基于CUDA加速的NVIDIA点云解决方案

CTSE
Dec 2022
haoyud@nvidia.com

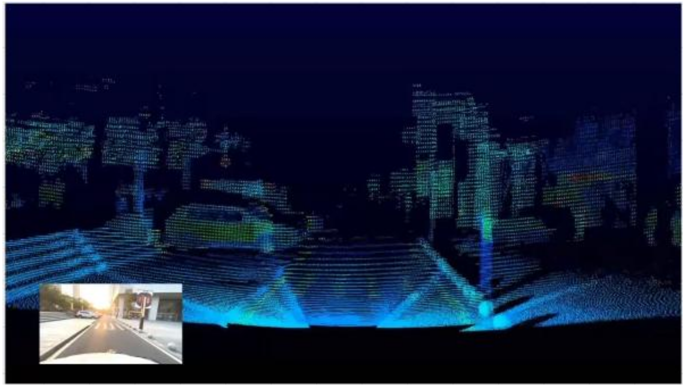
AGENDA

- Background
- cuPCL
- CUDA-PointPillars
- CUDA-CenterPoint
- End-to-end Solution

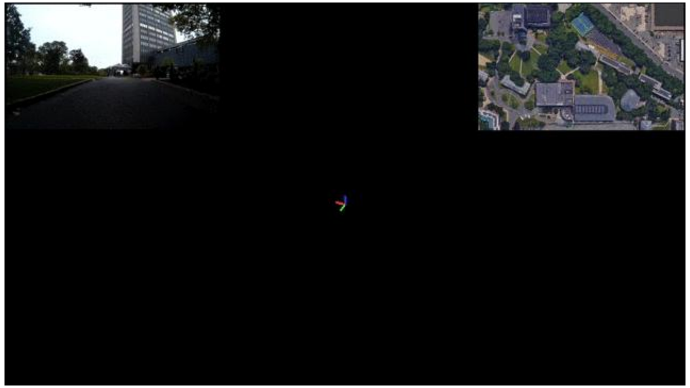
Background

BACKGROUND

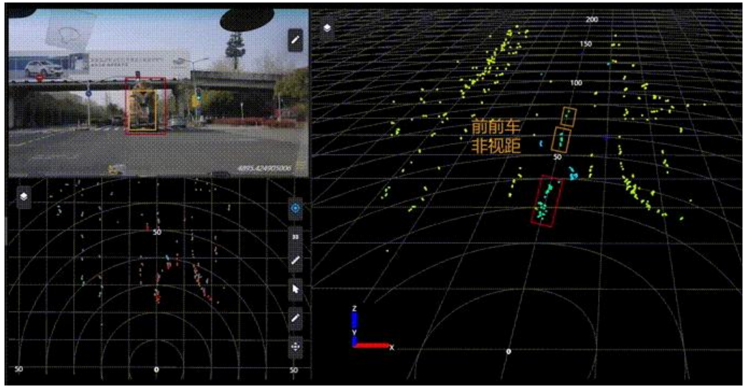
Lidar/Radar are becoming key sensors of precise perception & localization for AV safety



Detection task with solid state lidar (Robosense)



SLAM task with mechanical lidar
(Velodyne)



Point cloud mode with MMV Radar
(Huawei)

Lidar/Radar has better spatial resolution for small targets and longer detection range than camera.

The denser point cloud is, the better accuracy and safety can be achieved for AV.

However, point cloud processing cost can grow rapidly!!!

Introduction to cuPCL

GITHUB & USERS

https://github.com/NVIDIA-AI-IOT/cuPCL

Our github:

About

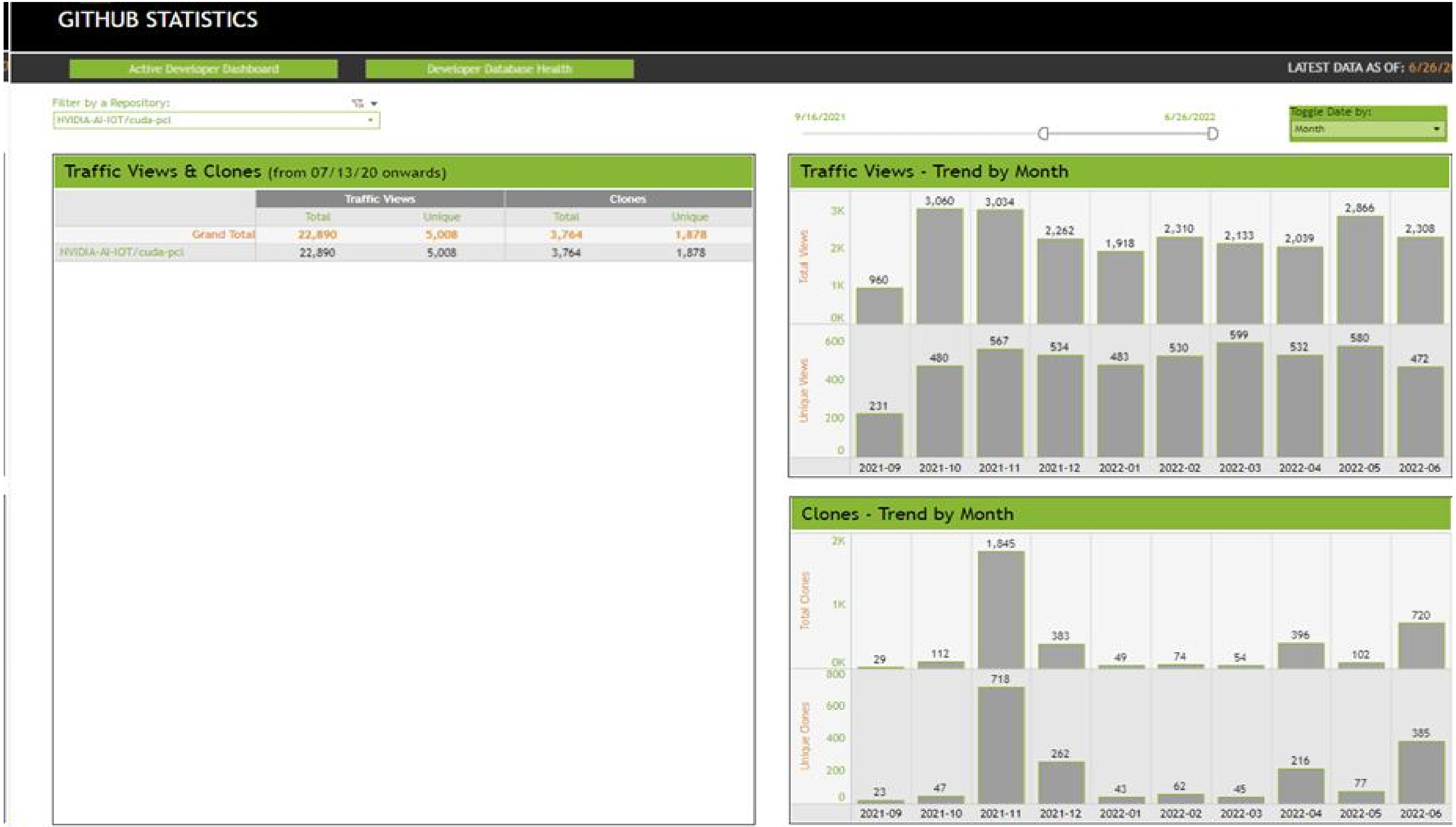
A project demonstrating how to use the libs of cuPCL.

