# **基于Cesium的地形可视域分析实现与应用研究**

**摘要**

地形可视域分析是地理信息系统（GIS）中的一项重要空间分析功能，广泛应用于军事侦察、通信选址、城市规划等领域。本文基于Cesium平台，提出一种实时、高效的地形可视域分析方法。通过结合射线投射与地形采样技术，实现在三维地形环境中从任意观察点出发的可视区域识别与可视效果渲染。实验结果表明，该方法具备良好的实时性与可视化表现力，适用于各类可视化场景需求。

**关键词**：Cesium；地形分析；可视域分析；WebGL；三维可视化

## **1 引言**

地形可视域分析（Viewshed Analysis）指从某一地理位置出发，识别在给定地形条件下可视范围的过程。传统的可视域分析多依赖于二维栅格分析方法，但无法准确反映地形遮挡关系。随着三维GIS技术的发展，基于WebGL的可视化平台如Cesium提供了更高效、交互性更强的分析手段。

Cesium是一个开源的Web端三维地球可视化平台，支持全球地形数据、空间数据可视化与交互操作。本文利用Cesium平台，提出一种结合射线投射与地形采样的可视域分析方法，并实现其可视化表达。

## **2 相关工作**

传统GIS平台（如ArcGIS）多使用视线扫描或DEM栅格分析方式进行可视域计算，存在分辨率受限、计算效率低的问题。近年来，随着WebGL和三维可视化技术的发展，基于Cesium、Potree、Three.js等平台的三维可视域分析逐渐兴起。

研究者如Xie等人（2020）提出基于Cesium的实时射线投射方法用于三维视野计算。刘等（2022）引入地形LOD优化以提升交互效率。本文在前人工作的基础上，进一步优化了地形采样策略与渲染表现。

## **3 技术实现**

### **3.1 Cesium平台概述**

Cesium使用WebGL进行三维地球渲染，支持地形加载、3D Tiles、地理定位与场景交互。

### **3.2 需求概述**

1、实现三维求的加载，拉近显示影像图；

2、在地图上添加高亮区域面，双击地图视角可以快速的聚焦到面显示的区域；

3、添加该区域的地形图数据，显示和现实类似的高低起伏的走势；

4、在地图上显示八个黄色的点数据；

5、拖动点的位置，放置到高亮面区域位置，可以根据地形判断四周遮挡情况，可视化效果使用绿色表示可视，红色表示不可视。

### **3.2 需求技术实现**

#### **3.1.1、三维地球的实现**

**1、需要引入css和js，如下：**

 <link rel="stylesheet" href="./cesium/Widgets/widgets.css">

<script type='text/javascript' src="./cesium/Cesium.js"></script>

**2、地图实例化：**

viewer = new Cesium.Viewer("map-container", {})

**3、添加影像瓦片地图**

此处以ArcGis影像瓦片为例

viewer = new Cesium.Viewer("map-container", {

    //添加ArcGis瓦片底图图层

   imageryProvider: new Cesium.ArcGisMapServerImageryProvider({

  url:"https://services.arcgisonline.com/ArcGIS/rest/services/World\_Imagery/MapServer",

})

})

#### **3.1.2、添加高亮区域面数据**

**1、高亮区域面数据**

代码如下：

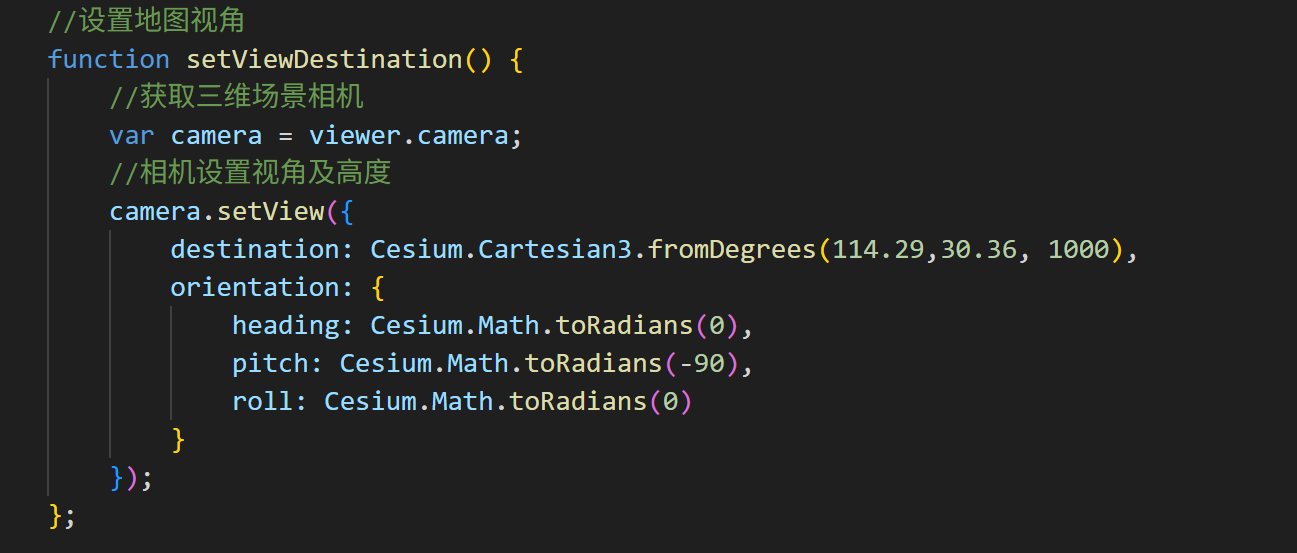


**参数介绍**

|  |  |
| --- | --- |
| **hierarchy** | 定义多边形的边界点（经纬度序列） |
| **material** | 填充多边形的颜色材质 |
| **outline** | 是否绘制多边形边界线 |
| **outlineColor** | 设置轮廓线颜色 |
| **clampToGround** | 是否将多边形贴地绘制 |

