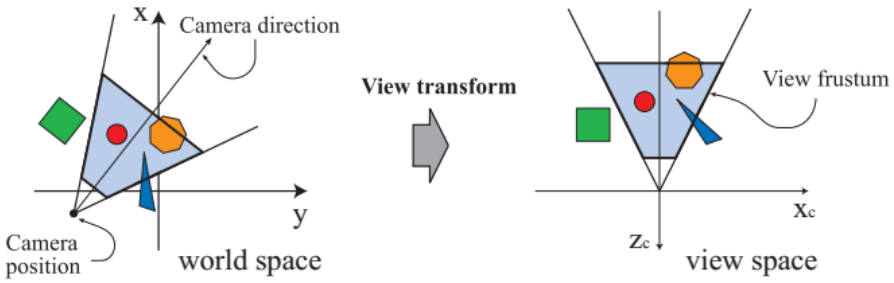
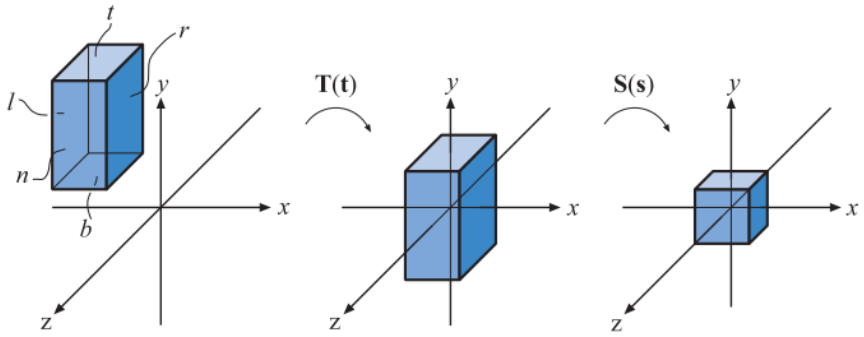
# 如何写一个软渲染器

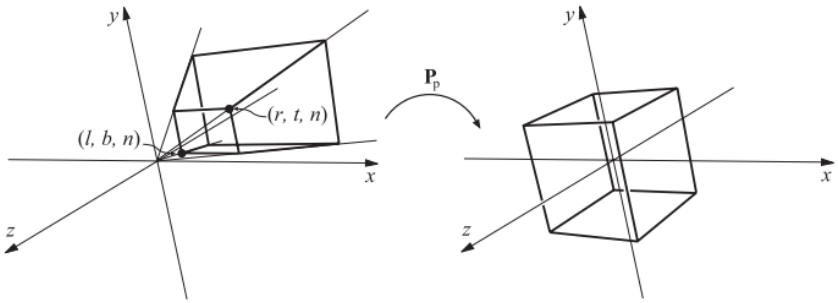
- 渲染管线概述
  - 功能概述
    - 虚拟相机、三维物体、光源、照明模式、以及纹理等诸多条件——>二维图像
  - 体系结构
    - 应用程序阶段(The Application Stage )
      - 输出：绘制图元（rendering primitives),如点、线、矩形
    - 几何阶段(The Geometry Stage)
      - 模型变换(Model Transform)
        - 将模型的顶点和法线变换到世界空间
      - 视图变换(View Transform)
        - 将相机放在坐标原点，方便后续的投影和裁剪操作

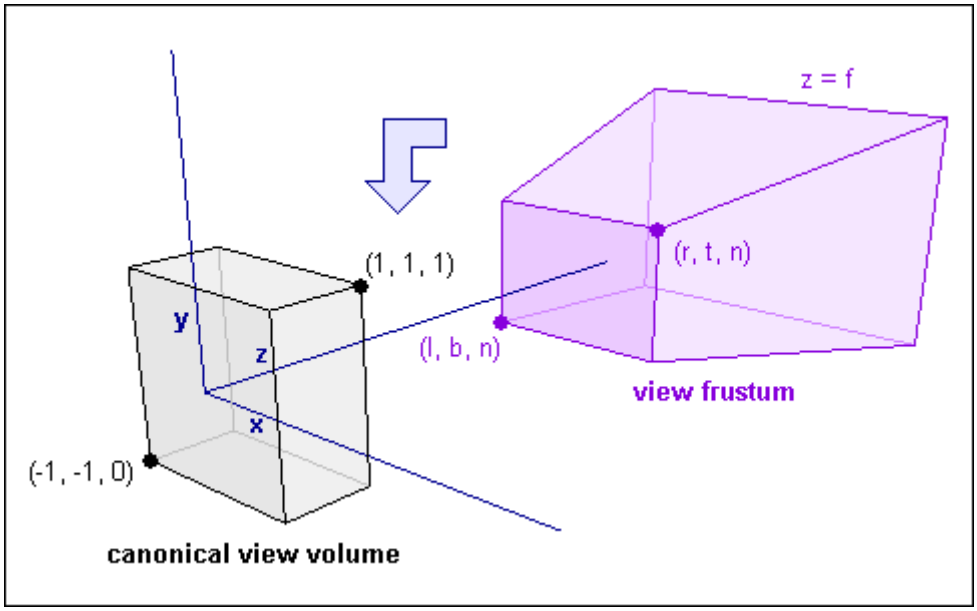



      - 顶点着色(Vertex Shading)
        - 目的：确定模型上顶点处材质的光照效果。
        - 计算位置：世界空间
      - 投影(Projection)
        - 规范立方体 (Canonical View Volume，CVV)
        - 正交投影
          - 可视体：长方体
          - 平移和缩放

