Processing Laser Data

ROS + PR2 Training Workshop

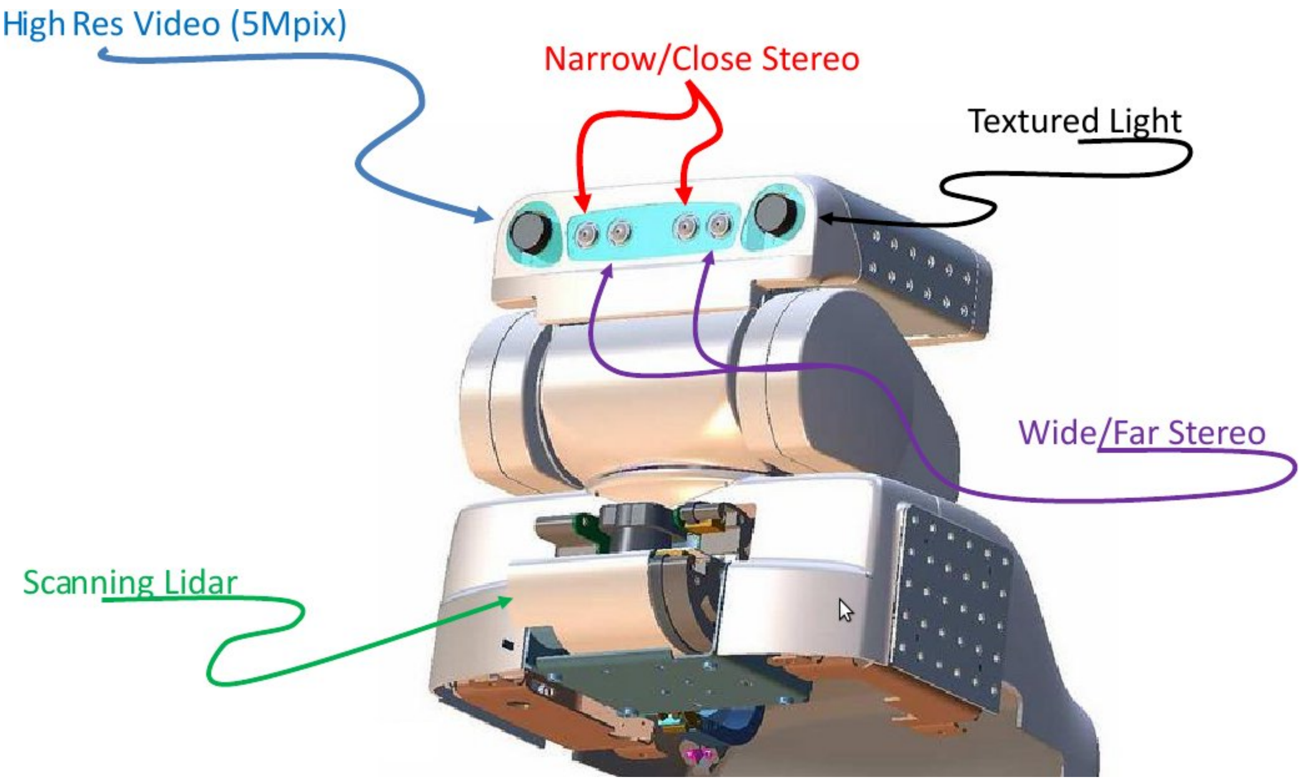
Outline

- Lasers and 3D sensing
- Visualizing Laser Scans

- From LaserScan to PointCloud
- What are Point Clouds?
- Data representation
- Visualizing PointCloud messages

- Preview :: ROS C-Turtle (latest)

