Surfel-LIO: Fast LiDAR-Inertial Odometry with Pre-computed Surfels and Hierarchical Z-order Voxel Hashing

Seungwon Choi[1], Dong-Gyu Park[2], Seo-Yeon Hwang[2] and Tae-Wan Kim[1]

***Abstract*— LiDAR-inertial odometry (LIO) is an active research area, as it enables accurate real-time state estimation in GPS-denied environments. Recent advances in map data structures and spatial indexing have significantly improved the efficiency of LIO systems. Nevertheless, we observe that two aspects may still leave room for improvement: (1) nearest neighbor search often requires examining multiple spatial units to gather sufficient points for plane fitting, and (2) plane parameters are typically recomputed at every iteration despite unchanged map geometry. Motivated by these observations, we propose Surfel-LIO, which employs a hierarchical voxel structure (hVox) with pre-computed surfel representation. This design enables O(1) correspondence retrieval without runtime neighbor enumeration or plane fitting, combined with Z-order curve encoding for cache-friendly spatial indexing. Experimental results on the M3DGR dataset demonstrate that our method achieves significantly faster processing speed compared to recent state-of-the-art methods while maintaining comparable state estimation accuracy. Our implementation is publicly available at `https://github.com/93won/lidar_inertial_odometry`.**

I. Introduction

LiDAR-Inertial Odometry (LIO) is a widely used technique for real-time state estimation in GPS-denied environments [1], [2]. By tightly coupling high-frequency IMU measurements with precise LiDAR point clouds, LIO systems provide accurate and robust performance even under challenging conditions such as fast motion and geometrically degenerate scenes [3], [4].

The computational pipeline of modern LIO systems consists of three stages: IMU propagation, LiDAR-based state update, and map update. The IMU propagation step is computationally lightweight, as it involves only the integration of inertial measurements. In contrast, the LiDAR-based state update and map update stages account for the majority of the computational cost. During LiDAR-based state update, each iteration requires nearest neighbor search and plane fitting for every input point. The map update stage must also insert new points while maintaining the map structure for subsequent queries. Recent state-of-the-art methods have made remarkable progress in addressing these challenges. Fast-LIO2 [5] established a highly influential framework by employing an incremental k-d tree (ikd-Tree) [6] to maintain a global map, performing point-to-plane ICP [7]–[9] within an Iterated Extended Kalman Filter (IEKF) [10]. Unlike conventional k-d trees that require costly reconstruction when points are added, the ikd-Tree supports efficient incremental updates while maintaining $O(\log N)$ search complexity. Faster-LIO [11] further accelerated the pipeline by replacing the tree structure with iVox (incremental Voxels), a hash-based voxel map inspired by spatial hashing techniques [12], [13], achieving $O(1)$ map insertion and demonstrating impressive real-time performance.

[1]Seungwon Choi and Tae-Wan Kim are with the Department of Naval Architecture and Ocean Engineering, Seoul National University, Seoul, Korea. {csw3575, taewan}@snu.ac.kr

[2]Donggyu Park and Seo-Yeon Hwang are with Clobot Co., Ltd., Seoul, Korea. {derrick, ian}@clobot.co.kr

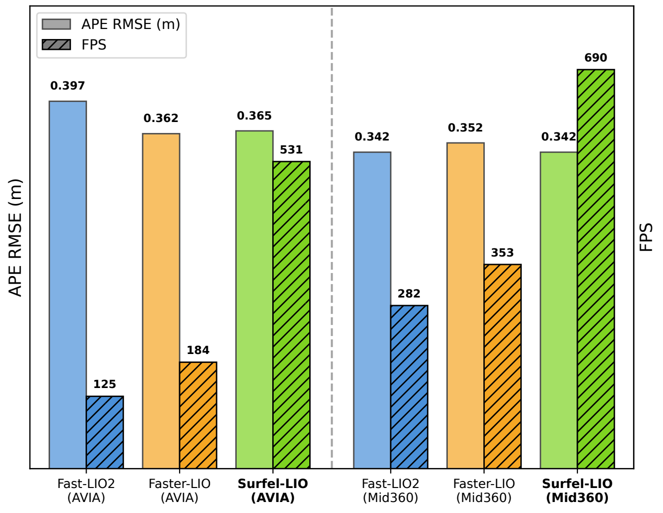

Fig. 1. Benchmark comparison on M3DGR dataset [14]. Our method achieves comparable accuracy with significantly higher throughput compared to Fast-LIO2 and Faster-LIO across different LiDAR sensors.

Although these methods have achieved remarkable performance, there is still room for improvement when considering the iterative nature of IEKF-based state estimation. First, while iVox provides efficient hash-based voxel access, gathering sufficient neighbor points for plane fitting requires examining multiple surrounding voxels—typically 7 to 27 depending on configuration—which can become a bottleneck when processing dense point clouds. Second, since IEKF performs multiple iterations per scan, both methods repeatedly compute plane parameters via least-squares fitting for the same set of correspondences, incurring $O(k^2)$ overhead at every iteration where k denotes the number of neighbors.