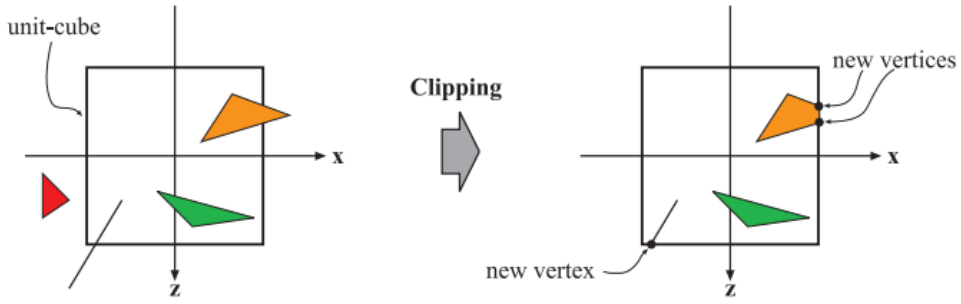



- 透视投影
  - 可视体：平截头体
  - 近大远小

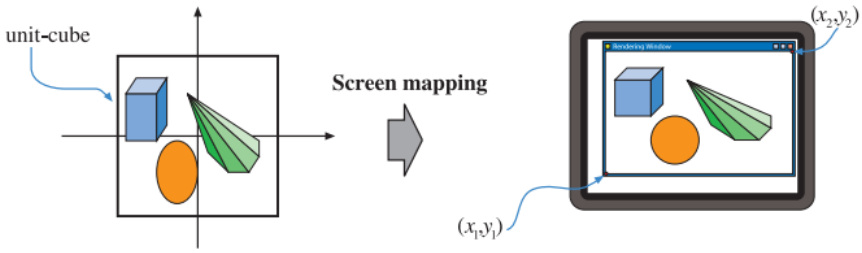



  - 目的：模型从三维空间投射到了二维的空间中的过程
- 裁剪(Clipping )
  - 目的：对部分位于视体内部的图元进行裁剪操作




- 屏幕映射(Screen Mapping)
  - 目的：将之前步骤得到的坐标映射到对应的屏幕坐标系



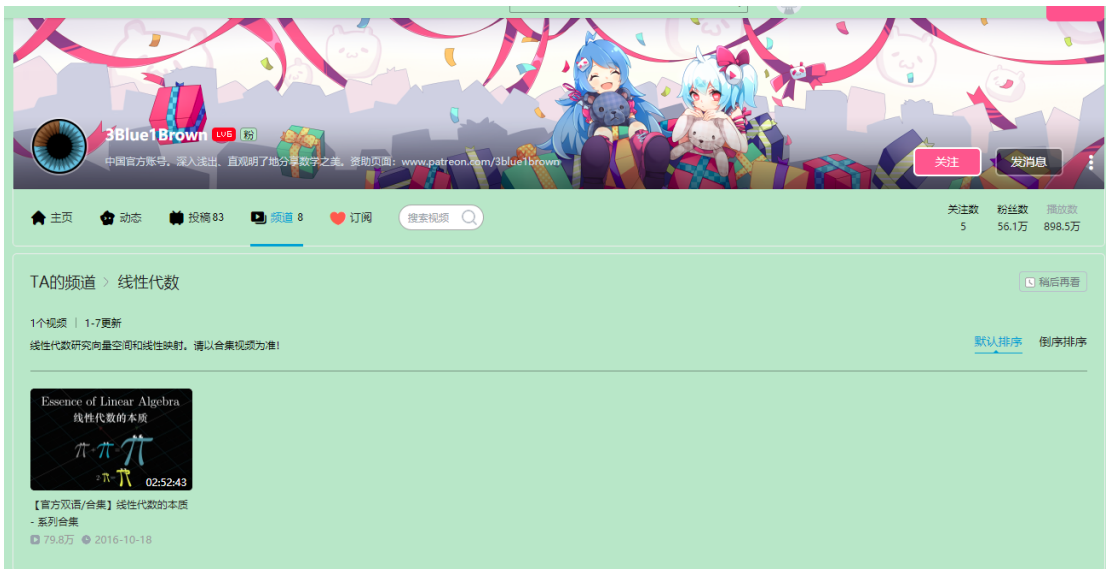

- 光栅化阶段(The Rasterization stage)
  - 概述：给定经过变换和投影之后的顶点，颜色以及纹理坐标（均来自于几何阶段），给每个像素 （Pixel）正确配色，以便正确绘制整幅图像
  - ZBuffer

# 软渲染器

## 准备工作

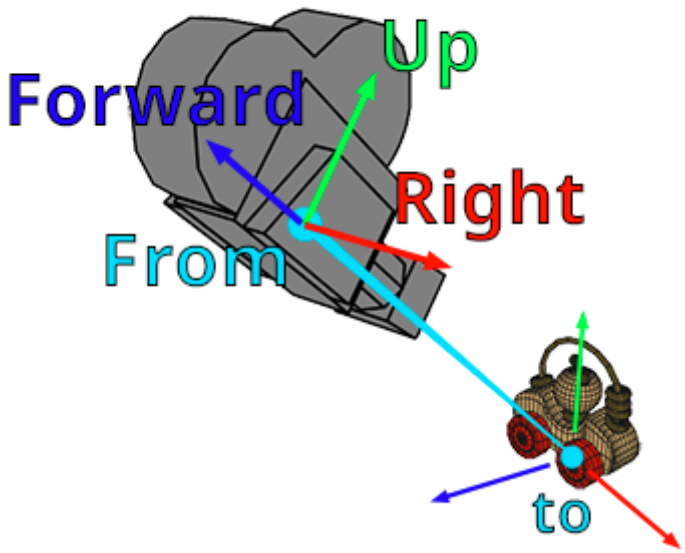
### 数学

- https://space.bilibili.com/88461692/channel/detail?cid=9450



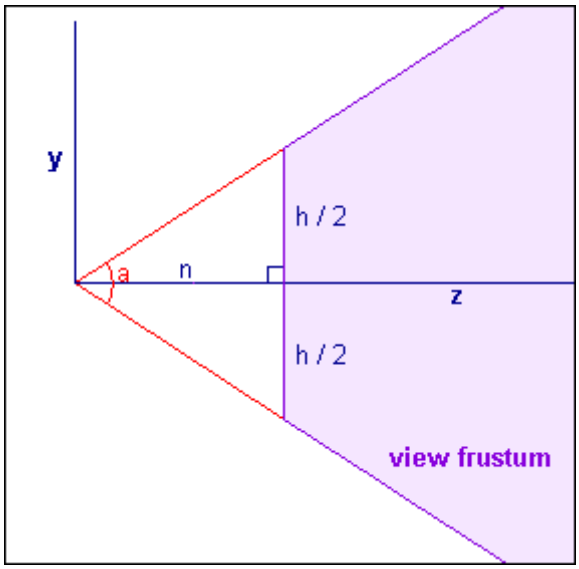

- 点，向量，法线

        - 坐标系
            - 维度
            - 左右手
        - 矩阵
            - 线性变换
            - 行主序/列主序
        - 齐次坐标
        - 线性插值方法
- 开始实现
    - 定义三维数据类型，并实现对这些类型的操作方法
        - 向量
            - 加、减、点积、叉积、归一化(Normalization)、长度
        - 矩阵
            - 矩阵乘法，矩阵与向量的乘法
    - 定义模型数据
        - 顶点
            - 位置
            - 法线向量
            - 颜色
            - UV
        - 三角形
        - 摄像机



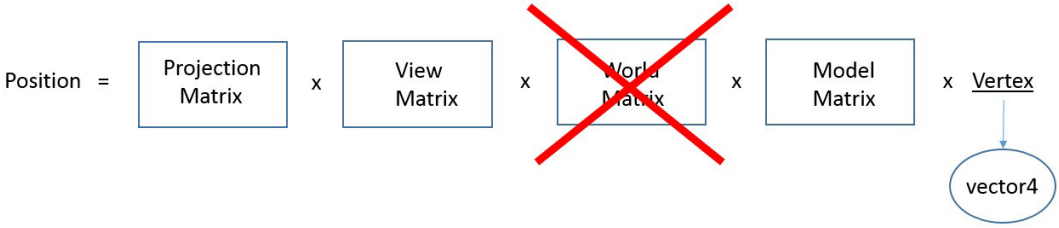

            - Eye：摄像机的位置

- Look：摄像机观察的位置
- Up：摄像机方向
- Fov：




- https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/lookat-function
- http://in2gpu.com/2015/05/17/view-matrix/
- 相机矩阵的推导
  - 1 计算forward(z)轴 Vector3 forward = (mLook - mEye).Normorlize();
  - 2 计算right(x)轴 Vector3 right = Vector3::Cross(up, forward);
  - 3 计算up(y)轴 Vector3 up = Vector3::Cross(forward, right);
  - 4 移动到Eye点

