渲染引擎

对各种模型进行染色、光照、阴影等渲染，绘制出屏幕画面的功能模块。 下面是渲染的五个阶段

- 工具阶段（制作模型，定义几何和表面特性）
- 资产调节阶段（处理几何和材质，转换模型格式）
- 应用程序阶段 CPU （为渲染管线做准备）
- 几何阶段 GPU（处理网格顶点数据 转为屏幕空间的顶点信息）
- 光栅化阶段 GPU（着色 输出最终结果）
    - 裁剪（Clipping） 将超出视景体范围的图元裁剪掉
    - 光栅化（Rasterization） 将图元转化为像素，生成片元（Fragment）
    - 片段着色器（Fragment Shader） 对每个片元进行着色，计算片元的颜色
    - 深度测试（Depth Test）和 混合（Blending）判断像素的遮挡关系和透明度，决定是否应该丢弃和怎么显示

渲染管线就是**一堆原始图形数据经过各种变化处理最终出现在屏幕的过程。**渲染管线可分为三个阶段，**应用程序阶段，几何阶段，和光栅化阶段。**

2、应用程序阶段

由CPU主要负责。CPU将GPU渲染需要的灯光、模型准备好，随后向GPU下达一个渲染指令Draw Call，即往命令缓冲区中放入命令，GPU则依次取出命令执行

3. 几何阶段

把输入的3D数据转换成2D数据。包括顶点着色器、图元装置、裁剪和屏幕映射几个过程。

顶点着色器主要进行顶点坐标变换。将输入的**模型空间**顶点坐标变换到**裁剪空间**顶点坐标。

图元装配将**顶点**装配成指定**图元**的形状。可以**细分**为外壳着色器、镶嵌器和域着色器。

几何着色器通过产生新顶点构造出新的图元来生成其他形状。

图元组装将输入的顶点组装成指定的图元。

图元组装后会进行屏幕映射的操作，包括**透视除法**（投影、裁剪转成2维）和**视口变换**（映射，适配到屏幕），将图元从三维空间映射到二维平面上，这是由**硬件**完成的。

4、光栅化阶段

把图元映射为最终屏幕上显示的颜色，包括光栅化，片段着色，透明度测试和模板测试和混合。

光栅化将顶点转为屏幕上的像素。会进行三角形遍历。**三角型遍历，**检测出所有被三角型覆盖的像素。（此处可拓展出怎么划分片元、怎么抗锯齿）

片段着色器计算每个像素的最终颜色。是一个**可编程的阶段**，主要的光照处理都在这个阶段。**透明度（Al**pha）测试**通过深度信息决定像素是否显示。****可设置阈值，显示的像素将与颜色缓冲区中颜色混合。

模板测试通过片元的模板值与模板缓冲区的模板值的比较来筛选像素。

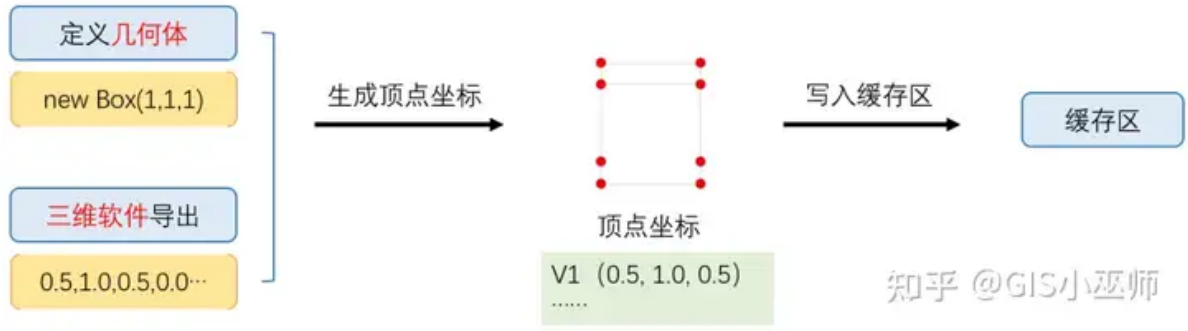
深度测试通过深度信息判断像素的遮挡关系。筛选掉被遮挡的像素。现在大多数 GPU 都支持**提前深度测试**(Early depth testing)，在片元着色器之前测试，由硬件功能完成。

