# 从零开始实现一个软渲染器

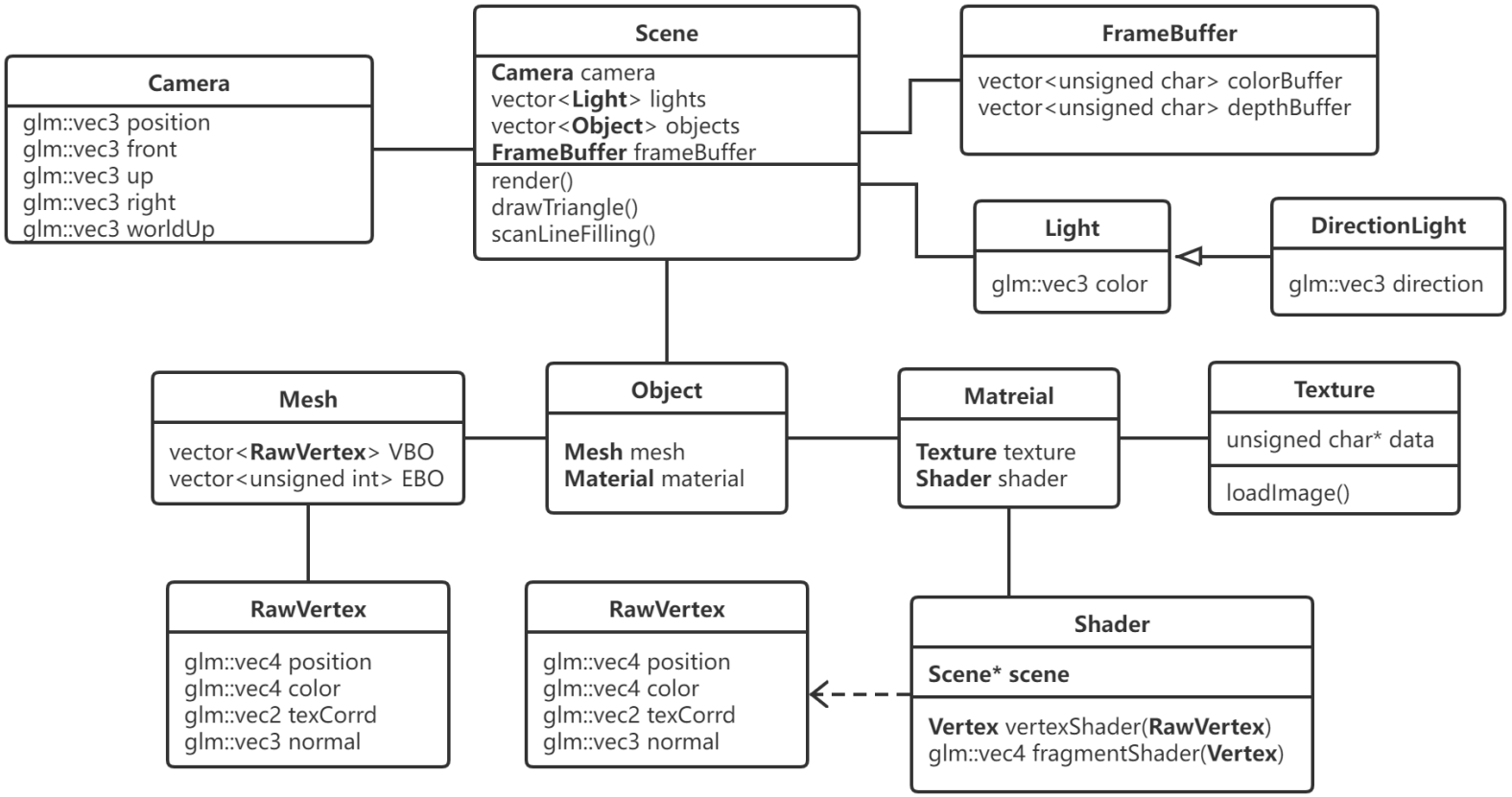
1953910 李林洲

# 依赖

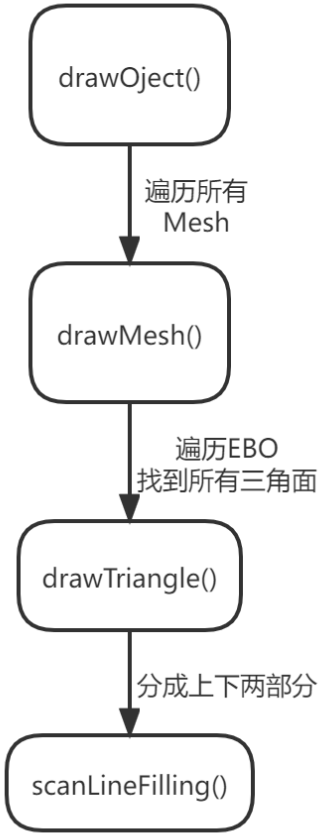
位于`include`和`lib`文件夹下。

- GLFW c++图形化界面库
- OpenGL 仅使用了`glViewport`与`glDrawPixels`两个函数
- glm 矩阵运算数学库
- stb_image.h 图片文件加载
- glad 用户加载OpenGL函数

# 类图


Camera
glm::vec3 position
glm::vec3 front
glm::vec3 up
glm::vec3 right
glm::vec3 worldUp
Scene
Camera camera
vector<Light> lights
vector<Object> objects
FrameBuffer frameBuffer
render()
drawTriangle()
scanLineFilling()
FrameBuffer
vector<unsigned char> colorBuffer
vector<unsigned char> depthBuffer
Light
glm::vec3 color
DirectionLight
glm::vec3 direction
Mesh
vector<RawVertex> VBO
vector<unsigned int> EBO
Object
Mesh mesh
Material material
Matreial
Texture texture
