# 概述

软渲染器主要有三个板块，BaseType, Model和Render。BaseType主要是维护向量和矩阵的数据结构，矩阵和向量之间的数学运算。

## 1.BaseType:

用到了c++模板，运算符重载没有遇到什么问题。

但是由于我网上找的参考代码对标的是directX，它使用的是左手坐标系，最开始没有注意到这个问题，直接借用其旋转矩阵和透视投影矩阵的生成方式。出错。

发现问题之后，因为这部分是整个渲染结果的基础，因此想找一个真的可以运行的代码做参考，便借鉴opengl中用的数学计算库，glm。虽然它是右手坐标系，但是其矩阵的存储方式与一般的手写不一样，它是列主序，这部分搞清楚了也就解决了。

这里没有使用任何virtual继承和virtual方法。前两天看《深入理解c++对象模型》，知道一个*Plain* old *data* structure运行时，没有额外时间和空间开销，所以在效率方面就没那么多顾虑了。

旋转矩阵直接使用的四元数转换为旋转矩阵的公式。

**优化问题：** 这里我只是使用了最简单的循环做数学运算，但是glm代码中应该使用了SIMD加速技术，有时间可以考虑优化。

# 2. Model:

使用Obj文件格式，这里就是一个文本文件的读取，处理，比较繁琐。

Model提供一个迭代器，该迭代器每次指向一个维护三角形的数据结构。这个类就是就是提供每个三角形的position、normals和texture coordinate。

# 3.Render:

### 私有域：

1. 四个变换矩阵，mat_*
2. vector<vector>维护的z-buffer
3. 渲染结果的Image数据结构

### 方法：

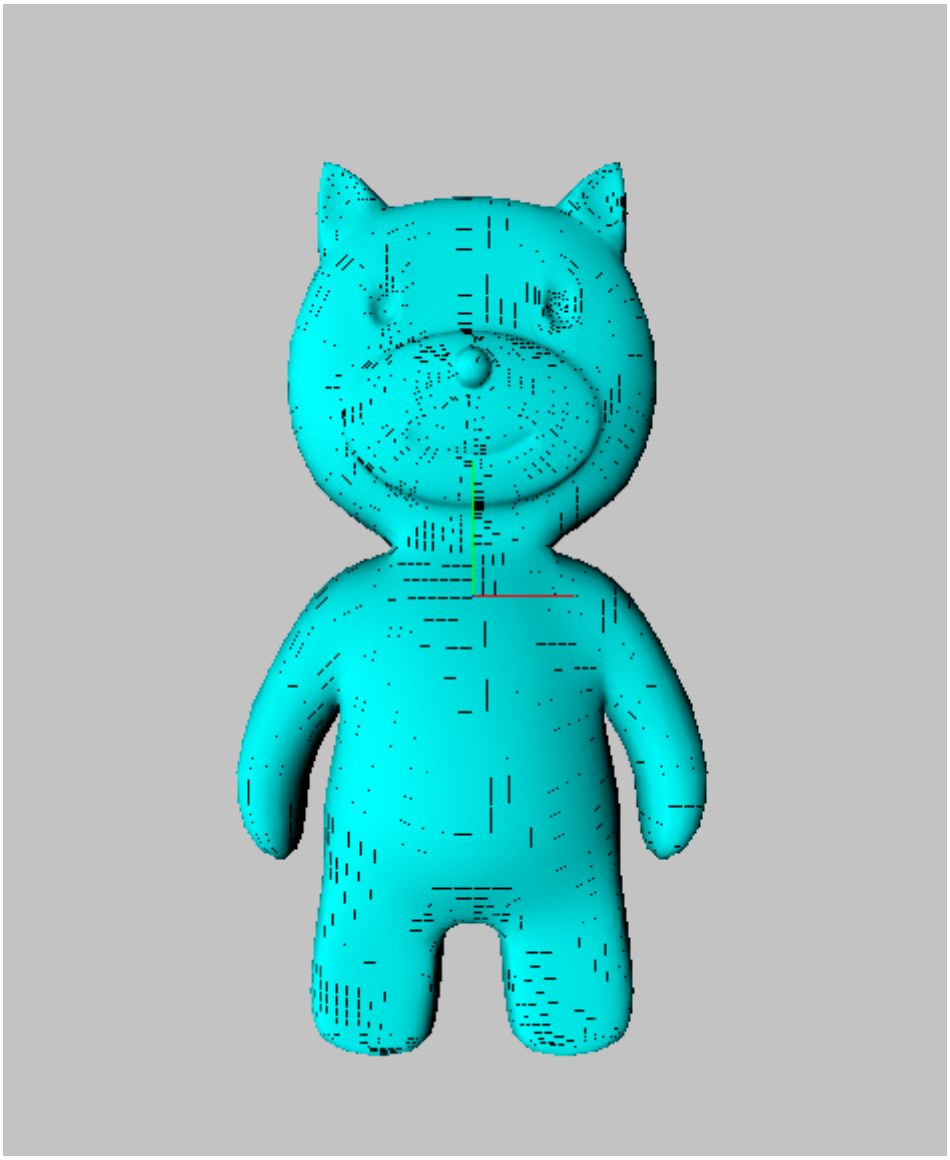
1. shader 方法。由model获得的每一个三角形做如下计算：
    - 对于每一个三角形个每一个顶点，position数据先做mvp然后做透视除法，再变换到屏幕空间（x, y）;normal数据根据光照模型所在空间（有时候可能在相机空间做一写光照上的计算）计算。每一个顶点还要存放深度信息。
    - 由三角形的屏幕可以得到再屏幕上包围这个三角形的一个矩形。
    - 颜色和光照只用了在world空间一个平行光照，计算颜色强度。
2. draw_primitive 插值出图像：
    - 使用插值方法画出三点组成的三条线，其实也就是模型的'边框'。
    - 计算shader方法得到的屏幕坐标围成的面积，如果为零，直接返回。