Motivated by these observations, we propose *hVox*, a two-level hierarchical voxel structure built upon the efficient hash-based design of prior works. Our key idea is to precompute surfel parameters (centroid, normal, and covariance) at map update time, so that during IEKF iterations the system can retrieve surface geometry directly without neighbor enumeration or least-squares fitting. To ensure robust surfel estimation, we organize voxels into two levels: fine-

TABLE I

COMPARISON OF LIDAR-INERTIAL ODOMETRY METHODS

Method	**Estimation**	**Map Structure**	**NN Search**	**Plane Estimation**	**Map Update**
LIO-Mapping [15]	Sliding Window Optimization	k-d tree (rebuild)	$O(\log N)$	$O(k^2)$	$O(N \log N)$
LIO-SAM [3]	iSAM2 Factor Graph	k-d tree (rebuild)	$O(\log N)$	$O(k^2)$	$O(N \log N)$
LINS [4]	Iterated Extended Kalman Filter	k-d tree (rebuild)	$O(\log N)$	$O(k^2)$	$O(N \log N)$
Fast-LIO [16]	Iterated Extended Kalman Filter	k-d tree (rebuild)	$O(\log N)$	$O(k^2)$	$O(N \log N)$
Fast-LIO2 [5]	Iterated Extended Kalman Filter	ikd-Tree	$O(\log N)$	$O(k^2)$	$O(\log N)$
Faster-LIO [11]†	Iterated Extended Kalman Filter	iVox (hash)	$O(m)$	$O(k^2)$	$O(1)$
Surfel-LIO (Ours)	**Iterated Extended Kalman Filter**	**hVox + Surfel**	$\boldsymbol{O(1)}$	$\boldsymbol{O(1)}$	$\boldsymbol{O(1)}$

N: number of map points, k: number of neighbors for plane fitting, m: number of neighboring voxels searched (typically 7–27).
†iVox achieves $O(1)$ hash-based voxel access, but gathering sufficient neighbors requires examining multiple surrounding voxels.

resolution voxels (L0) alone often contain too few points for reliable normal estimation, especially in sparse regions. By grouping multiple L0 voxels into coarser L1 voxels, we accumulate sufficient points to compute stable surfel parameters while preserving fine-grained spatial resolution for accurate point localization [17]–[19].

For efficient spatial indexing, we employ the Z-order curve (Morton code) [20] to encode 3D voxel coordinates into a single integer key. By leveraging the fact that consecutive LiDAR points are spatially adjacent due to the sensor's scanning pattern, the Z-order curve's locality-preserving property ensures that nearby points map to nearby hash keys, improving cache efficiency during batch queries [21]. Furthermore, the hierarchical relationship between L0 and L1 voxels can be computed through simple bit-shift operations, enabling efficient parent-child lookups without additional data structures.

To validate our approach, we evaluate on the M3DGR dataset [14], which contains sequences recorded with two distinct Livox LiDAR sensors—AVIA (non-repetitive scanning, 70° FoV) and Mid-360 (360° FoV, higher point density)—along with ground-truth trajectories from motion capture and RTK-GPS. As shown in Fig. 1, by combining pre-computed surfels with cache-friendly Z-order indexing, our method runs significantly faster than both Fast-LIO2 and Faster-LIO while maintaining competitive accuracy.

In summary, the main contributions of this paper are:

- A two-level hierarchical voxel map (*hVox*) that pre-computes surfel parameters during map updates, enabling $O(1)$ correspondence retrieval without neighbor search or plane fitting.
- Z-order curve encoding that exploits LiDAR scanning locality for cache-friendly voxel access.
- Experimental validation on the M3DGR dataset, demonstrating improved efficiency over Fast-LIO2 and Faster-LIO while maintaining comparable accuracy.

In addition, we publicly release our implementation under a permissive license to facilitate reproducibility and encourage further research.

II. RELATED WORK

LiDAR-based simultaneous localization and mapping (SLAM) research has advanced rapidly since Zhang and Singh proposed LOAM [22]. LOAM extracts edge and planar features from 3D LiDAR point clouds based on curvature and estimates 6-DoF pose through scan-to-map matching with a k-d tree. By separating odometry and mapping into different execution frequencies, LOAM achieved both computational efficiency and accuracy. This feature extraction and k-d tree-based correspondence search structure has been adopted as a core pipeline in numerous subsequent LiDAR odometry studies. However, LiDAR-only approaches exhibit degraded performance under fast dynamic motion and in geometrically degenerate environments such as long corridors or open spaces, where insufficient geometric constraints lead to pose estimation failures.

To handle these challenges, LiDAR-Inertial Odometry (LIO) has gained significant attention by tightly fusing LiDAR measurements with IMU data. LiDAR-IMU fusion methods are broadly categorized into factor graph-based optimization and Kalman filter-based approaches. LIO-Mapping [15] presented a tightly-coupled structure that simultaneously optimizes LiDAR scan matching and IMU preintegration factors through sliding window factor graph optimization. LIO-SAM [3] adopted the iSAM2 [23]-based incremental smoothing framework for efficient factor graph optimization. However, factor graph-based methods incur computational costs that scale with the optimization window size, limiting real-time processing on resource-constrained platforms.

In contrast, Kalman filter-based approaches maintain relatively low computational complexity by performing state propagation and measurement updates sequentially. LINS [4] introduced the iterated extended Kalman filter (IEKF) with robocentric formulation for LiDAR-IMU fusion. Fast-LIO [16] inherited the IEKF framework from LINS while adopting a direct registration approach that uses raw points without feature extraction, improving computational efficiency. Furthermore, by formulating Kalman gain computation in state dimension rather than measurement dimension, it efficiently processes thousands of points. This IEKF-based structure has

