UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

New! 2021

3D Augmented Reality

A.Y. 2021/2022

LiDAR Simulator & Point Cloud Visualization

LAB experience 5

Elena Camuffo

elena.camuffo@unipd.it

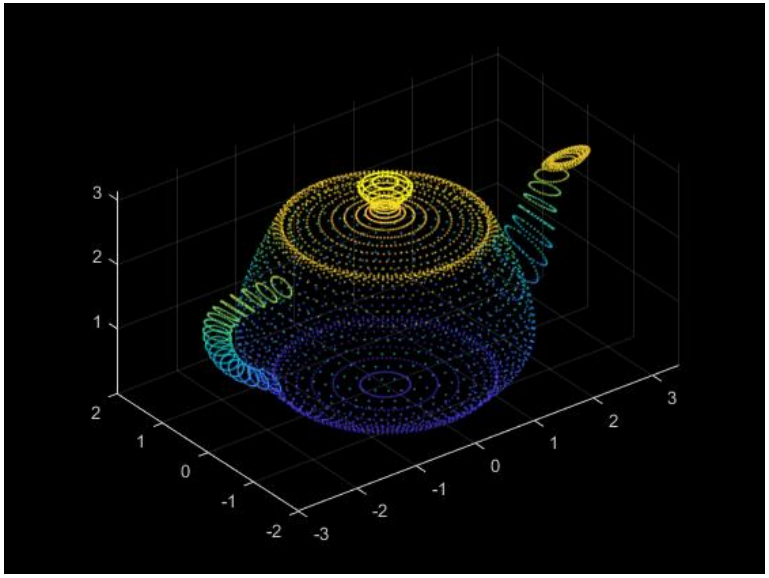
Point Clouds

- A **point cloud** is a *set of data points in the 3D space*. Each point is spatially defined by a triplet of coordinates and a combination of such points can be used to describe the geometry of an object or the complete scene.