**2、地图视角聚焦**

可以通过viewer.zoomTo(polygonEntity)的方式或者以下方式：



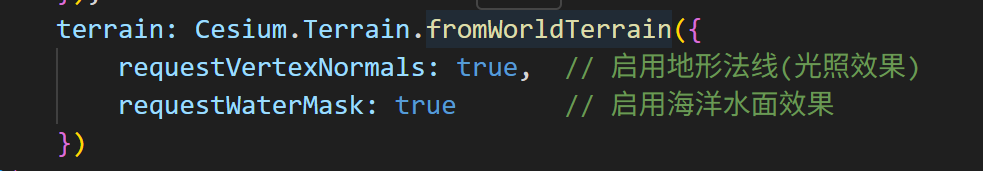
**viewer.camera**：Cesium 的相机对象，负责控制场景中观察者的位置与方向，支持平移、旋转、缩放、飞行等操作。

|  |  |
| --- | --- |
| **destination** | 相机的位置，即从哪儿看 |
| **heading** | 相机朝向水平方向的朝向 |
| **pitch** | 相机朝向绕 X 轴旋转，即俯仰角 |
| **roll** | 绕 Y 轴旋转，表示图像是否倾斜 |

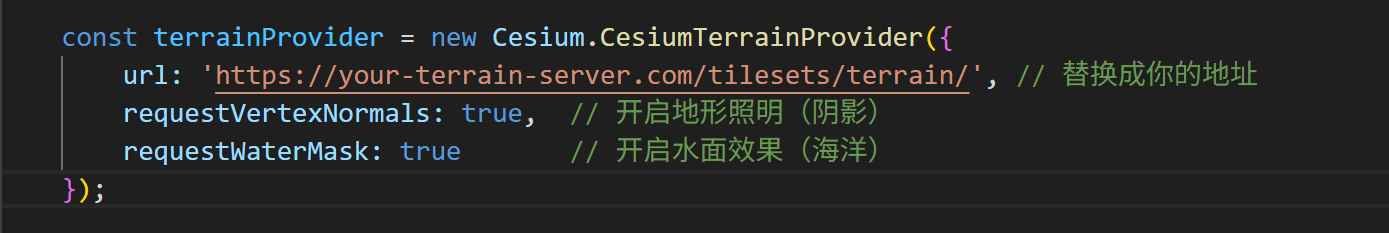
#### **3.1.3、地图上添加地形图数据**

在Cesium添加地形图方式有多多种，可以使用内置的地形，亦可以自己发布地形图片数据加载。

**1、使用内置的地形，代码如下：**



**2、自己发布地形图加载，代码如下：**



地形贴图（Terrain Textures）是通过将纹理（Texture）图像映射到3D模型表面来提升可视化效果的一种技术。在地理信息系统（GIS）和三维渲染中，地形贴图通常用于模拟地形的外观，如山脉、平原、湖泊等。具体到实现，地形贴图通常依赖于地形网格和纹理图的结合，以下是其底层实现的几个关键步骤：

**1. 地形网格的生成**

地形网格通常是一个高度图（Heightmap）与网格结构的结合。高度图是一个二维的数组或图像，其中的每个像素值代表地面的高度。基于这些高度信息，生成相应的三维地形网格。

* 网格划分：将整个地形划分为多个小单元（如三角形网格）。常见的方式是使用四叉树（Quadtree）或三角剖分（Triangulation）来优化地形的细节和渲染效率。
* 顶点生成：每个网格的顶点位置根据高度图上的值来设置，例如，(x, y) 坐标代表网格的平面位置，而 z 坐标则是从高度图读取的高度值。

**2. 纹理映射**

一旦地形网格被生成，接下来就是纹理映射，将地面纹理贴到这些网格表面。

* 纹理坐标：每个网格顶点都有相应的纹理坐标，这些坐标定义了纹理图像中每个点与网格点的关系。通常，纹理坐标是在生成网格时根据地形的经纬度或网格坐标来计算的。
* 多重纹理：为了增强地形的表现效果，可以使用多重纹理映射（Multi-texturing）。例如，一种纹理可能用于覆盖远距离的地面，而另一种纹理则用于近距离细节。

**3. LOD（Level of Detail）技术**

地形渲染通常会使用LOD（细节层级）技术来优化性能。LOD是通过根据观察者与地形的距离来决定显示地形的细节层级，从而减少渲染计算。

* LOD技术：常见的LOD技术包括基于网格细分的动态细节层级调整，以及基于视距的距离裁剪。在远距离时，使用低分辨率网格和纹理；在近距离时，使用高分辨率网格和纹理。
* 细节切换：地形贴图的细节切换可以通过逐渐切换纹理分辨率，或根据摄像机的位置和视角动态加载不同细节级别的纹理。

**4. 动态纹理生成（可选）**

某些情况下，地形贴图可能会包括动态生成的纹理，比如添加天气效果、季节性变化、或者与地形交互后的变化。

* 实时纹理生成：通过算法动态调整地形纹理，例如添加雪、雨、或者岩石表面在风化后的纹理效果。
* 纹理混合：在一些高度动态的场景中，纹理可以根据高度或坡度进行混合。例如，在高山地带可能会有冰雪覆盖，而在低谷则可能是泥土或草地的纹理。

**5. 着色器（Shaders）的应用**

在渲染过程中，着色器扮演着至关重要的角色，尤其是地形的外观。

* 顶点着色器：通过顶点着色器，将网格的每个顶点转换到屏幕空间，同时进行相关的光照计算。
* 片段着色器：通过片段着色器来实现纹理采样和细节处理，最终决定每个像素的颜色。对于地形纹理，常常在这里实现纹理混合、法线贴图、环境光等效果。

