Final Report

Group Nr. 9

1 Motivation

The goal of Kinect Fusion is a cheap and fast way for 3D scanning stationary objects with commonly available hardware.

2 Kinect Fusion

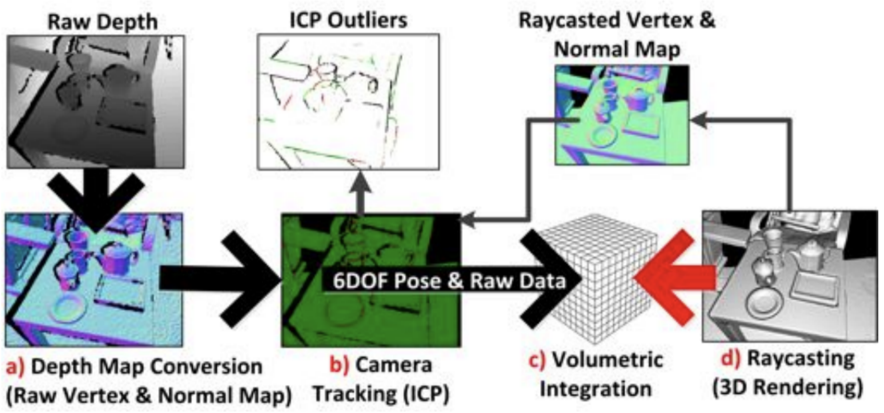
Raw Depth
ICP Outliers
Raycasted Vertex & Normal Map
6DOF Pose & Raw Data
a) Depth Map Conversion (Raw Vertex & Normal Map)
b) Camera Tracking (ICP)
c) Volumetric Integration
d) Raycasting (3D Rendering)

Figure 1: Method overview. [2]

2.1 Surface Measurement

For each timestep k the sensor provides a raw depth map R_k with a depth measurement $R_k(u)$ at each image pixel $\mathbf{u} = (u, v)^T$. Back-Projection of a depth measurements will result in a point measurement $p_k = R_k(u)K^{-1}$ in the sensor frame of reference. To improve the quality of the normals a bilateral filter is applied [4]:

$$\begin{aligned}
\mathscr{N}_\sigma(t) &= exp(\frac{-2^t}{2^\sigma}),\\
W_p(u,q) &= \mathscr{N}_{\sigma_s}(||u-q||_2)\mathscr{N}_{\sigma_r}(||R_k(u)-R_k(q)||_2),\\
D_k(u) &= \frac{\sum_{q\in\mathscr{U}} R_k(q)W_p(u,q)}{\sum_{q\in\mathscr{U}} W_p(u,q)},
\end{aligned}$$

Through back-projection into the sensor's frame, we receive the vertex for the filtered depth measurement $D_k(u)$. Only the camera intrinsics are needed for back-projection as the camera extrinsics for this frame is unknown for now.

$$V_k(u) = D_k(u)K^{-1}\dot{u}.$$

Invalid point measurements get marked with invalid values so that they can later be identified and ignored. The corresponding normal is computed with neighbouring points.

$$N_k(u) = v[(V_k(u+1,v) - V_k(u,v)) \times (V_k(u,v+1) - V_k(u,v))].$$

, where $v[x] = normalize(x)$. Only valid vertices are used to produce normals and invalid normals are also marked for later filtering.

2.2 Pose Estimation/Camera Tracking

Listing 1 Projective point-plane data association.

```
for each image pixel $\mathbf{u} \in$ depth map $\mathbf{D_i}$ in parallel do
  if $\mathbf{D}_i(\mathbf{u}) > \mathbf{0}$ then
    $\mathbf{v}_{i-1} \leftarrow \mathbf{T}_{i-1}^{-1}\mathbf{v}_{i-1}^{\mathrm{g}}$
    $\mathbf{p} \leftarrow$ perspective project vertex $\mathbf{v}_{i-1}$
    if $\mathbf{p} \in$ vertex map $\mathbf{V}_i$ then
      $\mathbf{v} \leftarrow \mathbf{T}_{i-1}\mathbf{V}_i(\mathbf{p})$
      $\mathbf{n} \leftarrow \mathbf{R}_{i-1}\mathbf{N}_i(\mathbf{p})$
      if $||\mathbf{v} - \mathbf{v}_{i-1}^{\mathrm{g}}|| <$ distance threshold and
      $\mathbf{n} \cdot \mathbf{n}_{i-1}^{\mathrm{g}} <$ normal threshold then
        point correspondence found
```

