CSCI 5563: Assignment #5

Depth Fusion

1 Submission

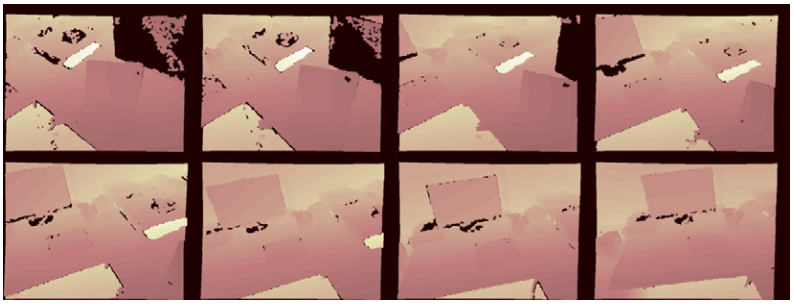
- Assignment due: May 7 (11:55pm)
- Skeletal code and data can be downloaded from: `https://www-users.cs.umn.edu/~hspark/csci5563_S2021/hw5.zip`.
- Individual assignment
- Up to 3 page summary write-up with resulting visualization (more than 3 page assignment will be automatically returned.).
- Submission through Canvas.
- Use Python 3
- You will complete HW5.py including the following functions:
  - `Get3D`
  - `CreateTSDF`
  - `ImageRays.cast`
  - `ComputeTSDFNormal`
  - `FindCorrespondence`
  - `SolveForPose`
  - `FuseTSDF`
- DO NOT SUBMIT THE PROVIDED IMAGE DATA
- The function that does not comply with its specification will NOT BE GRADED.
- You are not allowed to use computer vision related package functions unless explicitly mentioned here. Please consult with TAs if you are not sure about the list of allowed functions.

CSCI 5563: Assignment #5

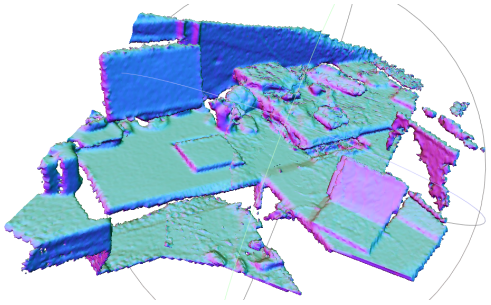
Depth Fusion

2 Overview

In this assignment, you will implement a method to create a coherent 3D geometry in the form of Truncated Sign Distance Function (TSDF) by fusing a sequence of depth images as shown in Figure 1. The pseudo code can be found in Algorithm 1.



(a) Input depth images



(b) Depth fusion

Figure 1: Given (a) the input depth images (the darker, the closer), you will fuse them to create (b) a coherent TSDF.

