KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera

Presenter: Sudhanshu Bahety

Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, Andrew Fitzgibbon
Motivation

- Reconstructs 3D model of a physical scene in real time
- High quality reconstruction
- Low Cost
- Real time dynamic interactions
- Augmented Reality
Problem with Kinect Depth Data

- Inherently Noisy data
- Consists numerous holes where no readings are observed
- Incomplete mesh from a single viewpoint

Figure 2: RGB image of scene (A). Extracted normals (B) and surface reconstruction (C) from a single bilateral filtered Kinect depth map. 3D model generated from KinectFusion showing surface normals (D) and rendered with Phong shading (E).
High Level Description

- 6DOF pose of the camera is tracked continuously
- Live depth data from the camera is fused into a single global 3D model in real-time
- With new views of the physical scene, new points are fused into the same model.
- Holes are filled, and the model becomes more complete and refined over time

https://www.youtube.com/watch?v=quGhaggn3cQ
Figure 4: A) User rotating object in front of fixed Kinect. B) 360° 3D reconstruction. C) 3D model imported into SolidWorks. D) 3D printout from reconstruction.
GPU Implementation

Figure 11: Overview of tracking and reconstruction pipeline from raw depth map to rendered view of 3D scene.
GPU Implementation (Depth Map Conversion)

- All CUDA threads operate in parallel in each pixel $u$
- $D_i(u) \rightarrow$ Incoming depth map
- $K \rightarrow$ Camera calibration matrix
- $v_i(u) \rightarrow$ 3D point in camera co-ordinate space
- $n_i \rightarrow$ Normal vectors for each point in CC space

$$v_i(u) = D_i(u) K^{-1} [u, 1].$$

$$n_i(u) = (v_i(x+1, y) - v_i(x, y)) \times (v_i(x, y+1) - v_i(x, y)).$$
GPU Implementation (Camera Tracking)

- Camera pose (6DOF) is calculated to align points in the current frame with the previous frame.
- ICP algorithm is used.
- $T_i$ (global camera pose) can be calculated incrementally.

$$T_{w,k} = \begin{bmatrix} R_{w,k} & t_{w,k} \\ 0^T & 1 \end{bmatrix} \in SE_3$$

$SE_3 := \{ R, t \mid R \in SO_3, t \in \mathbb{R}^3 \}$
GPU Implementation (Camera Tracking)

- Step 1: Find Correspondences
  - *Projective Data Association* used to find the initial correspondences

**Listing 1** Projective point-plane data association.

1: for each image pixel $u \in$ depth map $D_i$ in parallel do
2:   if $D_i(u) > 0$ then
3:     $v_{i-1} \leftarrow T_{i-1}^{-1} v_{i-1}^g$
4:   $p \leftarrow$ perspective project vertex $v_{i-1}$
5:   if $p \in$ vertex map $V_i$ then
6:     $v \leftarrow T_i V_i(p)$
7:     $n \leftarrow R_i N_i(p)$
8:     if $\|v - v_{i-1}^g\| <$ distance threshold and $abs(n \cdot n_{i-1}^g) <$ normal threshold then
9:       point correspondence found
GPU Implementation (Camera Tracking)

- Step 1: Find Correspondences

(a) point pair correspondence
(b) compatibility test

projective association of point pairs (t and t-1) along rays
point distance threshold
normal angle threshold
GPU Implementation (Camera Tracking)

- **Step 2: Finding T**
  - Output of each ICP iteration is a single transformation matrix $T$ that minimizes the point-to-plane error metric.
  - $T$ is initialized to be $T_{(i-1)}$ and is updated incrementally.
  - Linear Approximation made

$$\arg \min \sum_{u \in D_i(u)>0} \| (Tv_i(u) - v_{i-1}^g(u)) \cdot n_{i-1}^g(u) \|^2$$
Step 2: Finding T

\[ R(\alpha, \beta, \gamma) = \begin{pmatrix} 1 & \alpha\beta - \gamma & \alpha\gamma + \beta & 0 \\ \gamma & \alpha\beta\gamma + 1 & \beta\gamma - \alpha & 0 \\ -\beta & \alpha & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \hat{R}(\alpha, \beta, \gamma). \]