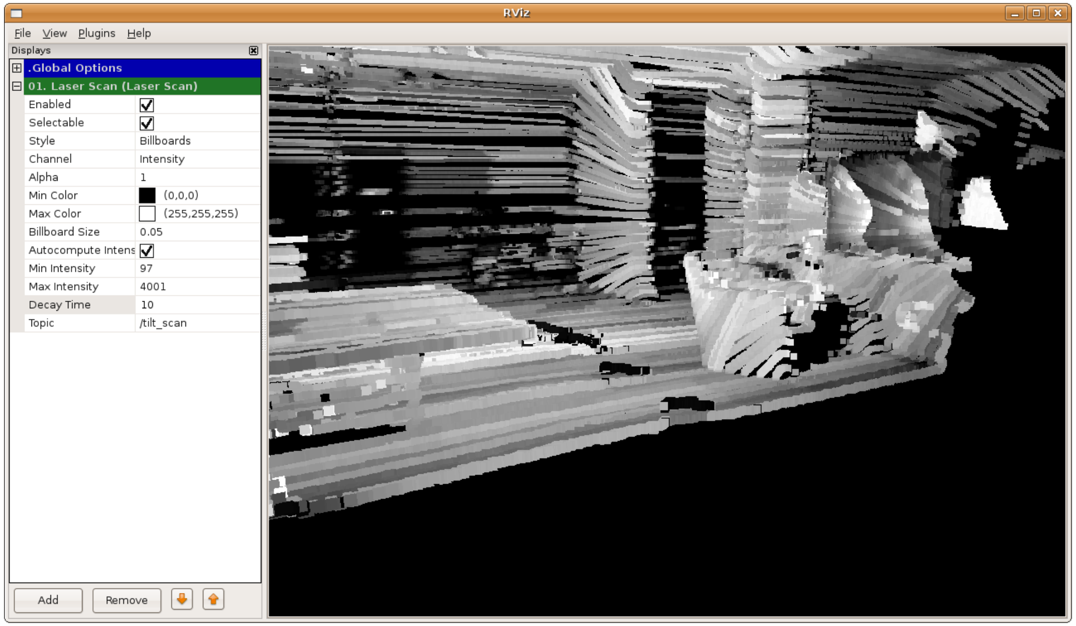
Lasers and 3D sensing



- Stereo cameras in the head
- Tilting laser range finder
- Base laser range finder

Visualizing Laser Scans

- rviz (http://www.ros.org/wiki/rviz/DisplayTypes/LaserScan)
  - $ rosrun rviz rviz
  - Add a Laser Scan display
  - Set the topic and the TF frames (Fixed/Target)