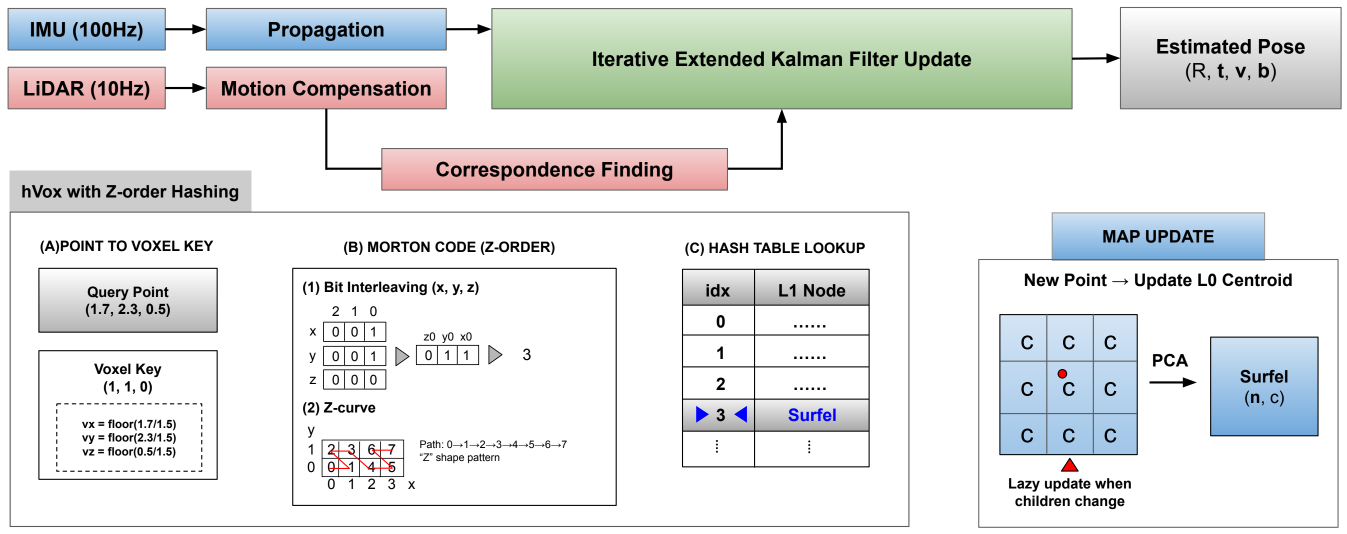

Fig. 2. Overview of the Surfel-LIO pipeline and correspondence finding procedure. Top: the system fuses IMU propagation and LiDAR measurements through IEKF to estimate pose. Bottom left: correspondence finding via hVox with Z-order hashing, consisting of (A) point-to-voxel key conversion, (B) Morton code computation through bit interleaving, and (C) O(1) hash table lookup to retrieve the pre-computed surfel. Bottom right: map update procedure where new points update L0 centroids, triggering lazy surfel recomputation via PCA when children change. The Morton code example uses 3-bit coordinates for illustration; the actual implementation uses 21 bits per axis to support large-scale environments.

become the core estimation framework for subsequent Fast-LIO series research.

Map data structure design is a critical factor determining real-time processing performance in LIO systems. Since nearest neighbor (NN) search and plane estimation must be repeatedly performed for thousands to tens of thousands of points per scan, the complexity of these operations dominates overall processing time. Fast-LIO [16] performed NN search by completely rebuilding the k-d tree every frame, incurring $O(N \log N)$ construction cost and inevitable performance degradation as map size increases.

To reduce this overhead, Fast-LIO2 [5] proposed the incremental k-d tree (ikd-Tree) [6]. Unlike conventional k-d trees that require costly reconstruction when points are added, the ikd-Tree supports incremental insertion, deletion, and re-balancing, enabling dynamic map updates without tree reconstruction while maintaining $O(\log N)$ complexity for both insertion and NN search. It also efficiently manages local maps during robot movement through box-wise deletion operations. However, computing point-to-plane residuals requires searching for k neighboring points per query point and then performing Principal Component Analysis (PCA) on them to estimate plane parameters. Since covariance computation and eigenvalue decomposition of a $3 \times k$ matrix are required, plane estimation incurs additional $O(k^2)$ complexity. Thus, the total complexity per query is $O(\log N + k^2)$.

Faster-LIO [11] proposed spatial hashing-based incremental voxels (iVox) instead of tree structures. iVox partitions 3D space into uniform voxels and stores each voxel in a hash table, enabling $O(1)$ access to any individual voxel. However, since a single voxel often contains insufficient points for reliable plane estimation, iVox must examine multiple neighboring voxels—typically 7 to 27 depending on configuration—to gather enough points. This neighboring voxel search introduces $O(m)$ complexity where m is the number of voxels examined. Furthermore, Faster-LIO still performs PCA at query time for plane estimation, maintaining $O(k^2)$ plane estimation complexity. Consequently, the total complexity per query is $O(m + k^2)$.

In this manuscript, we propose a novel map representation combining hierarchical voxel maps (hVox) with pre-computed surfels. hVox adopts a coarse-to-fine 2-level hierarchical structure (L1→L0) to partition space hierarchically, with Morton code-based spatial hashing [20] at each level enabling $O(1)$ voxel access. Morton codes encode 3D coordinates into a single integer, arranging spatially adjacent voxels adjacently in the hash table to improve cache efficiency.

The key differentiator lies in pre-computing plane information. While existing methods collect neighboring points at query time to perform PCA, our method incrementally updates plane parameters (normal vector, centroid, point count) whenever points are inserted into each voxel. Specifically, when a new point is added to a voxel, the running covariance matrix is updated with $O(1)$ operations using Welford's algorithm [24], and once sufficient points accumulate, plane parameters are computed via eigenvalue decomposition and stored as surfels [17]. Subsequently, at query time, plane information can be immediately obtained by directly referencing the voxel's surfel, eliminating the need for separate PCA computation.