**6. 优化与缓存**

由于地形通常很大，直接加载全部的地形数据可能会造成性能瓶颈。因此，通常会采用一些缓存策略来优化性能。

* 地形分块：将地形划分为多个块，并根据视角动态加载当前视野内的地形块。这种方式类似于虚拟地球（如Cesium）中常见的瓦片加载方式。
* 纹理压缩：为了提高性能和减少内存占用，纹理通常会进行压缩。常见的压缩格式包括DDS、PVR和ETC。

**7. GPU加速渲染**

现代的地形渲染通常会利用GPU进行加速，尤其是在大规模地形或复杂场景中。

* GPU计算：GPU可以并行处理地形网格的顶点计算、纹理映射和光照计算，从而大大提升渲染效率。
* 多线程渲染：在一些高级渲染引擎中，地形的渲染过程可能会分为多个线程进行处理，从而进一步提升渲染效率。

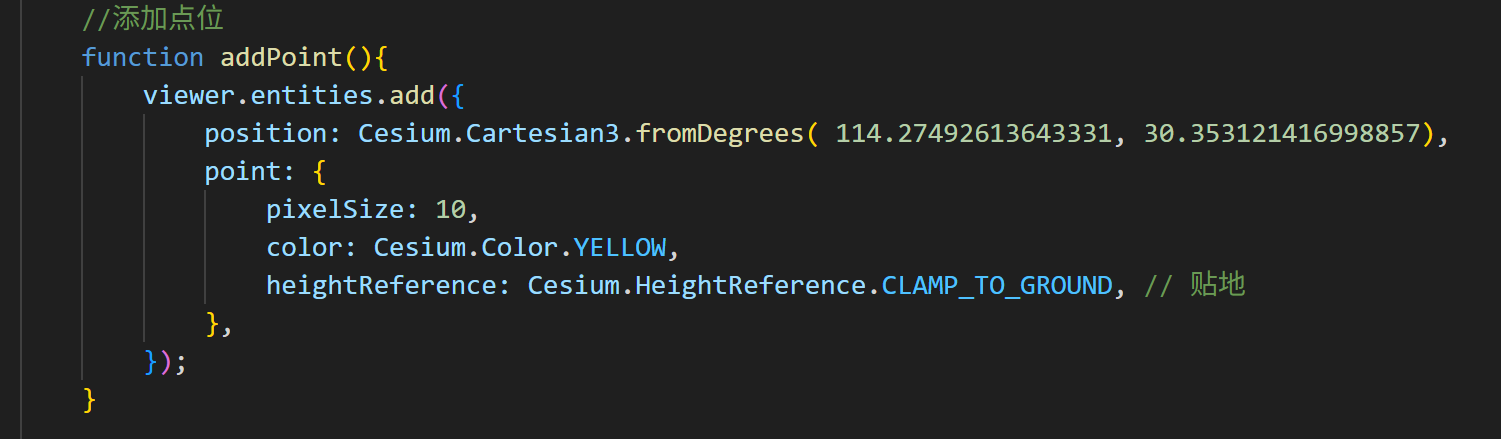
8. 优化技术

为了进一步优化渲染性能和提升视觉效果，常见的技术还包括：

* 法线贴图（Normal Mapping）：用来模拟地表的细节，而不是实际改变网格的几何形状。通过将法线信息嵌入纹理中，能够呈现出更高的细节感。
* 环境光遮蔽（Ambient Occlusion）：增强地形的阴影效果，让地形的凹凸部分更加自然。

#### **3.1.4、地图上添加八个黄色的点**

实现方法如下：



**参数介绍：**

|  |  |
| --- | --- |
| **position** | **地图定位的经纬度坐标点** |
| **pixelSize** | **像素点大小** |
| **color** | **像素点颜色黄色** |
| **heightReference** | **高度参考** |