$$P = \bigcup_{p_i \in R^3} p_i \,, \qquad p_i = (x_i, y_i, z_i)$$

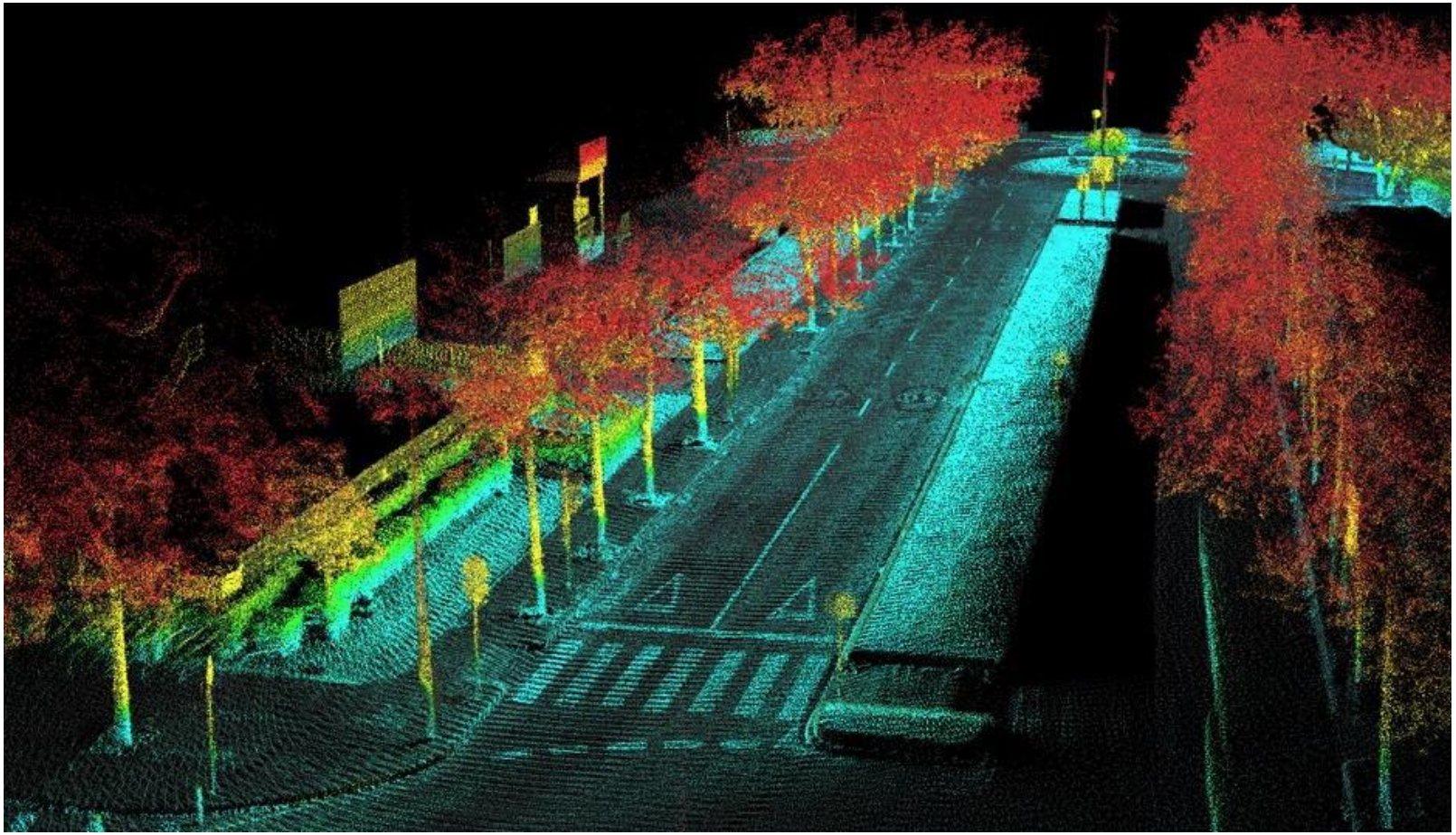
single point

point cloud



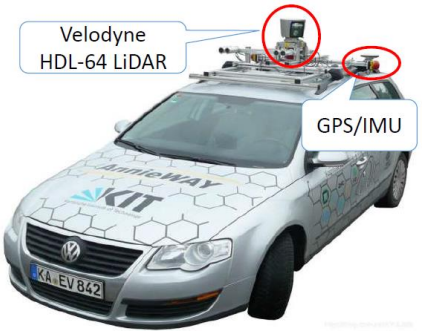
- They are easily obtained as a *map of the external surface of objects* through reality capture devices, like **LiDAR sensors**.

LiDAR Point Clouds

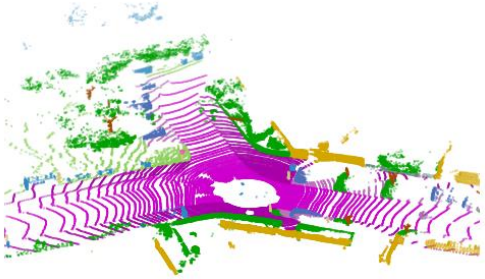


LiDAR Acquisition Sensor

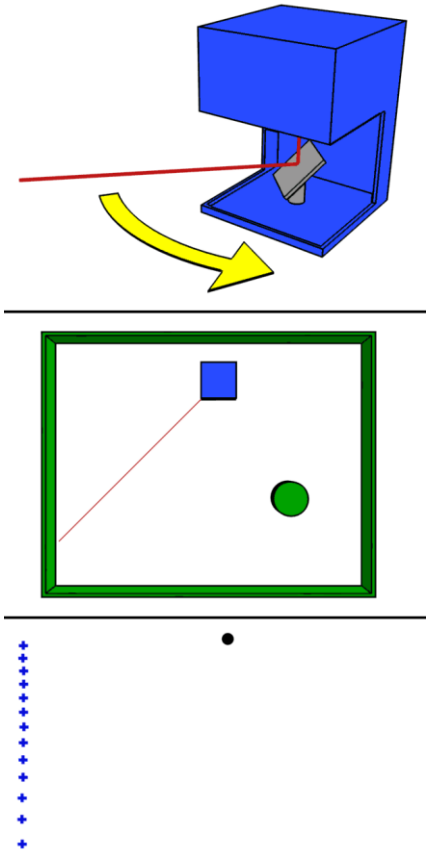
- Light Detection and Ranging or **Laser Imaging Detection and Ranging (LiDAR)** are particular detection systems which work on the principle of radar but use light from a laser.
- These sensors produce a **sparse prediction** of the environment composed of point cloud data.



