glm::lookAt 函数

glm::lookAt() 函数用于构建视图矩阵（View Matrix），将世界坐标系中的点转换到相机坐标系。以下是其数学推导过程的分步解释：

1. 输入参数

函数原型为：

```
glm::mat4 lookAt(
    glm::vec3 eye,    // 相机位置
    glm::vec3 center, // 目标点（相机对准的位置）
    glm::vec3 up      // 上方向向量（通常为(0,1,0)）
);
```

2. 坐标系定义

- 相机坐标系：相机位于原点，看向 $-z$ 方向，x 向右，y 向上（右手坐标系）。
- 目标：将世界坐标系中的点变换到相机坐标系。

3. 推导步骤

(1)计算前向向量（Forward Direction）

从相机位置指向目标点的方向向量，并归一化：

$$\vec{f} = \left\| \overrightarrow{\text{center}} - \overrightarrow{\text{eye}} \right\|$$

在相机坐标系中，前向方向为 $-z$，因此实际使用的前向向量是 $-\vec{f}$。

(2) 计算右向量（Right Direction）

使用叉乘计算相机的右方向（x 轴）：

$$\vec{r} = \left\| \vec{f} \times \overrightarrow{\text{up}} \right\|$$

注意：叉乘顺序需符合右手法则（OpenGL 使用右手系）。

(3) 计算上向量（Up Direction）

通过右向量和前向向量重新计算正交化的上方向：

$$\vec{u} = \left\| \vec{r} \times \vec{f} \right\|$$

(4) 构造旋转矩阵

将相机坐标系对齐到世界坐标系。旋转矩阵的列向量为相机的右、上、前方向：

$$R = \begin{bmatrix} \vec{r}_x & \vec{u}_x & -\vec{f}_x & 0 \\ \vec{r}_y & \vec{u}_y & -\vec{f}_y & 0 \\ \vec{r}_z & \vec{u}_z & -\vec{f}_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

由于前向向量实际是 $-\vec{f}$，因此在矩阵中使用负值。

(5) 构造平移矩阵

将相机位置平移到原点：

$$T = \begin{bmatrix} 1 & 0 & 0 & -\overrightarrow{\text{eye}}_x \\ 0 & 1 & 0 & -\overrightarrow{\text{eye}}_y \\ 0 & 0 & 1 & -\overrightarrow{\text{eye}}_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(6) 组合旋转和平移

视图矩阵是 先旋转再平移，即：

$$M_{\text{view}} = R \cdot T$$

展开后为：

$$M_{\text{view}} = \begin{bmatrix} \vec{r}_x & \vec{u}_x & -\vec{f}_x & -\vec{r} \cdot \overrightarrow{\text{eye}} \\ \vec{r}_y & \vec{u}_y & -\vec{f}_y & -\vec{u} \cdot \overrightarrow{\text{eye}} \\ \vec{r}_z & \vec{u}_z & -\vec{f}_z & \vec{f} \cdot \overrightarrow{\text{eye}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

其中平移分量通过点乘计算，将相机的世界坐标转换到新坐标系。

4. 代码实现

GLM 的实现与上述推导一致：

```
template<typename T, qualifier Q>
GLM_FUNC_QUALIFIER mat<4, 4, T, Q> lookAtRH(vec<3, T, Q> const& eye, vec<3, T, Q> const&
center, vec<3, T, Q> const& up)
{
  vec<3, T, Q> const f(normalize(center - eye));
  vec<3, T, Q> const s(normalize(cross(f, up)));
  vec<3, T, Q> const u(cross(s, f));

  mat<4, 4, T, Q> Result(1);
  Result[0][0] = s.x;
  Result[1][0] = s.y;
  Result[2][0] = s.z;
  Result[0][1] = u.x;
  Result[1][1] = u.y;
  Result[2][1] = u.z;
  Result[0][2] =-f.x;
  Result[1][2] =-f.y;
  Result[2][2] =-f.z;
  Result[3][0] =-dot(s, eye);
  Result[3][1] =-dot(u, eye);
  Result[3][2] = dot(f, eye);
  return Result;
}
}
```