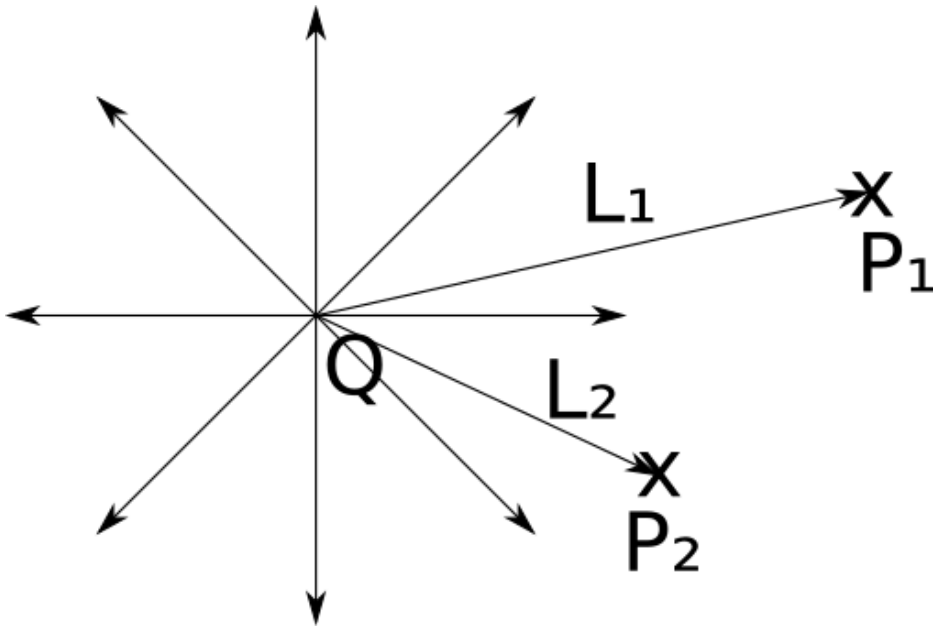
- 几何阶段(The Geometry Stage)




  - 模型变换(Model Transform)
    - 直接定义一个在世界空间下的三角形，或者长方体
    - World Transform：单位矩阵
  - 视图变化(View Transform)
    - http://in2gpu.com/2015/05/17/view-matrix/
    - 由摄像机矩阵推导出视图矩阵

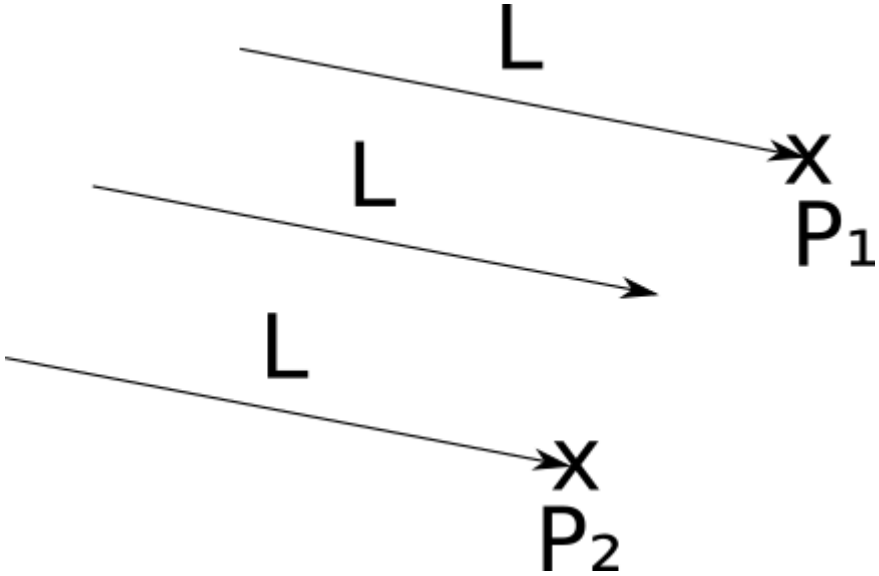
```
Matrix4D CreateLookAtRH(const Vector3& eye, const Vector3& lookat, const Vector3& upaxis)
{
    Vector3 zaxis = (eye - lookat).Normalize();
    Vector3 xaxis = Vector3::Cross(upaxis, zaxis).Normalize();
    Vector3 yaxis = Vector3::Cross(zaxis, xaxis);

    float xeye = -Vector3::Dot(xaxis, eye);
    float yeye = -Vector3::Dot(yaxis, eye);
    float zeye = -Vector3::Dot(zaxis, eye);

    return Matrix4D(
        xaxis.x, yaxis.x, zaxis.x, 0.0f,
        xaxis.y, yaxis.y, zaxis.y, 0.0f,
        xaxis.z, yaxis.z, zaxis.z, 0.0f,
        xeye, yeye, zeye, 1.0f);
}
```

- 顶点着色(Vertex Shading)
  - 光照模型https://www.gabrielgambetta.com/computer-graphics-from-scratch/light.html
    - 光源(Light sources)
      - 点光源(Point lights)




```
light {
  type = point
  intensity = 0.6
  position = (2, 1, 0)
}
```

        - 位置
        - 强度
      - 直射光源(Directional lights)




```
light {
  type = directional
  intensity = 0.2
  direction = (1, 4, 4)
}
```

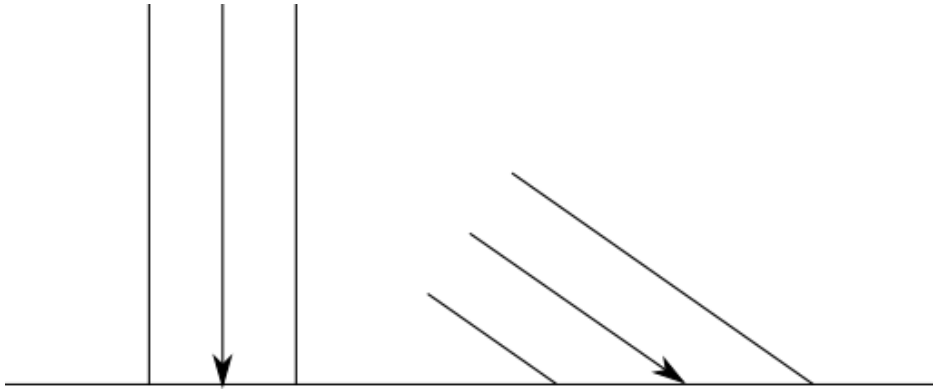
- 方向
- 强度

- 环境光(Ambient light)

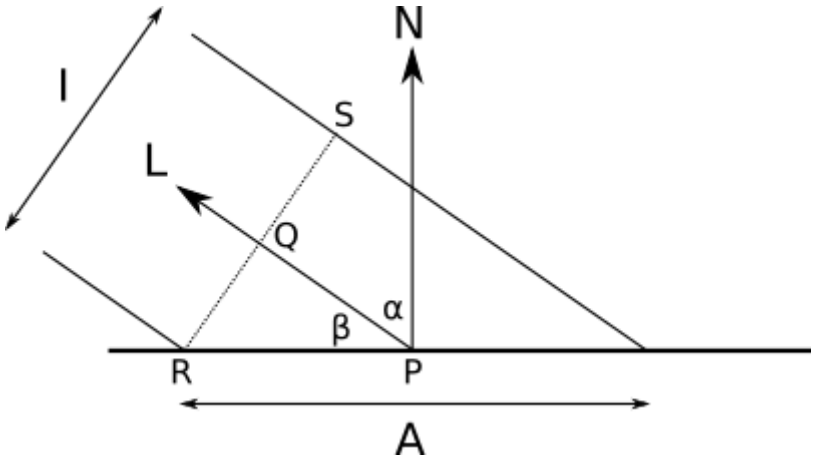
```
light {
  type = ambient
  intensity = 0.2
}
```

- 强度

- 光照计算(Illumination of a single point)
  - 漫反射(Diffuse reflection)