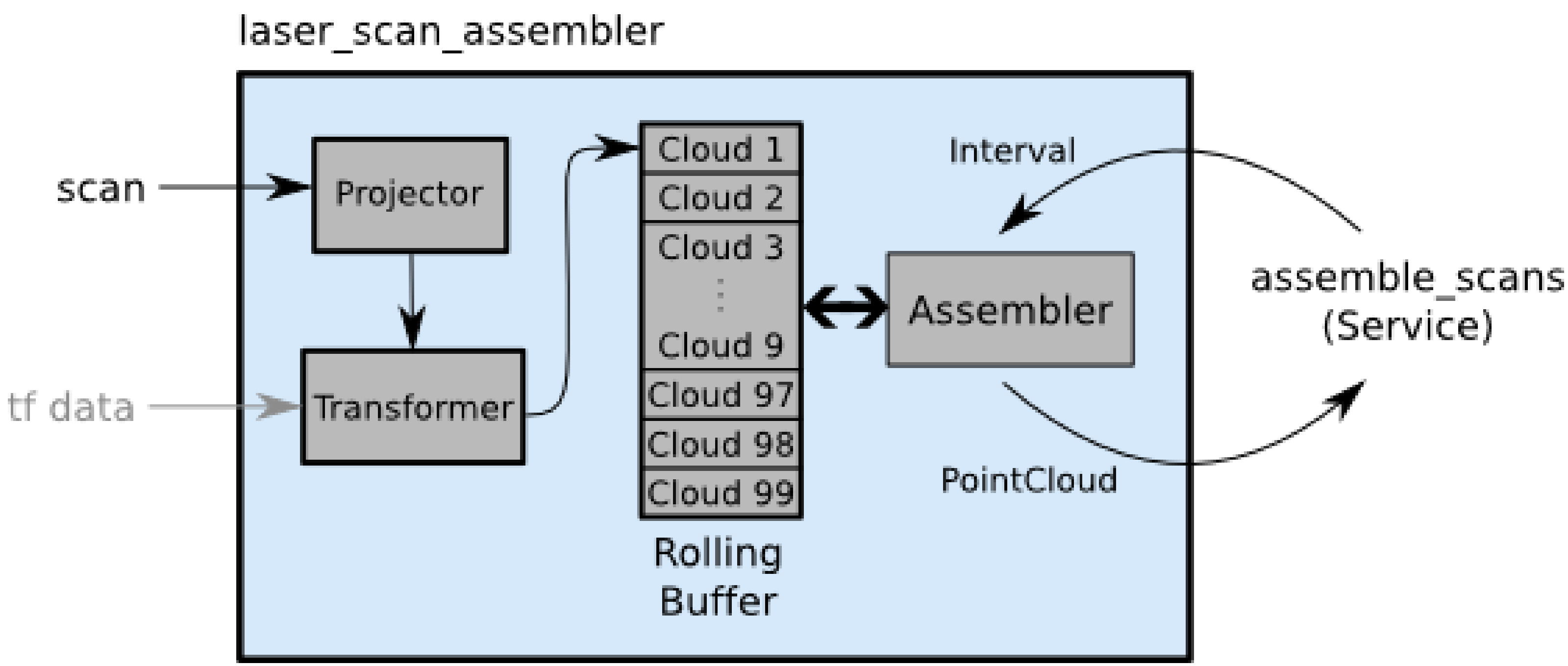


Outline

- Lasers and 3D sensing
- Visualizing Laser Scans
- From LaserScan to PointCloud
- What are Point Clouds?
- Data representation
- Visualizing PointCloud messages
- Preview :: ROS C-Turtle (latest)

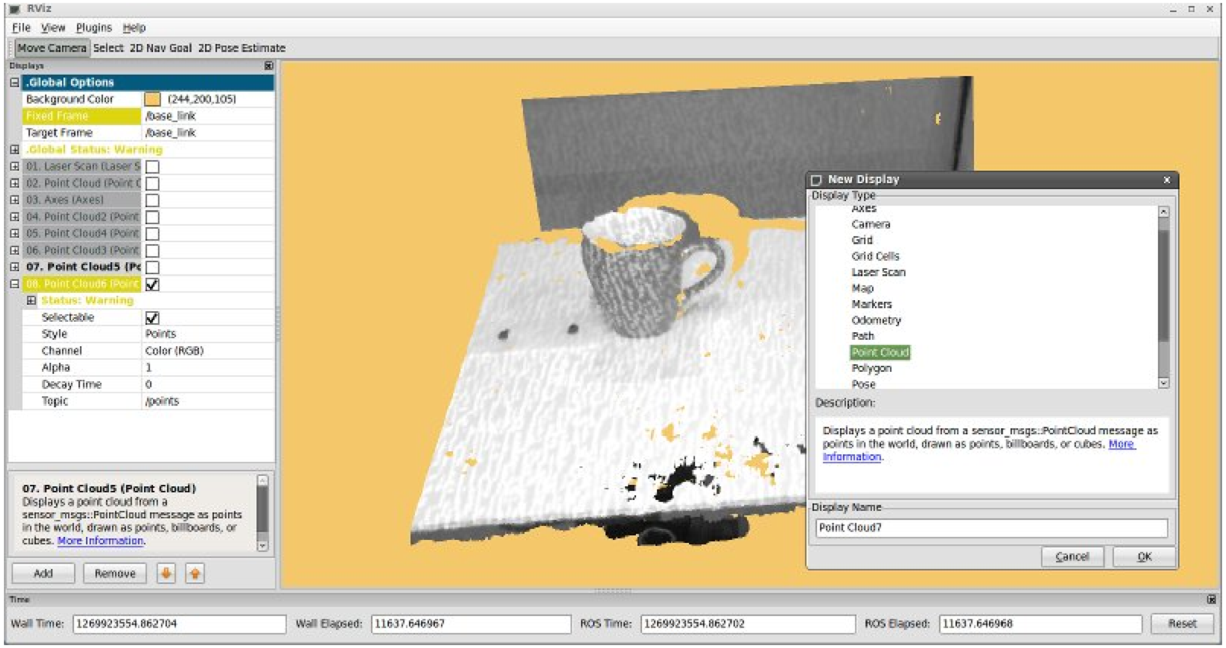
From LaserScan to PointCloud

- Check if the tilt laser is being actuated
  (http://www.ros.org/wiki/pr2_mechanism_controllers/LaserScannerTrajController)
- laser_assembler (http://www.ros.org/wiki/laser_assembler)
  - create PointCloud from LaserScan messages