Shader shader
Texture
unsigned char* data
loadImage()
RawVertex
glm::vec4 position
glm::vec4 color
glm::vec2 texCorrd
glm::vec3 normal
RawVertex
glm::vec4 position
glm::vec4 color
glm::vec2 texCorrd
glm::vec3 normal
Shader
Scene* scene
Vertex vertexShader(RawVertex)
glm::vec4 fragmentShader(Vertex)

# 渲染流程


drawOject()
遍历所有
Mesh
drawMesh()
遍历EBO
找到所有三角面
drawTriangle()
分成上下两部分
scanLineFilling()

# drawMesh

## 顶点着色器

- 计算世界坐标(World Position)
- 计算窗口坐标(Window Position)

Window Position = Project Matrix * View Matrix * Model Matrix * Position (MVP矩阵)

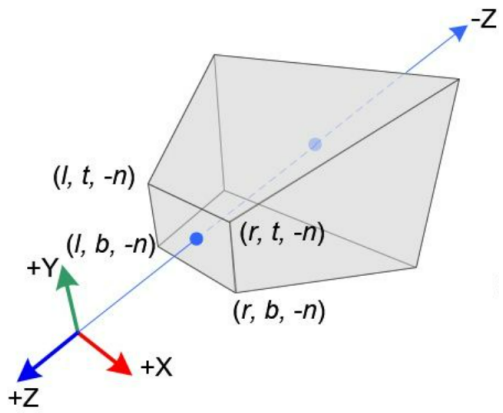
透视投影矩阵：

$$\begin{pmatrix} \frac{n}{r} & 0 & 0 & 0 \\ 0 & \frac{n}{t} & 0 & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

