作业一：三角形光栅化

代码的目录结构如下：

```
homework1/
├── core/
│   ├── __init__.py
│   ├── math_utils.py        # 数学工具（矩阵变换、四元数等）
│   ├── geometry.py          # 几何体定义（三角形类等）
│   └── rasterizer.py        # 光栅化器
├── examples/
│   ├── __init__.py
│   ├── basic_rendering.py # 基础渲染示例
│   ├── projection_demo.py # 投影对比示例
│   ├── antialiasing_demo.py # 抗锯齿示例
│   ├── depth_test_demo.py # 深度测试示例
│   └── rotation_demo.py     # 旋转插值示例
└── main.py                  # 主程序入口
```

给定三角形三个顶点 A,B,C 的三维坐标、相机视图矩阵、投影矩阵等参数，需实现三角形光栅化后的图片。具体，需完成以下任务：

基础实现 （9 分）

1. 实现函数 LookAt(eye, center, up)以构建相机的视图矩阵 (1 分)

   修改 core/ math_utils.py 里的 LookAt 函数的实现细节，给出正确的视图矩阵

```python
def LookAt(eye, center, up):
    """构建视图矩阵
    参数:
        eye: 相机位置
        center: 相机看向的位置
        up: 相机上方向
    返回:
        4x4视图矩阵
    """

    return np.eye(4)
```

2. 实现函数 Ortho(left, right, bottom, top, near, far)以构建相机的正交投影矩阵 (1 分)

修改 core/ math_utils.py 里的 Ortho 函数的实现细节，给出正确的正交投影矩阵

```python
def Ortho(left, right, bottom, top, near, far):
    """构建正交投影矩阵
    参数:
        left (float): 左边界，视锥体左侧平面的x坐标
        right (float): 右边界，视锥体右侧平面的x坐标
        bottom (float): 下边界，视锥体底部平面的y坐标
        top (float): 上边界，视锥体顶部平面的y坐标
        near (float): 近平面，视锥体近端平面到相机的距离 （正值）
        far (float): 远平面，视锥体远端平面到相机的距离 （正值， near < far)
    返回:
        4x4正交投影矩阵
    """


    return np.eye(4)
```

3. 实现函数 Perspective(fov, aspect, near, far)以构建相机的正交投影矩阵（1 分）

修改 core/ math_utils.py 里的 Perspective 函数的实现细节，给出正确的透视投影矩阵

```python
def Perspective(fov, aspect=1.0, near=0.1, far=10.0):
    """构建透视投影矩阵
    参数:
        fov (float): 垂直视野角度（Field of View Y），单位为度（degrees）
        aspect (float): 宽高比（aspect ratio），默认值为1.0
        near (float): 近平面，视锥体近端平面到相机的距离 （正值）
        far (float): 远平面，视锥体远端平面到相机的距离 （正值， near < far)
    返回:
        4x4透视投影矩阵
    """

    return np.eye(4)
```

4. 实现 inside 函数，判断点是否在三角形里（1 分）

修改 core/ geometry.py 里的 Triangle 类的 inside 函数的实现细节，判断点是否在三角形里

```python
    def inside(self, x, y, z):
        """判断点(x,y)是否在三角形内"""
        v0, v1, v2 = self.vertices

        return True
```

5. 实现 rotate_norm 函数，计算三角形绕其法线旋转后的新位置（2 分）

修改 core/ geometry.py 里的 Triangle 类的 compute_normal 和 rotate_norm 函数的实现细节。compute_normal 函数是计算三角形法线方向，请提供计算法向的实现细节。rotate_norm 函数是绕三角形法线方向旋转相应角度，更新三角形三个顶点的坐标，请更新函数里的旋转矩阵 R 的计算细节。

```python
    def compute_normal(self):
        """计算三角形的法线向量"""

        return np.array([0, 0, 1])

    def rotate_norm(self, theta):
        """绕法线旋转三角形"""
        center = np.mean(self.vertices, axis=0)
        normal = self.compute_normal()

        R = np.eye(3)

        for i in range(3):
            self.vertices[i] = center + R @ (self.vertices[i] - center)
```

6. 实现四元数相关功能函数 to_rotation_matrix 以及 slerp 插值函数（2 分）

   修改 core/ math_utils.py 里的四元数 Quaternion 类的 to_rotation_matrix 和 slerp 函数的实现细节。to_rotation_matrix 函数是将四元数转换成旋转矩阵的形式，slerp 函是用来做两个表示旋转的四元数的球面线性插值。

```python
    def to_rotation_matrix(self):
        """转换为3x3旋转矩阵"""
        self.normalize()

        return np.eye(3)

    @staticmethod
    def slerp(q1, q2, t):
        """球面线性插值"""
        q1.normalize()
        q2.normalize()


        return q1
```

