Preprocess.py

1. detect_keypoints 函数的补全

补全思路:
1.使用 OpenCV 中的 cv2.SIFT_create() 创建一个 SIFT 检测器对象。
2.通过 detectAndCompute() 方法，传入图像来检测关键点并计算对应的描述符。
3.处理完成后，将检测到的关键点和描述符保存到文件中。

2. create_feature_matches 函数的补全

补全思路:
1.使用 cv2.BFMatcher() 创建一个暴力匹配器对象。
2.通过 knnMatch 方法找到每个关键点的两个匹配点。
3.使用 Lowe 比例测试来筛选出好的匹配点。具体来说，对于每个匹配点，如果第一个匹配点的距离比第二个匹配点小，并且满足比例条件，那么就认为这个匹配点是一个“好匹配”。

3. create_ransac_matches 函数的补全

补全思路:
使用 cv2.findEssentialMat() 来计算本质矩阵，进而验证匹配的几何一致性。
findEssentialMat 使用 RANSAC 方法来估算本质矩阵，找出哪些匹配点是“内点”。
最后返回几何验证后的匹配点和本质矩阵。

4. create_scene_graph 函数的补全

补全思路:
遍历所有图像对，加载每对图像的 RANSAC 匹配结果。
如果匹配的内点数超过给定的阈值 min_num_inliers，就添加一条边到图中。
这个图是使用 NetworkX 库来构建的。

Sfm.py

1.get_init_image_ids 函数的补全

补全思路:
通过遍历查找有着最大内点数的匹配

2. get_init_extrinsics 函数的补全

补全思路:
1.使用 cv2.recoverPose() 从本质矩阵和匹配点恢复旋转矩阵和位移。
2.将第一张图像的外参设置为单位矩阵，将第二张图像的外参通过旋转矩阵和位移向量构建。

3. get_reprojection_residuals 函数的补全

补全思路:
1. 将 3d 点进行齐次化并获得其结果的转置
2. 计算投影矩阵
3.进行 3d 点阵的投影
4.计算残差并返回

4. solve_pnp 函数的补全

补全思路:
1.根据已知的 3D 点和 2D 点，使用 cv2.solvePnP() 来估计相机的旋转向量和位移向量。
2.然后使用 cv2.Rodrigues()计算旋转矩阵。
3.最后使用 get_reprojection_residuals()计算残差

5. get_next_pair 函数的补全

补全思路:
1.从已注册图像中选取一个图像。
2.遍历该图像的所有邻居，找到尚未注册的图像。
3.对每一对图像，计算其匹配内点的数量。
4.更新最大内点数的图像对。
5.返回内点数最多的图像对的 ID。

bundle_adjustment.py

1.compute_ba_residuals 函数的补全

补全思路:
1.选择对应的三维点:
使用 points3d_idxs 来获取与每个二维点相关联的三维点。这些三维点将在重投影过程中作为基础。
2.齐次化三维点:

由于计算机视觉中的投影变换通常使用齐次坐标（4 维），因此将三维点从 (x, y, z) 转换为齐次坐标形式 (x, y, z, 1)。
使用 np.concatenate 来将一个 1 的列加到三维点的末尾，形成一个 N x 4 的齐次坐标数组。
3.计算投影矩阵：
通过提取每个相机的旋转矩阵 rot_mtx 和位移向量 tvec，将它们与内参矩阵 intrinsics 结合来计算每个相机的投影矩阵。
投影矩阵由内参矩阵与外参矩阵（旋转矩阵和平移向量）结合得到：P = intrinsics @ extrinsics，其中 extrinsics 是 3 x 4 的外参矩阵，表示旋转和平移。
4.计算二维投影：
对于每个三维点，使用相应的投影矩阵将其投影到二维图像平面。
投影公式为：[u, v, w]^T = P * [X, Y, Z, 1]^T，其中 [u, v] 是二维点，[X, Y, Z] 是三维点。
然后，将齐次坐标 [u, v, w] 除以 w，得到实际的二维坐标 [u/w, v/w]。
5.计算残差：
最后，计算残差，即每个二维点的实际坐标与通过投影得到的坐标之间的欧几里得距离：
residual = sqrt((u_actual - u_calculated)^2 + (v_actual - v_calculated)^2)

三维重建结果的可视化