



Katholieke
Universiteit
Leuven

Department of
Computer Science

ADVANCED COMPUTER ARCHITECTURE

project

Thibaut Beck
Wannes Paesschesoone

Academic year 2025–2026

Contents

1	Introduction	2
2	Theory	2
2.1	Point Cloud	2
2.2	Voxel	2
2.3	Pointcloud to Voxel Grid Filtering	2
2.4	Morton Codes	2
2.5	CUDA Thrust Library	2
2.6	LAS Files	2
3	Implementation	2
4	Morton and Sort Voxelizer	3
5	Dynamic Hash Map Voxelizer	3
6	Results	4

1 Introduction

This report talks about the implementation of a point cloud to voxel grid filter.

2 Theory

2.1 Point Cloud

A **point cloud** is a collection of data points defined in a three-dimensional coordinate system. Each point represents a position in space, typically described by (x, y, z) coordinates. Point clouds are commonly acquired using 3D scanners, LiDAR sensors, or photogrammetry systems. They are used in applications such as 3D modeling, robotics, mapping, and computer vision.

2.2 Voxel

A **voxel** (volumetric pixel) is the smallest unit of a 3D grid, similar to how a pixel is the smallest unit of a 2D image. Voxels divide 3D space into uniform cubes, allowing volumetric representations of shapes or environments. They are used in 3D reconstruction, simulation, gaming, and medical imaging.

2.3 Pointcloud to Voxel Grid Filtering

Pointcloud to voxel grid filtering converts an unstructured point cloud into a regular 3D grid. Space is divided into equal-sized voxels, and each point is assigned to its corresponding voxel. This reduces data density, removes redundant points, and creates a structured representation that is easier and faster to process for tasks such as downsampling and spatial queries.

2.4 Morton Codes

Morton codes (also called Z-order curves) are a method of encoding multi-dimensional coordinates into a single integer value. They work by bit-interleaving the binary coordinates (e.g., x, y, z). This encoding preserves spatial locality, making it useful for data structures such as octrees and for accelerating spatial queries on GPUs.

2.5 CUDA Thrust Library

The **CUDA Thrust library** is a parallel algorithms library for NVIDIA GPUs. It provides high-level abstractions similar to the C++ Standard Template Library (STL), including parallel sorting, scanning, reduction, and vector operations. Thrust simplifies GPU programming by offering ready-to-use, highly optimized parallel primitives.

2.6 LAS Files

LAS files are a standardized file format used for storing LiDAR point cloud data. The format supports 3D coordinates, intensity values, classification labels, GPS time, color information, and other metadata. LAS is widely used in geospatial applications, surveying, and remote sensing due to its efficiency and interoperability.

3 Implementation

In this chapter, the practical implementation details of the point cloud to voxel grid filter are presented. To leverage the massive parallelism of modern hardware, the algorithms were developed using CUDA for GPU acceleration. Two distinct parallel strategies were designed to solve the voxelization problem: a sorting-based approach using Morton encoding, and a scattering approach using a GPU-resident dynamic hash map.

4 Morton and Sort Voxelizer

The first approach utilizes the Thrust library to perform a sort-and-reduce operation. This method relies on the principle of organizing data such that points belonging to the same voxel are stored contiguously in memory.

The algorithm proceeds in the following stages:

1. Quantization: First, the global bounding box of the point cloud is calculated. For every point, discrete grid coordinates are computed based on the user-defined voxel size.
2. Encoding: The 3D grid coordinates are mapped to a 1D scalar value using Z-order curve encoding. This is achieved by bit-interleaving the integer coordinates to produce a 64-bit integer, which serves as the unique key for the voxel.
3. Sorting: The point indices are sorted based on their generated Morton codes. This ensures that all points residing in the same spatial voxel are grouped together in the linear array.
4. Reduction: A reduction-by-key operation is performed. This step iterates through the sorted array, identifies segments of identical Morton codes, and sums the coordinate and color data for each segment.
5. Centroid Calculation: Finally, the accumulated sums are divided by the point count per voxel to yield the final downsampled point cloud.

The theoretical complexity of this approach is dominated by the sorting phase. Given N input points, the time complexity is $O(N \log N)$. This method offers deterministic memory usage and stable performance regardless of the spatial distribution of the points.