acquisition system



example point cloud
of KITTI dataset



LiDAR Point Clouds famous datasets

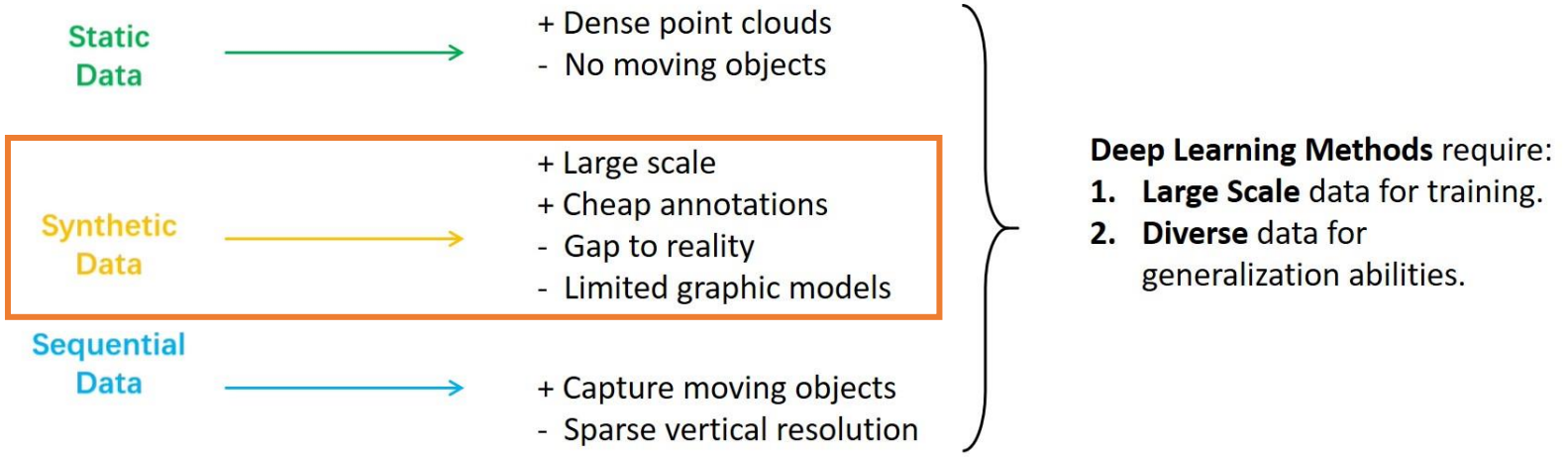
- There are not yet many LiDAR publicly available datasets, and most of them are outdoor environments of autonomous driving scenarios.
- Just a couple of them are **Synthetic Datasets,** i.e., acquired with simulated LiDAR in a virtual environment.

	Dataset	Year	Frames	Points	Scene	Instance	Sequential
Static Data	Oakland	2009	17	1.6M	outdoor	✗	✗
	Paris-rue-Madame	2014	2	20M	outdoor	✓	✗
	TerraMobilita/IQmulus	2015	10	12M	outdoor	✓	✗
	S3DIS	2016	5	215M	indoor	✗	✗
	Semantic3D	2017	30	4009M	outdoor	✗	✗
	Paris-lille-3D	2018	3	143M	outdoor	✓	✗
Synthetic Data	GTA-V	2018	/	/	outdoor	✗	✓
	SynthCity	2019	75000	367.9M	outdoor	✗	✓
Sequential Data	Sydney Urban	2013	631	/	outdoor	✓	✓
	SemanticKITTI	2019	43552	4549M	outdoor	✓	✓
	SemanticPOSS	2020	2988	216M	outdoor	✓	✓