Figure 1: Point-to-plane error between two surfaces.

\[ \arg\min_{\mathbf{u}} \sum_{\mathbf{D}_i(\mathbf{u}) > 0} \| (\mathbf{T}v_i(\mathbf{u}) - v_{i-1}^g(\mathbf{u})) \cdot n_{i-1}^g(\mathbf{u}) \|^2 \]
GPU Implementation (Camera Tracking)

- Step 2: Finding T
GPU Implementation (Volumetric Representation)

- camera pose + camera coordinates in the $i_{th}$ instance
- $v_i^g(u) = T_i v_i(u)$ and $n_i^g(u) = R_i n_i(u)$ estimated
- Volumetric representation of data used
- Volume is subdivided uniformly into a 3D grid of voxels
- TSDF computed that specifies the relative distance of each vertices in 3D space to the actual surface
- $+$ve $\rightarrow$ Front of the surface, $-$ve $\rightarrow$ Behind the surface
GPU Implementation (SDF)

- $D(x)$ is a continuous implicit function
- Weighted signed distance of each point to the nearest range surface along the line of sight to the sensor
- For $1 \ldots n$ observations of $x$, we get sign $d_i(x)$ and weight $w_i(x)$

$$D(x) = \frac{\sum w_i(x)d_i(x)}{\sum w_i(x)}$$

$$W(x) = \sum w_i(x)$$

$d_1(x), d_2(x), \ldots, d_n(x)$

$w_1(x), w_2(x), \ldots, w_n(x)$
GPU Implementation (SDF - Uniform Weighting)
GPU Implementation (TSDF)

- The distance $d_i(x)$ and weighting functions $w_i(x)$ should extend indefinitely in either direction.
- TSDF restricts the functions to the vicinity of the surface and the values do not extend indefinitely.
GPU Implementation (Volumetric Integration)

- Pseudocode for computing tsdf

```plaintext
for each voxel g in x,y volume slice in parallel do
  while sweeping from front slice to back do
    v^g ← convert g from grid to global 3D position
    v ← T_i^{-1} v^g
    p ← perspective project vertex v
    if v in camera view frustum then
      sdf_i ← ||t_i - v^g|| - D_i(p)
      if (sdf_i > 0) then
        tsdf_i ← min(1, sdf_i / max truncation)
      else
        tsdf_i ← max(-1, sdf_i / min truncation)
    w_i ← min(max weight, w_i-1 + 1)
    tsdfavg ← \frac{tsdf_{i-1} w_{i-1} + tsdf_i w_i}{w_{i-1} + w_i}
    store w_i and tsdfavg at voxel g
```
GPU Implementation (Volumetric Integration)

- Visualization 1
GPU Implementation (Volumetric Integration)

- Visualization 2
GPU Implementation (Volumetric Integration)

- Visualization 3

```
-0.9 -0.4 -0.1 0.2 0.9 1 1 1 1 1
-0.9 -0.2 0.1 0.5 0.9 1 1 1 1 1
-0.9 -0.3 0.2 0.8 1 1 1 1 1
-0.9 -0.4 0.2 0.8 1 1 1 1 1
-0.8 -0.1 0.2 0.6 0.8 1 1 1 1 1
-0.9 -0.3 0.3 0.7 0.9 1 1 1 1 1
-0.9 -0.4 0.3 0.8 1 1 1 1 1 1
0.9 -0.5 0.0 0.4 0.9 1 1 1 1 1
-0.9 0.0 0.1 0.4 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
```
GPU Implementation (Volumetric Integration)

Visualization 4
The aim is to render the surface which is stored in the voxel model.
GPU Implementation (Raycast Rendering)