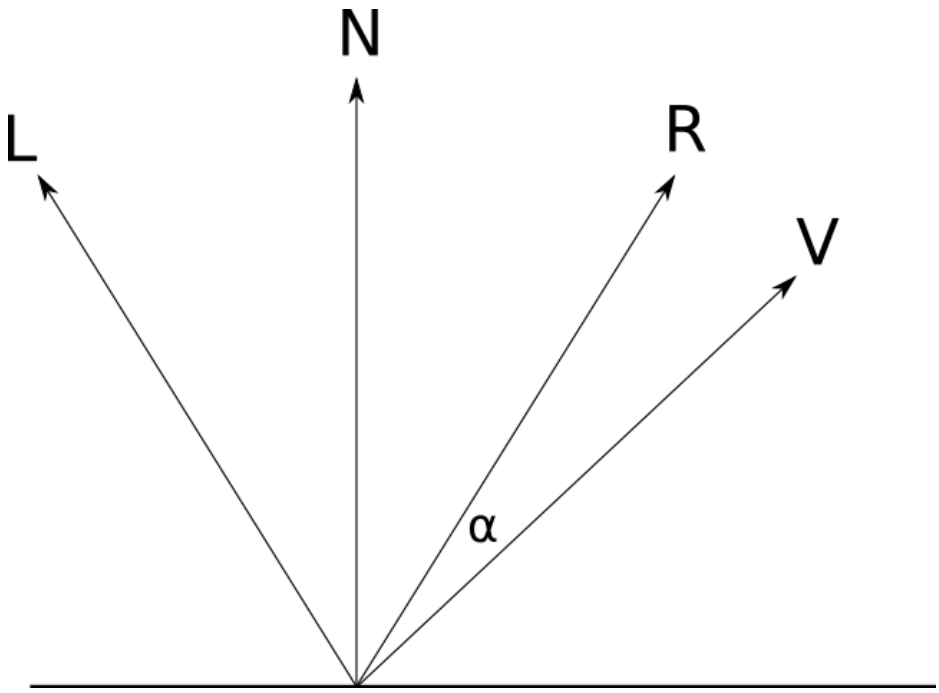
- 光的方向L，光的强度I，平面的法线N




- 公式

$$I_P = I_A + \sum_{i=1}^{n} I_i \frac{\langle \vec{N}, \vec{L_i} \rangle}{|\vec{N}||\vec{L_i}|}$$

- 镜面反射(Specular reflection)

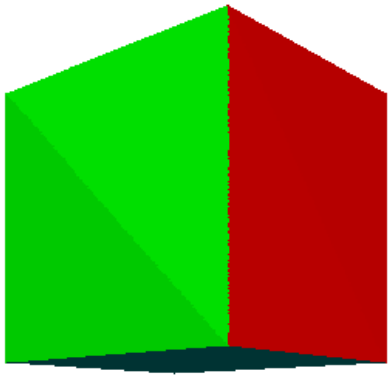


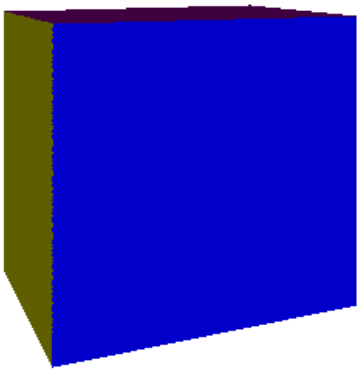

$$\vec{R} = 2\vec{N}\langle \vec{N}, \vec{L} \rangle - \vec{L}$$

$$I_S = I_L \left( \frac{\langle \vec{R}, \vec{V} \rangle}{|\vec{R}||\vec{V}|} \right)^s$$

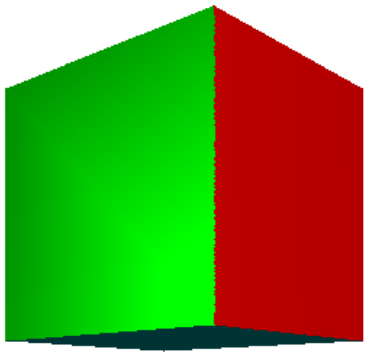
```
ComputeLighting(P, N, V, s) {
    i = 0.0
    for light in scene.Lights {
        if light.type == ambient {
            i += light.intensity
        } else {
            if light.type == point
                L = light.position - P
            else
                L = light.direction

            # Diffuse
            n_dot_l = dot(N, L)
            if n_dot_l > 0
                i += light.intensity*n_dot_l/(length(N)*length(L))

            # Specular
            if s != -1 {
                R = 2*N*dot(N, L) - L
                r_dot_v = dot(R, V)
                if r_dot_v > 0
                    i += light.intensity*pow(r_dot_v/(length(R)*length(V)), s)
            }
        }
    }
    return i
}
```

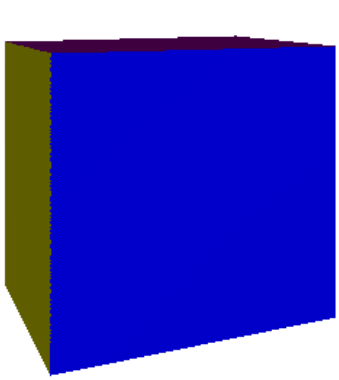
- 着色模型
  - https://www.gabrielgambetta.com/computer-graphics-from-scratch/shading.html
  - 平滑着色(Flat shading)
    - 计算三角形的一个顶点的颜色值，然后用这个值对整个三角形着色

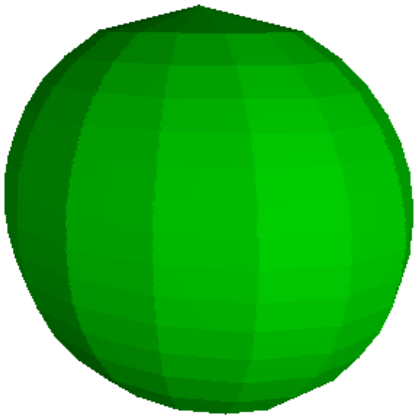


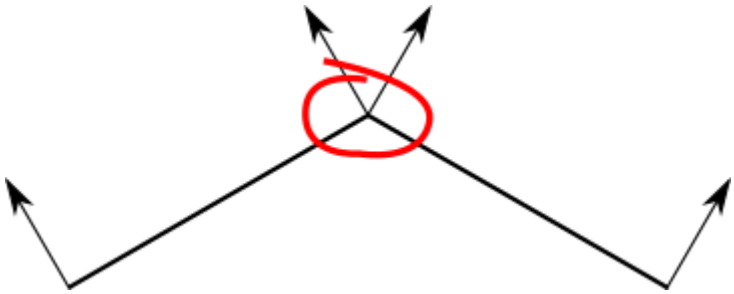


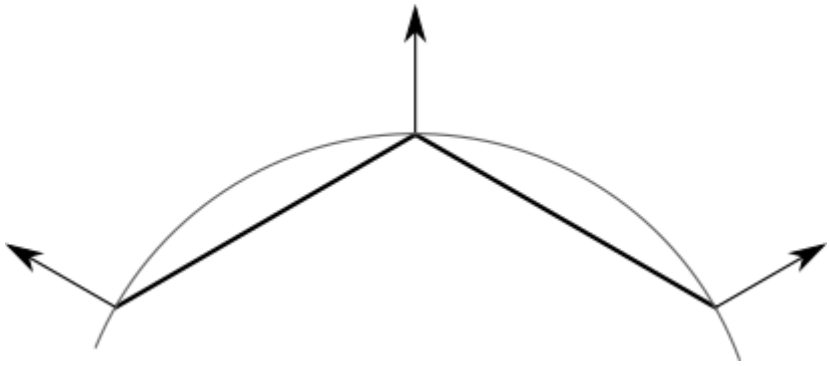
- 高洛德着色(Gouraud shading)
  - 逐顶点光照(Vertex lighting)
    - 计算三个顶点的颜色，然后对每个三角形内的像素进行线性插值得到每一个像素的颜色