Point Cloud Semantic Segmentation Datasets

LiDAR Point Clouds famous datasets

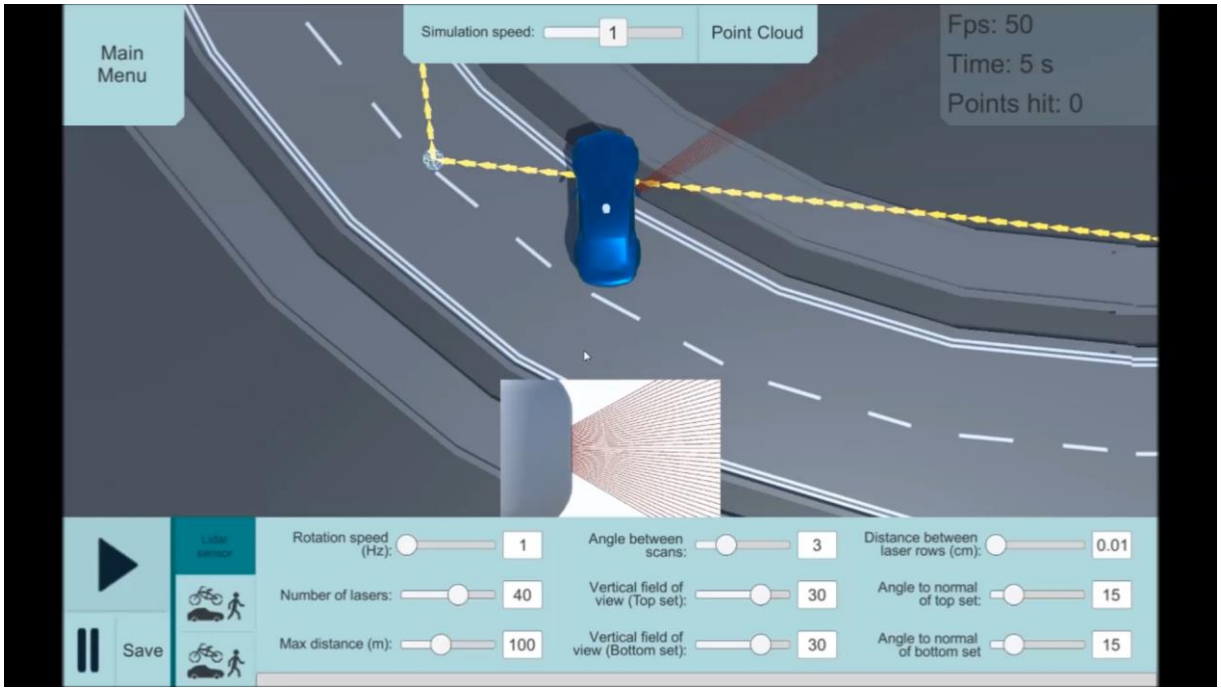
- There are not yet many LiDAR publicly available datasets, and most of them are outdoor environments of autonomous driving scenarios.
- Just a couple of them are **Synthetic Datasets,** i.e., acquired with simulated LiDAR in a virtual environment.
- These datasets are **newer** with respect to the others and can hold thousands of thousands of points, collecting and annotating them easily.



LiDAR Simulator in Unity Example

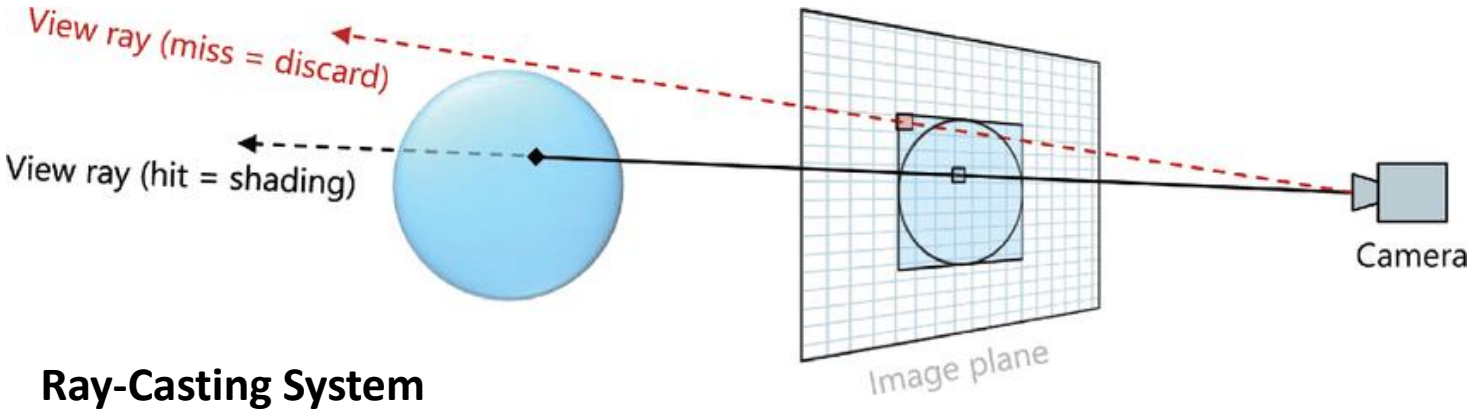
- This example is a complete environment for **simulating an autonomous driving scenario**.
- You can download the whole project cloning the following GitHub repository:

  https://github.com/ptibom/Lidar-Simulator



Simulation of a LiDAR Sensor

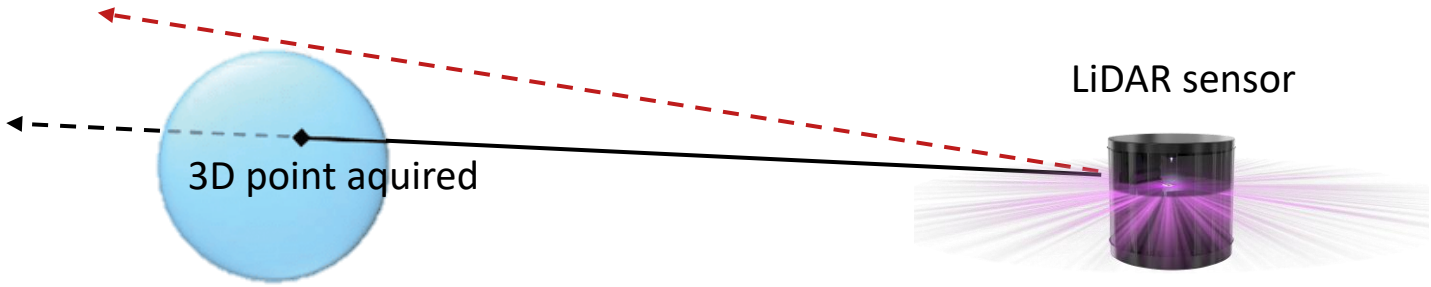
- The **Velodyne HDL-64E sensor** is used as a model for this LiDAR simulator.
- Each LiDAR sensor has some distinguishable key components that identify it, such as the **number of lasers**, each individual laser **position and angle**, and the **rotational speed**.
- Each laser is represented in the game engine by using a method called **ray-casting,** *i.e.,* a *directional three-dimensional vector, which checks for intersections with other geometries, and then returns a coordinate of the intersected position*.



Ray-Casting System

Simulation of a LiDAR Sensor

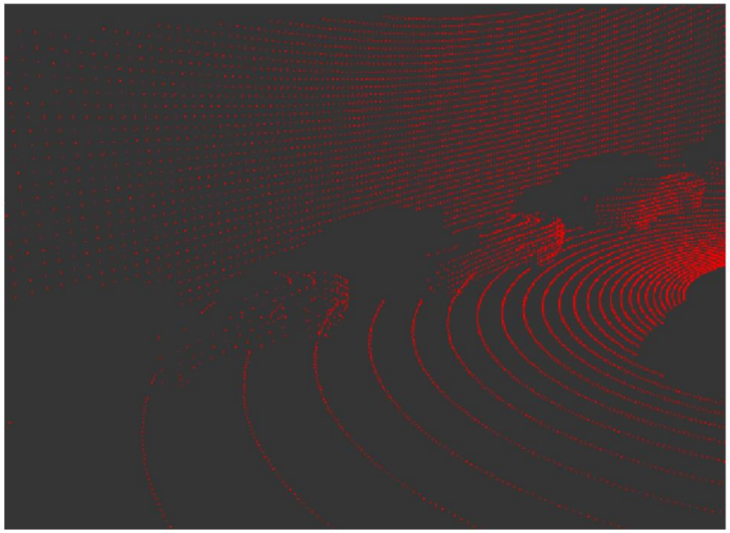
- The **Velodyne HDL-64E sensor** is used as a model for this LiDAR simulator.
- Each LiDAR sensor has some distinguishable key components that identify it, such as the **number of lasers**, each individual laser **position and angle**, and the **rotational speed**.
- Each laser is represented in the game engine by using a method called **ray-casting,** *i.e.,* a *directional three-dimensional vector, which checks for intersections with other geometries, and then returns a coordinate of the intersected position*.