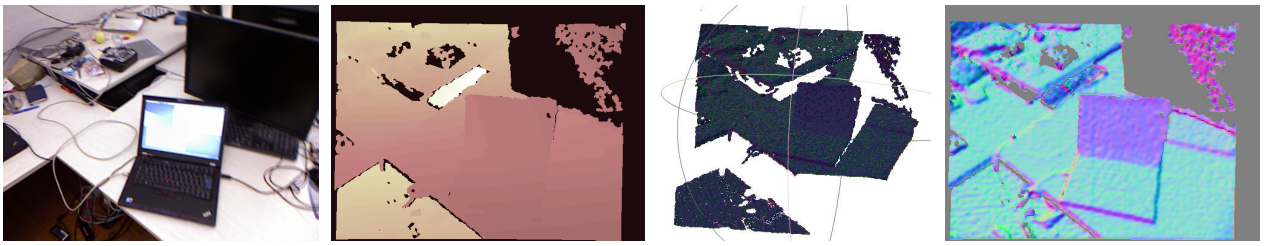
Algorithm 1 Depth Fusion

Load the first depth image.
$\mathbf{T} = \mathbf{I}_4$ ▷ Initialization transformation
Initialize TSDF using the first depth image. ▷ `CreateTSDF`
for i^{th} image in all depth images except for the first image **do**
    Initialize current transformation $\mathbf{T}_{\text{cur}} = \mathbf{T}$
    Get camera extrinsic from $\mathbf{T}_{\text{cur}}$
    Predict points from the TSDF by ray casting ▷ `ImageRays.cast`
    Predict normals from the predicted points ▷ `ComputeTSDFNormal`
    Load the i^{th} depth image
    Compute 3D points and surface normals. ▷ `Get3D`
    for j^{th} iteration $< n_{\text{iter}}$ **do**
        Find correspondences ▷ `FindCorrespondence`
        Compute ΔT ▷ `SolveForPose`
        Update $\mathbf{T}_{\text{curr}}$ and make sure $\det(\mathbf{R}_{\text{curr}}) = 1$
    end for
    Update transformation $\mathbf{T} = \mathbf{T}_{\text{curr}}$
    Create new TSDF for the i^{th} depth with $\mathbf{T}$ ▷ `CreateTSDF`
    Fuse TSDFs ▷ `FuseTSDF`
end for

CSCI 5563: Assignment #5

Depth Fusion

3 3D Points and Surface Normals from Depth



(a) Image (b) Depth (c) 3D point (d) Surface normal

Figure 2: Given a depth image (b), you will generate 3D points and surface normal using inverse projection.

Given a depth image, you will reconstruct 3D points and surface normals for all pixels. Given a pixel location $\mathbf{u} = (u, v)$, and its depth d, the 3D point $\mathbf{x} \in \mathbb{R}^3$ can be computed using inverse projection:

$$\mathbf{p}_c(u, v) = \mathbf{p}_c(\mathbf{u}) = d\mathbf{K}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}, \tag{1}$$

where $\mathbf{K}$ is the intrinsic parameter. Note that $\mathbf{p}_c$ is represented in the camera coordinate system.

The surface normal of $\mathbf{p}_c$ can be derived by taking spatial derivative of the 3D point:

$$\mathbf{n}_c(\mathbf{u}) = \frac{(\mathbf{p}_c(u+1, v) - \mathbf{p}_c(u, v)) \times (\mathbf{p}_c(u, v+1) - \mathbf{p}_c(u, v))}{\|(\mathbf{p}_c(u+1, v) - \mathbf{p}_c(u, v)) \times (\mathbf{p}_c(u, v+1) - \mathbf{p}_c(u, v))\|}. \tag{2}$$

Note that the surface normal is an unit vector where it needs to be normalized by its magnitude.

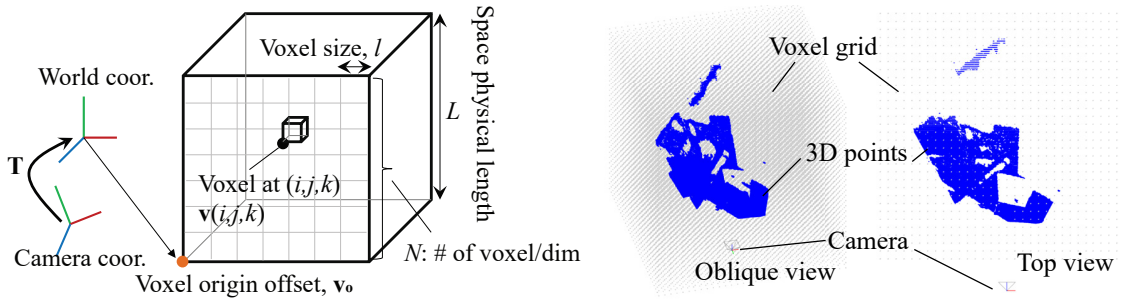
```
def Get3D(depth, K):
  ...
  return point, normal
```

Input: `depth` $\in \mathbb{R}_{+}^{H \times W}$ is the depth image with the height H and width W, and `K` $\in \mathbb{R}^{3 \times 3}$ is the intrinsic parameter.
Output: `point` $\in \mathbb{R}^{3 \times H \times W}$ and `normal` $\in \mathbb{R}^{3 \times H \times W}$ are points and surface normals for all coordinates in the depth image.
Note: You can use the provided function `SavePointNormal(filename, point, normal)` to save the points and normals into a `*.ply` file that can be visualized using `MeshLab` software as shown in Figure 2(c). The point color encodes the surface normal.

CSCI 5563: Assignment #5

Depth Fusion

4 TSDF from Depths



(a) Voxel geometry (b) Voxel visualization

Figure 3: Given a depth image (b), you will generate 3D points and surface normal using inverse projection.