WebGL 绘制流程

- 1、获取顶点坐标
- 2、图元装配 （画出一个个三角形）
- 3、光栅化（生成片元，即一个个像素点）

获取顶点坐标

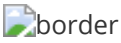
顶点坐标一般从三维软件导出，或者框架生成。



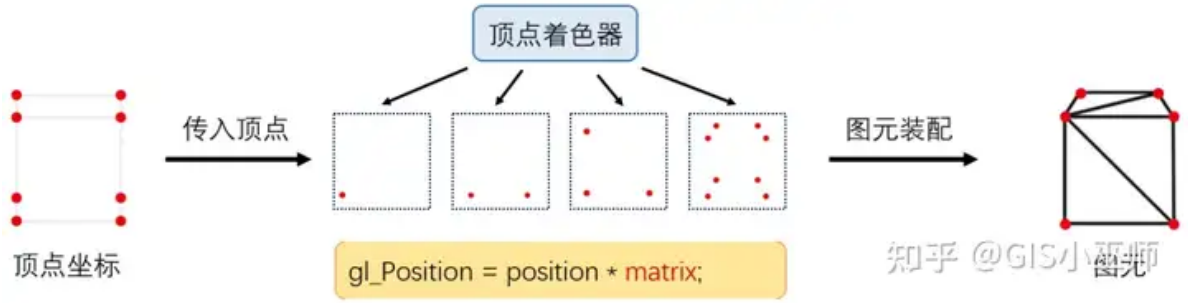
写入缓冲区是大量的顶点数据，会存在显卡，也就是缓存区内，方便GPU快速读取。

图元装配

图元装配就是生成一个个图元（三角形）。这里我们可编程渲染管线，为了更好的控制顶点位置。

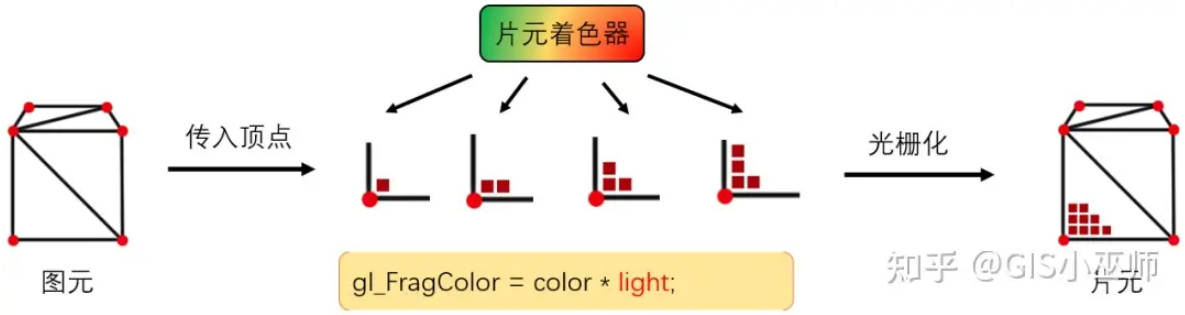


顶点着色器处理流程



顶点着色器会先将坐标转换完毕，然后由 GPU 进行图元装配，有多少顶点，着色器就运行多少次。投影矩阵将三维坐标转为屏幕坐标，这个矩阵叫做投影矩阵。

光栅化



图元完成以后，就需要给模型上色，这就是 运行在 GPU的 片元着色器来完成。模型看起来是什么质地（颜色，漫反射 贴图）、灯光等都是片元着色器来计算。

```
const FragmentColor = `
  precision mediump float;
  uniform vec4 u_FragColor;
  void main() {
    // gl_FragColor 输出的颜色值
    gl_FragColor = u_FragColor;
  }
`;
```

片元着色器生成多少片元（像素）就运行多少次。

顶点着色器（Vertex Shader）

接收 attributes，计算和操作每个顶点的位置，传递额外的的数据（varings）给片元着色器。

Fragment Shader

片元是在栅格化之后，形成像素之前的数据。片元着色器是每个片元调用一次的程序。shader 会根据屏幕不同的位置执行不同的操作。

4.3、WebGL的完整工作流程

1、准备数据阶段
在这个阶段，我们需要提供顶点坐标、索引（三角形绘制顺序）、uv（决定贴图坐标）、法线（决定光照效果），以及各种矩阵（比如投影矩阵）。其中顶点数据存储在缓存区（因为数量巨大），以修饰符attribute传递给顶点着色器；矩阵则以修饰符uniform传递给顶点着色器。

