NUS CS5477: 3D Computer	April 8, 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) in the reference coordinate. Firstly, I used `concat_extrinsic_matrix(ref_pose, invert_extrinsic(images[i].pose_mat))` to calculate `relative_pose`, which can convert each image to reference coordinate. Then this `relative_pose` is used in `get_plane_sweep_homographies()`. The `homographies` is used to call `cv2.warpPerspective()` to get `warped_images` and `warped_maskes`. `warped_maskes` are used to update `accum_count`. I tried two approaches to calculate the variance. One way is purely "variance": Two iterations are used to calculate the mean and variance for the same pixel and depth of `warped_images`. But this result is quite noisy. Then I tried to use L1 loss as variance, and set `kernel_size=(3,3)`, this method produced a better result. This result is so good that makes the effect of my `post_process()` is not much noticeable.

- `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 $(3, 5, 5)$. Then, I use sigma detection for each depth to filter out unreliable positions. Specifically, I iterate over all pixels to verify if `accum_count[idx[h, w], h, w] < accum_mean[idx[h, w]] - accum_var[idx[h, w]]` or `smooth_ps_volume[idx[h, w], h, w] > ps_mean[idx[h, w]] + ps_var[idx[h, w]]`. If not, I set the mask as false.

- `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]`.