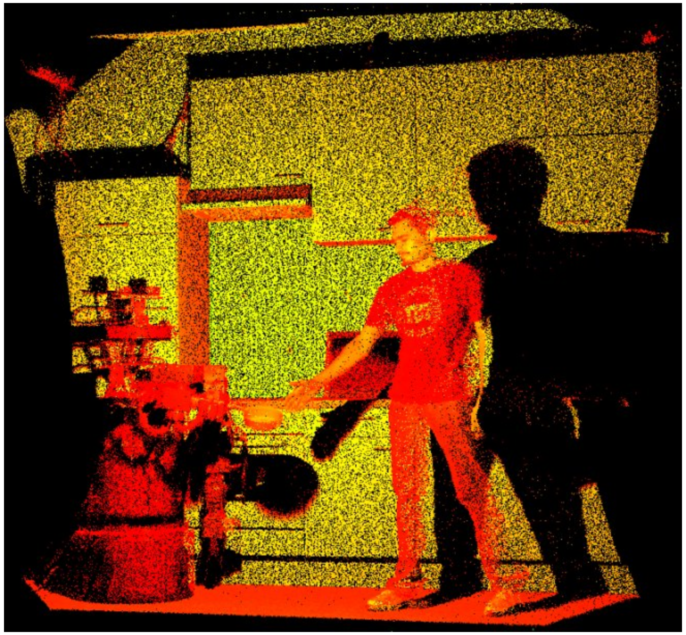


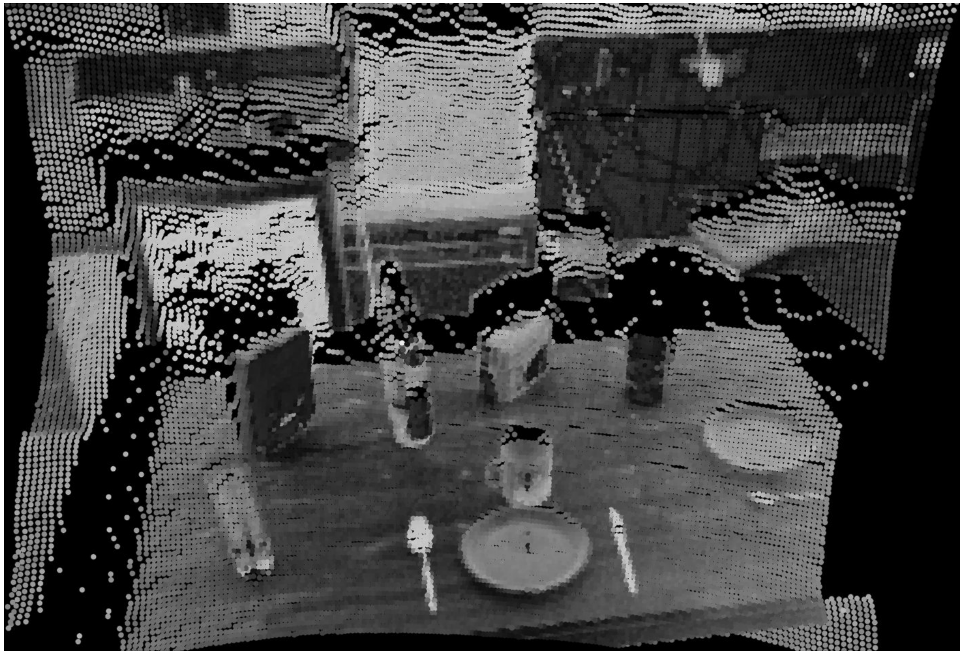
Visualizing Point Clouds

- rviz (http://www.ros.org/wiki/rviz/DisplayTypes/PointCloud)
  - $ rosrun rviz rviz
  - Add a Point Cloud display
  - Set the topic and the TF frames (Fixed/Target)



What are Point Clouds?