Truncated sign distance field (TSDF) can be constructed by using a depth image and its transformation $\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}$. TSDF can be defined over the 3D cubic regular grid with the physical size of the space, L, the number of voxels per dimension, N, and voxel origin offset, $\mathbf{v}_o$. The location of the $(i, j, k)^{\text{th}}$ cell in the world coordinate system, $\mathbf{v}_w(i, j, k) \in \mathbb{R}^3$ can be written as:

$$\mathbf{v}_w(i, j, k) = \mathbf{v}_o + l \begin{bmatrix} i \\ j \\ k \end{bmatrix}, \tag{3}$$

where $l = L/N$ is the size of each voxel.

The sign distance field SDF at (i, j, k) is defined as:

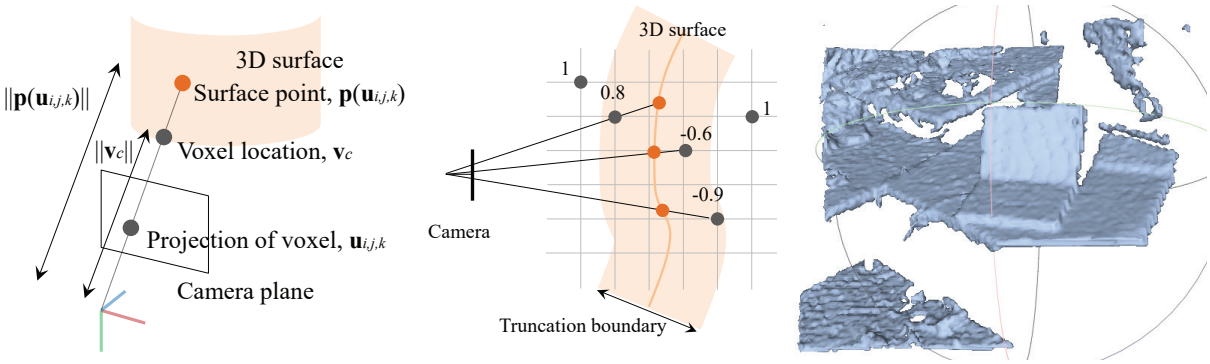
$$SDF(i, j, k) = \|\mathbf{p}_c(\mathbf{u}_{i,j,k})\| - \|\mathbf{v}_c(i, j, k)\|, \tag{4}$$

where $\mathbf{v}_c(i, j, k)$ is the location of the cell in the camera coordinate system, i.e., $\mathbf{v}_w = \mathbf{R}\mathbf{v}_c + \mathbf{t}$ where $\mathbf{T}$ is the transform that takes a point in the camera coordinate system to the world coordinate system, i.e., $\mathbf{T}$ is the inverse of camera extrinsic parameter $(\mathbf{R}_c, \mathbf{C})$ that transforms a point in the world coordinate to the camera coordinate. The subscript w and c indicate the world and camera coordinate systems, respectively.

$\mathbf{p}_c(\mathbf{u}_{i,j,k})$ is the 3D point that is reconstructed by the depth image at the projected location of $\mathbf{v}_c$ as shown in Figure 4(a). $\mathbf{v}_c$ is projected onto the camera plane to form 2d projection $\mathbf{u}_{i,j,k}$, and the point $\mathbf{p}_c(\mathbf{u}_{i,j,k})$ can be reconstructed using its depth (Equation (1)).

CSCI 5563: Assignment #5

Depth Fusion



(a) 3D geometry of SDF (b) TSDF in 2D (c) TSDF visualization

Figure 4: You will generate TSDF value and weight for each voxel.

You can construct the truncated signed distance field F from SDF:

$$F(i,j,k) = \begin{cases} 1 & \text{if } |SDF(i,j,k)| > \gamma \\ SDF(i,j,k)/\gamma & \text{otherwise} \end{cases}, \tag{5}$$

where γ is the truncation threshold that defines the boundary of truncation as shown in Figure 4(b). The value of TSDF is positive in front of the surface, and negative behind the surface up to the truncation boundary.

