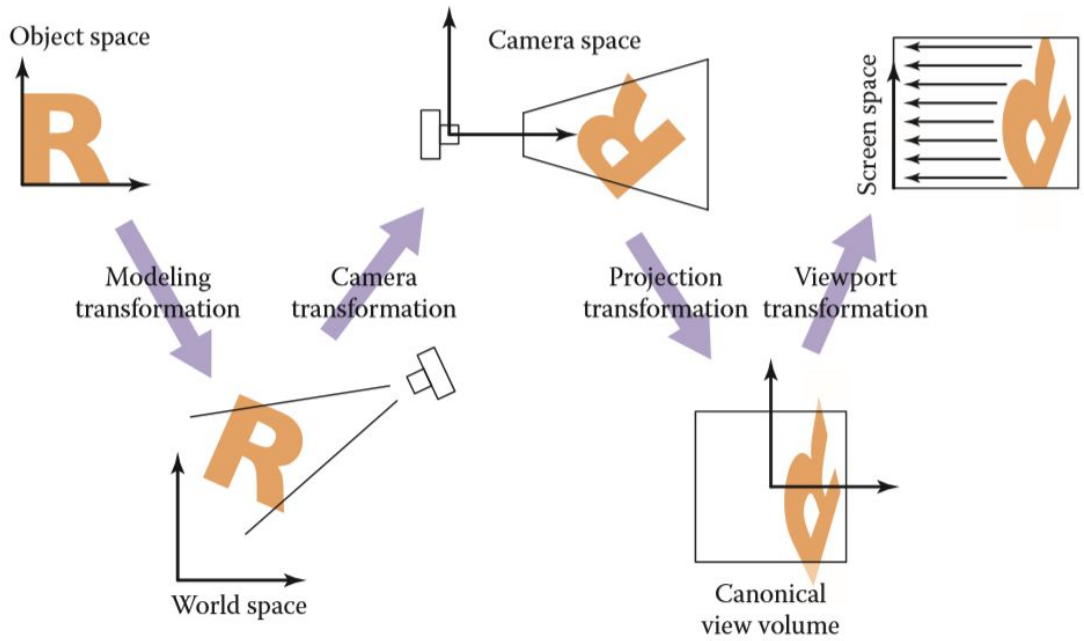
作业3

思来想去，还是决定说明一下代码框架，以及整个渲染器是怎么工作的，这对于理清笔者的思路也有帮助

代码框架说明

- 从main()函数开始，首先设定了观察模型的角度angle为140°，然后读取模型数据，存储每个三角形的顶点坐标，顶点法线坐标，对应的纹理坐标，再将三角形存储在一个三角形列表中
- 然后初始化渲染器，设定屏幕像素宽为700，高为700，设置相关路径，初始化默认采用phong模型渲染
- 当接收到相应的命令行指令，设置对应的shader函数、MVP矩阵
- 随后调用draw()函数进行图形的绘制；在draw()函数中，先得到了三角形的每个顶点在视线空间 (经过了model transformation和view transformation) 的坐标，然后进行点的mvp变换，并化为齐次坐标，而对于法线则不相同，因为模型存储的顶点的法线是相对于切线空间而言的，要变换到世界空间的法线，则要进行mv逆矩阵的变换，最后对顶点进行视口变换



- 最后调用rasterize_triangle()函数进行三角形的渲染，先创建三角形的二维bounding box，遍历此bounding box内的所有像素 (使用其整数索引)，然后利用三个叉乘判断该像素中心是否位于三角形内，如果在内部，通过重心坐标插值得到α、β、γ值 (此

时得到的这三个值是经过投影变换的，而投影变换并不保证重心坐标的一致性，所以需要通过公式矫正得到投影变换前的重心坐标，原理见以下链接)，通过这三个值，对该像素的深度进行插值，如果其深度大于深度缓冲的深度值，则该像素能被摄像机看见，所以继续进行相应属性的插值，便可以得到顶点的颜色、法线、纹理坐标、view space坐标

https://www.cs.ucr.edu/~craigs/courses/2018-fall-cs-130/lectures/perspective-correct-interpolation.pdf