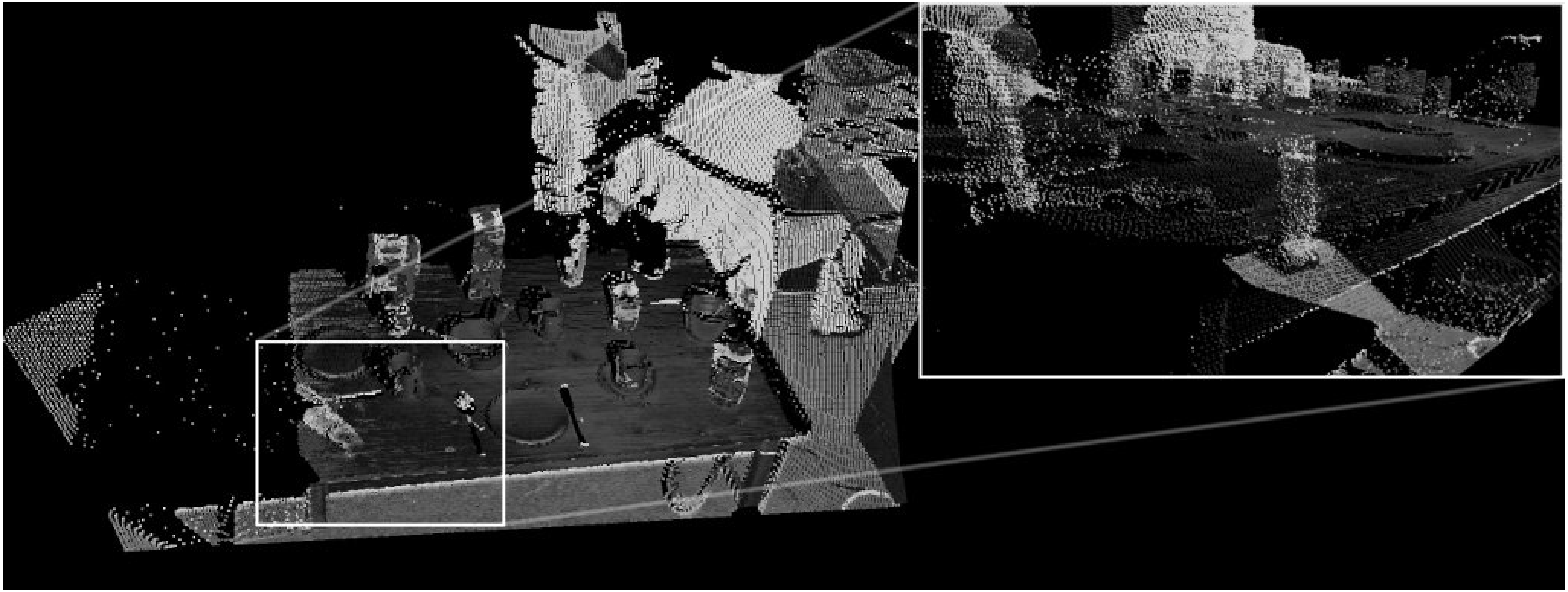


- Point Cloud = a “cloud” (i.e., collection) of nD points (usually n = 3)
- $p = \{x, y, z\} \rightarrow P = \{p_1, p_2, \ldots, p_n\}$
- represent 3D information about the world

What are Point Clouds?

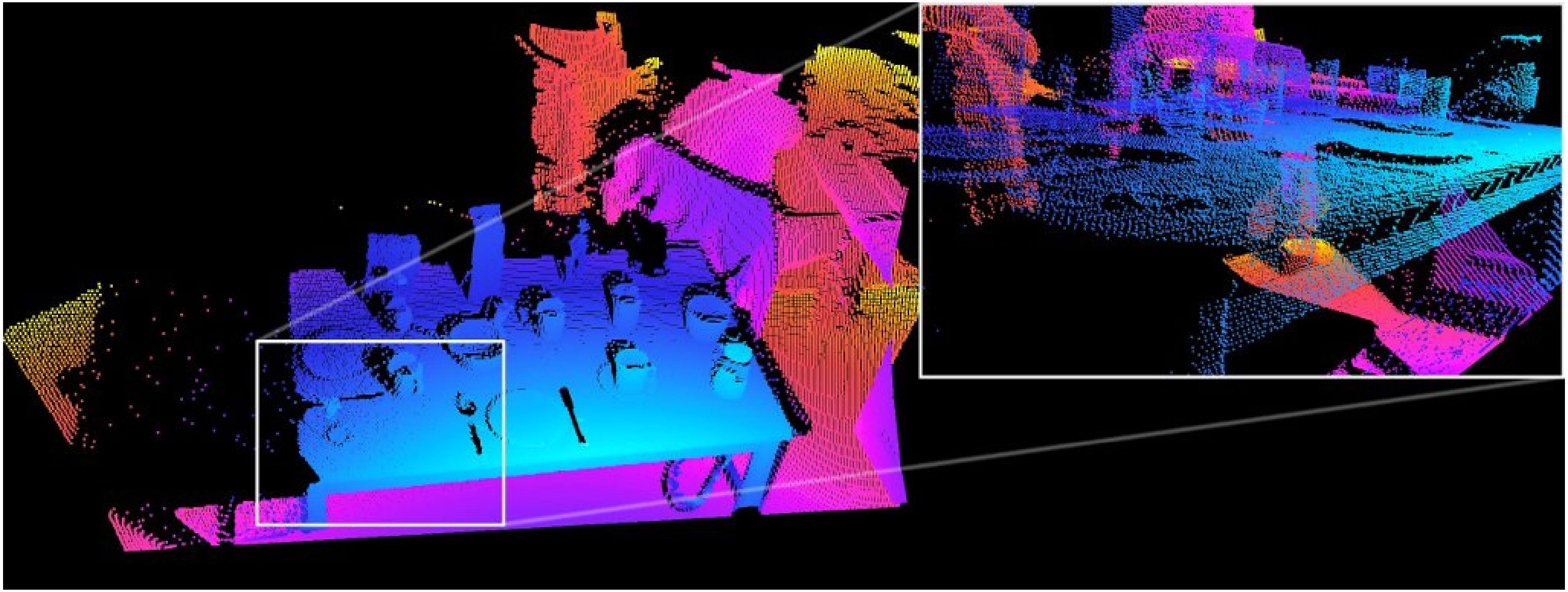
- besides XYZ data, each point ***p*** can hold additional information
- examples include: RGB colors, intensity values, distances, segmentation results, etc