In addition to the TSDF value, its weight needs to be set:

$$W(i,j,k) = \begin{cases} 1 & \text{if } |F(i,j,k)| < 1 \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

This weight of the TSDF will be used for fusing two TSDFs.

```
def CreateTSDF(depth, T, K, voxel_param):
  ...
  return tsdf
```

Input: `depth` $\in \mathbb{R}_{+}^{H\times W}$ is the depth image with the height H and width W, `K` $\in \mathbb{R}^{3\times 3}$ is the intrinsic parameter, `T` $\in \mathbb{R}^{4\times 4}$ is the transformation from the camera coordinate to the world coordinate, and `voxel_param` is a class instance that include the parameters of voxels (e.g., L, N, $\mathbf{v}_o$, and γ). See `VoxelParam` class definition.
Output: `tsdf` is a class instance for TSDF that includes the values and weights. `tsdf.value` $\in \mathbb{R}^{N\times N\times N}$ is the value of TSDF, and `tsdf.weight` $\in \mathbb{R}^{N\times N\times N}$ is the weight of the TSDF.
Note: (1) The `VoxelParam` and `TSDF` classes are provided in the `tsdf.py`. (2) You can use the provided function `SaveTSDFtoMesh(filename, tsdf)` to save the mesh generated by Marching Cubes algorithm [1] into a `*.ply` file that can be visualized using `MeshLab` software as shown in Figure 4(c). This function is located in `utils.py`.

CSCI 5563: Assignment #5

Depth Fusion

5 Surface Prediction

(a) Ray casting (b) Depth prediction (c) Normal prediction

Figure 5: Given a TSDF and camera pose, you will predict the 3D points and surface normals using ray casting.

Given a TSDF and camera pose, you will predict the 3D points and surface normals using ray casting. Consider a ray emitted from a camera with the transformation matrix $\mathbf{T} = \begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix}$, i.e., that brings camera to world coordinate. Evenly spaced points on the ray can be written in the world coordinate as:

$$\mathbf{r}_i = \mathbf{t} + \lambda_i \mathbf{R}\mathbf{d}, \tag{7}$$

where $\mathbf{r}_i$ and λ_i are the i^{th} point along the ray and its magnitude, and $\mathbf{d}$ is the direction of the ray as shown in Figure 5(a), i.e., $\mathbf{d} = \frac{\mathbf{K}^{-1}\tilde{\mathbf{u}}}{\|\mathbf{K}^{-1}\tilde{\mathbf{u}}\|}$. Note that $\mathbf{d}$ is a unit vector.

The surface is located at the zero cross of TSDF value, i.e., there exist consecutive sampled points that change the sign of their TSDF values, i.e., $\exists\ j$ s.t. $F(\mathbf{r}_j) > 0, F(\mathbf{r}_{j+1}) \leq 0$. By abuse of notation, we denote the TSDF value sampled at $\mathbf{r}$ by $F(\mathbf{r})$. The predicted surface $\widehat{\mathbf{p}}$ at zero crossing can be computed by:

$$\widehat{\mathbf{p}}_w(\mathbf{u}) = \begin{bmatrix}\widehat{\mathbf{p}}_x \\ \widehat{\mathbf{p}}_y \\ \widehat{\mathbf{p}}_z\end{bmatrix} = -\frac{F(\mathbf{r}_j)}{F(\mathbf{r}_{j+1}) - F(\mathbf{r}_j)}(\mathbf{r}_{j+1} - \mathbf{r}_j) + \mathbf{r}_j. \tag{8}$$

Cast rays are not valid if there is no zero crossing.

CSCI 5563: Assignment #5

Depth Fusion

Given the estimated surface, its surface normal can be derived by taking spatial derivative of the TSDF values:

$$\widehat{\mathbf{n}}_w(\widehat{\mathbf{p}}) = \begin{bmatrix} \widehat{\mathbf{n}}_x \\ \widehat{\mathbf{n}}_y \\ \widehat{\mathbf{n}}_z \end{bmatrix}, \tag{9}$$

where

$$\widehat{\mathbf{n}}_x = \frac{\partial F}{\partial x} = \frac{F(\widehat{\mathbf{p}}_x + 1, \widehat{\mathbf{p}}_y, \widehat{\mathbf{p}}_z) - F(\widehat{\mathbf{p}}_x, \widehat{\mathbf{p}}_y, \widehat{\mathbf{p}}_z)}{\|F(\widehat{\mathbf{p}}_x + 1, \widehat{\mathbf{p}}_y, \widehat{\mathbf{p}}_z) - F(\widehat{\mathbf{p}}_x, \widehat{\mathbf{p}}_y, \widehat{\mathbf{p}}_z)\| + \epsilon}, \tag{10}$$

where $0 < \epsilon \ll 1$ is a constant that ensure avoiding zero division, e.g., $\epsilon = 10^{-5}$. The same partial derivative applies for the y and z directions.

```
class ImageRays:
  def __init__(self, K, voxel_param, im_size)
    self.rays_d = ...
    self.lambda_step = ...
```