7. 实现计算三角形重心坐标的函数 compute_barycentric。（1 分）

   修改 core/ geometry.py 里的 Triangle 类的 compute_barycentric 函数的实现重心坐标的计算，主要用于光栅化时颜色等的插值，在实现这个类的时候，可认为当前的三角形是投影至成像平面上的三角形，因此在计算重心坐标时可忽略 z 值坐标。

```python
    def compute_barycentric(self, x, y, z):
        """计算重心坐标"""
        v0, v1, v2 = self.vertices


        return np.array([1/3, 1/3, 1/3])
```

渲染要求 （11 分）

a) 透视投影图像和正交投影图像对比（1 分）

补全基础实现里的相关函数的具体实现，执行
python examples/projection_demo.py 命令，即可渲染出正交投影效果图片（orthographic_projection.png）和透视投影效果图片（perspective_projection.png）。

b) 三角形颜色插值：根据重心坐标进行颜色插值；三角形绕其自身中点旋转 45,90,135,180 度后的透视投影图像 。(3 分)

补全基础实现里的相关函数的具体实现，执行
python examples/basic_rendering.py 命令，即可渲染出颜色插值效果图片（basic_triangle.png）和三角形旋转不同角度效果图片（triangle_rotated_*.png）。

c) 旋转插值：渲染旋转动画的关键帧，包含位置和颜色插值（2 分）

补全基础实现里的相关函数的具体实现，执行
python examples/rotation_demo.py 命令，即可渲染出渲染旋转动画插帧效果图片（rotation_slerp_t_*.png）。

d) 深度测试：渲染两个远近三角形，考虑深度测试（2 分）

修改 core/ rasterizer.py 里的 Rasterization 类的_rasterize_standard 函数，利用 self.depth_buf 深度缓存，进行深度测试。当前的光栅化渲染并没有考虑深度缓存，请在当前的基础上修改，考虑深度测试。

```
116     def _rasterize_standard(self, t, min_x, max_x, min_y, max_y):
117         """标准光栅化"""
118         for y in range(min_y, max_y + 1):
119             for x in range(min_x, max_x + 1):
120                 if t.inside(x + 0.5, y + 0.5, 0):
121                     barycentric = t.compute_barycentric(x + 0.5, y + 0.5, 0)
122 
123                     color = t.interpolate_color(barycentric)
124                     self.color_buf[y, x] = color
125 
```

修改完后，执行 python examples/depth_test_demo.py 命令，即可渲染出不带深度测试的效果图片（depth_test_disabled.png）和带深度测试的效果图片（depth_test_enabled.png）。

e) **走样和反走样对比**：展示锯齿效应和 MSAA 抗锯齿效果（3 分）

修改 core/ rasterizer.py 里的 Rasterization 类的_rasterize_msaa 函数，利用 self.sample_points 采样点（相对于像素中心的偏移量 dx 和 dy），实现 MSAA 抗锯齿功能。当前的光栅化渲染并没有考虑抗锯齿，请在当前的基础上修改，实现抗锯齿。

```python
def _rasterize_msaa(self, t, min_x, max_x, min_y, max_y):
    """MSAA抗锯齿光栅化"""

    for y in range(min_y, max_y + 1):
        for x in range(min_x, max_x + 1):
            if t.inside(x + 0.5, y + 0.5, 0):
                barycentric = t.compute_barycentric(x + 0.5, y + 0.5, 0)

                color = t.interpolate_color(barycentric)
                self.color_buf[y, x] = color
```

修改完后，执行 python examples/antialiasing_demo.py 命令，即可渲染出走样效果图片（thin_triangles_aliasing.png）和反走样效果图片（thin_triangles_antialiasing.png）。

PS：当上述所有要求修改的函数实现完后，也可直接执行 python main.py 一键生成所有的效果图片。

作业提交要求：

i) 代码 + 报告

j) 报告里需简要说明每个考核点的实现过程，并给出最终的结果。其中 1-7 需给出实现过程，a)-b)需给出最终渲染的结果图像，c)-e)不仅要描述功能实现的步骤，同时需要给出最终渲染的结果图像。

k) 文件命名：学号+姓名+作业一.zip 上传地址为

https://bhpan.buaa.edu.cn/link/AA826639BCDC84471BBB6D686006D58A3B

文件夹名：作业一
有效期限：2025-10-31 23:59
提取码：CG25