#### **3.15、点的可视域分析**

可视域分析在许多领域有着广泛的应用：

**1、军事领域**：评估观察哨、雷达站的覆盖范围，**识别视野死角和最佳观测位置**。

**2、城市规划**：分析新建筑对周围景观视野的影响，**确保重要景点的可见性**。

**3、通信网络规划：优化信号塔的位置，确保最大的信号覆盖范围。**

**4、安防监控**：评估监控摄像头的覆盖范围，**消除监控盲区**。

**5、旅游景点设计：确定最佳观景平台位置，提升游客体验。**

**6、环境影响评估：分析新项目对周围环境视觉影响的范围。**

**技术原理解析**

Cesium中实现可视域分析的核心原理主要是利用阴影映射（Shadow Mapping）技术。该技术原本用于渲染阴影，但我们可以巧妙地应用于可视域分析中：

1、创建虚拟光源相机：将观测点作为光源，设置相机的视锥体参数（如视距、水平视角、垂直视角等）。

2、生成深度图：从虚拟光源相机角度渲染场景，生成深度图（Shadow Map）。

3、可见性判断：对于每个像素点，比较其到观测点的实际距离与深度图中记录的距离，判断该点是否可见。

4、着色处理：根据可见性结果，对可见区域和不可见区域分别使用不同颜色进行着色。

**相关代码解析：**

**1、设置观测点；**

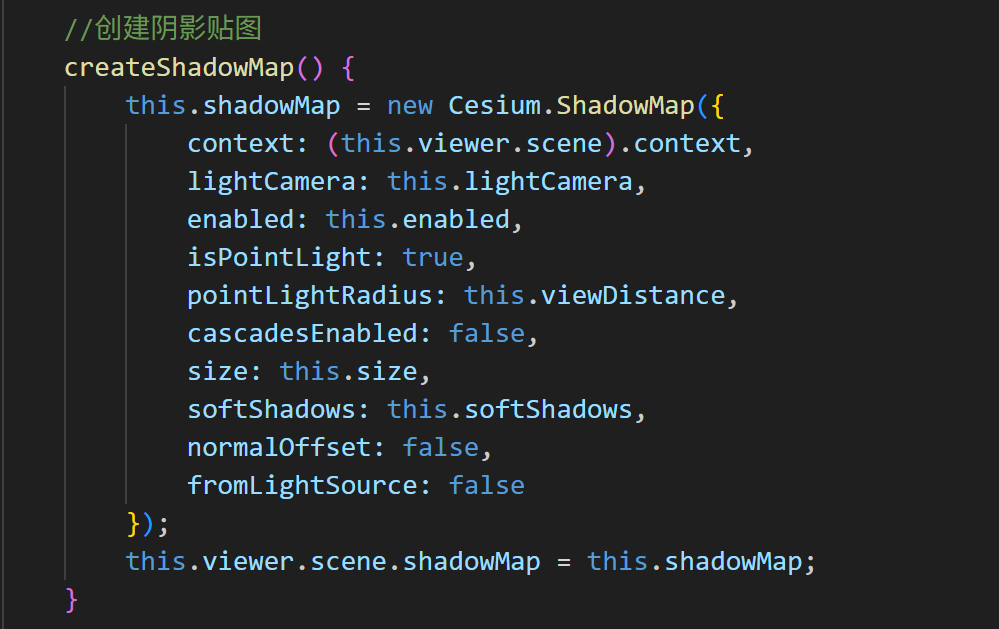
可以手动交互式的，也可以自己设定

**2、创建虚拟点光源，代码如下：**



这个方法创建一个虚拟相机作为光源，位于观测点位置，朝向观测方向，并设置视锥体参数（近平面、远平面、视场角、纵横比等）。这个相机将用于生成深度图。

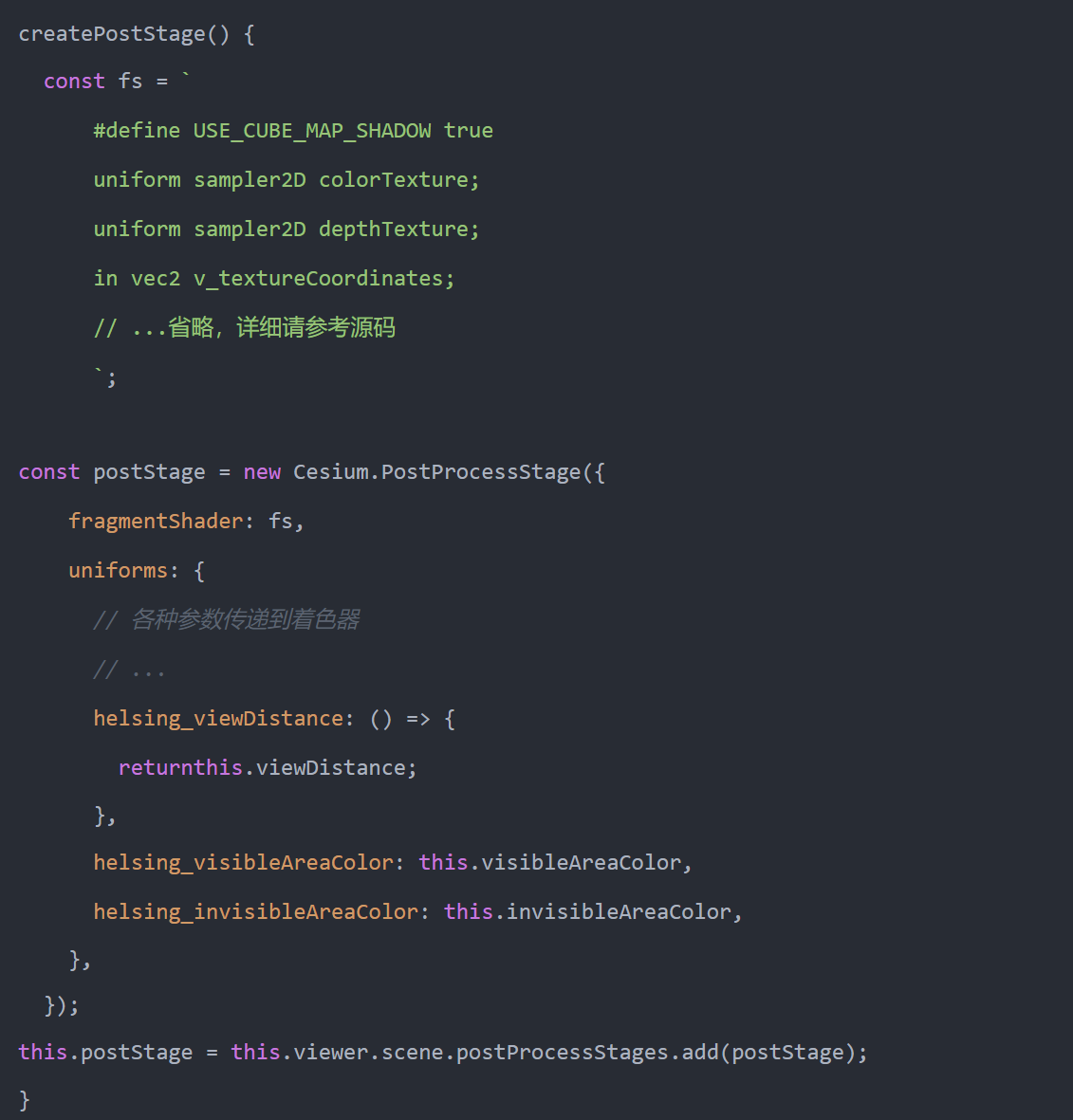
**3、创建阴影贴图，代码如下**



这个方法创建了一个ShadowMap对象，用于从光源（观测点）角度生成场景的深度信息。设置isPointLight为true表示从点光源出发，可以向各个方向投射视线。

4、创建后处理

后处理是可视域分析中最关键的部分，注意：着色器需要根据Cesium版本进行语法变更，webgl1跟webgl2的语法略有不同。 通过自定义着色器来渲染可视域分析的结果；