Through this approach, the proposed method achieves $O(1)$ complexity for both NN search and plane estimation. This advantage becomes particularly pronounced during IEKF iterative updates, where plane information for the same points must be referenced multiple times. We provide a comparative summary of existing LIO methods in Table I.

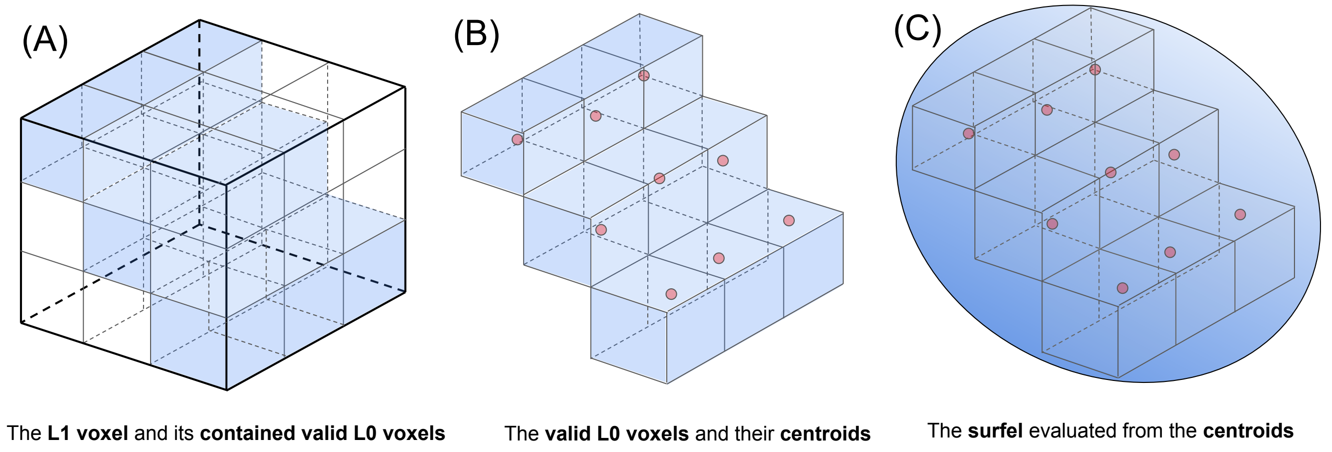
(A)
(B)
(C)
The **L1 voxel** and its **contained valid L0 voxels**
The **valid L0 voxels** and their **centroids**
The **surfel** evaluated from the **centroids**

Fig. 3. Hierarchical voxel structure and surfel representation. (A) An L1 voxel encompasses a $3 \times 3 \times 3$ grid of L0 child voxels, where shaded cells indicate occupied voxels. (B) Each occupied L0 voxel stores only its centroid (red points), computed incrementally as points accumulate. (C) A surfel is derived by applying PCA to the L0 centroids, yielding a compact plane representation $(\mathbf{n}, \mathbf{c})$ for point-to-plane registration.

III. Methodology

A. System Overview

Fig. 2 presents the overall architecture of Surfel-LIO, a tightly-coupled LiDAR-inertial odometry system designed for real-time state estimation. The system processes IMU measurements at high frequency (100-400Hz) and LiDAR scans at lower frequency (10Hz), fusing them through an Iterated Extended Kalman Filter (IEKF).

Upon receiving an IMU measurement, we propagate the state estimate using standard IMU kinematics, predicting the rotation, position, and velocity of the sensor platform. When a LiDAR scan arrives, the raw point cloud is first undistorted using the IMU-propagated trajectory to compensate for motion during the scan acquisition. Each point is then transformed to the world frame using the current state estimate and queried against our hierarchical voxel map to find its corresponding surfel. Unlike existing methods that perform nearest neighbor search followed by plane fitting, our approach retrieves pre-computed plane parameters directly from the voxel map in O(1) time. The retrieved surfels define point-to-plane residuals that drive the IEKF measurement update, which iterates until convergence. Finally, the optimized pose is used to add the registered scan points to the map, where L0 voxel centroids are updated incrementally and L1 surfels are recomputed as needed.

B. Hierarchical Voxel Map

We propose a two-level hierarchical voxel map (hVox) that separates point aggregation from plane representation, enabling efficient incremental updates while providing accurate geometric information for registration. Fig. 3 illustrates the structure.

The finest level, Level-0 (L0), consists of small voxels with edge length s_0. Rather than storing raw points, each L0 voxel maintains only the incrementally updated centroid of all points that have fallen within it (Fig. 3(B)):

$$\mathbf{c}^{(n+1)} = \frac{n \cdot \mathbf{c}^{(n)} + \mathbf{p}_{\text{new}}}{n+1} \tag{1}$$

where n is the accumulated point count and $\mathbf{p}_{\text{new}}$ is the newly inserted point. This formulation requires only O(1) storage per voxel regardless of how many points it contains.

Level-1 (L1) voxels serve as parent nodes, each encompassing a $3 \times 3 \times 3$ grid of L0 children with edge length $s_1 = 3s_0$ (Fig. 3(A)). Instead of aggregating points, each L1 voxel stores a pre-computed surfel—a compact planar primitive derived from the centroids of its occupied L0 children (Fig. 3(C)). Given the set of centroids $\{\mathbf{c}_i\}_{i=1}^m$ from m occupied L0 children, we compute the surfel via Principal Component Analysis [25]. The mean centroid is:

$$\bar{\mathbf{c}} = \frac{1}{m}\sum_{i=1}^{m} \mathbf{c}_i \tag{2}$$

The covariance matrix is constructed as:

$$\mathbf{C} = \frac{1}{m}\sum_{i=1}^{m} (\mathbf{c}_i - \bar{\mathbf{c}})(\mathbf{c}_i - \bar{\mathbf{c}})^\top \tag{3}$$

Eigendecomposition of $\mathbf{C}$ yields eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3$ with corresponding eigenvectors $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$. The surfel normal is taken as the eigenvector corresponding to the smallest eigenvalue:

$$\mathbf{n} = \mathbf{v}_3 \tag{4}$$

representing the direction of least variance. The planarity score quantifies fitting confidence:

$$\rho = \frac{\lambda_2 - \lambda_3}{\lambda_1 + \epsilon} \tag{5}$$

where ϵ is a small constant for numerical stability. Surfels with ρ below a threshold are rejected during correspondence search to avoid unreliable measurements.