What are Point Clouds?



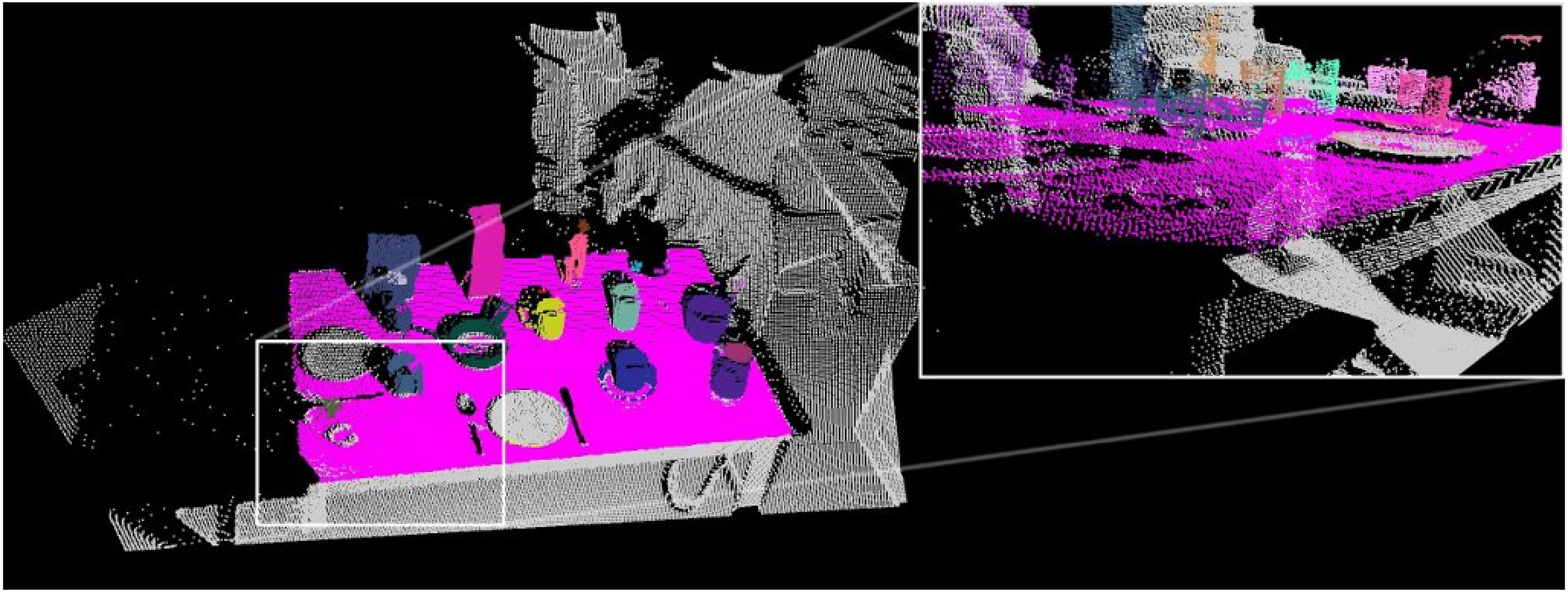
- intensity data

What are Point Clouds?