5 Dynamic Hash Map Voxelizer

The second approach implements a custom open-addressing hash table directly in GPU global memory. This method is designed for maximum throughput, avoiding the global synchronization overhead required by sorting.

Crucially, this implementation re-purposes the Morton encoding scheme employed in the previous method. Instead of using the code for sorting, the 64-bit Morton integer serves as a compact, unique hash key. This allows a complex three-dimensional coordinate triplet to be treated as a single primitive type, enabling efficient atomic operations within the hardware registers.

The architecture operates through a high-throughput scatter-and-accumulate pipeline:

1. Initialization: A hash table is allocated in VRAM. To mitigate the risk of collisions inherent to hashing, the capacity is set significantly larger than the number of input points (typically a factor of 2.0 to 4.0).
2. Insertion and Accumulation: A custom kernel processes points in parallel. For each point, the grid coordinates are converted into a Morton code. This code is then hashed to find an initial slot in the table. The algorithm uses a linear probing strategy combined with lock-free atomic primitives:
 - A thread attempts to claim a bucket using an atomic Compare-and-Swap (`atomicCAS`).
 - If the bucket is empty, the thread successfully claims the voxel ownership.
 - If the bucket holds the same Morton key, the thread accumulates its point data into the existing voxel using `atomicAdd`.
 - If the bucket is occupied by a different key (a collision), the thread probes the subsequent memory slot until a valid location is found.
3. Compaction: Because the hashing process leaves gaps in the table, a final collection pass scans the memory to extract only the valid, populated voxels and writes them densely into the output buffer.

Ideally, this approach achieves $O(N)$ complexity, processing each point in constant time. By treating the Morton code as a hash key, the algorithm creates a "race" where threads compete to update voxel data simultaneously. While this is generally faster than sorting, performance is sensitive to the load factor. As the table fills, threads must probe further to find open slots, which can degrade performance. However, for massive point clouds with sufficient memory available, this method typically yields the highest processing speeds.