Algorithm 1 Map Update

```
Require: Point cloud $\mathcal{P}$, transformation $\mathbf{T}_{WL}$
 1: for each point $\mathbf{p} \in \mathcal{P}$ do
 2:     $\mathbf{p}^W \leftarrow \mathbf{T}_{WL} \cdot \mathbf{p}$
 3:     $\mathbf{k}_0 \leftarrow \lfloor \mathbf{p}^W / s_0 \rfloor$
 4:     if L0[$\mathbf{k}_0$] exists then
 5:         Update centroid via Eq. 1
 6:     else
 7:         Create L0[$\mathbf{k}_0$] with centroid $\mathbf{p}^W$
 8:         $\mathbf{k}_1 \leftarrow \lfloor \mathbf{k}_0 / 3 \rfloor$
 9:         Register L0[$\mathbf{k}_0$] as child of L1[$\mathbf{k}_1$]
10:     end if
11:     Mark L1[$\mathbf{k}_1$] surfel as dirty
12: end for
13: for each dirty L1 voxel do
14:     Recompute surfel via Eq. 2–5
15: end for
```

To avoid redundant computation, we employ a lazy update strategy: when a new point modifies an L0 centroid, the parent L1 surfel is marked dirty rather than immediately recomputed. This is effective because multiple points from a single scan often fall into the same L1 voxel, and L1 voxels never queried during registration need not be updated at all. Algorithm 1 summarizes the map update procedure.

C. Z-order Spatial Hashing

Both L0 and L1 voxels are stored in hash tables for O(1) average-case access. However, standard hash tables suffer from poor cache locality: spatially adjacent voxels may be scattered across memory, causing frequent cache misses as the map grows. Since modern CPUs access RAM in cache lines (typically 64 bytes), loading a single voxel from a random memory location incurs the full memory latency ($\sim$100 cycles), whereas accessing nearby data already in CPU cache costs only $\sim$4 cycles. This disparity makes cache-friendly memory layouts critical for practical performance.

We employ Z-order (Morton) encoding [20], [21] as our hash function to preserve spatial locality. The bottom-left of Fig. 2 illustrates the complete correspondence finding procedure using this encoding. Given a query point, step (A) converts the 3D coordinate to an integer voxel key by dividing by the voxel size and flooring. For example, a point $(1.7, 2.3, 0.5)$ with voxel size $s = 1.5$ yields key $(1, 1, 0)$. Step (B) computes the Morton code by interleaving the bits of each coordinate in z, y, x order. As shown in Fig. 2(B), coordinates $x = 1, y = 1, z = 0$ (binary: 001, 001, 000) produce interleaved bits $z_0 y_0 x_0 = 011$, yielding Morton code 3. The Z-curve diagram shows how this interleaving traces a Z-shaped path through 2D space (path: $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow \cdots$), naturally clustering spatially adjacent cells into consecutive indices.

The Morton code is computed as:

$$M(k_x, k_y, k_z) = \sum_{i=0}^{20} \left(x_i \cdot 4^i + y_i \cdot 2 \cdot 4^i + z_i \cdot 4 \cdot 4^i \right) \quad (6)$$

Algorithm 2 Correspondence Finding

```
Require: Point $\mathbf{p}^W$ in world frame, voxel size $s_1$
Ensure: Surfel $(\mathbf{n}, \mathbf{c}, \rho)$ or $\emptyset$
 1: $\mathbf{k} \leftarrow \lfloor \mathbf{p}^W / s_1 \rfloor$            ▷ Coordinate quantization
 2: $m \leftarrow$ MORTON$(k_x, k_y, k_z)$                ▷ Bit interleaving
 3: if L1[$m$] exists and L1[$m$].$\rho > \rho_{\min}$ then
 4:     return L1[$m$].surfel
 5: else
 6:     return $\emptyset$                       ▷ No valid correspondence
 7: end if
```

where x_i, y_i, z_i are the i-th bits of each coordinate, and 21 bits per axis support coordinates up to $\pm 2^{20}$ voxels from the origin. This encoding ensures that consecutive LiDAR points—which typically fall in nearby spatial regions—map to similar Morton codes and thus access neighboring hash table entries, improving cache hit rates. Algorithm 2 summarizes the correspondence finding procedure.

Finally, step (C) uses the Morton code as the hash key to perform O(1) lookup into the hash table, directly retrieving the L1 voxel and its pre-computed surfel. This three-step process—coordinate quantization, Morton encoding, and hash lookup—replaces the traditional kNN search and plane fitting with a single hash table access.

For the hash table implementation, we adopt Robin Hood hashing [26], an open-addressing scheme that minimizes probe sequence variance by redistributing entries based on their displacement from the ideal position. Combined with Morton encoding, this provides both spatial locality in key distribution and memory locality in collision resolution, further improving cache efficiency.