Input: `K` $\in \mathbb{R}^{3\times3}$ is the camera intrinsic parameter, and `T` $\in \mathbb{R}^{4\times4}$ is the matrix that transforms camera to world coordinate. `voxel_param` is an instance of `VoxelParams`. `im_size` $= (H, W)$ is the size of depth image.
Class parameters: `self.rays_d` $\in \mathbb{R}^{3\times H\times W}$ is the ray bundle from an image w.r.t the camera coordinate, and `self.lambda_step` is the number of step to cast each ray, computed based on `voxel_param`.
Note: The `__init__` function of `ImageRays` class is provided in the file `rays.py`.

```
def ImageRays.cast(self, T, voxel_param, tsdf):
  ...
  return point_pred, validity
```

Input: `T` $\in \mathbb{R}^{4\times4}$ is the transformation from the camera coordinate to the world coordinate, `voxel_param` is an instance of `VoxelParams`, and `tsdf` is a truncated signed distance function.
Output: `point_pred` $\in \mathbb{R}^{3\times H\times W}$ is the predicted points, and `validity` $\in \{0, 1\}^{H\times W}$ is its validity where one if valid, and zero otherwise.
Note: You may use python built-in functions for 3D interpolation, e.g., `scipy.ndimage.map_coordinates`. The predicted points can be projected back to the camera plane to visualize the depth prediction as shown in Figure 5(b).

```
def ComputeTSDFNormal(point, tsdf, voxel_param):
  ...
  return normal
```

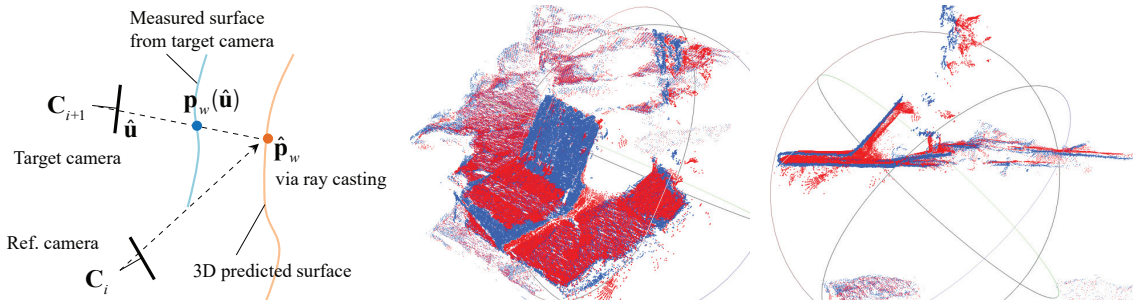
Input: `point` $\in \mathbb{R}^{3\times H\times W}$ is the predicted points from the TSDF values.
Output: `normal` $\in \mathbb{R}^{3\times H\times W}$ is the predicted normal.

Note: The normal vector must be a unit vector. The predicted normals can be projected back to the camera plane to visualize the normal prediction as shown in Figure 5(c).

CSCI 5563: Assignment #5

Depth Fusion

6 Projective Correspondence Estimation



(a) Geometry of correspondence (b) Oblique view (c) Side view

Figure 6: Correspondence between the predicted (red) and measured (blue) points are visualized.

Given surface prediction, you will estimate the projective correspondences between the predicted points and measured points for pose estimation. The predicted points in the world coordinate system can be computed by the ray casting from the reference camera (i^{th} time instant) described in Section 5. You will find the matching points in the target camera ($(i+1)^{\text{th}}$ time instant) by projecting onto the target camera pose, i.e.,

$$\mathbf{p}_w(\widehat{\mathbf{u}}) = \mathbf{R}\mathbf{p}_c(\widehat{\mathbf{u}}) + \mathbf{t}, \quad \mathbf{n}_w(\widehat{\mathbf{u}}) = \mathbf{R}\mathbf{n}_c(\widehat{\mathbf{u}}), \tag{11}$$

where $\mathbf{p}_w$ and $\mathbf{n}_w$ are points and normals represented in the world coordinate system. $\mathbf{R}$ and $\mathbf{t}$ are the transformation of the target camera. $\widehat{\mathbf{u}}$ is the projection of the predicted points (Figure 6(a)):

$$\lambda \begin{bmatrix} \widehat{\mathbf{u}} \\ 1 \end{bmatrix} = \mathbf{K}\mathbf{R}_c(\widehat{\mathbf{p}}_w - \mathbf{C}). \tag{12}$$