2、生成顶点着色器
根据我们需要，由Javascript定义一段顶点着色器（opengl es）程序的字符串，生成并且编译成一段着色器程序传递给GPU。

3、图元装配
GPU根据顶点数量，挨个执行顶点着色器程序，生成顶点最终的坐标，完成坐标转换。

4、生成片元着色器
模型是什么颜色，看起来是什么质地，光照效果，阴影（流程较复杂，需要先渲染到纹理，可以先不关注），都在这个阶段处理。

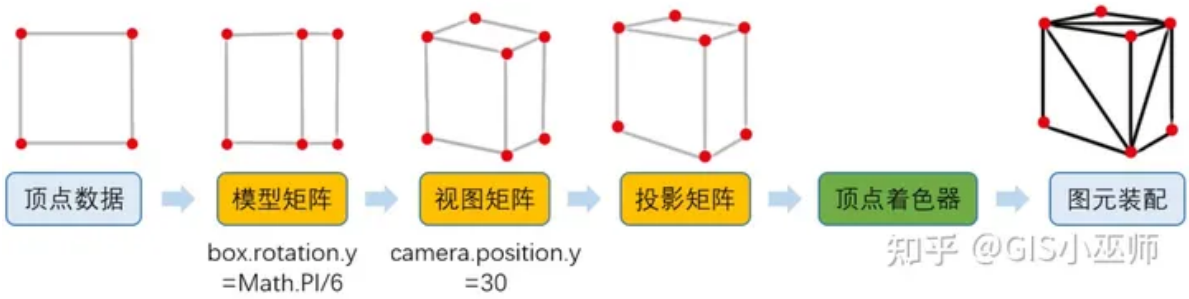
5、光栅化
能过片元着色器，我们确定好了每个片元的颜色，以及根据深度缓存区判断哪些片元被挡住了，不需要渲染，最终将片元信息存储到颜色缓存区，最终完成整个渲染。

Threejs 做了什么?

threejs 的流程

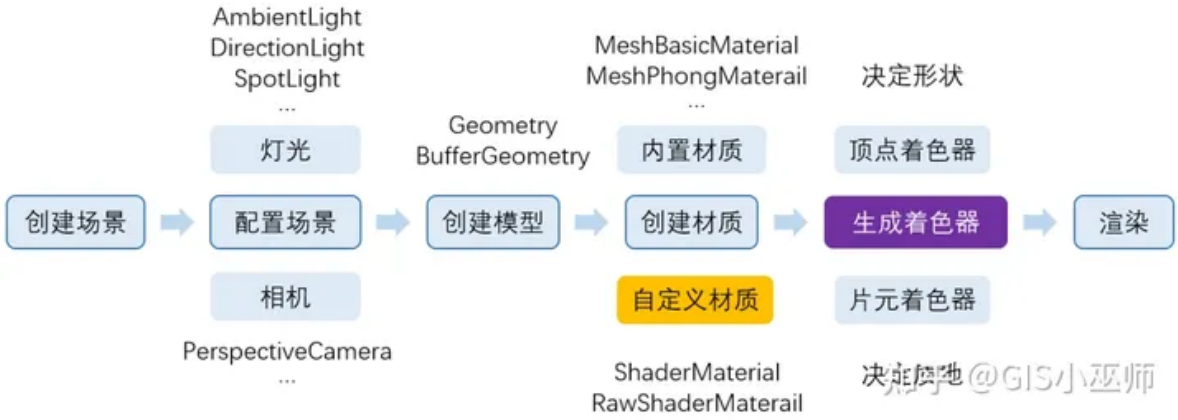
黄色和绿色部分，都是three.js参与的部分，其中黄色是javascript部分，绿色是opengl es部分。

- 辅助我们导出了模型数据；
- 自动生成了各种矩阵；
- 生成了顶点着色器；
- 辅助我们生成材质，配置灯光；
- 根据我们设置的材质生成了片元着色器。
- 而且将webGL基于光栅化的2D API，封装成了我们人类能看懂的 3D API。



矩阵运算能帮助我们节省大量时间来计算坐标变换。

片元着色器处理流程



1、获取 `webgl` 上下文、初始化着色器
2、获取顶点坐标（框架生成，三维软件导出）
3、写入缓冲区