CSci 5563 Assignment 5

Luis Guzman

Friday May 7, 2021

This assignment is based on the paper *KinectFusion: Real-Time Dense Surface Mapping and Tracking* and consists of fusing multiple depth images together to form a coherent 3D representation of a scene. The full algorithm is shown in figure 1.

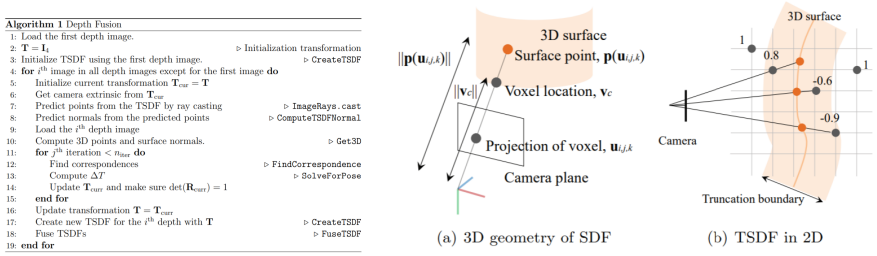

Figure 1: Full depth fusion algorithm (left) and the truncated signed distance function (right)

Every pixel in a given image represents part of a surface, and the depth image tells up how far away that surface is from the camera. In order efficiently store the scene geometry, we use the truncated signed distance function (TSDF), which represents the 3D space as discrete voxels. Each voxel stores the signed projective distance to the image surface. Positive values are defined as being in front of the scene geometry and negative values are behind the surface. All values above/below ± 1 are truncated to ± 1. Figure 1 visualizes this operation. Additional details are provided below, but an understanding of the TSDF is important to conceptualize early steps in the method.

In order to construct the initial TRDF, we build an initial estimate of the surface geometry and normals from the first depth image. The 3D point corresponding to each image pixel is

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

where d is the depth given by the depth image, and K is the intrinsic parameter. The surface normal can be calculated with

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

My calculated normals are shown in figure 4.

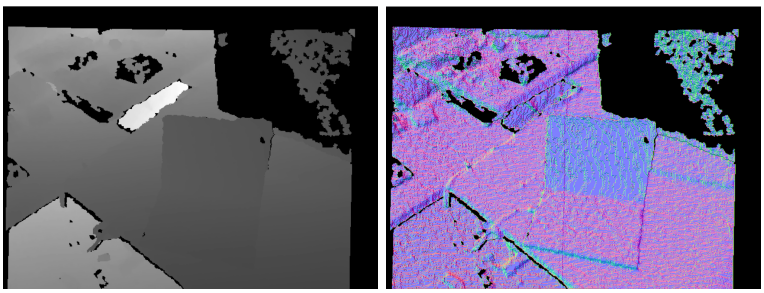

Figure 2: The initial depth image (left) and my normal estimation (right)

Next, I calculate an initial estimate of the TSDF from the calculated normals. The TSDF sections the 3D world into $256 \times 256 \times 256$ discrete voxels. The $(i,j,k)^{\text{th}}$ voxel has a location in the world coorinate system of

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

I use this 3D position to calculate the distance from each voxel to the nearest point on my estimated surface from the previous step. This distance defines the SDF as

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

I also only consider voxels that are visible from the camera and have non-zero distance value to be valid. The TSDF values and weights can then be initialized as

$$F(i,j,k) = \begin{cases} 1 & \text{if } |SDF(i,j,k)| > \gamma \\ SDF(i,j,k)/\gamma & \text{otherwise} \end{cases} \qquad W(i,j,k) = \begin{cases} 1 & \text{if } |F(i,j,k)| < 1 \\ 0 & \text{otherwise.} \end{cases}$$

My initial estimate of the TSDF is shown in figure 3.