Among the correspondences $\mathbf{p}_w(\widehat{\mathbf{u}}) \leftrightarrow \widehat{\mathbf{p}}_w$, you will filter out the ones that are too far or with different surface normals:

$$\begin{cases} \|\widehat{\mathbf{p}}_w - \mathbf{p}_w(\widehat{\mathbf{u}})\| < \epsilon_p \\ \widehat{\mathbf{p}}_w^{\mathsf{T}} \mathbf{p}_w(\widehat{\mathbf{u}}) > \cos(\epsilon_n) \end{cases} \tag{13}$$

In practice, you also need to filter out the projections beyond the image boundary and the 3D points behind the target camera.

CSCI 5563: Assignment #5

Depth Fusion

```
def FindCorrespondence(T, p_pred, n_pred, valid_rays, p, n, K, e_p, e_n):
  ...
  return p_pred_corr, n_pred_corr, p_corr, n_corr
```

Input: `T` $\in \mathbb{R}^{4\times4}$ transforms a 3D point in the camera coordinate to the world coordinate, `p_pred`, `n_pred` $\in \mathbb{R}^{3\times H\times W}$ are points and normals in the world coordinate system predicted from the reference camera using ray casting, and `valid_rays` $\in 0, 1^{H\times W}$ indicates the validity of predictions. `p`, `n` $\in \mathbb{R}^{3\times H\times W}$ are the points and normals in the target camera coordinate system, `K` is the camera intrinsic parameter, and `e_p`, `e_n` $\in \mathbb{R}_+$ are the error thresholds for point and normal, respectively, in Equation (13).

Output: `p_pred_corr`, `p_corr`, `n_pred_corr`, `n_corr` $\in \mathbb{R}^{3\times m}$ are m correspondences, i.e., each row in `p_pred_corr` and `p_corr` specifies point correspondence, and each row in `n_pred_corr` and `n_corr` specifies normal correspondence.

Note: You can use the provided function `SavePointDepth(filename, point)` to save the point cloud and visualize the correspondences in `MeshLab` software as shown in Figure 6(b) and 6(c). This function is located in `utils.py`.

CSCI 5563: Assignment #5

Depth Fusion

7 Pose Estimation

Using the correspondences, you will solve for the target camera pose by minimizing the plane to point distance:

$$\begin{aligned} &\underset{\mathbf{R},\mathbf{t}}{\text{minimize}} \quad \sum_{i=1}^{m} \left| \left(\mathbf{p}_w(\widehat{\mathbf{u}}_i) - \widehat{\mathbf{p}}_i\right)^{\mathsf{T}} \widehat{\mathbf{n}}_i \right| \\ =&\underset{\mathbf{R},\mathbf{t}}{\text{minimize}} \quad \sum_{i=1}^{m} \left| \left(\mathbf{R}\mathbf{p}_c(\widehat{\mathbf{u}}_i) + \mathbf{t} - \widehat{\mathbf{p}}_i\right)^{\mathsf{T}} \widehat{\mathbf{n}}_i \right|, \end{aligned} \tag{14}$$

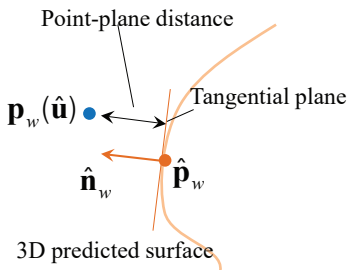


Figure 7: Point-to-plane error

where $\left| \left(\mathbf{p}_w(\widehat{\mathbf{u}}_i) - \widehat{\mathbf{p}}_i\right)^{\mathsf{T}} \widehat{\mathbf{n}}_i \right|$ measures the point-to-plane distance as shown in Figure 7, and m is the number of correspondences.