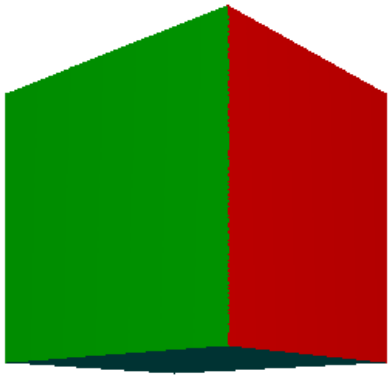


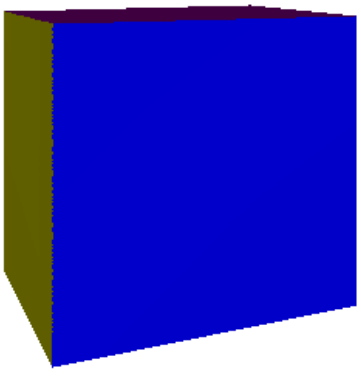


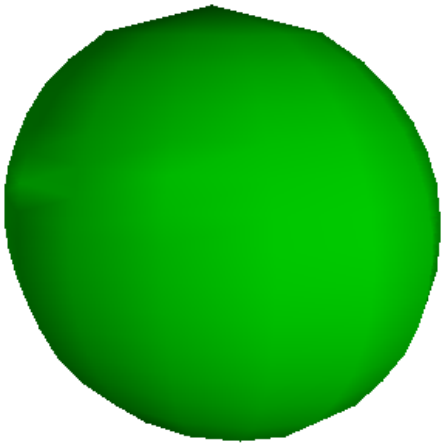


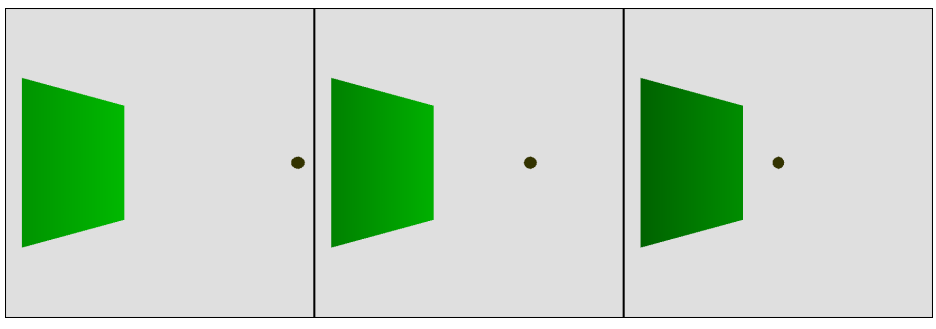


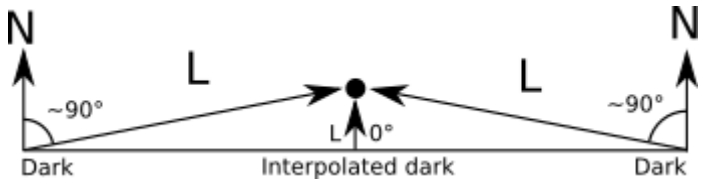




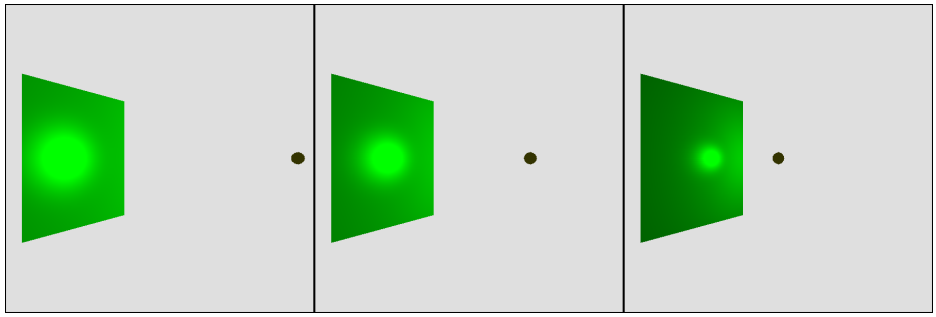






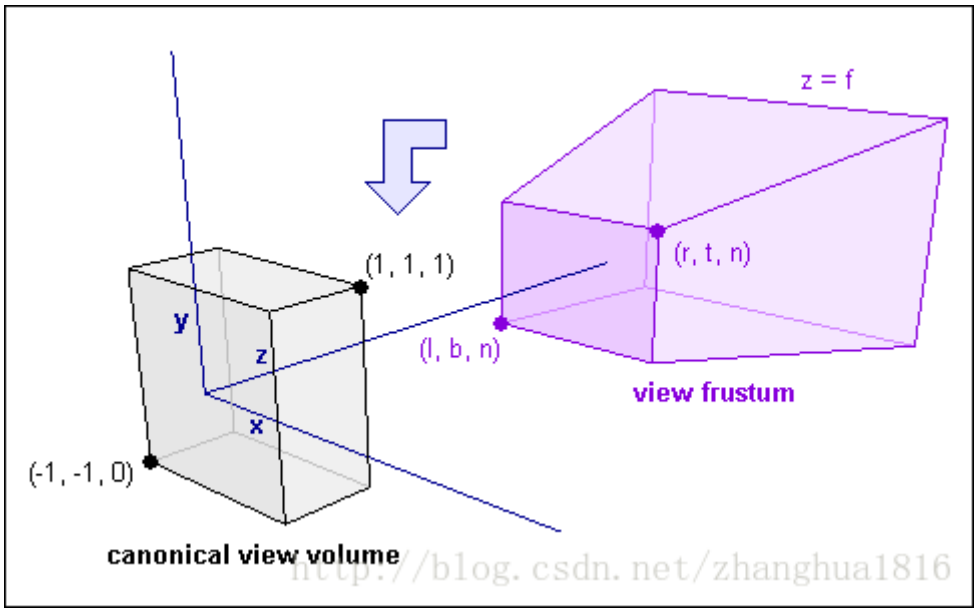


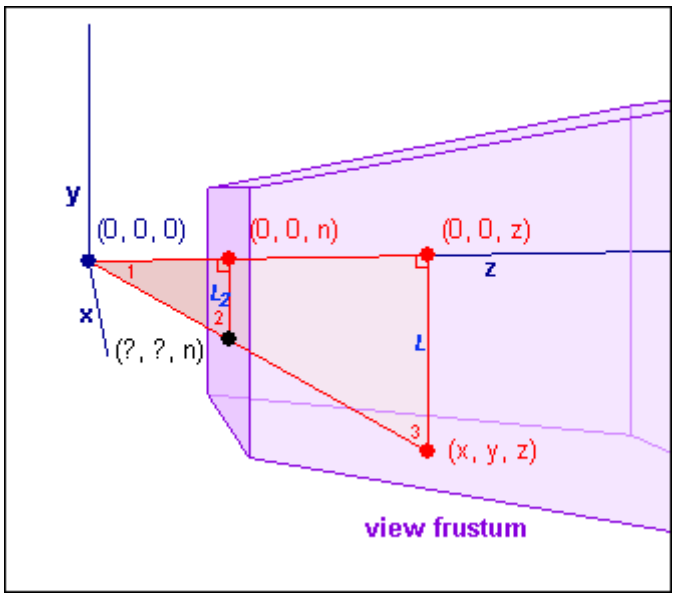

- 冯氏着色(Phong shading)
  - 逐像素光照(Per-pixel lighting)