- 随后调用对应的shader，进行相关的计算，得到最后的颜色值，然后进行对该元素的渲染

基础部分

基础部分的代码编写并不难，不过有一些知识有些朝纲，但是老师很贴心地把思路写在了代码注释里，按照老师思路来进行便没有什么错误，这部分代码就不贴出来了

需要说明的细节

- 在Blinn-Phong反射模型中的镜面反射公式中需要一个指数p，添加该项的原因很直接，因为离反射光越远就越不应该看见反射光，需要一个指数p加速衰减

提高部分

双线性插值

双线性插值的实现并不难，代码如下：

```
Eigen::Vector3f getColorBilinear(float u, float v)
    {
        if(u<0) u=0;
        if(u>1) u=1;
        if(v<0) v=0;
        if(v>1) v=1;
        float u_img = u * width;
        float v_img = (1-v) * height;

        auto u_min = std::floor(u_img);
        auto u_max = std::min((float)width,std::ceil(u_img));
        auto v_min = std::floor(v_img);
        auto v_max = std::min((float)height,std::ceil(v_img));

        auto A00 = image_data.at<cv::Vec3b>(v_max, u_min);
        auto A01 = image_data.at<cv::Vec3b>(v_max, u_max);
        auto A10 = image_data.at<cv::Vec3b>(v_min, u_min);
        auto A11 = image_data.at<cv::Vec3b>(v_min, u_max);
```

```
        float w1 = (u_img-u_min)/(u_max-u_min);
        float w2 = (v_img-v_min)/(v_max-v_min);

        auto bot = (1-w1) * A10 + w1*A11;
        auto top = (1-w1) * A00 + w1*A01;
        auto color = (1-w2)*bot+w2*top;

        return Eigen::Vector3f(color[0], color[1], color[2]);
    }
```

需要注意的一些细节

- z轴的正负问题，这个在作业2已经有了详细的说明
- 关于法线空间、切线空间的相关术语知识

图形学 | Shader |用一篇文章理解法线变换、切线空间、法线贴图

在做blinn-phong光照计算时，法线信息必不可少。而在shader中，可以得到 app 传入 vertex shader的模型空间的的法线信息（一个顶点含有一个法向量），但是计算时往往需要将法线从模型空间转换为世界空间，同时

知 https://zhuanlan.zhihu.com/p/261667233#:~:text=%E5%88%87%E7%BA%BF%E7%A9%BA%E9%97%B4%E6%98%AF%E4%BD%8D%E4%BA%8E%E4%B8%89%E8%A7%92%E5%BD%A2%E8%A1%A8%E9%9D%A2%E4%B9%8B%E4%B8%8A%E7%9A%84%E7%A9%BA%E9%97%B4%EF%BC%8C%20%E5%88%87%E7%BA%BF%E7%A9%BA%E9%97%B4%E4%B8%AD%E7%9A%84xyz%E8%BD%B4%E5%88%86%E5%88%AB%E6%98%AFt%E8%BD%B4%EF%BC%88%E5%88%87%E7%BA%BF%E6%96%B9%E5%90%91%EF%BC%89%E3%80%81b%E8%BD%B4%EF%BC%88%E5%89%AF%E5%88%87%E7%BA%BF%E6%96%B9%E5%90%91%EF%BC%89%E5%92%8Cn%E8%BD%B4,%E3%80%82%20n%E8%BD%B4%E4%BB%A3%E8%A1%A8%E7%9A%84%E6%98%AF%E8%AF%A5%E7%82%B9%E7%9A%84%E6%B3%95%E7%BA%BF%E6%96%B9%E5%90%91%EF%BC%8C%E5%9B%A0%E6%AD%A4%E7%A1%AE%E5%AE%9A%E7%9A%84%E4%B8%80%E7%82%B9%E6%98%AF%EF%BC%9A%E5%88%87%E7%BA%BF%E7%A9%BA%E9%97%B4%E7%9A%84z%20%E8%BD%B4%E6%AD%A3%E6%96%B9%E5%BD%A2%E4%B8%8E%E6%B3%95%E7%BA%BFn%E5%90%8C%E5%90%91%E3%80%82

- 在bump mapping的实现中，只修改了法线；而在displacement mapping中，不仅修改了法线，还修改了顶点的位置，还考虑了光照