- 迭代包围三角形矩阵里每一个像素（屏幕坐标）,先用重心坐标计算出权重，然后不处理不在三角形之中的点。再用重心坐标插值出深度，颜色。深度测试通过便覆盖写入。

3. 暂时先在shander计算出三角形的深度，颜色信息，之后考虑在这个方法里验证其他的光照模型。

而draw_primitive方法就插值出三角形内每一个像素点的像素值。

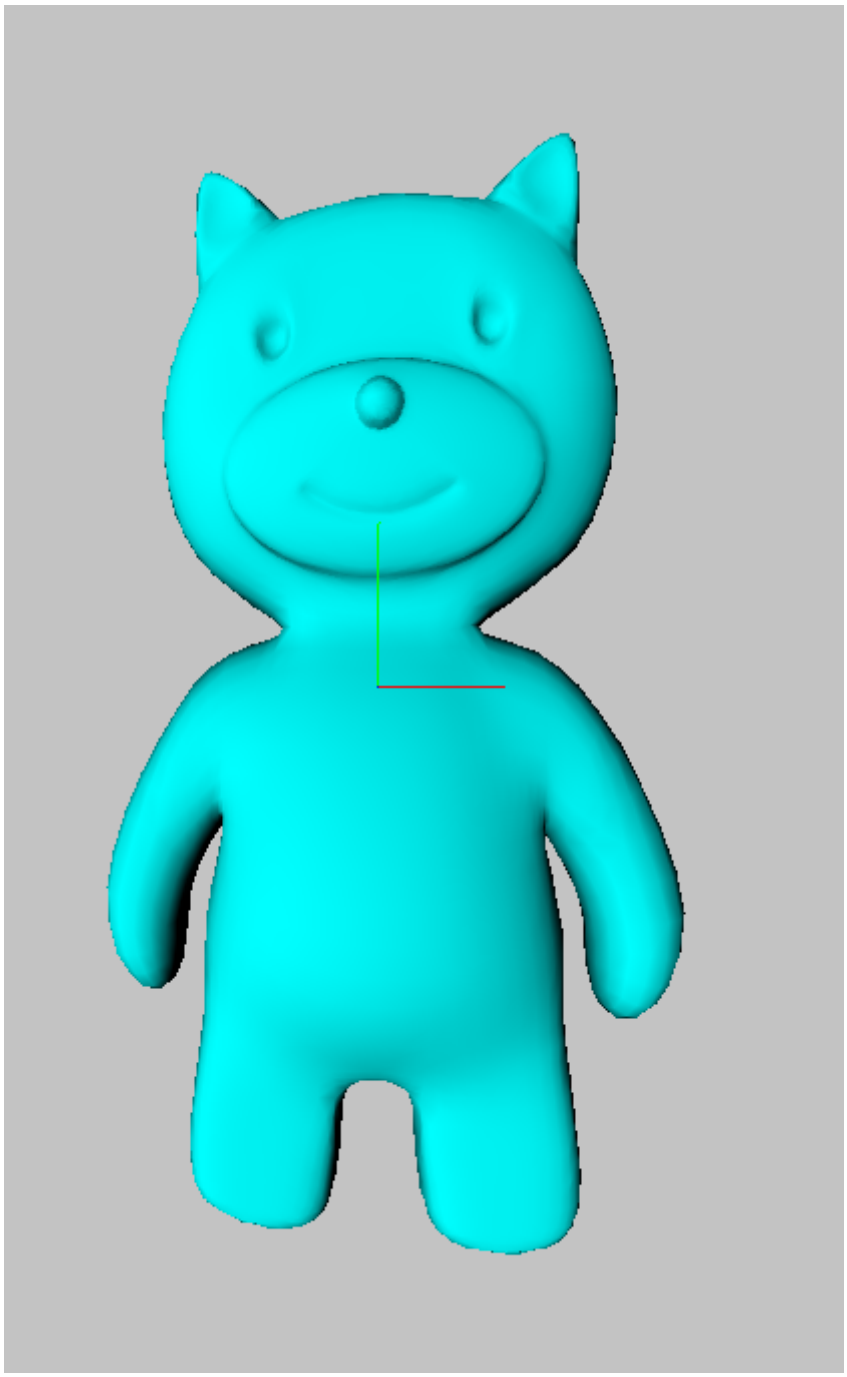
**坑点：**



之前的渲染效果一直是这样，问题出在重心坐标插值的时候，对于出现在三角形的边上的点，会计算出一个0，虽然数学模型上计算出零没有问题，但是在计算机中，如果p3 = 1- p1-p2;而p1+p2=0。那么p3是一个非常接近但是不为0.0的一个浮点数，这样直接用p3>=0这样的条件判断就会跟预期不符合。之前的解决方案是对于三个边框，为其实现一个draw_line方法。更快的解决方案就是对p3做一些处理

```
float p1, p2, p3;
auto a_pair = braycentic_coord(fragments);
p1 = a_pair.first;
p2 = a_pair.second;
p3 = 1 - p1 - p2;
p3 = abs(p3) < 1e-3 ? 0.0f : p3; //重新加的代码，1e-3是实验数据，1e-5还是有黑点
if (p1 >=0 && p2 >=0 && p3 >= 0)
{}
```

这样之后可以实现



## 4.Next:

1. 使用一些颜色贴图和法线贴图。然后实现经典的光照模型，实现对透明物体的渲染。在考虑要不是在复现一下软光线追踪?
2. 理论上，学习pbr渲染。