PA1 实验报告

代码逻辑

光线发射

我在`camera.hpp`中模拟了光线从相机发射的逻辑。

1. 计算光线发射点。从屏幕中心向屏幕朝向的反方向移动一段距离，使得上下视角恰为传入的angle参数，这就是光线的发射点。
2. 发射光线。计算光线发射点到屏幕上坐标为`[x, y]`的点的方向，就是光线发射的方向。

```
Ray PerspectiveCamera::generateRay(const Vector2f &point) override {
    float distance = height / 2 / tan(angle / 2);
    Vector3f rayDirection = distance * direction + (point.x() - width / 2)
* horizontal + (point.y() - height / 2) * up;
    return Ray(center, rayDirection.normalized());
}
```

光线求交

1. 平面求交。使用直接矩阵计算解析解来实现。
2. 球体求交。使用几何法，分内部和外部两种情况讨论。
3. 三角形求交。先进行平面求交，再用Cramer定理判断交点是否在三角形内部。
4. 组求交。对组里的物体挨个求交即可。

Phong模型

```
Vector3f Shade(const Ray &ray, const Hit &hit,
               const Vector3f &dirToLight, const Vector3f &lightColor) {
    Vector3f N = hit.getNormal();
    Vector3f L = dirToLight.normalized();
    Vector3f V = -ray.getDirection();
    Vector3f R = N * 2 * Vector3f::dot(N, L) - L;

    // diffuse shade
    float diffuseFactor = std::max(0.0f, Vector3f::dot(N, L));
    Vector3f diffuseShade = diffuseColor * lightColor * diffuseFactor;

    // specular shade
    float specFactor = std::max(0.0f, Vector3f::dot(R, V));
    // float specFactor = std::max(0.0f, Vector3f::dot(H, N));
    Vector3f specularShade = specularColor * lightColor * pow(specFactor,
shininess);

    // ambient color neglected
    Vector3f shaded = diffuseShade + specularShade;
```

```
    return shaded;
}
```

遇到的问题

1. 我一开始修改了`ray.hpp`，在其中将光线的方向进行了标准化，这样就避免了光线求交的时候每次都要标准化。但是这会导致`transform.hpp`中对光线的变换错误（调试了3天才发现）。
2. 我一开始忽略了光从内部与球体相交的情况，但是这好像没有导致什么错误。不过我还是改过来了。
3. 课上讲，Phong模型一般不计算`R * V`而是使用`H * N`来代替，但是这次PA中这样错会被判错。

代码参考

1. 除了`sphere.hpp`和`ray.hpp`，其他代码由我独立完成，其中平面求交、Phong模型等完全使用课上讲的算法。
2. 我的“复杂兔子”一直不太对，因此我让GPT帮我检查了我的所有代码。`sphere.hpp`中没有考虑到内部相交的错误就是它发现的。尽管这并没有解决问题。
3. 经过在群聊里询问，我发现王钰琨同学“复杂兔子”的输出和我一模一样，因此我参考了他解决问题的方法，并由此发现了我的`ray.hpp`存在问题。