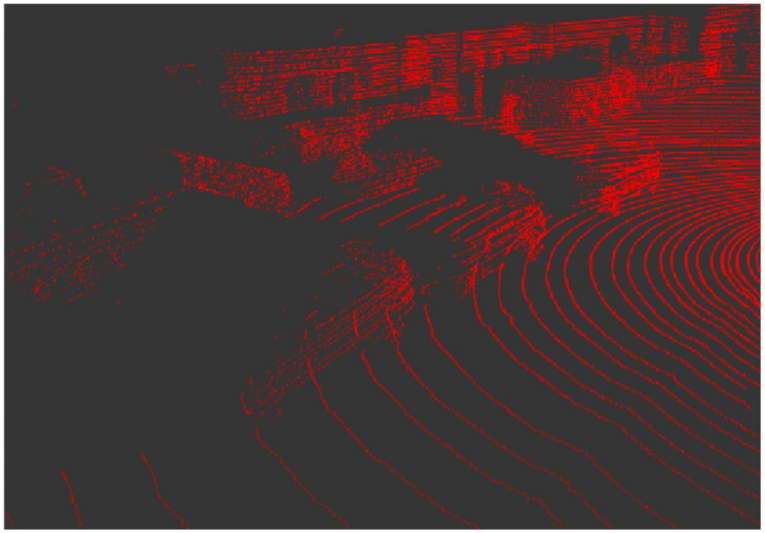
Ray-Casting for
Simulating LiDAR in Unity

Simulated vs Real LiDAR scans

- The simulated LiDAR is built from an **indefinite number of lasers** whose parameters can be adjusted during the simulation. With an accurate parameter tuning the result is similar to a real LiDAR point cloud.



Generated with Simulator

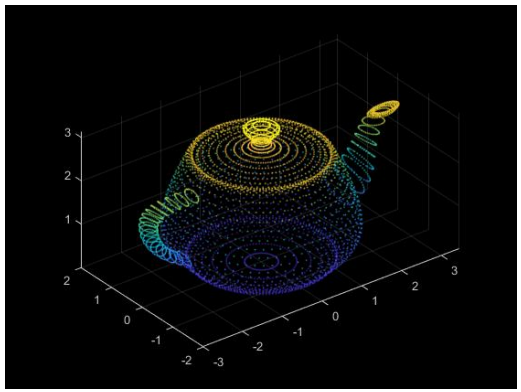


KITTI dataset [1]

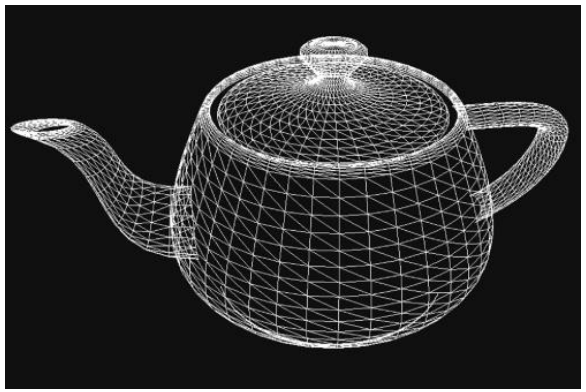
[1] Geiger, Andreas, et al. "Vision meets robotics: The kitti dataset." *The International Journal of Robotics Research* 32.11 (2013): 1231-1237.
Further details on the laser implementation here: https://publications.lib.chalmers.se/records/fulltext/251700/251700.pdf

LiDAR Data Management

- The LiDAR sensor **generates large amounts of data** that we can use for many purposes, such as data processing with Deep Learning methods. The data captured store the *coordinates of the scanned area during a span of time*.
- Once acquired, a **visualization tool** can be useful to allow the user inspect the collected data. The visualization can be achieved in two ways:
    1. By means of a surface via the **creation of a mesh**, *offline*.
    2. using a particle system with **fixed particles** in a space, *in real-time*.