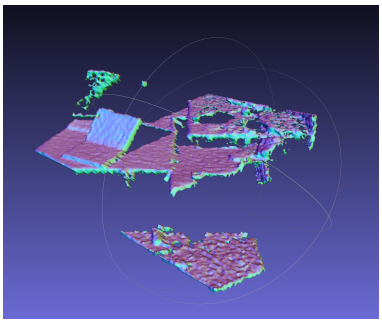

Figure 3: The initial TSDF estimate

The next step is to estimate the surface using zero-crossing of the TSDF. Because the TSDF is signed, depending on which side of the surface a given voxel is on, the surface itself corresponds to a sign of zero. Defining the surface in this way can give a more accurate representation than our inital estimate, and makes it easier later on to add additional depth data to our geometric representation.

In order to find zero-crossings, we use the method of ray-casting. By shooting a ray through every camera pixel and recording the TSDF of every voxel it passes through, we can define a zero-crossing as

$$F[\lambda, h, w] * F[\lambda + 1, h, w] < 0$$

where λ is the number of steps along a given ray, and the ray itself is defined as

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

with $\mathbf{d} = \frac{\mathbf{K}\mathbf{u}}{||\mathbf{K}\mathbf{u}||}$ and R, t defining the camera location. The zero crossing will then be somewhere between $F[\lambda, h, w]$ and $F[\lambda + 1, h, w]$ so we interpolate to find the true value with

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

The normals can the be calculated from the trdf as well. The result of my point and normal estimation is shown in figure 4.

Next, we need to align multiple depth images to the world coordinate system. We do this using the functions `FindCorrespondence` and `SolveForPose`. The correspondence is solved by taking each predicted point, finding the corresponding pixel coordinate in the new camera, then getting the new point for that pixel. The transformation is calculated as

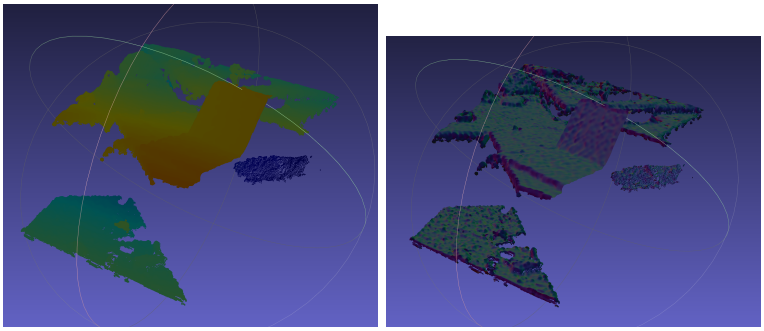

Figure 4: The estimations of the 3D points (left) and the normals (right) from the TSDF

$$\mathbf{p}_w(\widehat{\mathbf{u}}) = \mathbf{R}\mathbf{p}_c(\widehat{\mathbf{u}}) + \mathbf{t}, \quad \mathbf{n}_w(\widehat{\mathbf{u}}) = \mathbf{R}\mathbf{n}_c(\widehat{\mathbf{u}}), \qquad \lambda \begin{bmatrix} \widehat{\mathbf{u}} \\ 1 \end{bmatrix} = \mathbf{K}\mathbf{R}_c(\widehat{\mathbf{p}}_w - \mathbf{C}).$$

Corresponding points must fall within the camera pixel boundaries and follow the criteria

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

Next, in `SolveForPose` the new camera pose is approximated using 3 linear transforms. Since each step is assumed to be small, the linear approximation is valid and we converge to the new camera pose. The new pose is solved for as a $Ax = b$ linear system.

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

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

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

Lastly, the TSDF is updated using

$$F(i,j,k) = \frac{W(i,j,k)F(i,j,k) + W_{\text{new}}(i,j,k)F_{\text{new}}(i,j,k)}{W(i,j,k) + W_{\text{new}}(i,j,k)},$$
$$W(i,j,k) = W(i,j,k) + W_{\text{new}}(i,j,k).$$

Unfortunately I wasn't able to get the alignment code working due to time constraints. I have implemented every function and ensured that it is running without errors, but the meshes do not align properly. However, I have visualized my implementation up to the normal estimation, so I'm pretty confident up to that section. The full depth fusion requires further debugging.