- Readme
- MIT license
- 277 stars
- 14 watching
- 57 forks

Our users:

Since cuPCL has been placed on github, it is available for all developers

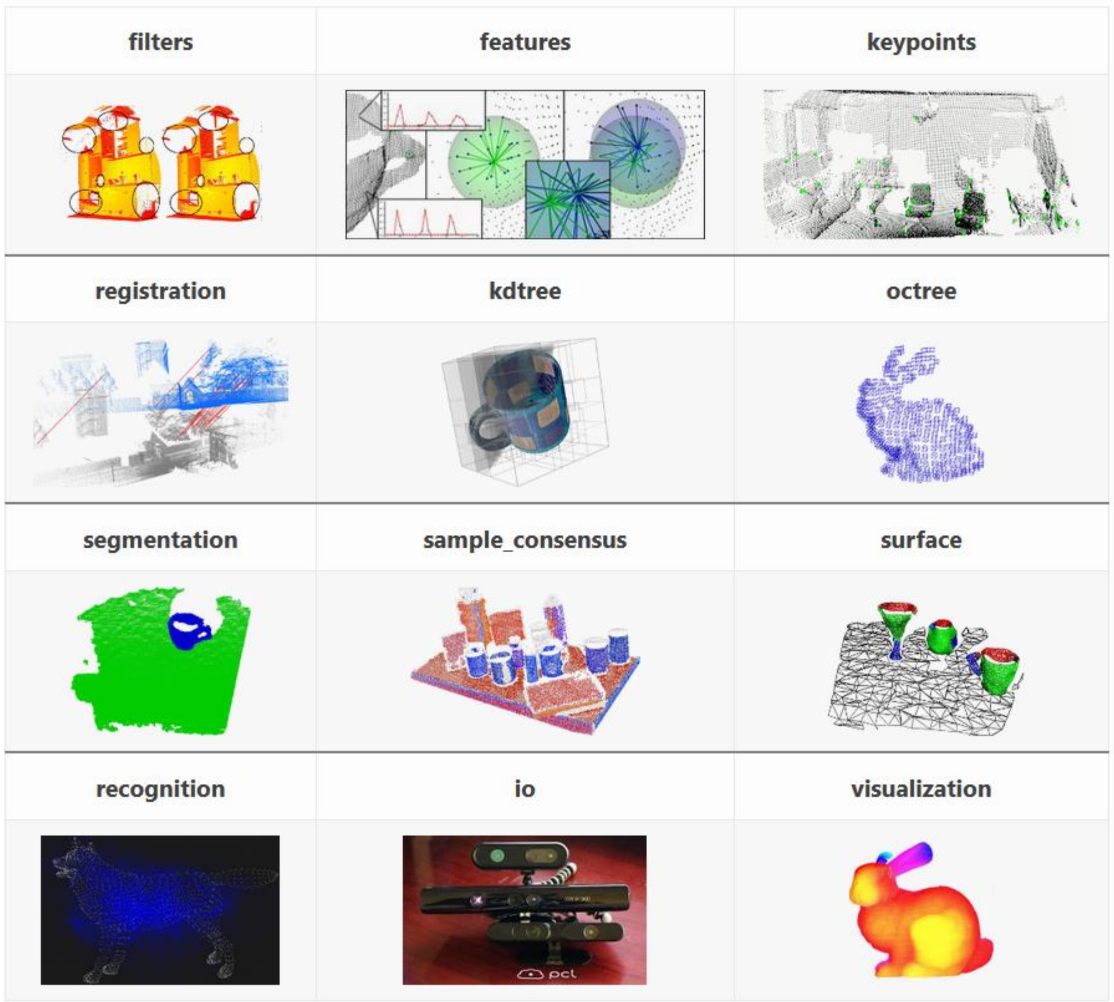
Close to 2000 unique downloads



WHAT IS PCL?