Figure 2: Linearized ICP using projective data association (source: [1])

We use frame-to-model Linearized ICP for estimating the pose of the frame. As mentioned in the paper we use projective data association to find the correspondences. The pseudocode for projective data association is listed in figure 2. For each pixel we launch a CUDA thread which finds the correspondence and adds the point-to-plane contraint. To achieve better performance for solving the matrix system we compute the 6×6 $(A^T A)$ matrix and 6×1 $(A^T b)$ matrix as follows:

$$\tilde{T}^z_{g,k}\dot{V}_k(u) = \tilde{R}^z\tilde{V}^g_k(u) + \tilde{t^z} = G(u)x + \tilde{V}^g_k(u),$$

, where the 3×6 matrix $\mathbf{G}$ is formed with the skew-symmetric matrix form of $\tilde{V}^g_k(u)$:

$$G(u) = \left[\left[\tilde{V}^g_k\right]_\times |I_{3\times 3} \right],$$

To solve the matrix system quickly the equations are reformed as such, where $\mathscr{C}$ represents all correspondences found:

$$\sum_{(\hat{u},u)\in\mathscr{C}} (A^T A)x = \sum A^T b,$$

$$A^T = G^T(u)\hat{N}^g_{k-1}(\hat{u}),$$

$$b = \hat{N}^g_{k-1}(\hat{u})^T(\hat{V}^g_{k-1}(\hat{u}) - \hat{V}^g_k(u))$$

The matrices $A^T A$ and $A^T b$ can be computed quickly in parallel using CUDA and are then reduced to a single matrix via a tree-based reduction algorithm or via CUB's reduction algorithm [3]. The resulting symmetric 6×6 matrix system can be solved as the upper triangular matrix for quicker results. The result vector is then moved back into a 4×4 transformation matrix $\tilde{T}^{inc}_{g,k}$ and composed with $\tilde{T}^{z-1}_{g,k}$ to get $\tilde{T}^{z}_{g,k}$. This step is repeated multiple times for a close alignment of the correspondences.

2.3 Volumetric Fusion

We are fusing each depth frame into a single 3D volumetric grid using their camera poses and truncated signed distance function (TSDF). Volumetric grid contains voxels and each voxel contains the current truncated signed distance value $F_k(p)$ and a weight $W_k(p)$. At each frame, the current frame is integrated into the signed distance field by updating each voxel's distance $F_k(p)$ and weight $W_k(p)$ values. This step is called the integration step and it can be easily parallelized in GPU.

First, the volumetric grid is projected into the current camera frame using the current camera pose and eliminate the points which has unvalid depth values.Then points are projected into the image space using the camera intrinsics and the pixel values outside of the image boundries are again eliminated. Next we retrieve the corresponding depth value of each voxel in pixel space using the depth map and compute the distance between the camera space and the depth map.

$$(\lambda^{-1}||t_{g,k} - p||_2 - R_k(x))$$

After computing the distance, we will eliminated the distances that are smaller than truncation margin $-\mu$ and for others, we compute the TSDF distance values.

$$\psi(\eta) = \begin{cases} min(1, \frac{\eta}{\mu} sgn(\eta)) & \text{, iff } \eta \geq -\mu \\ \text{null} & \text{, otherwise} \end{cases}$$

As a result, we can determine the distance values of each voxel as:

$$F_{R_k}(p) = \psi(\lambda^{-1}||t_{g,k} - p||_2 - R_k(x))$$

Finally, we can update the volumetric grid distances as:

$$F_k(p) = \frac{W_{k-1}(p)F_{k-1}(p) + W_{R_k}(p)F_{R_k}(k)}{W_{k-1}(p) + W_{R_k}(p)}$$

and weights as :

$$W_k(p) = W_{k-1}(p) + W_{R_k}(p)$$

We further observed that setting $W_{R_k}(p) = 1$ provides good results. All those steps are repeated for the next frame until no more depth measurements are provided.