An alternative approach is the Pseudo-Hilbert Curve (PHC) used in Faster-LIO [11], which provides even better locality preservation than Z-order. However, PHC requires maintaining a sorted data structure and periodic re-sorting as new voxels are inserted, adding computational overhead. In contrast, Morton encoding is computed directly from coordinates via simple bit operations, requiring no auxiliary data structures. Given that our surfel pre-computation already shifts computational burden to the update phase, we opt for the lightweight Morton encoding to keep hashing overhead minimal.

D. State Estimation

We employ an Iterated Extended Kalman Filter (IEKF) to fuse IMU measurements with LiDAR observations, following the error-state formulation commonly used in LIO systems [16]. The state vector is defined as:

$$\mathbf{x} = \begin{bmatrix} \mathbf{R}^\top & \mathbf{p}^\top & \mathbf{v}^\top & \mathbf{b}_g^\top & \mathbf{b}_a^\top \end{bmatrix}^\top \in SO(3) \times \mathbb{R}^{12} \quad (7)$$

where $\mathbf{R} \in SO(3)$ is the rotation from body to world frame, $\mathbf{p} \in \mathbb{R}^3$ is the position, $\mathbf{v} \in \mathbb{R}^3$ is the velocity, and $\mathbf{b}_g, \mathbf{b}_a \in \mathbb{R}^3$ are the gyroscope and accelerometer biases respectively.

Upon receiving IMU measurements, the state is propagated using the discrete-time kinematics:

$$\hat{\mathbf{R}}_{k+1} = \hat{\mathbf{R}}_k \cdot \mathrm{Exp}((\boldsymbol{\omega}_k - \mathbf{b}_g)\Delta t) \quad (8)$$

TABLE II

PER-OPERATION TIMING COMPARISON AVERAGED OVER ALL SEQUENCES.

Operation	**Fast-LIO2**	**Faster-LIO**	**Surfel-LIO**
NN Search (μs/pt)	1.42	2.76	**0.05**
Plane Est. (μs/pt)	0.17	0.61	**0.01**
Map Update (ms/fr)	0.34	**0.03**	**0.03**

$$\hat{\mathbf{v}}_{k+1} = \hat{\mathbf{v}}_k + (\hat{\mathbf{R}}_k(\mathbf{a}_k - \mathbf{b}_a) + \mathbf{g})\Delta t \tag{9}$$

$$\hat{\mathbf{p}}_{k+1} = \hat{\mathbf{p}}_k + \hat{\mathbf{v}}_k\Delta t + \frac{1}{2}(\hat{\mathbf{R}}_k(\mathbf{a}_k - \mathbf{b}_a) + \mathbf{g})\Delta t^2 \tag{10}$$

where $\boldsymbol{\omega}_k$ and $\mathbf{a}_k$ are the gyroscope and accelerometer readings, $\mathbf{g}$ is the gravity vector, Δt is the time interval, and $\text{Exp}(\cdot)$ denotes the exponential map from $\mathfrak{so}(3)$ to $SO(3)$.

When a LiDAR scan arrives, each point $\mathbf{p}_i^L$ in the LiDAR frame is transformed to the world frame and matched to its corresponding surfel via the hash table lookup described in Section III-C. Given a surfel with normal $\mathbf{n}_i$ and centroid $\mathbf{c}_i$, the point-to-plane residual is:

$$r_i = \mathbf{n}_i^\top \left(\mathbf{R}\mathbf{p}_i^L + \mathbf{p} - \mathbf{c}_i\right) \tag{11}$$

The IEKF iteratively refines the state estimate by linearizing this residual around the current estimate. Let $\delta\mathbf{x} = [\delta\boldsymbol{\theta}^\top, \delta\mathbf{p}^\top, \delta\mathbf{v}^\top, \delta\mathbf{b}_g^\top, \delta\mathbf{b}_a^\top]^\top$ denote the error state, where $\delta\boldsymbol{\theta} \in \mathbb{R}^3$ parameterizes the rotation error via $\mathbf{R}_{\text{true}} = \mathbf{R} \cdot \text{Exp}(\delta\boldsymbol{\theta})$. The Jacobian of the residual with respect to the error state is:

$$\mathbf{H}_i = \frac{\partial r_i}{\partial \delta\mathbf{x}} = \begin{bmatrix} -\mathbf{n}_i^\top \mathbf{R}[\mathbf{p}_i^L]_\times & \mathbf{n}_i^\top & \mathbf{0}_{1\times 9} \end{bmatrix} \tag{12}$$

where $[\cdot]_\times$ denotes the skew-symmetric matrix. Stacking all valid correspondences, the measurement update follows the standard Kalman filter equations with iteration until the state correction converges below a threshold.

IV. EXPERIMENTAL RESULT

A. Experimental Setup

We evaluate our method on the M3DGR dataset [14], a multi-sensor dataset designed for ground robot localization. We select nine sequences (Dark01-02, Dynamic03-04, Occlusion03-04, Varying-illu03-05) that provide centimeter-level ground truth from motion capture systems and RTK-GPS. These sequences present various challenges: Dark sequences feature low-light indoor environments where visual sensors struggle, Dynamic sequences contain moving pedestrians and objects that may corrupt map consistency, Occlusion sequences include partial sensor blockage from nearby obstacles, and Varying-illu sequences exhibit rapid lighting changes during indoor-outdoor transitions.

The dataset provides two Livox LiDAR sensors with different characteristics: AVIA with non-repetitive scanning pattern and 70° circular FOV generating approximately 24,000 points per scan, and Mid-360 with 360° horizontal FOV generating approximately 15,000 points per scan. This setup allows us to examine how different point densities and scanning patterns affect each method's computational load and estimation quality.