# 剔除

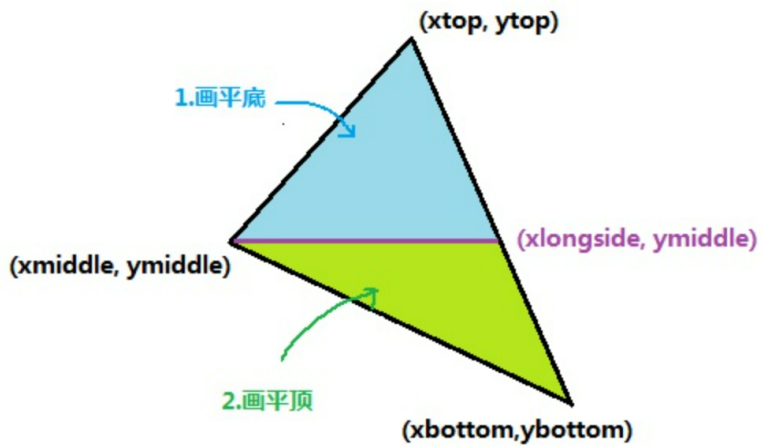
背面剔除：相机正方向与三角面不一致时剔除

视锥剔除：计算当前摄像机对应的视锥平面(上下左右远近)，三个点都不在视锥中的平面剔除

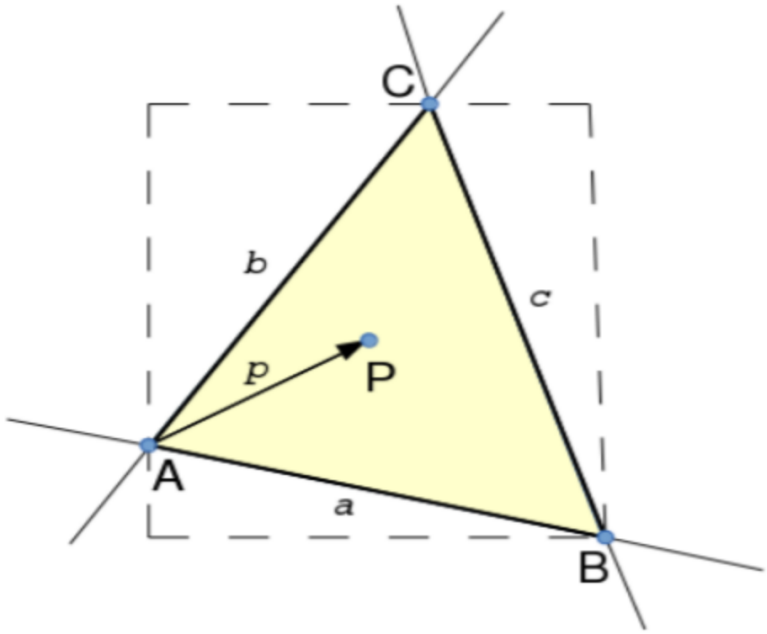


# drawTriangle

## 扫描线算法

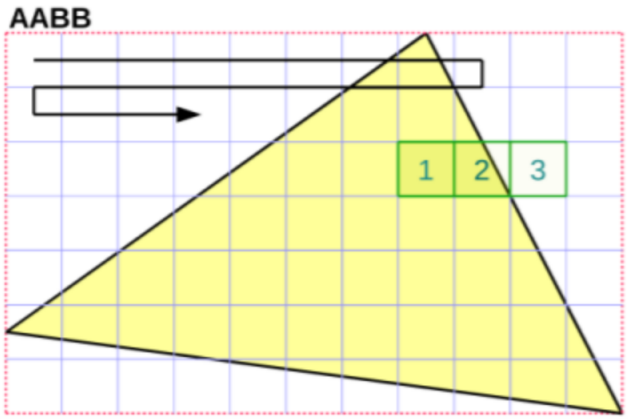


# 边界函数算法



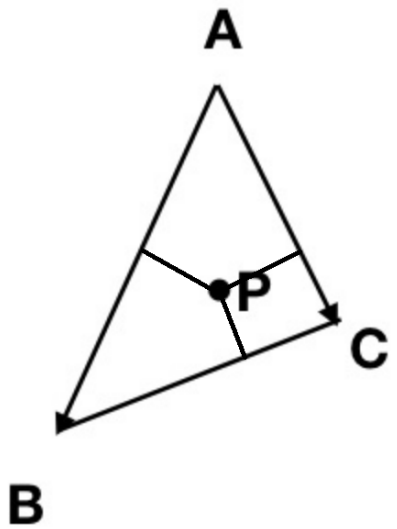
# 分块优化：

1. 完全在三角形内
2. 部分在三角形内
3. 完全在三角形外



## 重心坐标系：

- 重心坐标系下，x,y,z坐标均大于零则此点在三角形中
- 方便插值

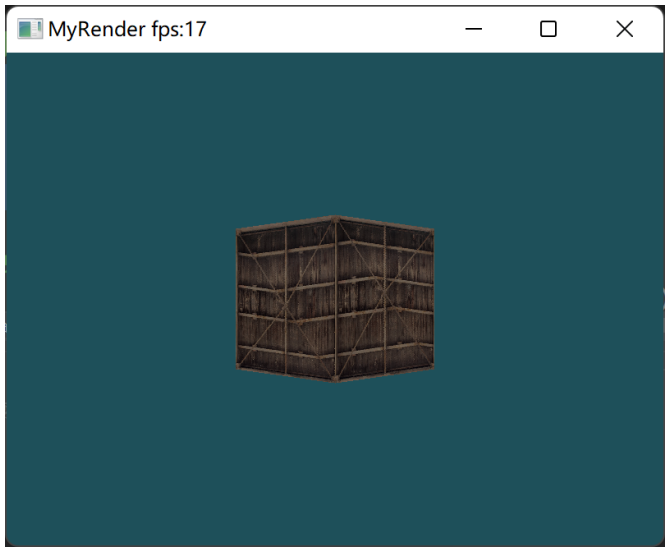



结果：单线程运行，边界函数算法反而比扫描线算法更慢

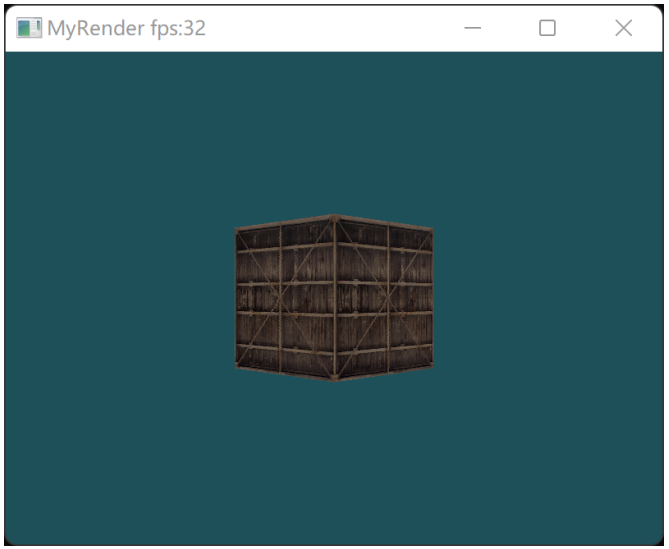