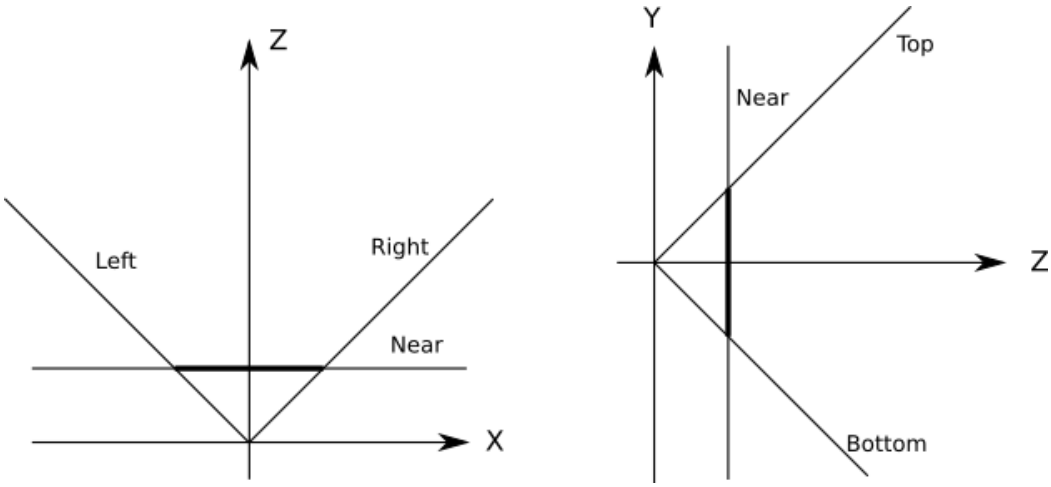
    - 光的方向L
    - 反射方向V
    - 法线V

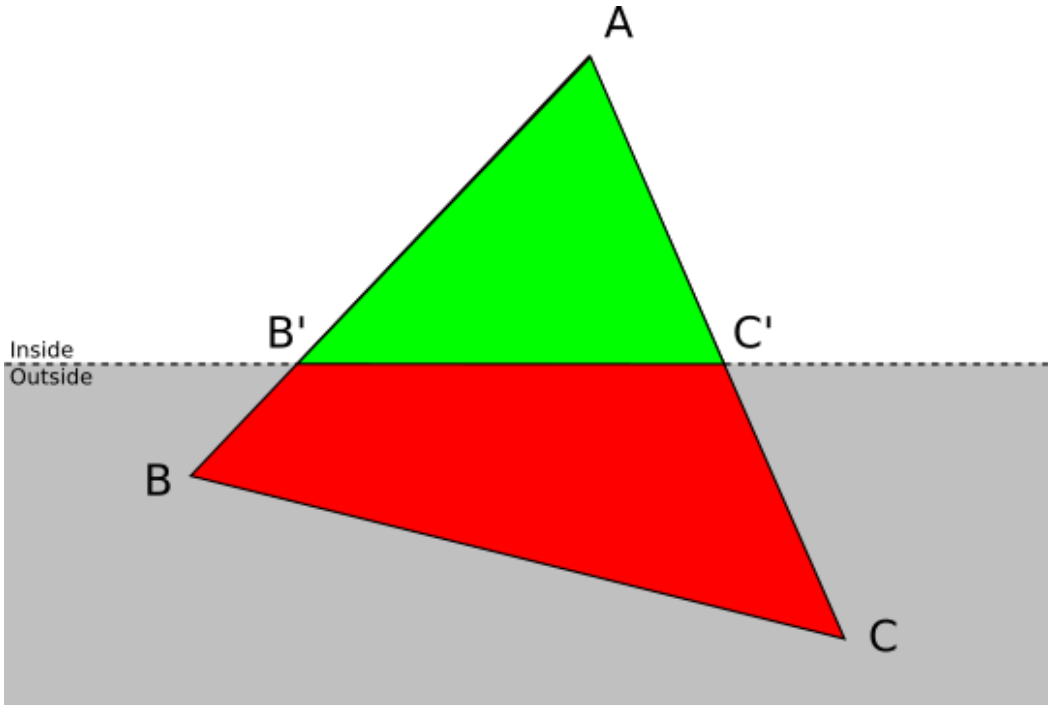
- 投影(Projection)
  - https://blog.csdn.net/weixin_33817333/article/details/86340159
  - http://www.codeguru.com/cpp/misc/misc/math/article.php/c10123__1/Deriving-Projection-Matrices.htm
  - 透视投影

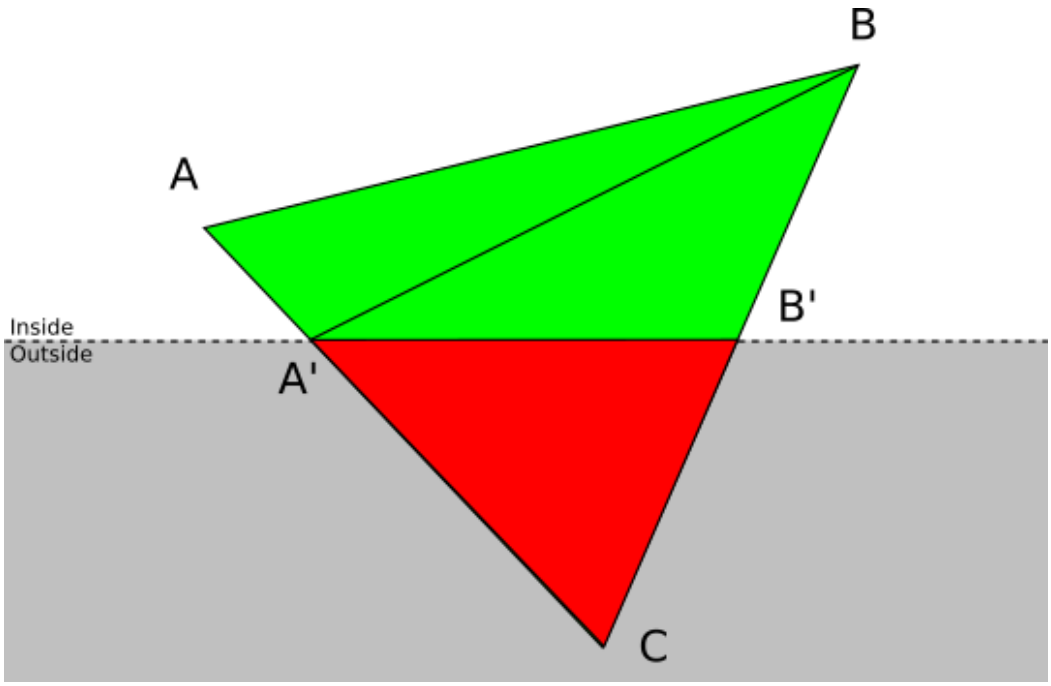





- 裁剪(Clipping )
  - https://www.gabrielgambetta.com/computer-graphics-from-scratch/clipping.html
  - https://www.gabrielgambetta.com/computer-graphics-from-scratch/hidden-surface-removal.html
  - 视锥裁剪