2.4 Raycasting/Surface Prediction

The surface reconstruction can be performed with a per pixel raycast with the computed TSDF as depictured in figure 3. We march along the ray $T_{g,k}K^{-1}\dot{\mathbf{u}}$ from the minimum depth for the pixel and stop when we hit a zero crossing in the TSDF. If the zero crossing is hit from the wrong side (negative to positive), or when we marched out of the volume, the pixel is handeled as if it didn't hit the surface. As the gradient of the volume at $\mathbf{p}$ can be assumed to be orthogonal to the zero level set, we can compute the surface normal for u directly from the distance value F_k with a numerical derivative of the volume:

$$R_{g,k}\hat{\mathbf{N}} = \hat{\mathbf{N}}^g_n(\mathbf{u}) = \nu\nabla F(\mathbf{p})], \nabla F = \left[\frac{\delta F}{\delta x}, \frac{\delta F}{\delta y}, \frac{\delta F}{\delta z}\right]^T$$

The resulting vertex and normal map for each pixel can be used to render the surface.

Listing 3 Raycasting to extract the implicit surface, composite virtual 3D graphics, and perform lighting operations.

```
for each pixel $\mathbf{u} \in$ output image in parallel do
    $\mathbf{ray}^{\text{start}} \leftarrow$ back project $[\mathbf{u}, 0]$; convert to grid pos
    $\mathbf{ray}^{\text{next}} \leftarrow$ back project $[\mathbf{u}, 1]$; convert to grid pos
    $\mathbf{ray}^{\text{dir}} \leftarrow$ normalize $(\mathbf{ray}^{\text{next}} - \mathbf{ray}^{\text{start}})$
    $\mathbf{ray}^{\text{len}} \leftarrow \mathbf{0}$
    $\mathbf{g} \leftarrow$ first voxel along $\mathbf{ray}^{\text{dir}}$
    $\mathbf{m} \leftarrow$ convert global mesh vertex to grid pos
    $\mathbf{m}^{\text{dist}} \leftarrow ||\mathbf{ray}^{\text{start}} - \mathbf{m}||$
    while voxel $\mathbf{g}$ within volume bounds do
        $\mathbf{ray}^{\text{len}} \leftarrow \mathbf{ray}^{\text{len}} + \mathbf{1}$
        $\mathbf{g}^{\text{prev}} \leftarrow \mathbf{g}$
        $\mathbf{g} \leftarrow$ traverse next voxel along $\mathbf{ray}^{\text{dir}}$
        if zero crossing from $\mathbf{g}$ to $\mathbf{g}^{\text{prev}}$ then
            $\mathbf{p} \leftarrow$ extract trilinear interpolated grid position
            $\mathbf{v} \leftarrow$ convert $\mathbf{p}$ from grid to global 3D position
            $\mathbf{n} \leftarrow$ extract surface gradient as $\nabla\mathbf{tsdf}(\mathbf{p})$
            shade pixel for oriented point $(\mathbf{v}, \mathbf{n})$ or
            follow secondary ray (shadows, reflections, etc)
        if $\mathbf{ray}^{\text{len}} > \mathbf{m}^{\text{dist}}$ then
            shade pixel using inputed mesh maps or
            follow secondary ray (shadows, reflections, etc)
```

Figure 3: Raycasting on TSDF Volumetric Grid [1]

3 Results

3.1 Performance

A key aspect of KinectFusion is being able to scan the envorionment and build a realtime representation of the scanned input. To be able to achieve this, only a limited time per frame is available to achieve realtime performance.

A full quality camera stream from a Kinect Camera is a 640x480 pixel stream @ 30 frames per seconds, leaving about 30ms of time per frame. The tables below show the performance of our implementation of specific steps of the algorithm, while using a simple bilateral Filter and 20 iterations of the ICP algorithm on the full resolution:

Filter	Measurement	ICP	Fusion	Raycasting
∼20 ms	∼1 ms	∼25 ms	∼20 ms	∼15 ms

Summed up, this leads to ∼100 ms per Frame, so full-scale realtime performance is not possible. Reducing FPS of the camera to about 10 frames per second, still would allow for good realtime results, although alignment and therefore accuracy may be worse.