# fragmentShader

## 纹理映射

## 仿射纹理映射



# 透视纹理映射



# 光照

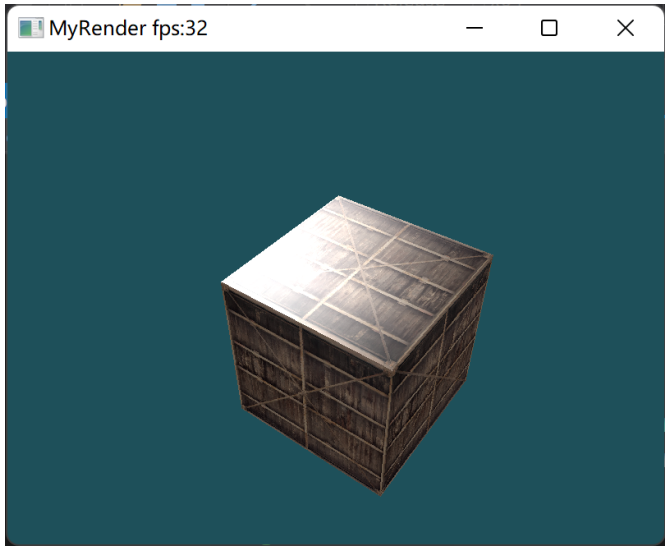
漫反射：从各处看到的反射光线都是一致的，漫反射的强度只跟物体表面与光线夹角有关。

```
float diff = dot(normal, -lightDir);
vec3 result = diff * lightColor * vertexColor * Intensity;
```

镜面反射：强度取决于反射光线与观察方向

```
vec3 reflectDir = normalize(lightDir - 2 * dot(normal, lightDir) * normal);
vec3 viewDir = normalize(cameraPos - worldPos); // 观察方向
float spec = pow(dot(reflectDir, viewDir), gloss);
vec3 result = spec * lightColor * Intensity
```

# 渲染效果



# Thanks