The Point Cloud Library (PCL) is a standalone, large scale, open project for 2D/3D image and point cloud processing. *[https://pointclouds.org/]*

PCL can be split into a series of modular libraries:



Function Lists

Now accelerated operations:

- CUDA-ICP
- CUDA-FILTER
- CUDA-SEGMENTATION
- CUDA-NDT
- CUDA-OCTREE
- CUDA-CLUSTER

CUDA-FILTER

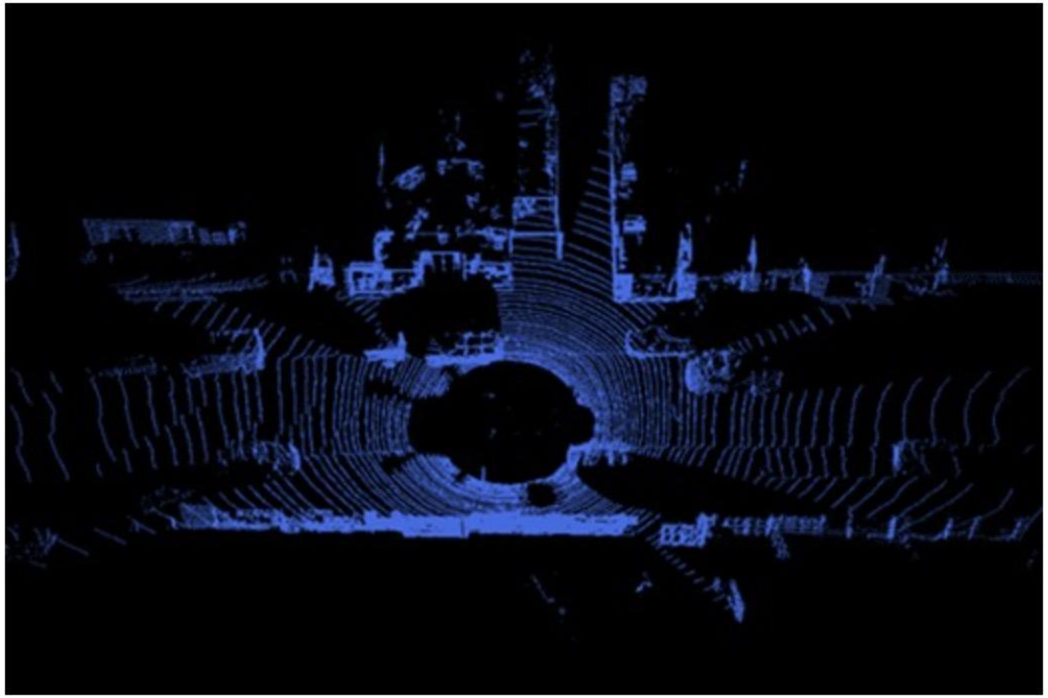
Overview:

- The first step in point cloud preprocessing
- Outliers removal
- Noise removal
- Downsampling

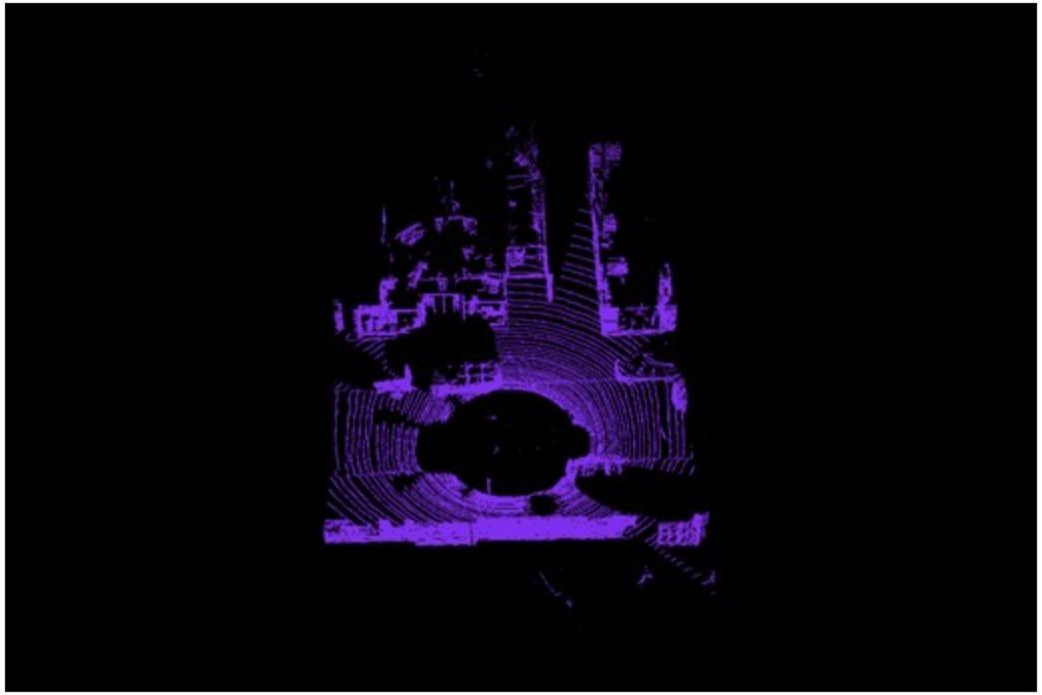
Now we support **PassThrough** filter and **VoxelGrid(Downsampling)** filter

CUDA-FILTER

An example of the PassThrough filter by constraint on the X axis.



Original point clouds



Point clouds filtered by constraint on the x axis

Official Blog - https://developer.nvidia.com/blog/accelerating-lidar-for-robotics-with-cuda-based-pcl/

CUDA-SEGMENTATION

Overview:

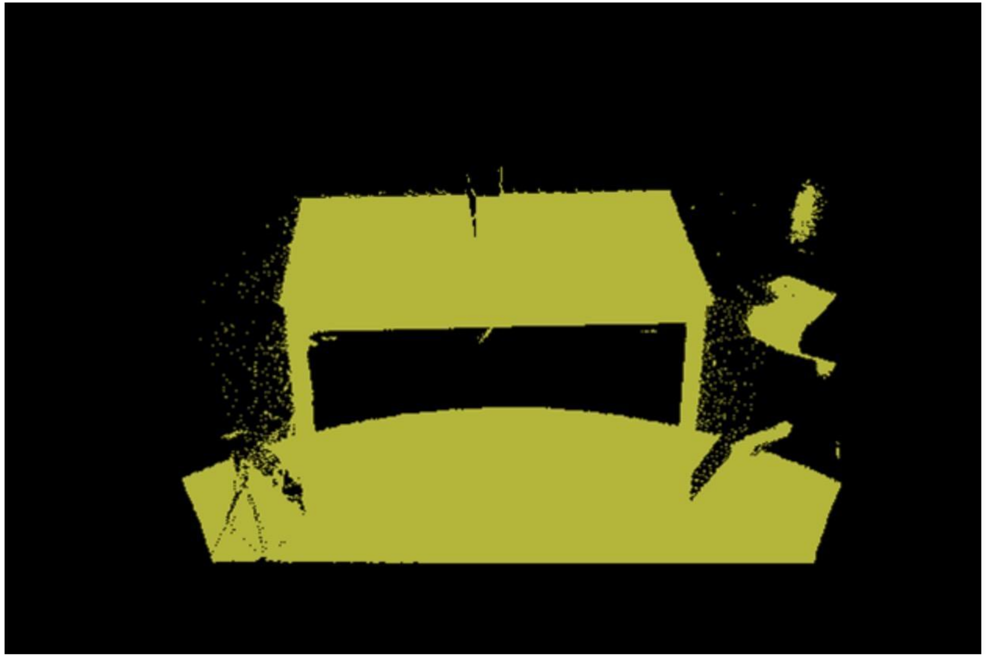
- Segmentation divides point clouds according to features such as space, geometry and texture, so that point clouds in the same division have similar features
- Effective segmentation of point clouds is often a prerequisite for many applications.
- For example, a point cloud map contains many ground points. This not only makes the whole map look messy but also brings trouble to the classification, identification, and tracking of subsequent obstacle point clouds, so it needs to be removed firstly

Now we support a **RandomSampleConsensus** with a **plane** model (SAC_RANSAC + SACMODEL_PLANE)

Official Blog - https://developer.nvidia.com/blog/accelerating-lidar-for-robotics-with-cuda-based-pcl/

CUDA-SEGMENTATION

The example below shows the original point cloud data and then a version processed with only obstacle-related point clouds remaining



Original image

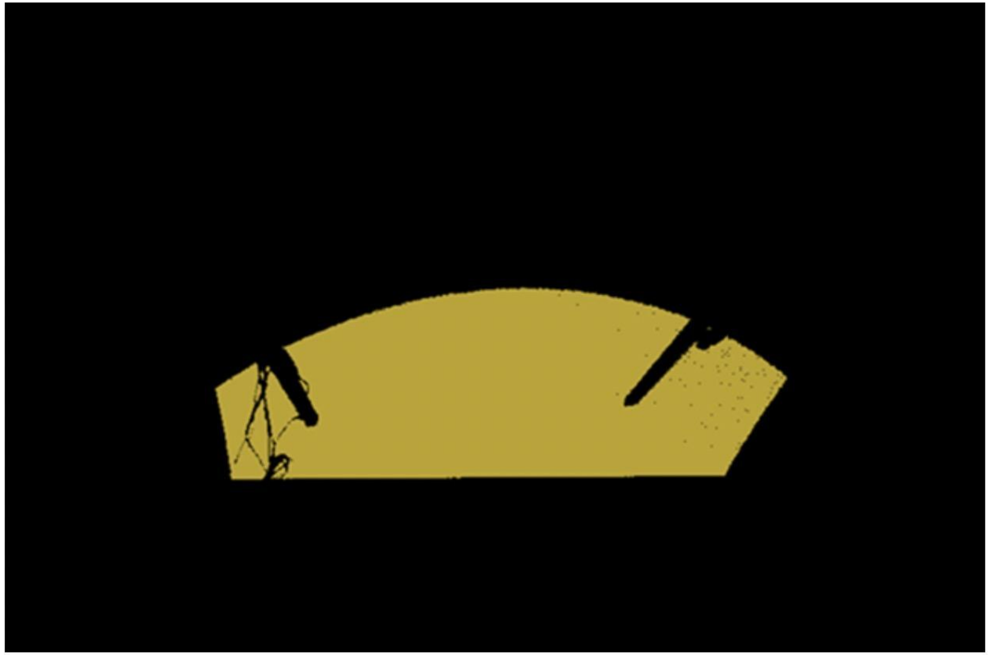


Image processed by CUDA-SEGMENTATION

Official Blog - https://developer.nvidia.com/blog/accelerating-lidar-for-robotics-with-cuda-based-pcl/

CUDA-ICP

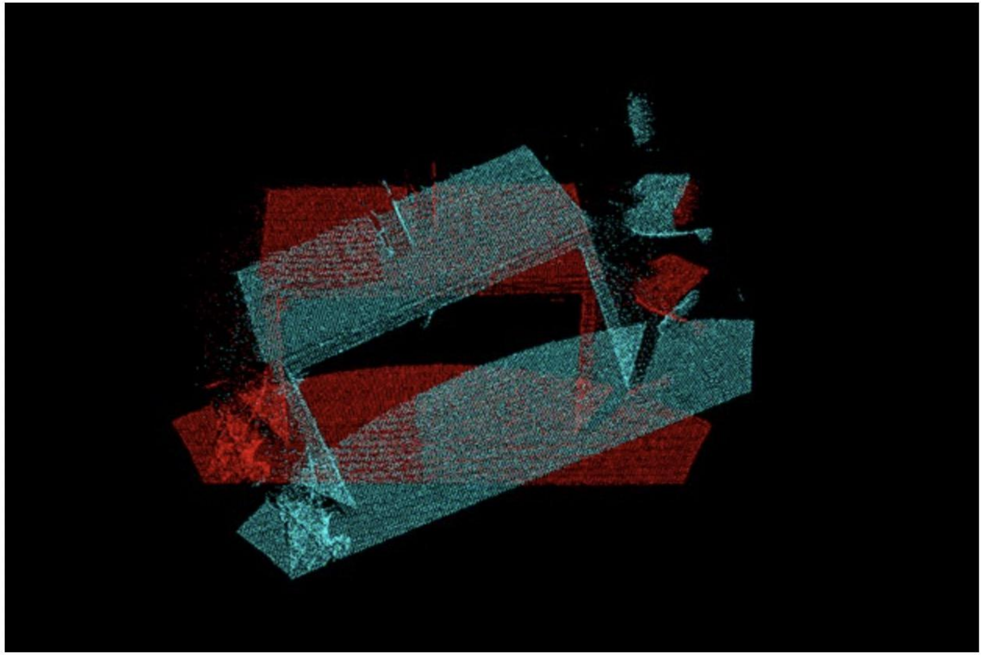
Overview:

- Iterative closest point --- iteratively revises the transformation (combination of translation and rotation) needed to minimize the distance from the source to the reference point cloud
- Registration Algorithm
- Merge point clouds of multiple views into a globally consistent model

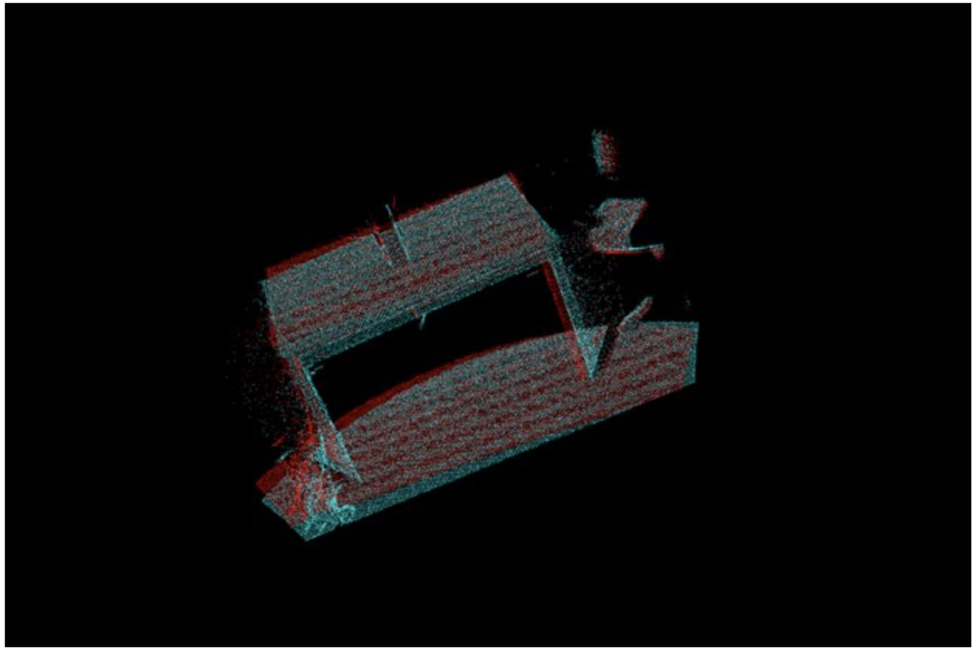
Official Blog - https://developer.nvidia.com/blog/accelerating-lidar-for-robotics-with-cuda-based-pcl/

CUDA-ICP

ICP takes the point clouds in two different coordinate systems so that one point cloud coordinate system is used as the **global coordinate system**, and the other point cloud is **rotated and translated**, and the overlapping parts of the two sets of point clouds completely overlap.



Two sets of point clouds before ICP



Two sets of point clouds after ICP

Official Blog - https://developer.nvidia.com/blog/accelerating-lidar-for-robotics-with-cuda-based-pcl/

CUDA-NDT

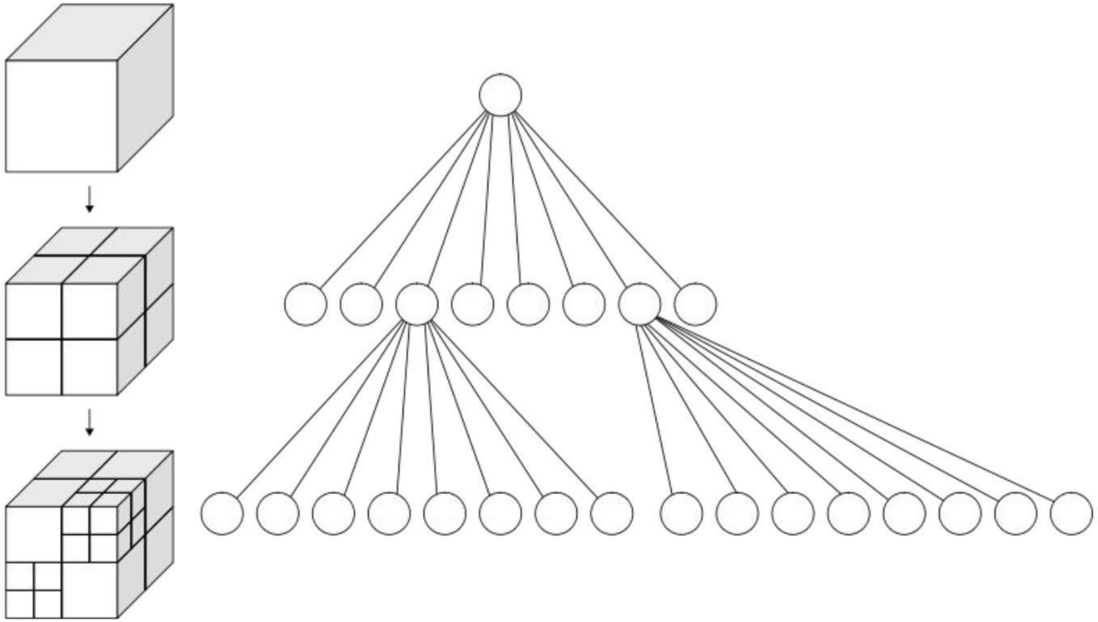
Overview:

- Another points cloud registration algorithm like ICP
- The NDT registration is faster but the accuracy is lower, while the ICP registration is slower but the accuracy is higher.
- So, the NDT and ICP algorithms can be combined to improve the registration accuracy and speed. First, the NDT algorithm can be used for rough registration to obtain the conversion parameters. Then use the ICP algorithm combined with the parameters for fine registration.