By linearizing Equation (14), you can derive a pose update equation:

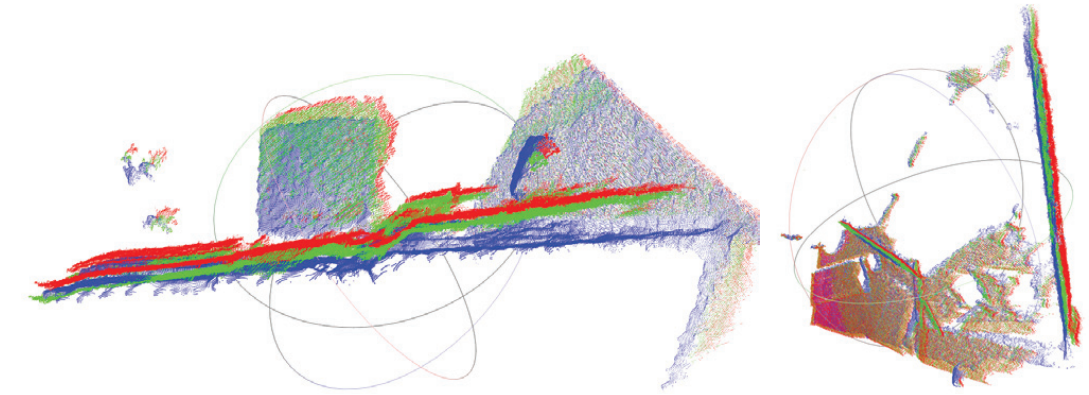
$$\mathbf{T} = (\Delta\mathbf{T})\,\mathbf{T} \qquad \text{where} \qquad \Delta\mathbf{T} = \begin{bmatrix} 1 & \alpha & -\gamma & t_x \\ -\alpha & 1 & \beta & t_y \\ \gamma & -\beta & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{15}$$

where the update parameters $(\beta, \gamma, \alpha, t_x, t_y, t_z)$ can be computed by solving the following linear system:

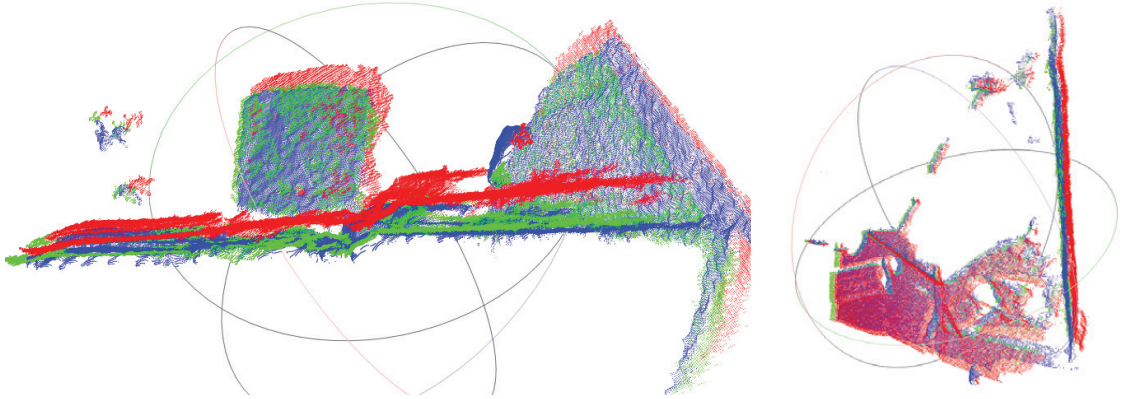
$$\begin{bmatrix} \widehat{\mathbf{n}}_1^{\mathsf{T}} \left[[\mathbf{p}_w(\widehat{\mathbf{u}}_1)]_\times \quad \mathbf{I}_3 \right] \\ \vdots \\ \widehat{\mathbf{n}}_m^{\mathsf{T}} \left[[\mathbf{p}_w(\widehat{\mathbf{u}}_m)]_\times \quad \mathbf{I}_3 \right] \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ \alpha \\ t_x \\ t_y \\ t_z \end{bmatrix} = \begin{bmatrix} \widehat{\mathbf{n}}_1^{\mathsf{T}} \left(\widehat{\mathbf{p}}_1 - \mathbf{p}_w(\widehat{\mathbf{u}}_1) \right) \\ \vdots \\ \widehat{\mathbf{n}}_m^{\mathsf{T}} \left(\widehat{\mathbf{p}}_m - \mathbf{p}_w(\widehat{\mathbf{u}}_m) \right) \end{bmatrix}. \tag{16}$$