We compare against Fast-LIO2 [5] and Faster-LIO [11], two widely-used tightly-coupled LIO systems representing different approaches to map management. Fast-LIO2 employs an incremental k-d tree (ikd-Tree) that supports efficient point insertion and deletion with $O(\log N)$ search complexity, where N denotes the number of map points. Faster-LIO uses hash-based incremental voxels (iVox) that provides $O(1)$ voxel access, but gathering sufficient neighbors for plane fitting requires examining 7–27 surrounding voxels depending on the query point's position relative to voxel boundaries.

All experiments are conducted on a desktop PC with Intel Core i7-14700KF CPU (20 cores, 5.6GHz boost) and 64GB DDR5 RAM using official open-source implementations. To ensure fair comparison, all methods use identical voxel resolution ($0.5m$) and local map extent ($200m \times 200m$).

B. Per-Operation Timing Analysis

To understand the computational characteristics of different map representations, we instrument each method to measure per-point execution time for core operations: nearest neighbor search, plane estimation, and map update. Table II presents the timing breakdown averaged over all sequences.

For nearest neighbor search, Fast-LIO2 requires $1.42\mu s$ per point due to ikd-Tree traversal, where search time grows logarithmically with the number of map points. Faster-LIO requires $2.76\mu s$ per point despite using hash-based voxel indexing; this is because finding k nearest neighbors for plane fitting requires examining multiple neighboring voxels, and each voxel may contain multiple points that must be distance-sorted. Our method requires only $0.05\mu s$ per point, as the query reduces to a single hash table lookup to retrieve the voxel containing the query point, which directly provides pre-computed plane parameters without additional neighbor gathering.

For plane estimation, Fast-LIO2 and Faster-LIO compute plane parameters at query time using SVD-based fitting on retrieved neighbor points, requiring $0.17\mu s$ and $0.61\mu s$ per point respectively. The difference may be attributed to implementation details and the number of points used for fitting. Our method requires only $0.01\mu s$ per point since plane parameters (normal vector and centroid) are pre-computed and stored as surfels during map update; the residual calculation reduces to a simple dot product between the query point and the stored normal vector.

For map update, all three methods show similar performance around $0.03ms$ per frame—Fast-LIO2 performs incremental tree balancing, Faster-LIO updates hash table entries, and our method updates surfel statistics through incremental covariance computation. The comparable update times suggest that the overhead of maintaining pre-computed surfels does not significantly increase map management cost.

TABLE III

DETAILED RESULTS ON M3DGR DATASET. APE DENOTES ABSOLUTE POSE ERROR RMSE (m), FPS DENOTES FRAMES PER SECOND. **RED BOLD** INDICATES BEST, **BLUE BOLD** INDICATES SECOND BEST.

	Livox AVIA						**Livox Mid360**					
	Fast-LIO2		**Faster-LIO**		**Surfel-LIO**		**Fast-LIO2**		**Faster-LIO**		**Surfel-LIO**	
Sequence	APE	FPS	APE	FPS	APE	FPS	APE	FPS	APE	FPS	APE	FPS
Dark01	0.258	140	0.223	203	0.118	670	0.177	589	0.173	925	0.185	1044
Dark02	0.729	120	0.645	195	0.692	488	0.239	337	0.212	401	0.310	720
Dynamic03	0.165	145	0.151	225	0.266	606	0.178	263	0.178	461	0.206	670
Dynamic04	0.279	138	0.261	201	0.392	603	0.216	265	0.214	363	0.246	657
Occlusion03	0.257	124	0.283	179	0.271	561	0.423	301	0.463	373	0.315	687
Occlusion04	0.479	130	0.337	203	0.295	501	0.216	251	0.284	385	0.345	596
Varying-illu03	0.897	120	1.032	165	0.961	436	1.221	242	1.189	259	0.957	618
Varying-illu04	0.102	93	0.119	144	0.125	435	0.161	205	0.163	170	0.206	664
Varying-illu05	0.402	133	0.207	172	0.167	576	0.245	297	0.290	498	0.307	704
Average	0.397	125	**0.362**	**184**	**0.365**	**531**	**0.342**	282	**0.352**	**353**	**0.342**	**690**

C. *Odometry Accuracy and Efficiency*

Table III presents end-to-end performance for both LiDAR sensors. APE (Absolute Pose Error) is computed as the root mean square error of translational differences between estimated and ground truth poses after SE(3) alignment using the evo evaluation toolkit [27].

On AVIA sequences with higher point density (∼24,000 points/scan), Surfel-LIO processes at 531 FPS on average, compared to 125 FPS for Fast-LIO2 and 184 FPS for Faster-LIO. On Mid-360 sequences with lower point density (∼15,000 points/scan), all methods achieve higher throughput: Surfel-LIO at 690 FPS, Fast-LIO2 at 282 FPS, and Faster-LIO at 353 FPS. The speed improvement is more pronounced on denser point clouds, which is consistent with the expectation that per-point query savings accumulate with more points per frame.

Regarding state estimation accuracy, all three methods achieve comparable performance across the tested sequences. On AVIA sequences, average APE is 0.397m for Fast-LIO2, 0.362m for Faster-LIO, and 0.365m for Surfel-LIO. On Mid-360 sequences, the three methods show similar accuracy: 0.342m for Fast-LIO2, 0.352m for Faster-LIO, and 0.342m for Surfel-LIO. Challenging sequences such as Dark02 and Varying-illu03 show higher errors across all methods, likely due to aggressive motion during low-light conditions and rapid illumination changes.