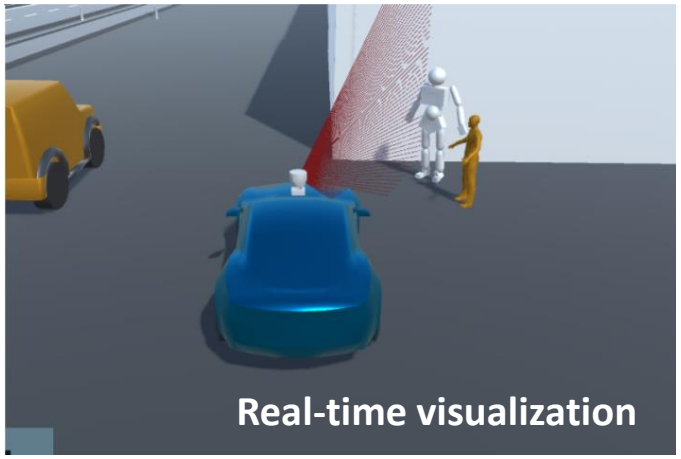
Point Cloud



Mesh

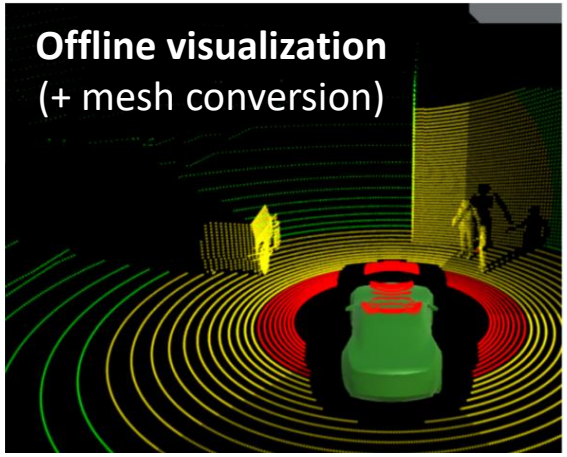
Real-time vs offline LiDAR scans

- Before the simulation it is possible to **place fixed and moving objects** in the environment.



LiDAR data

TXT




```
sep=
0 145.3044 157.6866 1.127946 1.916438 1.916438 0.01570796 0
0 145.3561 157.6828 1.122914 1.965633 1.965633 0.01570796 1
0 145.4105 157.6788 1.117616 2.017576 2.017576 0.01570796 2
0 145.4678 157.6747 1.112031 2.072503 2.072503 0.01570796 3
```

- Data are stored in *txt format*. Each row contains the **time stamp,** xyz-coordinates, the spherical coordinates where the car is in the origin, and the **id of the laser** that fired the ray.

Dealing with Point Cloud Data

- The example provides Point Clouds in simple **txt** format. However, Point Clouds can be stored in many other different formats, e.g., **CSV**, and the syntax depends on the specific format.
- The most common formats are **OFF, OBJ, PLY, FBX** and **binary ASCII.**

- In many cases Point Clouds are difficult data structures to deal with, and also not suitable for some purposes, so that they must be converted to **meshes** and/or **voxels.**

	Voxel	Point cloud	Polygon mesh
Memory efficiency	Poor	Not good	Good
Textures	Not good	No	Yes
For neural networks	Easy	Not easy	Not easy

- They are complex to be rendered. To produce any visual effect, **Shaders** can be employed.

Useful repository that provides tools for Point Cloud processing in **Python**: https://github.com/Dan8991/PCutils ***[Credits: Daniele Mari]***

Shaders in Unity

In Unity, shaders are divided into three broad categories. You use each category for different things, and work with them differently:

- **Shaders** that are **part of the graphics pipeline** are the most common type of shader. They perform calculations that *determine the color of pixels on the screen*. In Unity, you usually work with this type of shader by using Shader objects.
- **Compute shaders** perform calculations *on the GPU*, outside of the regular graphics pipeline.
- **Ray tracing** shaders perform calculations related to ray tracing.

You use Shader objects with materials to determine the **appearance of your scene**

Shaders in Unity

- A *shader asset* is an asset in your Unity project that defines a **Shader object**. It is a text file with a *.shader extension*. It contains *shader code*. There are two types of shaders: *vertex* and *fragment*.

```
1  // Shader for Vertex Color in the point
2  // cloud data
3  Shader "Custom/VertexColor"
4  {
5  Properties
6      {
7  _PointSize("PointSize", Float) = 10
8  }
9      SubShader
10     {
11     Pass
12         {
13         LOD 200
14
15         CGPROGRAM
16         #pragma vertex vert
17         #pragma fragment frag
18
19         // vertex shader inputs
20         struct VertexInput
21         {
22             float4 v : POSITION;
23             float4 color: COLOR;
24         };
25
26         // vertex shader outputs
27         struct VertexOutput
28         {
29             float4 pos : SV_POSITION;
30             float size : PSIZE;
31             float4 col : COLOR;
32         };
```

Shader

contains the name of the shader as a string and the whole shader.