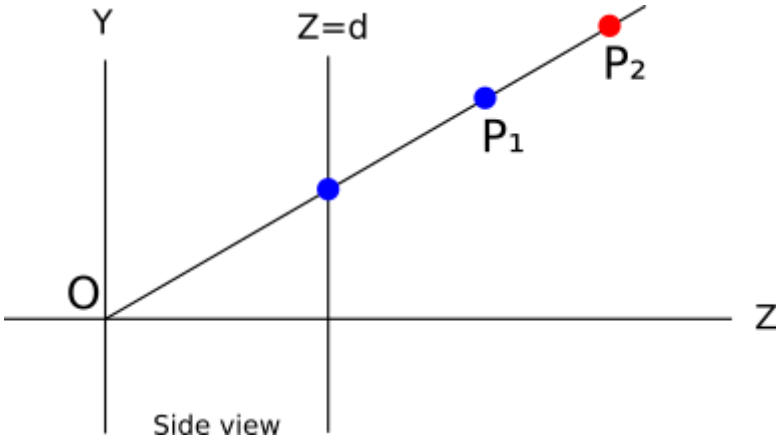




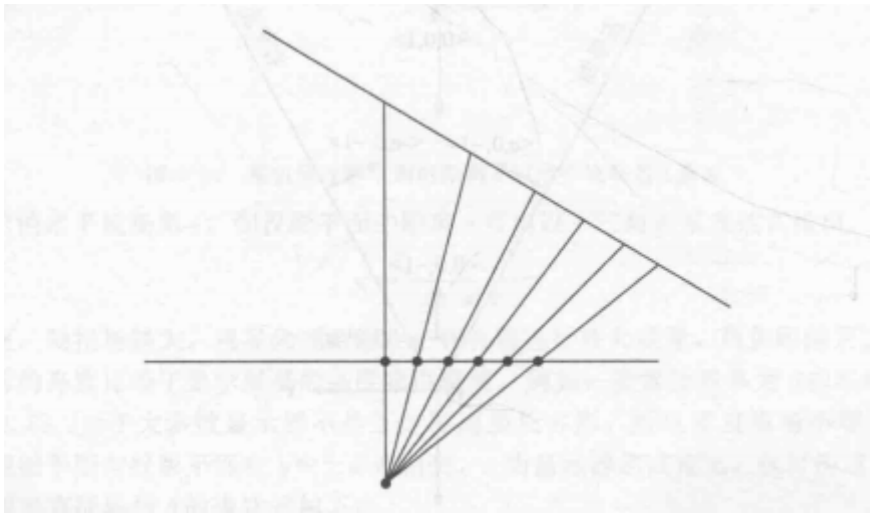




- CVV裁剪

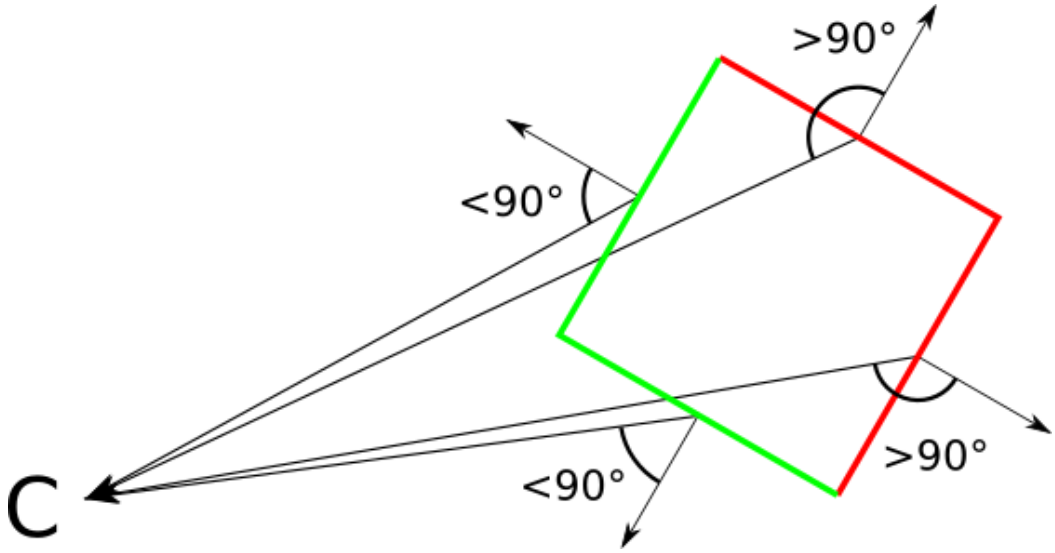
- 隐藏面消除(Hidden surface removal)
  - 深度缓冲



- 为每个像素点保存一个深度值，在为其填充颜色时比较其深度值来决定是否要将此像素填充成指定的颜色
- 在三角形内部的每个像素点的深度值由顶点的深度值插值得到
- 对1/z插值