These results suggest that the surfel-based map representation can improve processing efficiency while maintaining state estimation accuracy comparable to existing point-based methods. The consistent accuracy across methods indicates that the point-to-plane residual formulation is robust to different underlying map structures, with the primary distinction being how efficiently each method computes these residuals during iterative state estimation.

V. CONCLUSION

We presented Surfel-LIO, a LiDAR-inertial odometry system that employs pre-computed surfels for efficient correspondence search. By storing plane parameters as surfels during map construction, our approach reduces per-point query complexity from $O(\log N)$ or $O(m)$ to $O(1)$, eliminating the need for runtime nearest neighbor search and plane fitting.

Experimental results on the M3DGR dataset suggest that this design can improve processing efficiency compared to existing methods. Surfel-LIO achieved higher frame rates (531 FPS on AVIA, 690 FPS on Mid-360) while maintaining state estimation accuracy comparable to Fast-LIO2 and Faster-LIO. The per-operation timing analysis indicates that the computational savings primarily come from reduced query overhead, while the map update cost remains similar across all methods.

However, our approach has limitations that warrant further investigation. First, the surfel representation assumes locally planar geometry, which may be less suitable for environments with complex curved surfaces or sparse structures. Second, the current implementation uses a fixed voxel resolution, whereas adaptive resolution strategies could potentially improve both accuracy and efficiency. Third, our evaluation is limited to ground robot scenarios with Livox LiDAR sensors; the generalization to other platforms (e.g., aerial robots, handheld devices) and sensor types (e.g., spinning LiDARs) remains to be validated.

Future work includes extending the framework to handle non-planar geometric primitives such as lines and corners, investigating adaptive voxel sizing based on local geometry complexity, and integrating loop closure for globally consistent mapping. We also plan to explore the applicability of the surfel-based representation to other state estimation problems beyond LiDAR-inertial odometry.

References

[1] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[2] J. A. Placed, J. Strader, H. Carrillo, N. Atanasov, V. Indelman, L. Carlone, and J. A. Castellanos, "A survey on active simultaneous localization and mapping: State of the art and new frontiers," *IEEE Transactions on Robotics*, 2023.

[3] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti, and D. Rus, "Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 5135–5142.

[4] C. Qin, H. Ye, C. E. Pranata, J. Han, S. Zhang, and M. Liu, "Lins: A lidar-inertial state estimator for robust and efficient navigation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 8899–8906.

[5] W. Xu, Y. Cai, D. He, J. Lin, and F. Zhang, "Fast-lio2: Fast direct lidar-inertial odometry," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2053–2073, 2022.

[6] Y. Cai, W. Xu, and F. Zhang, "ikd-tree: An incremental kd tree for robotic applications," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3564–3571, 2021.

[7] P. J. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[8] Y. Chen and G. Medioni, "Object modelling by registration of multiple range images," in *IEEE International Conference on Robotics and Automation*, 1992, pp. 2724–2729.

[9] K.-L. Low, "Linear least-squares optimization for point-to-plane icp surface registration," University of North Carolina at Chapel Hill, Tech. Rep., 2004.

[10] B. M. Bell and F. W. Cathey, "The iterated kalman filter update as a gauss-newton method," *IEEE Transactions on Automatic Control*, vol. 38, no. 2, pp. 294–297, 1993.

[11] C. Bai, T. Xiao, Y. Chen, H. Wang, F. Zhang, and X. Gao, "Faster-lio: Lightweight tightly coupled lidar-inertial odometry using parallel sparse incremental voxels," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 4095–4102.

[12] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3d reconstruction at scale using voxel hashing," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 6, pp. 1–11, 2013.

[13] M. Teschner, B. Heidelberger, M. Müller, D. Pomerantes, and M. H. Gross, "Optimized spatial hashing for collision detection of deformable objects," in *VMV*, vol. 3, 2003, pp. 47–54.

[14] D. Zhang, J. Zhang, Y. Sun, T. Li, H. Yin, H. Xie, and J. Yin, "Towards robust sensor-fusion ground slam: A comprehensive benchmark and a resilient framework," *arXiv preprint arXiv:2507.08364*, 2025.

[15] H. Ye, Y. Chen, and M. Liu, "Tightly coupled 3d lidar inertial odometry and mapping," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019, pp. 3144–3150.

[16] W. Xu and F. Zhang, "FAST-LIO: A fast, robust lidar-inertial odometry package by tightly-coupled iterated kalman filter," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3317–3324, 2021.

[17] H. Pfister, M. Zwicker, J. Van Baar, and M. Gross, "Surfels: Surface elements as rendering primitives," in *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, 2000, pp. 335–342.

[18] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[19] E. Vespa, N. Nikolov, M. Grimm, L. Nardi, P. H. Kelly, and S. Leutenegger, "Efficient octree-based volumetric slam supporting signed-distance and occupancy mapping," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1144–1151, 2018.

[20] G. M. Morton, "A computer oriented geodetic data base and a new technique in file sequencing," IBM Ltd., Tech. Rep., 1966.

[21] H. Samet, *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006.

[22] J. Zhang and S. Singh, "LOAM: Lidar odometry and mapping in real-time," in *Robotics: Science and Systems (RSS)*, 2014.

[23] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, "iSAM2: Incremental smoothing and mapping using the bayes tree," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 216–235, 2012.

[24] B. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962.

[25] K. Pearson, "On lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[26] P. Celis, P.-Å. Larson, and J. I. Munro, "Robin hood hashing," University of Waterloo, Tech. Rep. CS-86-14, 1986.

[27] M. Grupp, "evo: Python package for the evaluation of odometry and slam," https://github.com/MichaelGrupp/evo, 2017.