6 Results

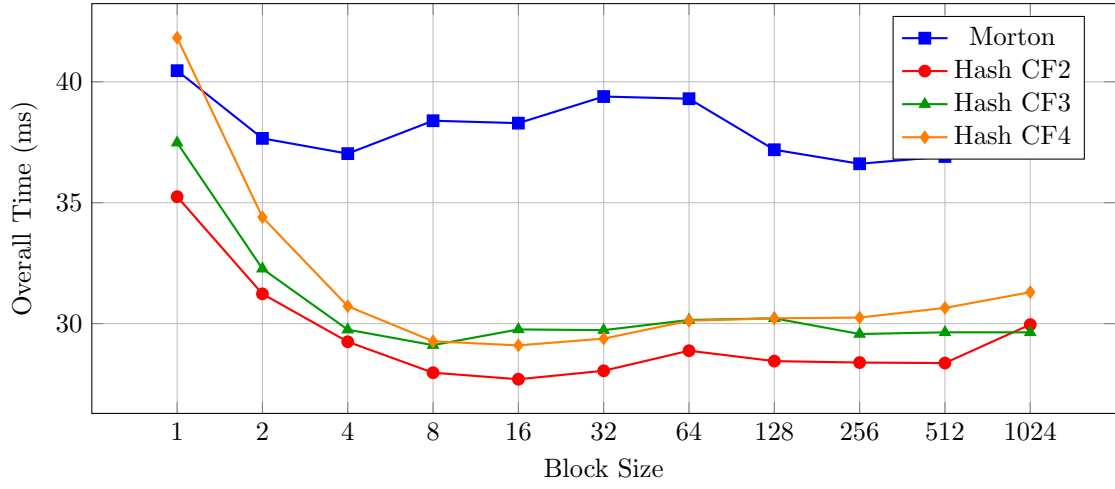


Figure 1: Overall execution time for voxel size 0.25

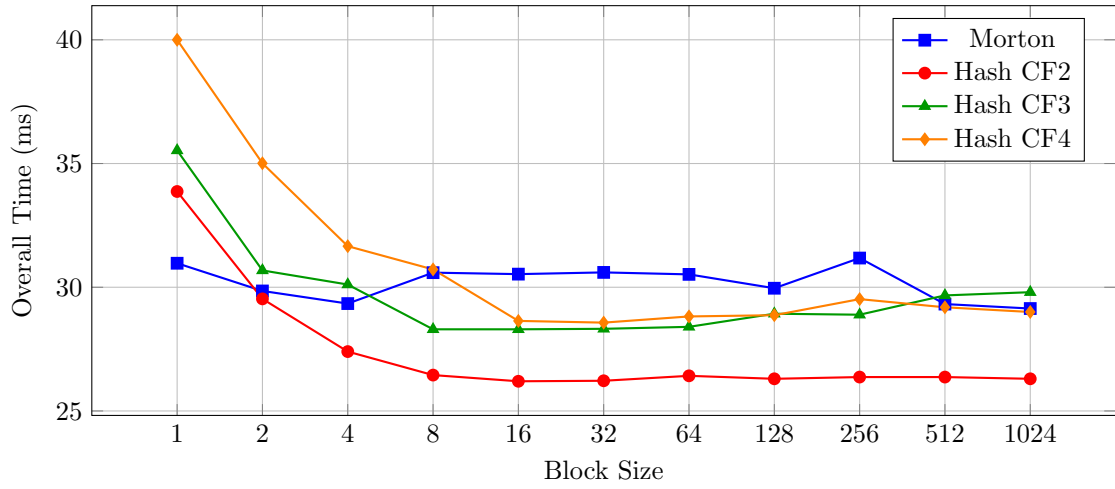


Figure 2: Overall execution time for voxel size 0.5

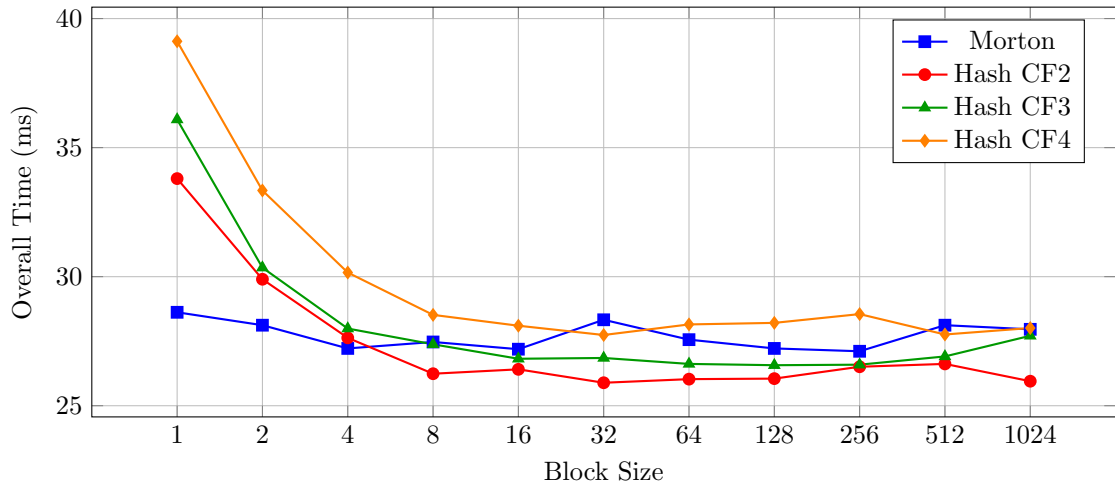


Figure 3: Overall execution time for voxel size 0.75

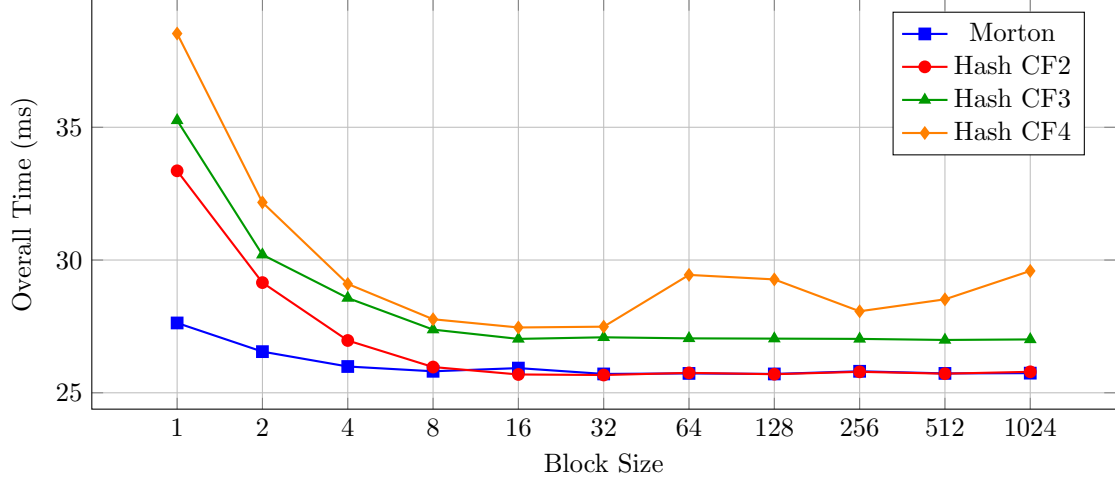


Figure 4: Overall execution time for voxel size 1.0

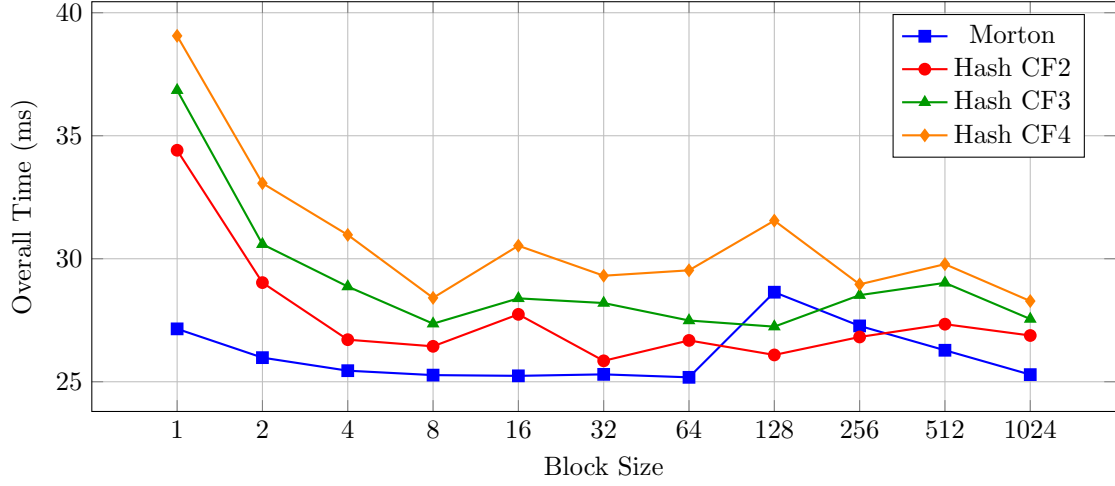


Figure 5: Overall execution time for voxel size 1.25

References

- [1] M. Nießner, M. Zollhöfer, S. Izadi & M. Stamminger, “Real-time 3D Reconstruction at Scale using Voxel Hashing,” **ACM Transactions on Graphics (TOG)**, vol. 32, no. 6, 2013.
- [2] “Morton encoding/decoding through bit interleaving: Implementations”, Forceflow — Jeroen Baert’s Blog, 7 October 2013. Available online: <https://www.forceflow.be/2013/10/07/morton-encodingdecoding-through-bit-interleaving-implementations/> (Accessed: 28 November 2025).
- [3] NVIDIA Corporation, “*Thrust — CUDA Core Compute Libraries (CCCL)*”. Available at: <https://nvidia.github.io/cccl/thrust/> (Accessed: 28 November 2025).
- [4] Wikipedia contributors, “*LAS file format*”, Wikipedia, The Free Encyclopedia. Available at: https://en.wikipedia.org/wiki/LAS_file_format (Accessed: 28 November 2025).
- [5] Ayushi Sharma, “*From Point Clouds to Voxel Grids: A Practical Guide to 3D Data Voxelization*”, Medium, 2021. Available at: <https://medium.com/@ayushi.sharma.3536/from-point-clouds-to-voxel-grids-a-practical-guide-to-3d-data-voxelization-cf5991c1e7bb> (Accessed: 28 November 2025).