CUDA-OCTREE

Overview:

- The octree is the scene management used in 3D space, which can quickly know the position of the object in the 3D scene, or detect whether there is a collision with other objects and whether it is within the visible range.
- Octree can be used to accelerate sorting algorithm.



Now we support **Approximate Nearest Search** and **Radius Search**

CUDA-CLUSTER

Overview:

- After obtaining the scene point cloud, clusteringing the point cloud is helpful to use the point cloud information to understand the content of the point cloud scene

- Labeling is required for each cluster

- Cluster based on the distance between points

Official Blog - https://developer.nvidia.com/blog/accelerating-lidar-for-robotics-with-cuda-based-pcl/

Step-by-step Guidance

STEP-BY-STEP GUIDANCE

Install Jetpack by SDKManager

Install PCL (Eigen included):

```
$sudo apt-get update
$sudo apt-get install libpcl-dev
```

Clone the ***repo*** and go to the folders you are interested in and Build:

```
$make
```

Boost CPU and GPU firstly:

```
$sudo nvpmodel -m 0
$sudo jetson_clocks
```

Run:

```
$./demo
```

Official Blog - https://developer.nvidia.com/blog/accelerating-lidar-for-robotics-with-cuda-based-pcl/

Performance Comparison

Performance Comparison

2~10x performance boost!

Iterative Closest Point (ICP)		GPU	GICP	ICP
	count of points cloud	7000	7000	7000
	maximum of iterations	20	20	20
	cost time(ms)	55.1	364.2	523.1
	fitness_score	0.514	0.644	0.525

Segmentation		GPU	CPU
	count of points cloud	11w+	11w+
	Points selected	7519	7519
	cost time(ms)	14.5712	67.2766

Filter	**PASSTHROUGH filter**	GPU	CPU
	count of points cloud	11w+	11w+
	dim	X	X
	down,up FilterLimits	(-0.5, 0.5)	(-0.5, 0.5)
	limitsNegative	false	false
	Points selected	5110	5110
	cost time(ms)	0.660954	2.97487
	VoxelGrid filter	GPU	CPU
	count of points cloud	11w+	11w+
	LeafSize	(1,1,1)	(1,1,1)
	Points selected	3440	3440
	cost time(ms)	3.12895	7.26262

Github - https://github.com/NVIDIA-AI-IOT/cuda-pcl.

Performance Comparsion

4~40x performance boost!

Octree	**Approximate Nearest**	GPU	CPU
	Point counts of tree	7000	7000
	Point counts of target	7000	7000
	Distance error	0.721	2.75
	cost time(ms)	2.55	11.67
	Radius search	GPU	CPU
	Point counts of tree	7000	7000
	Points selected	4751	4751
	cost time(ms)	0.083	1.29
Cluster		GPU	CPU
	count of points cloud	17w+	17w+
	cost time(ms)	10.3122	4016.85

Github - https://github.com/NVIDIA-AI-IOT/cuda-pcl.

Performance Comparsion

Environment:

- Jetson Xavier AGX 8GB
- Jetpack 4.4.1
- CUDA 10.2
- PCL 1.8
- Eigen 3

Github - https://github.com/NVIDIA-AI-IOT/cuda-pcl.

Introduction to CUDA-PointPillars

GITHUB & USERS

https://github.com/NVIDIA-AI-IOT/CUDA-PointPillars

Our github:

About

A project demonstrating how to use CUDA-PointPillars to deal with cloud points data from lidar.

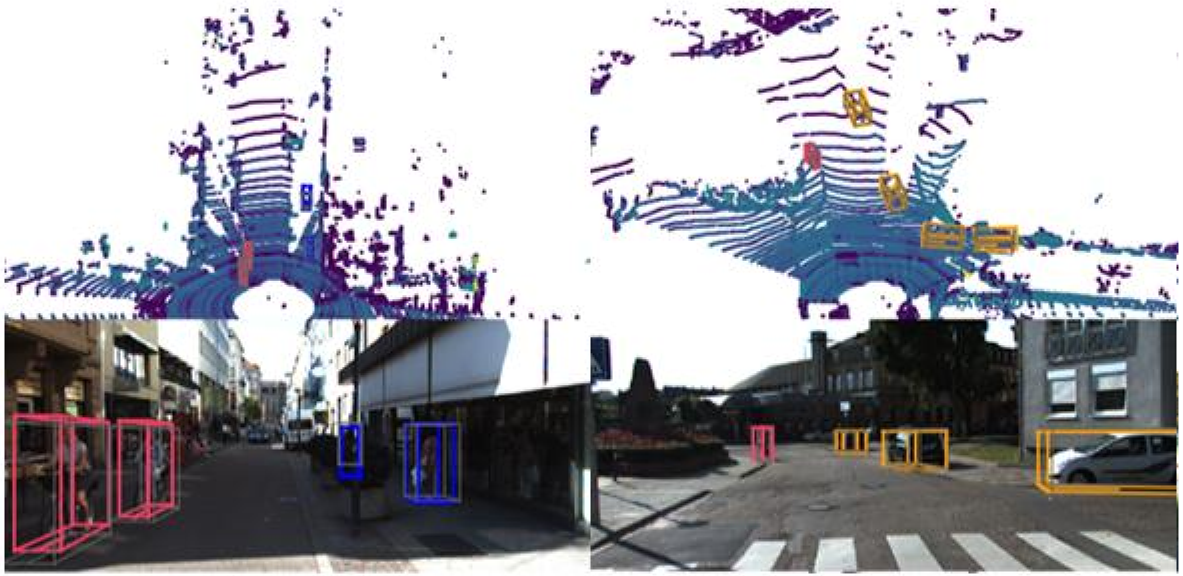
- Readme
- Apache-2.0 license
- 309 stars
- 9 watching
- 92 forks

Our users:

Since CUDA-PointPillars has been placed on github, it is available for all developers

WHAT IS PointPillars?

