleMesh，Triangle里面存有TriangleMesh的指针以及对应顶点在TriangleMesh中的索引。Scene由若干个Model构成。

Shader、Renderer、FrameBuffer、Window类：Shader参考opengl实现了顶点着色器和偏远着色器功能，每个Model有自己独立的shader。FrameBuffer里面包含了帧缓冲以及深度缓冲。Renderer用于渲染场景，渲染后的结果存储在FrameBuffer中。Window类将FrameBuffer中的内容展示到图形界面上。

ZBuffer、HiZBuffer、BVH、ScanlineZBuffer类：四种不同形式的zbuffer、其中FrameBuffer中包含ZBuffer、HiZBuffer、ScanlineZBuffer的指针，而BVH存储在Renderer类中，当Renderer渲染新的场景时会构建该场景的BVH。

加速

本项目主要使用了两个加速方法：

1. 使用了背面剔除，在ndc空间进行背面剔除，可以有效的减少光栅化的面片数量，往往能实现不错的帧率提升。
2. 在每个场景中构建完BVH后，由于在实际渲染时只需要渲染叶节点（每个叶节点包含一个三角形面片），于是我提前将叶节点存储起来，省去了遍历整颗二叉树的步骤。