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

1: for each pixel \( u \in \text{output image} \) in parallel do
2: \( \text{ray}^{\text{start}} \leftarrow \text{back project}[u, 0]; \text{convert to grid pos} \)
3: \( \text{ray}^{\text{next}} \leftarrow \text{back project}[u, 1]; \text{convert to grid pos} \)
4: \( \text{ray}^{\text{dir}} \leftarrow \text{normalize}(\text{ray}^{\text{next}} - \text{ray}^{\text{start}}) \)
5: \( \text{ray}^{\text{len}} \leftarrow 0 \)
6: \( g \leftarrow \text{first voxel along \text{ray}^{\text{dir}}} \)
7: \( m \leftarrow \text{convert global mesh vertex to grid pos} \)
8: \( m^{\text{dist}} \leftarrow ||\text{ray}^{\text{start}} - m|| \)
9: while voxel \( g \) within volume bounds do
10: \( \text{ray}^{\text{len}} \leftarrow \text{ray}^{\text{len}} + 1 \)
11: \( g^{\text{prev}} \leftarrow g \)
12: \( g \leftarrow \text{traverse next voxel along \text{ray}^{\text{dir}}} \)
13: if zero crossing from \( g \) to \( g^{\text{prev}} \) then
14: \( p \leftarrow \text{extract trilinear interpolated grid position} \)
15: \( v \leftarrow \text{convert } p \text{ from grid to global 3D position} \)
16: \( n \leftarrow \text{extract surface gradient as } \nabla \text{tsdf}(p) \)
17: shade pixel for oriented point \((v, n)\) or
18: follow secondary ray (shadows, reflections, etc)
19: if \( \text{ray}^{\text{len}} > m^{\text{dist}} \) then
20: shade pixel using inputed mesh maps or
21: follow secondary ray (shadows, reflections, etc)
GPU Implementation (Simulating Real World Physics)

- Taking the merging of real and virtual geometries further
- Particle simulation is implemented on the GPU
- Static particles are created during volume integration
- For each surface voxel, a static particle is instantiated
- Each particle —> 3D vertex, velocity vector, ID
- A spatially subdivided uniform grid to identify neighboring particles
- Dynamic or static particle is assigned a grid cell ID
- Each thread processes collisions by examining (33) neighborhood of cells
- Discrete Element Method (DEM) is used to calculate a velocity vector when two particles collide
GPU Implementation (Simulating Real World Physics)

Figure 7: Interactive simulation of physics directly on 3D model even during reconstruction. Thousands of particles interact with reconstructed scene. Reconstruction, camera tracking, and physics simulation all performed in real-time.
Till now, assumption that the scene will remain reasonably static

ICP tracking assumes a single rigid transform occurred per frame due to camera motion.

Two Challenges:

User interaction in front of the sensor break this assumption

Surface predictions are refined over time using a running weighted average of distance values.
Figure 14: Extended GPU pipeline for real-time foreground and background segmentation, tracking and reconstruction.
GPU Implementation (Detecting Touch on Arbitrary Surface)

- Observe intersections between foreground and background

**Listing 4** Create touch map – testing if foreground and background vertices overlap.

1. $V_{fg}^g \leftarrow$ raycasted vertex map from foreground volume
2. **for** each pixel $u \in O$ (touch map) **in parallel do**
3. cast single ray for $u$ (as Listing 3)
4. **if** zero crossing when walking ray **then**
5. $v_{bg}^g \leftarrow$ interpolated global zero crossing position
6. **if** $||v_{bg}^g - V_{fg}^g(u)|| <$ adaptive threshold **then**
7. $O(u) \leftarrow V_{fg}^g(u)$
GPU Implementation (Towards Modeling of Dynamic Scenes)

- Similar idea to that of ICP Outlier Segmentation
- Now in case of foreground, independently predicting the pose of
  the foreground object using another instance of ICP
- More weight to new measurements.
- Running average of the derivative of the TSDF
- For physics simulation, represent entire foreground
  reconstruction as static particles, allowing collisions between the
  moving user, and the dynamic particles
- Independently track 6DOF of the object moving