CSCI 5563: Assignment #5

Depth Fusion



(a) First iteration



(b) Third iteration

Figure 8: Blue: predicted points $\widehat{\mathbf{p}}$; Red: measured points $\mathbf{p}_w$; Green: transformed measured points ($\Delta\mathbf{T}\mathbf{p}_w$). The target transformation is updated through iterations, which aligns the measured points to the predicted points. After the third iterations, the green points align with the blue points.

```
def SolveForPose(p_pred, n_pred, p):
  ...
  return deltaT
```

Input: `p_pred`, `n_pred`, `p` $\in \mathbb{R}^{3\times m}$ are the predicted points, predicted normals, and measured points in the world coordinate system.

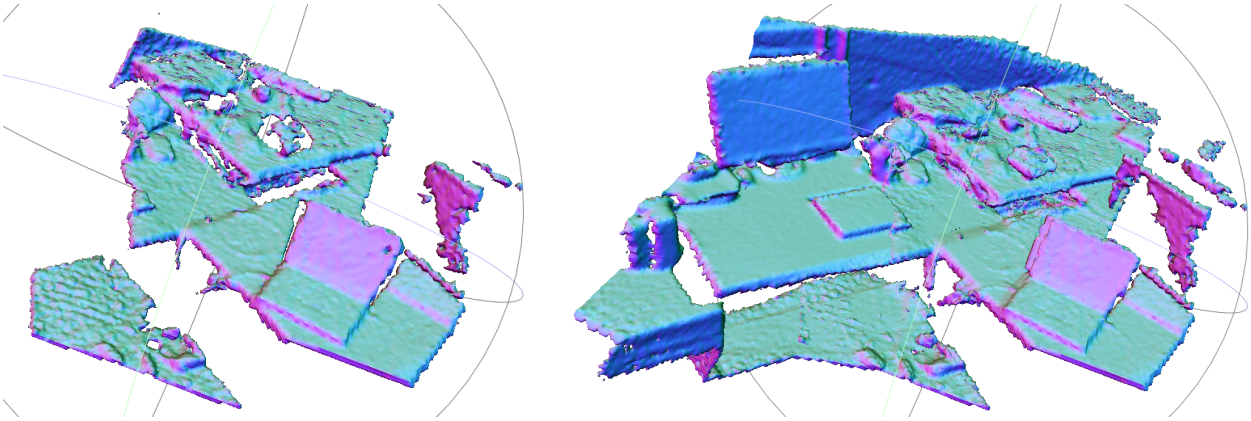
Output: `deltaT` $\in \mathbb{R}^{4\times 4}$ is the incremental update transformation using Equation (15).

Note: You can use the provided function `SavePoints(filename, point, color)` to save the points with a color ($[0, 255]^3$) to visualize the correspondences in `MeshLab` software. Figure 8(a) and 8(b) illustrate the updated transformation, i.e., the target transformation is updated, which aligns the measured points to the predicted points. After the third iterations, the green points align with the blue points.

CSCI 5563: Assignment #5

Depth Fusion

8 TSDF Fusion



(a) Initial TSDF (b) Final TSDF with 40 depth fusion

Figure 9: TSDF can be fused over frames to complete the scene.

Given the TSDF from previous frames (F) and the TSDF from the new frame (F_{new}), you will fuse the TSDF to update F based on weighted average:

$$\begin{aligned} F(i,j,k) &= \frac{W(i,j,k)F(i,j,k) + W_{\text{new}}(i,j,k)F_{\text{new}}(i,j,k)}{W(i,j,k) + W_{\text{new}}(i,j,k)}, \\ W(i,j,k) &= W(i,j,k) + W_{\text{new}}(i,j,k). \end{aligned} \tag{17}$$

```
def FuseTSDF(tsdf, tsdf_new):
  ...
  return tsdf
```

Input: `tsdf`, `tsdf_new` are the TSDF class from previous frames and new frame, respectively.
Output: `tsdf` is the updated TSDF.

References

[1] T. Lewiner, H. Lopes, A. W. Vieira, and G. Tavares, "Efficient implementation of marching cubes' cases with topological guarantees," *Journal of Graphics Tools*, 2003.