NUS CS5477: 3D Computer April 4, 2024

Assignment 4

Lecturer: Lee Gim Hee *Student: Zhang Rongqi A0276566M*

1 Implementation

- `get_plane_sweep_homographies()`: In this function, a list of homograph matrices are computed. Firstly I recover the $\mathbf{R}$ and $\mathbf{C}$ from `relative_pose` matrix, specifically, $\mathbf{M}_{rel} = [\mathbf{R}^\intercal|\mathbf{R}^\intercal\mathbf{C}]$ (don't know why it is not $[\mathbf{R}^\intercal|-\mathbf{R}^\intercal\mathbf{C}]$). In this case, $\mathbf{K} = \mathbf{K}_{ref}$ and $\mathbf{n} = (0,0,1)^\intercal$. Then `homographies` is calculated by $\mathbf{K}(\mathbf{R}^\intercal + \frac{\mathbf{R}^\intercal\mathbf{C}\mathbf{n}^\intercal}{d})\mathbf{K}^{-1}$.

- `compute_plane_sweep_volume()`: In this function, I iterate each (D, H, W) pixel and depth in the reference coordinate. Firstly, I used `concat_extrinsic_matrix( ref_pose, invert_extrinsic(images[i].pose_mat))` to calculate `relative_pose` (don't know why it is not `concat_extrinsic_matrix(images[i].pose_mat, invert_extrinsic(ref_pose))`). Then this `relative_pose` can be used in `get_plane_sweep_homographies()`. Then the calculated `homographies` is used to call `cv2.warpPerspective()` to get `warped_images` and `warped_maskes`. `warped_maskes` are used to update `accum_count`. Two iterations are used to calculate the mean and variance (`ps_volume`) for the same pixel and depth of `warped_images`, respectively.

- `compute_depths()`: In this function, I simply picked the depth which corresponds to the minimal variance.

- `post_process()`: Firstly, I utilized `scipy.ndimage.median_filter()` to smooth `ps_volume`. I set the window size to $(1, 5, 5)$ because I only intend to smooth images at the same depth, while images at different depths should remain unaltered. Then, I set `accum_threshold` and `variance_threshold` to filter out unreliable choices. Specifically, I iterate over all pixels to verify if `smooth_ps_volume` and `accum_threshold` meet these criteria.

- `unproject_depth_map()`: Firstly I checked the parameter `mask` exists or not. If not `mask` is initialized as all one. Then I used `np.meshgrid()` to get all the indices, and used the `mask` to filter the unreliable indices. In this case, the projective matrix is simply set as $\mathbf{K}[\mathbf{I}|\mathbf{0}]$. So 3D points can be calculated by `points3d` $= \mathbf{K}^{-1}(x, y, 1)^\intercal$, where the depth of `points3d` is corresponding reciprocal of `inv_depth_image` x and y are iterated from the image size, and corresponding `pointsrgb` is set as `image[y, x]`.