这个方法创建了一个后处理，其中包含自定义的片元着色器**。着色器通过比较每个像素点到观测点的距离与深度图中记录的距离，判断该点是否在视线范围内，然后对可见区域和不可见区域分别着色**。

**4、色器代码的工作流程是：**

获取当前像素的颜色和深度信息

转换到相机坐标系和世界坐标系

转换到观测点（虚拟相机）坐标系

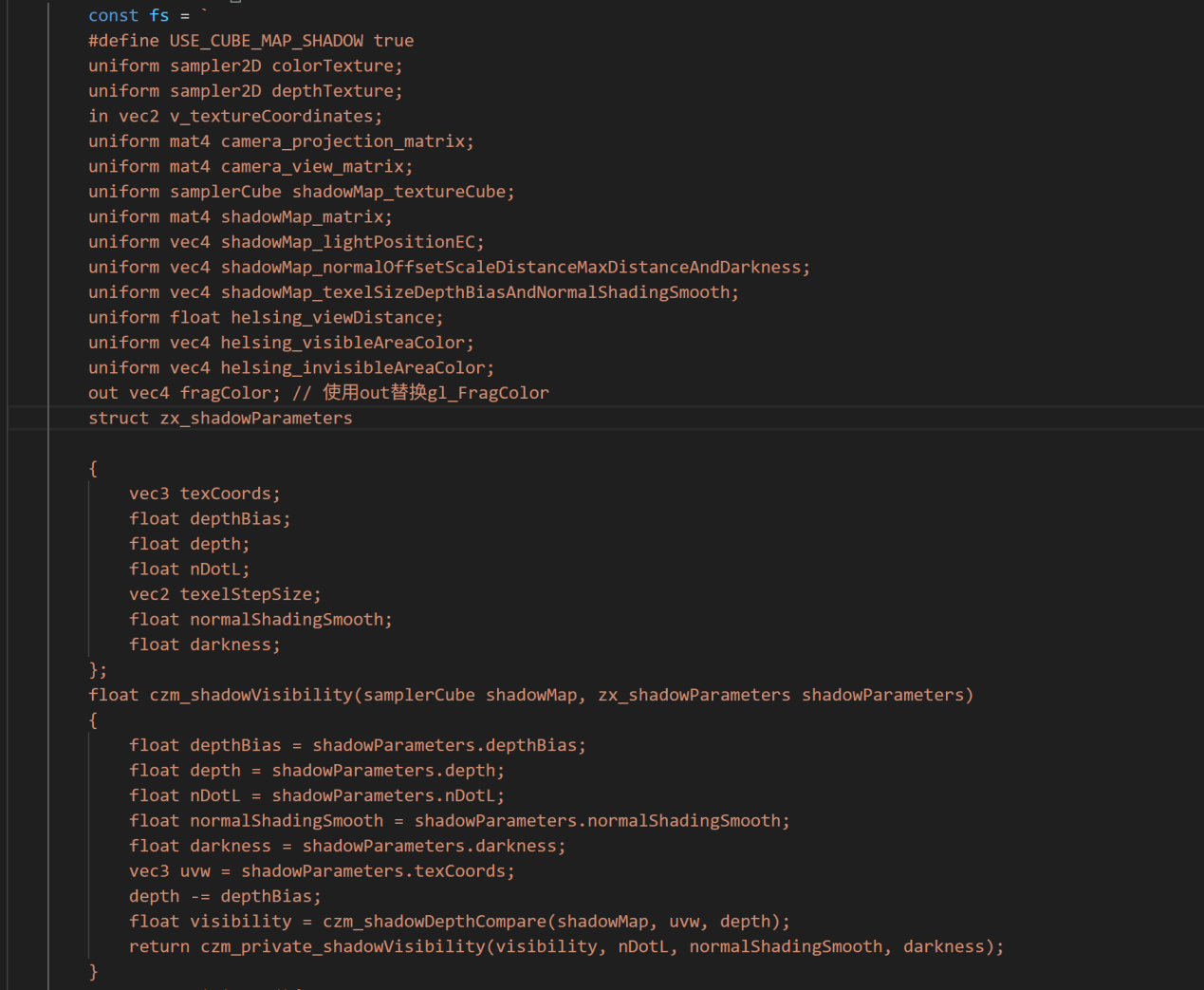
判断点是否在观测距离范围内

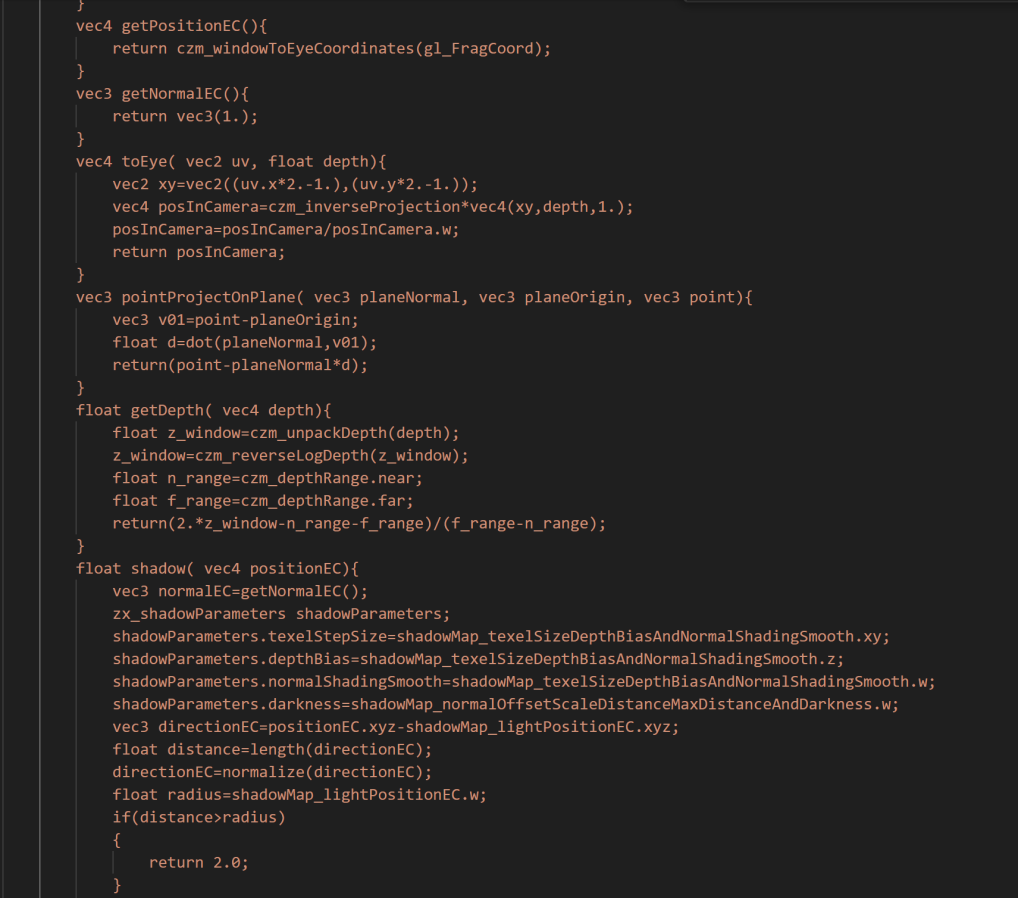
判断点是否在视锥体内

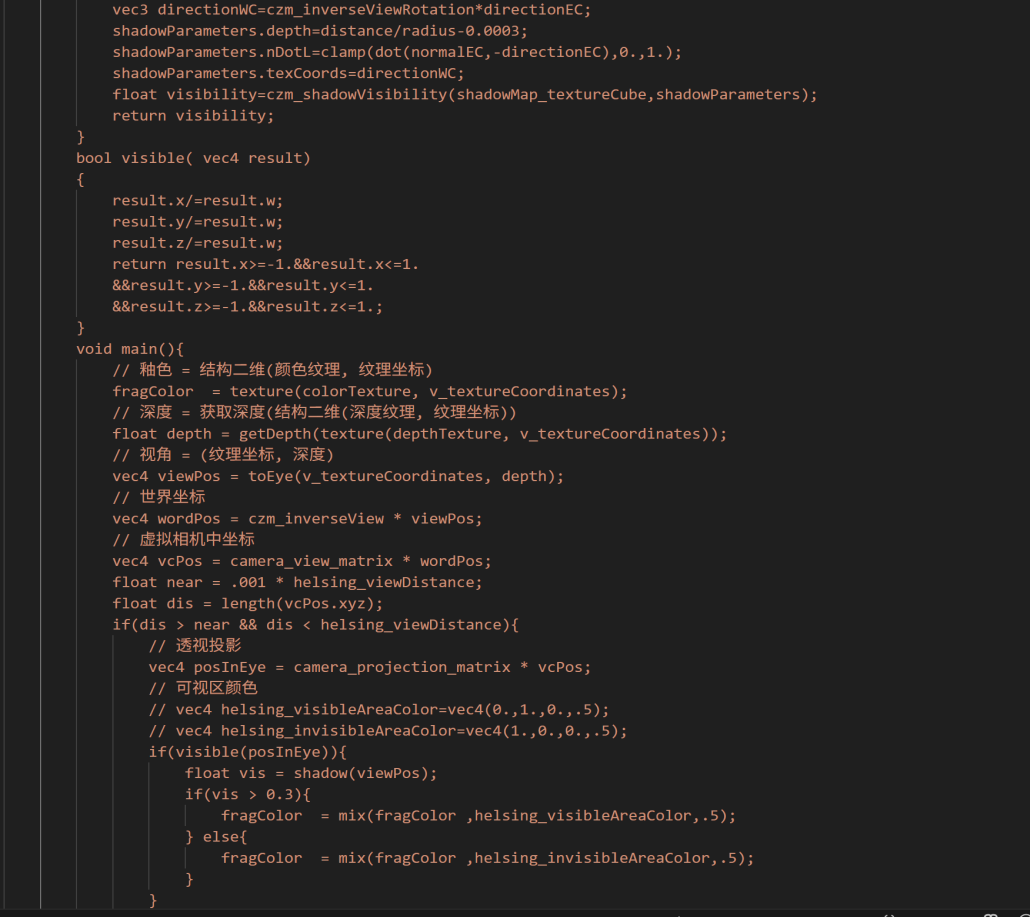
通过shadow函数判断点是否可见

