GCV-2021 report

Konstantin Soshin

March 2021

1 get_view() function

```
    pose_i = CameraPose(extrinsics[i])

    imaging_i = RaycastingImaging(intrinsics_dict[i]['
    resolution_image'], intrinsics_dict[i]['resolution_3d'])

    points_i = pose_i.camera_to_world(imaging_i.image_to_points(
    image_i))


```

Using ray casting we get pointcloud from depth. Using CameraPose class and camera inrinsics, stored in extrinsics[i], we perform transformation of pointcloud to world coordinates.

2 pairwise_interpolate_predictions() function

First, we transform pointcloud to camera i coordinate system as follows:

```
    reprojected_j = pose_i.world_to_camera(points_j)

```

Then, we for each point from pointcloud, obtained from camera j, find k nearest neighbours in pointcloud i, using cKDTree.

```
    uv_i = imaging_i.rays_origins[:, :2]
    _, nn_indexes_in_i = cKDTree(uv_i).query(reprojected_j[:, :2],
    k=nn_set_size)

```

We add depth as third coordinate from original image.

```
    point_from_j_nns = np.hstack((uv_i[point_nn_indexes], image_i.
    reshape((-1, 1))[point_nn_indexes]))#?

```

We find distances from each point in j to its nearest neighbours. If gap between them is big enough, we skip this point and don't interpolate it. As criterion, we will compare minimal distance to neighbors with some threshold.

```
    distances_to_nearest = np.linalg.norm(point_from_j_nns -
    point_from_j, axis=1)
    interp_mask[idx] = np.amin(distances_to_nearest) <
    distance_interpolation_threshold

```

If point is chosen to interpolate, we interpolate distance function in that point, using known distance values in points from i. For that we use interp2d function from SciPy.

```
    interpolator = interpolate.interp2d(point_from_j_nns[:, 0],
    point_from_j_nns[:, 1], distances_i.reshape((-1))[
    point_nn_indexes])
    distances_j_interp[idx] = interpolator(*point_from_j[:2])

```

3 Visualisation

Here we can see result of algorithm.

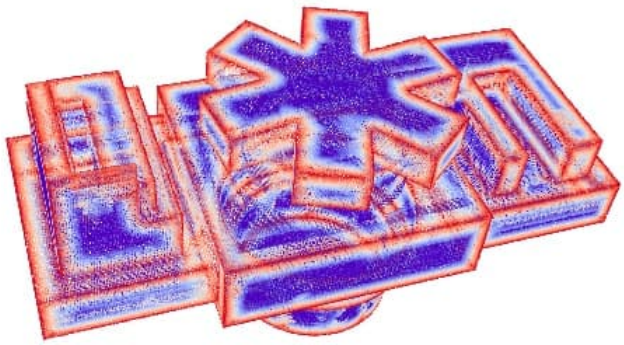

Figure 1: Result of fusion

4 Code

GitHub