3.2 ICP Alignment

Our implementation of the ICP does not produce perfectly aligned results, the meshes get aligned rotatioally, but are shifted by translation. This leads to incorrect results for all following frames, as the TSDF incorporates the incorrect mesh and the raycasting also then pulls data from the incorrect TSDF. Figure 4 shows the misalignment of the translation. You can notice that the ICP is aligning the frames a little bit but not perfectly.

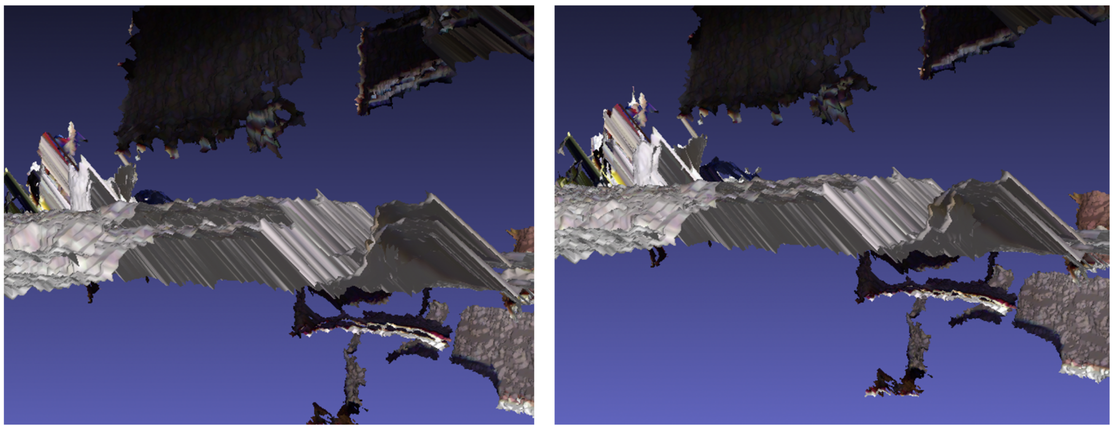

Figure 4: Left: Frame 4 and 5 before ICP, Right: Frame 4 and 5 after ICP

3.3 Volumetric Grid

Below we show the Point Cloud view of our volumetric grid after 5 frames.

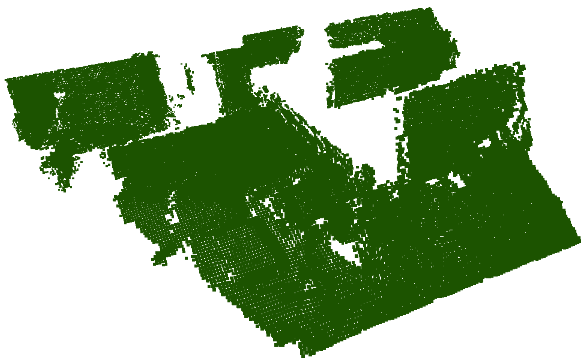
Figure 5: Volumetric Grid

3.4 Raycasting

Below we show the ray casted vertices we get from the volumetric grid for frame 5.

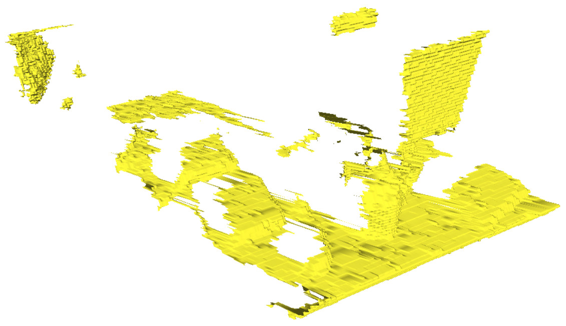
Figure 6: Ray casted result

4 Source Code

Source code is available at: https://github.com/mfaizanse/KinectFusion

References

[1] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, et al. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568, 2011.

[2] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136. IEEE, 2011.

[3] NVLabs. Nvidia cub. `http://nvlabs.github.io/cub/structcub_1_1_device_reduce.html`. Accessed: 2020-07-25.

[4] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *Sixth international conference on computer vision (IEEE Cat. No. 98CH36271)*, pages 839–846. IEEE, 1998.