根据可见性结果进行着色

代码如下：

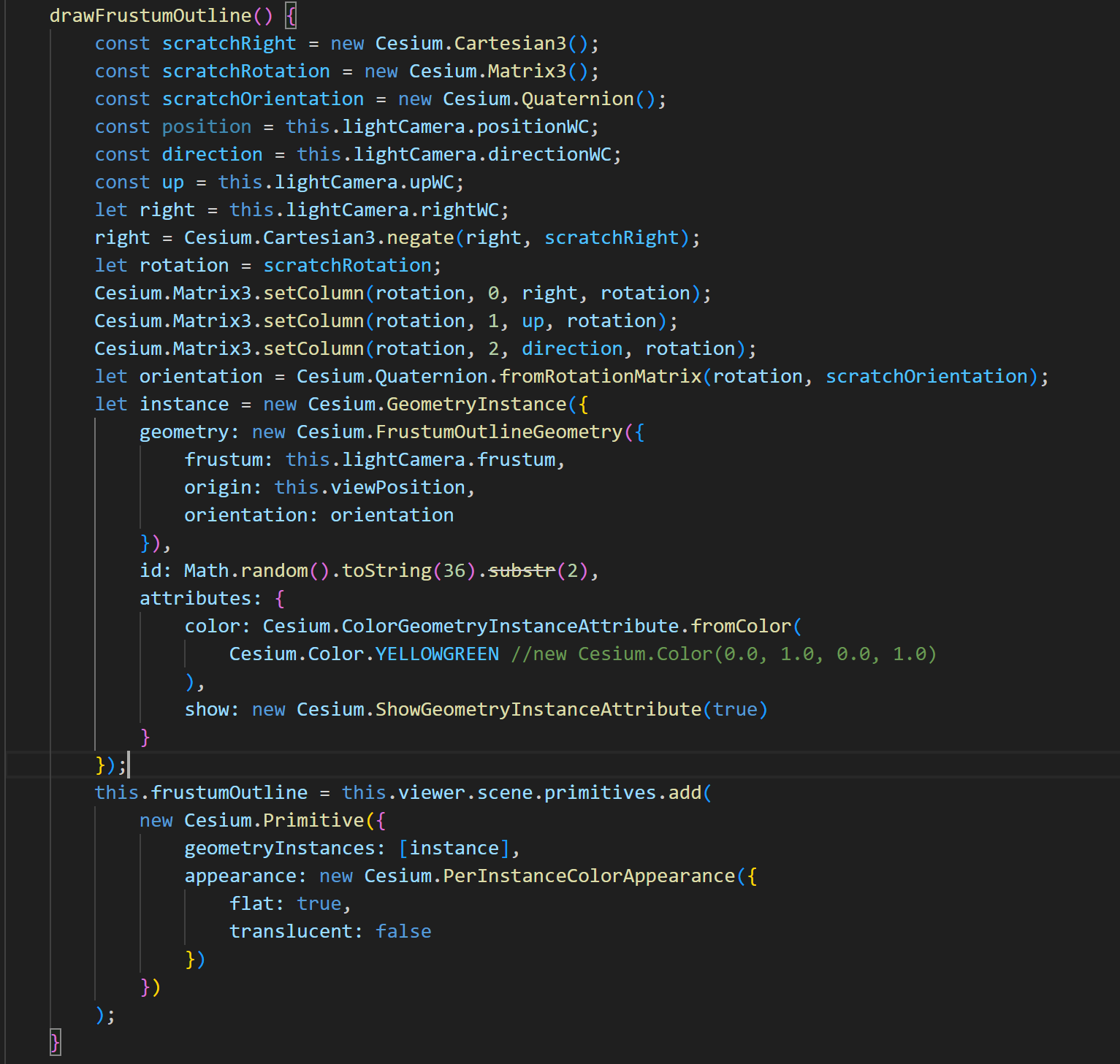




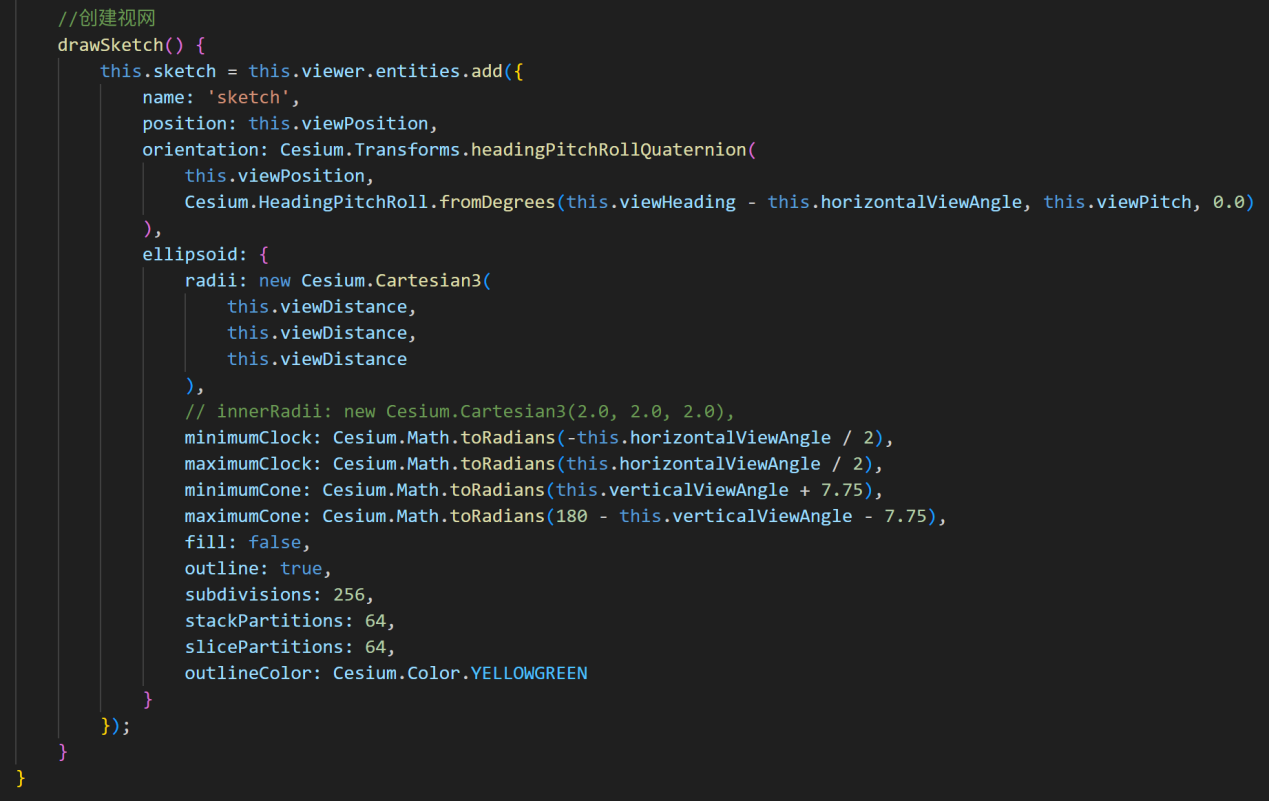


**5、创建视锥体和视网**

**创建视锥体：**



**创建视网：**



**注意：如果直接使用内置的地形，会出现都是可见区域这样的错误。**

**主要是因为**：Cesium 的默认地形（Cesium World Terrain）是基于 GPU 加速的地形渲染技术（如 GlobeSurfaceTileProvider），不是普通的 Primitive 或 Model。

而 ShadowMap 的原理依赖的是 从光源（或观察点）渲染视角下场景中所有参与阴影投影的图元（geometry）。