It is a novel encoder which utilizes PointNets to learn a representation of point clouds organized in vertical columns (pillars)
https://arxiv.org/abs/1812.05784



Our github contains sources and model for pointpillars inference using TensorRT

Github - https://github.com/NVIDIA-AI-IOT/CUDA-PointPillars

Pipeline of CUDA-PointPillars

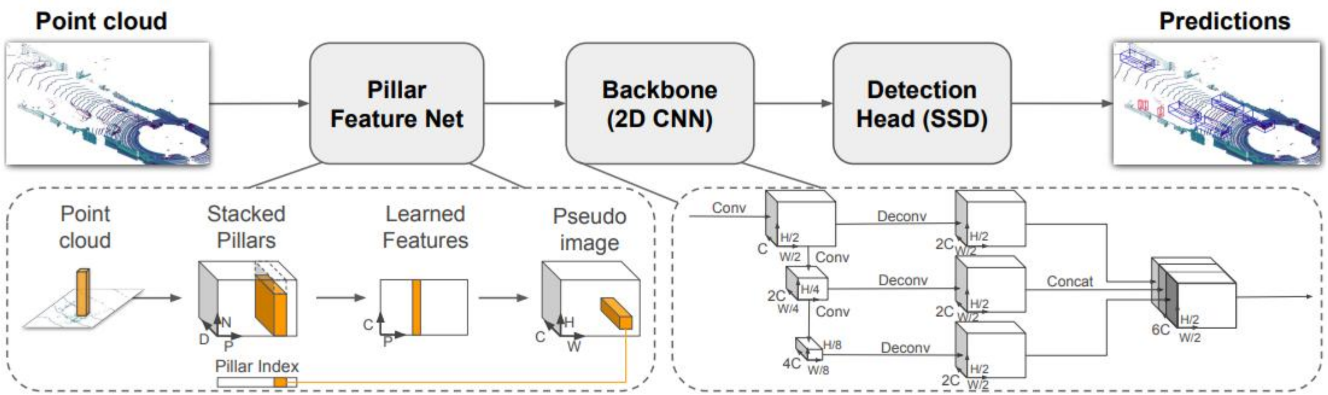
Overall inference has four phases:

- Convert points cloud into 4-channle voxels (**CUDA kernel**)
- Extend 4-channel voxels to 10-channel voxel features (**CUDA kernel**)
- Run TensorRT engine to get 3D-detection raw data (**TensorRT engine**)
- Parse bounding box, class type and direction (**CUDA kernel**)

Github - https://github.com/NVIDIA-AI-IOT/CUDA-PointPillars

Pipeline of CUDA-PointPillars

Network overview:



CUDA-PointPillars:

Points to Pillar (CUDA kernel)

PFN + 2D-CNN + SSD (TensorRT)

Decoder (CUDA kernel)

Github - https://github.com/NVIDIA-AI-IOT/CUDA-PointPillars

Step-by-step Guidance

STEP-BY-STEP GUIDANCE

Prerequisites:

TensorRT with PillarScatter layer is needed. PillarScatter layer plugin is already implemented as a plugin for TRT in the demo.

CUDA is needed.

Compile:

```
$ sudo apt-get install git-lfs
$ git lfs install
$ git clone https://github.com/NVIDIA-AI-IOT/CUDA-PointPillars.git && cd CUDA-PointPillars
$ mkdir build && cd build
$ cmake .. && make -j$(nproc)
```

Run:

```
$./demo
```

Github - https://github.com/NVIDIA-AI-IOT/CUDA-PointPillars

Performance Comparison

Performance Comparison

Performance of each parts within the pipeline:

Function(unit:ms)	Xavier	Orin
GenerateVoxels	0.29	0.14
GenerateFeatures	0.31	0.15
Inference	20.21	9.12
Postprocessing	3.38	1.77
Overall	24.19	11.18

Github - https://github.com/NVIDIA-AI-IOT/CUDA-PointPillars

Performance Comparison

3D detection performance of moderate difficulty on the val set of KITTI dataset:

	Car@R11	Pedestrian@R11	Cyclist@R11
CUDA-PointPillars	77.02	51.65	62.24
OpenPCDet	77.28	52.29	62.68

Github - https://github.com/NVIDIA-AI-IOT/CUDA-PointPillars

Performance Comparison

Environment:

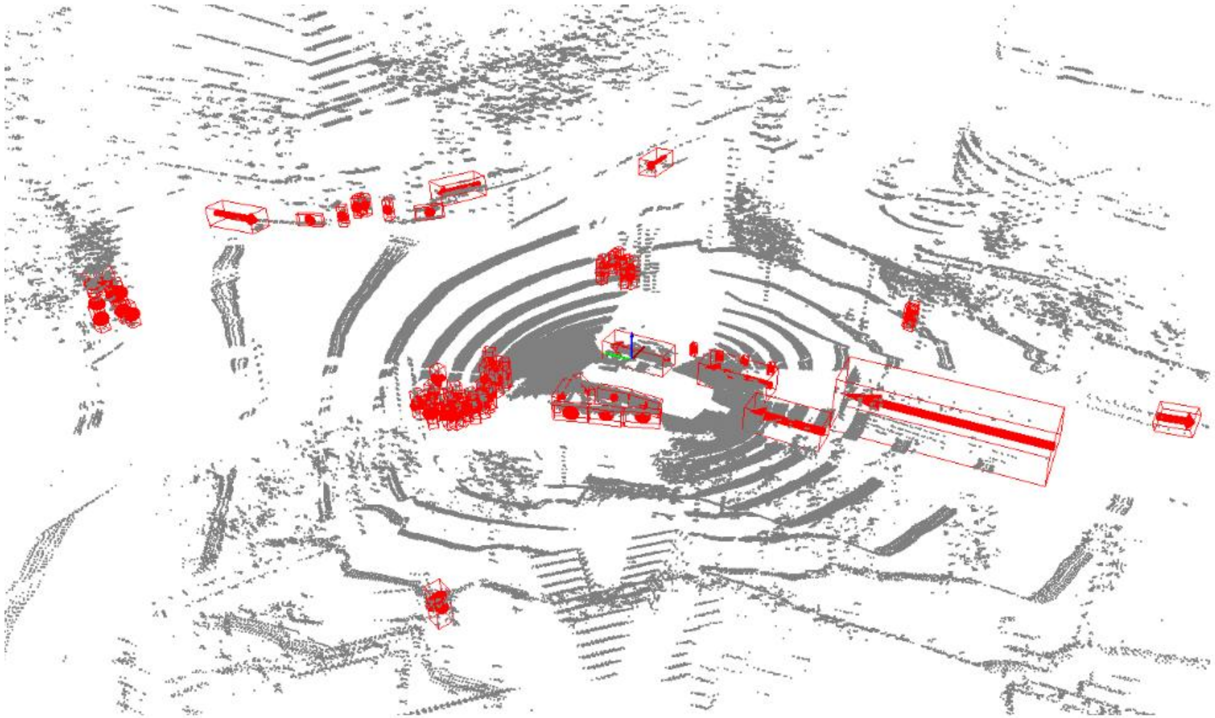
- Nvidia Jetson Xavier/Orin
- Jetpack 5.0
- CUDA 11.4
- cuDNN 8.3.2
- TensorRT 8.4.0

Github - https://github.com/NVIDIA-AI-IOT/CUDA-PointPillars

Introduction to CUDA-CenterPoint

WHAT IS CenterPoint?

CenterPoint, first detects centers of objects using a keypoint detector and regresses to other attributes, including 3D size, 3D orientation, and velocity. In a second stage, it refines these estimates using additional point features on the object.
https://arxiv.org/abs/2006.11275



For now we haven’t published it, we are still in the process of optimization. But feel free to contact me if you are interested!

NVIDIA.

Pipeline of CUDA-CenterPoint

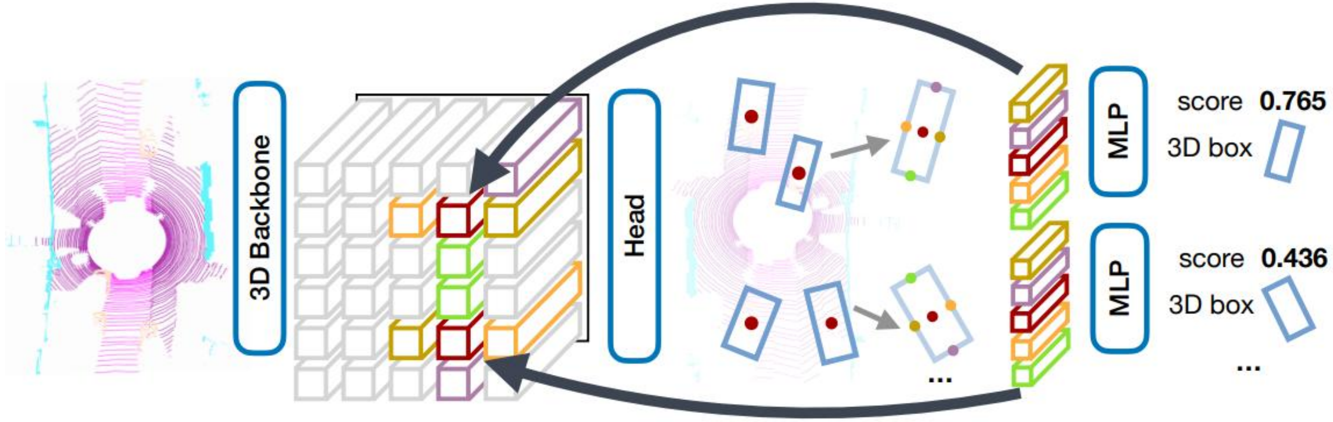
Overall inference has four phases:

- Voxelization (**CUDA kernel**)
- Voxels feature extraction (**CUDA kernel**)
- Run TensorRT engine to get 3D-detection raw data (3D backbone + RPN + Center Head: **TensorRT engine**)
- Parse bounding box, class type and direction (**CUDA kernel**)

Pipeline of CUDA-CenterPoint

Center-based 3D Object Detection and Tracking

Network:



CUDA-CenterPoint:

Points to Voxel (CUDA kernel)

SCN (Mini-Engine) + RPN + Center Head (TensorRT)

Decoder + NMS (CUDA kernel)

Performance Comparison

Performance Comparison

Average perf on A30 and Orin using the nuScenes validation dataset:

Unit : ms	Precision	A30	Orin
Voxelization	FP32	4.7	6.0
3D Backbone	FP16	9.8	22.3
RPN + Head	FP16/INT8	3.8/2.5	11.3/7.0
Decode + NMS	FP16	3.1	4.4
Total	MIXED	21.3/20.0	44.0/39.7

Performance Comparison

Environment:

- Nvidia Drive Orin 6.0.3.0
- CUDA 11.4 + cuDNN 8.3.3 + TensorRT 8.4.10.4
- &&
- Ubuntu20.04 x86_64 with Nvidia Tesla A30
- CUDA 11.4 + cuDNN 8.4.1 + TensorRT 8.4.12.5

End-to-end Solution

OVERVIEW

3D tasks call for more TOPs, thus more powerful GPU on AV

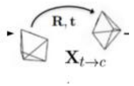
Perception Task

Raw scans → Filter/Ground Removal → Learned model → Rotated NMS → Detections

Localization Task

Raw scans → Learned model → Abs Pose → KF/EKF → 6-D Pose

Raw scans → ICP/NDT → Transformation → Rel Pose → KF/EKF



NVIDIA