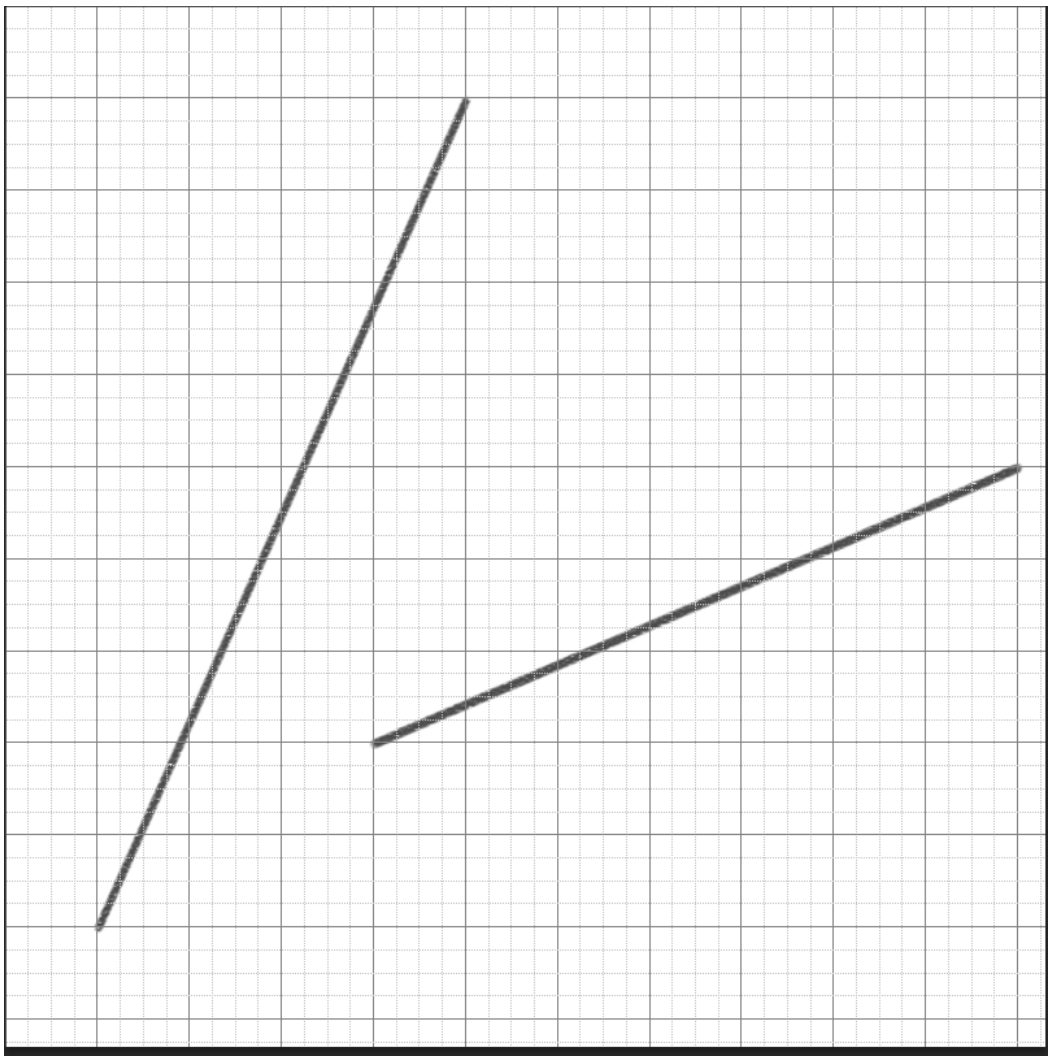


- 背面剔除

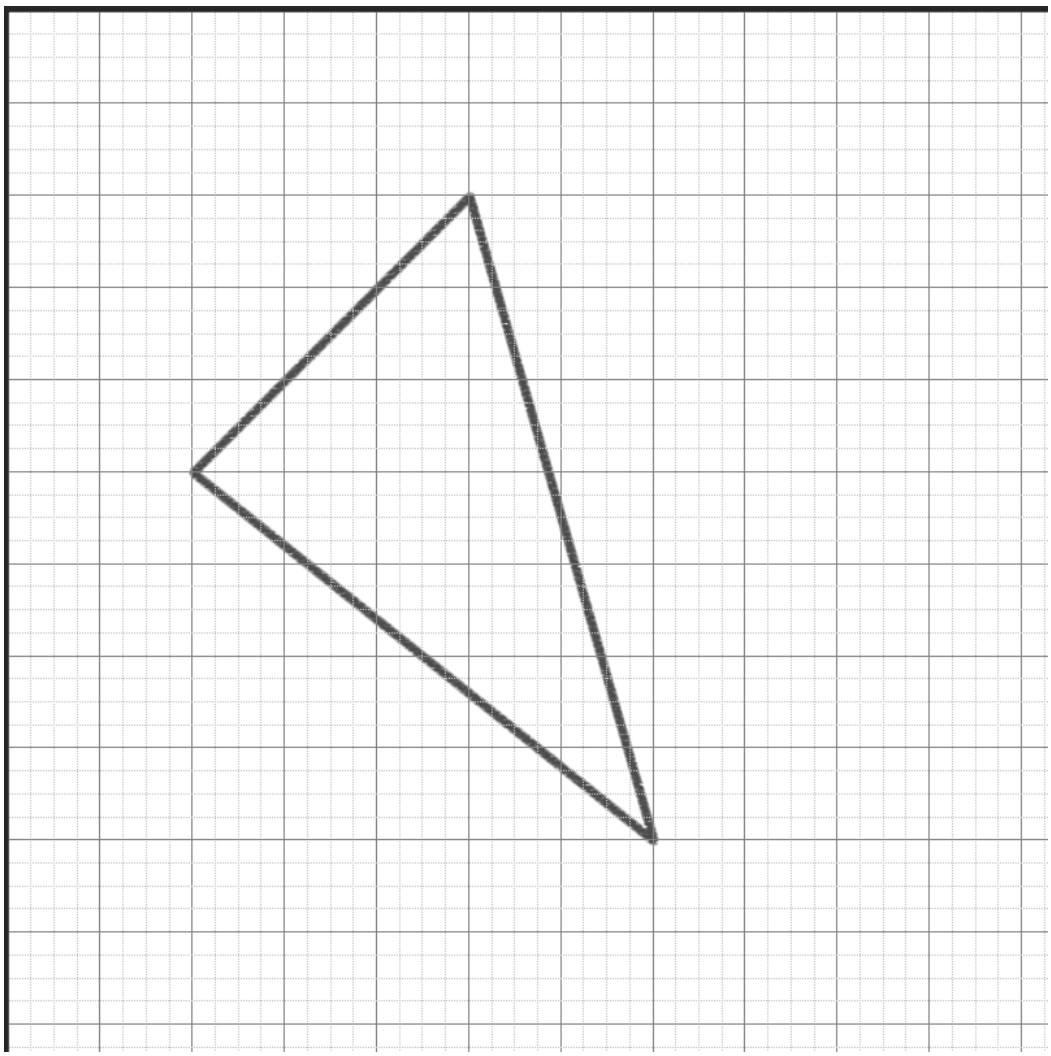



- 屏幕映射(Screen Mapping)

- 光栅化阶段(The Rasterization stage)
  - 画线
    - Ax+by+C= 0
    - https://www.gabrielgambetta.com/computer-graphics-from-scratch/lines.html

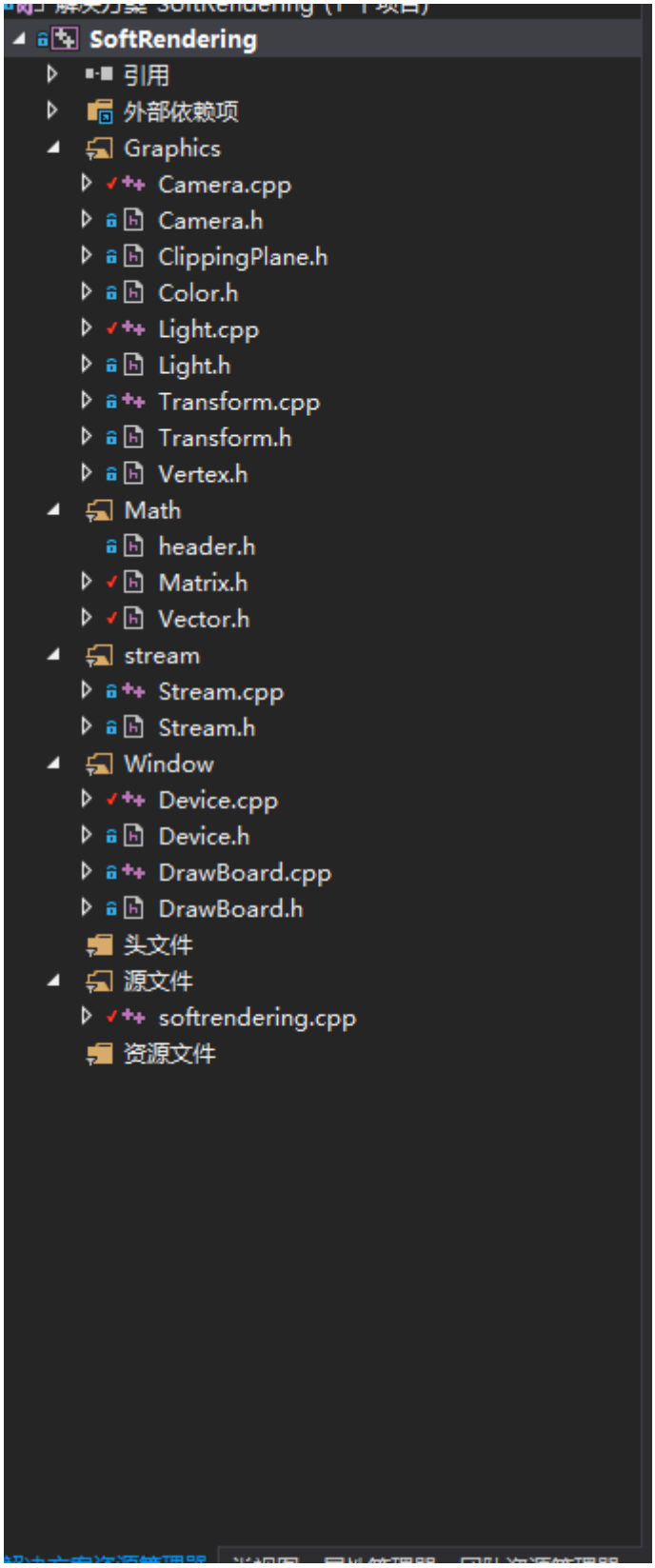


- 画三角形
  - https://www.gabrielgambetta.com/computer-graphics-from-scratch/filled-triangles.html



- 贴图(Textures)
  - https://www.gabrielgambetta.com/computer-graphics-from-scratch/textures.html
  - 每个顶点有一对uv值，插值得到每个像素的uv
  - Pixel(u(w−1),v(h−1))

- https://github.com/sherererlock/SoftRendering
-


SoftRendering
引用
外部依赖项
Graphics
Camera.cpp
Camera.h
ClippingPlane.h
Color.h
Light.cpp
Light.h
Transform.cpp
Transform.h
Vertex.h
Math
header.h
Matrix.h
Vector.h
stream
Stream.cpp
Stream.h
Window
Device.cpp
Device.h
DrawBoard.cpp
DrawBoard.h
头文件
源文件
softrendering.cpp
资源文件