**解决方案：**

**方法 1：使用可投影的“实体地形”**

如果你只需要在一个小区域做精确分析：

使用自定义的地形 mesh，比如：

通过 3D Tiles 加载地形模型。

把 DEM 数据转成 .glb / .gltf 模型加载进场景。

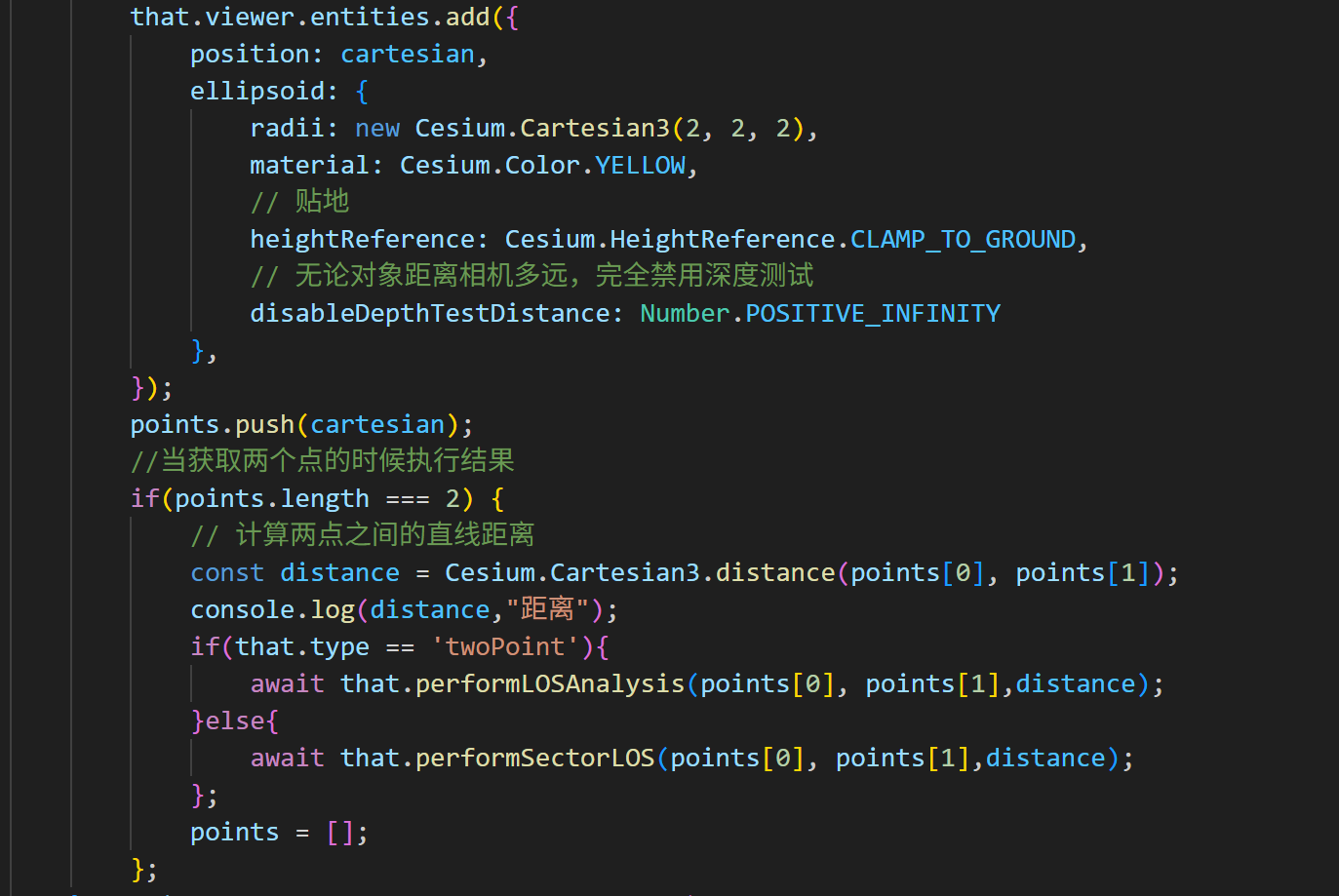
使用 Primitive 加载高程面片（可接受阴影）。

这样它们就能被 ShadowMap 采样，从而参与可见性判断

**方法 2：高级做法：自定义 Shader / 自定义 Ray Casting**

**核心代码如下：**

步骤1、设定初始视角位置点和终点，如下所示



步骤二、根据绘制起点和终点，结合地形计算可视域；详细步骤如下：

**1、根据起点和终点计算方法向量，归一化**

**2、根据起点和归一化方向向量获得Cesium射线**

**3、计算起点和终点的距离；**

**4、设置取样间隔，获取取样数量，取消间隔越小，越准确；**

**5、根据Cesium射线获取取样点的笛卡尔坐标集合；**

**6、根据viewer.terrainProvider地形,和取样点集合，获取对应集合点的高程**

**7、根据射线获取的集合点高度，和该段取样高程对比判断，如果大于，则可见，小于则不可见**

**8、然后根据可见为绿色，不可加为红色，循环绘制。**

**实现代码如下：**