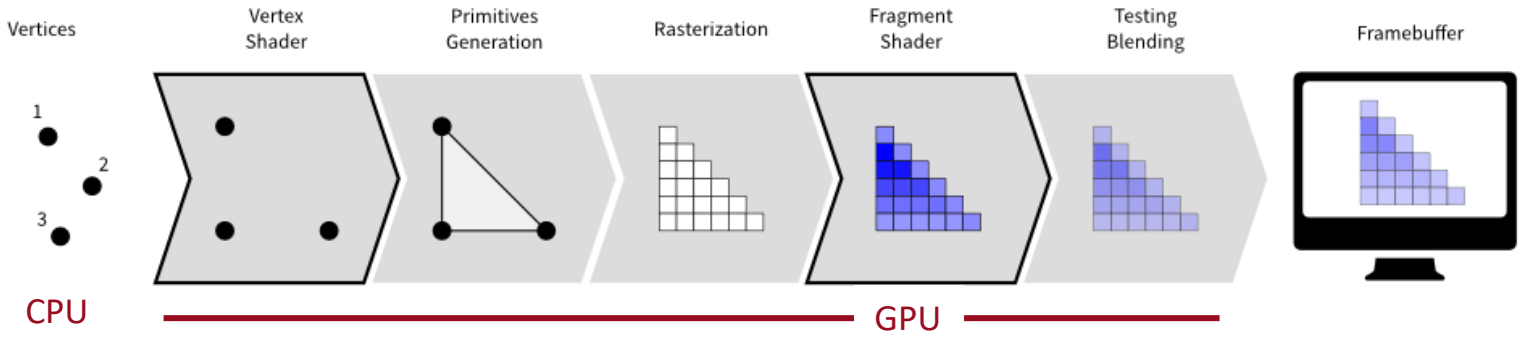
Properties

contains **shader variables** (textures, colors etc.) that will be saved as **part of the Material** and displayed in the material inspector.

Pass

represents an execution of the vertex and fragment code for the same object rendered with the material of the shader.

Shaders in Unity




```
32          float _PointSize;
33
34          // vertex shader
35          VertexOutput vert(VertexInput v)
36          {
37
38              VertexOutput o;
39              o.pos = UnityObjectToClipPos(v.v);
40              o.size = _PointSize;
41              o.col = v.color;
42
43              return o;
44          }
45
46          // fragment shader
47          float4 frag(VertexOutput o) : COLOR
48          {
49              return o.col;
50          }
51
52          ENDCG
53          }
54      }
55  }
```

The **Vertex Shader** is a program that runs on each vertex of the 3D model. Quite often it does not do anything particularly interesting, e.g., transform vertex position to rasterize the object on screen.

The **Fragment Shader** is a program that runs on every pixel that object occupies on-screen and is usually used to *calculate and output the color* of each pixel.

EXERCISE: Point Clouds Visualization Tool

1. **Export LiDAR Point Cloud with simulator**

   Use the demo LiDAR simulator to export your own .txt Point Cloud files and convert to CSV files, after properly set the environment.

2. **Write your Shader**

   Folder *Point Clouds* contains the material of the *Point Cloud Free Viewer* package downloaded from the Asset store. Try the default point cloud exportation of the mesh included in the package. Write a Shader modifying *VertexColor* shader properties, changing the appearance of the point clouds in such a way that the points look like a Neon material.

3. **Synthetic Point Cloud parsing**

   The only supported format for parsing is .OFF format, but our Point Clouds. Starting from that script, parse also the .txt data provided by the LiDAR (lidar.txt example file).

   Once parsed the .txt file, render it as a mesh, like it is performed for the .OFF files. Otherwise, you can write a script to convert your .txt point cloud to a .OFF file.

4. **(OPTIONAL) Visualize Real Point Clouds**

   You can try to repeat the procedure with different data formats and popular datasets of Point Clouds.