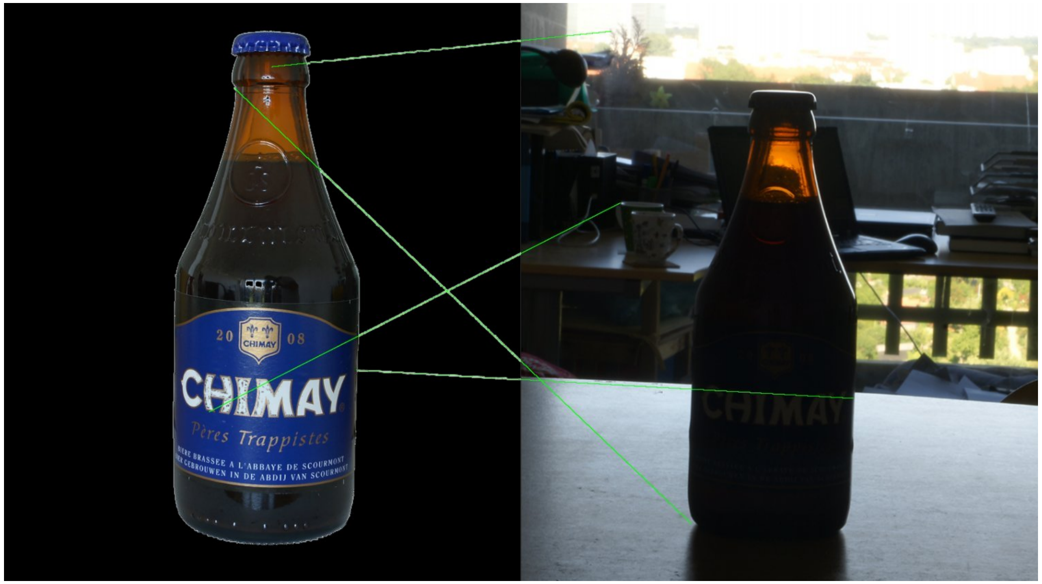
- distance data

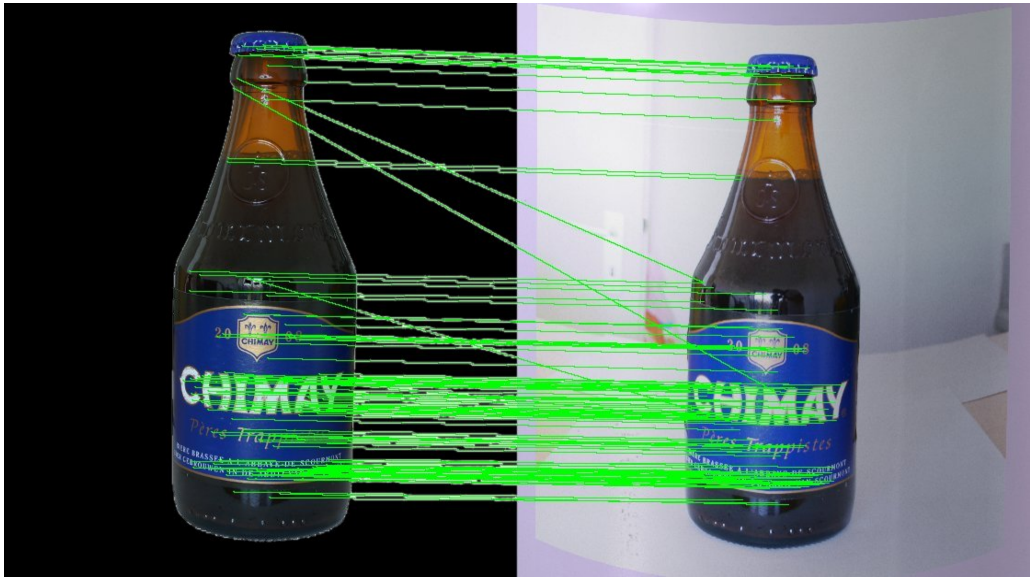
What are Point Clouds?



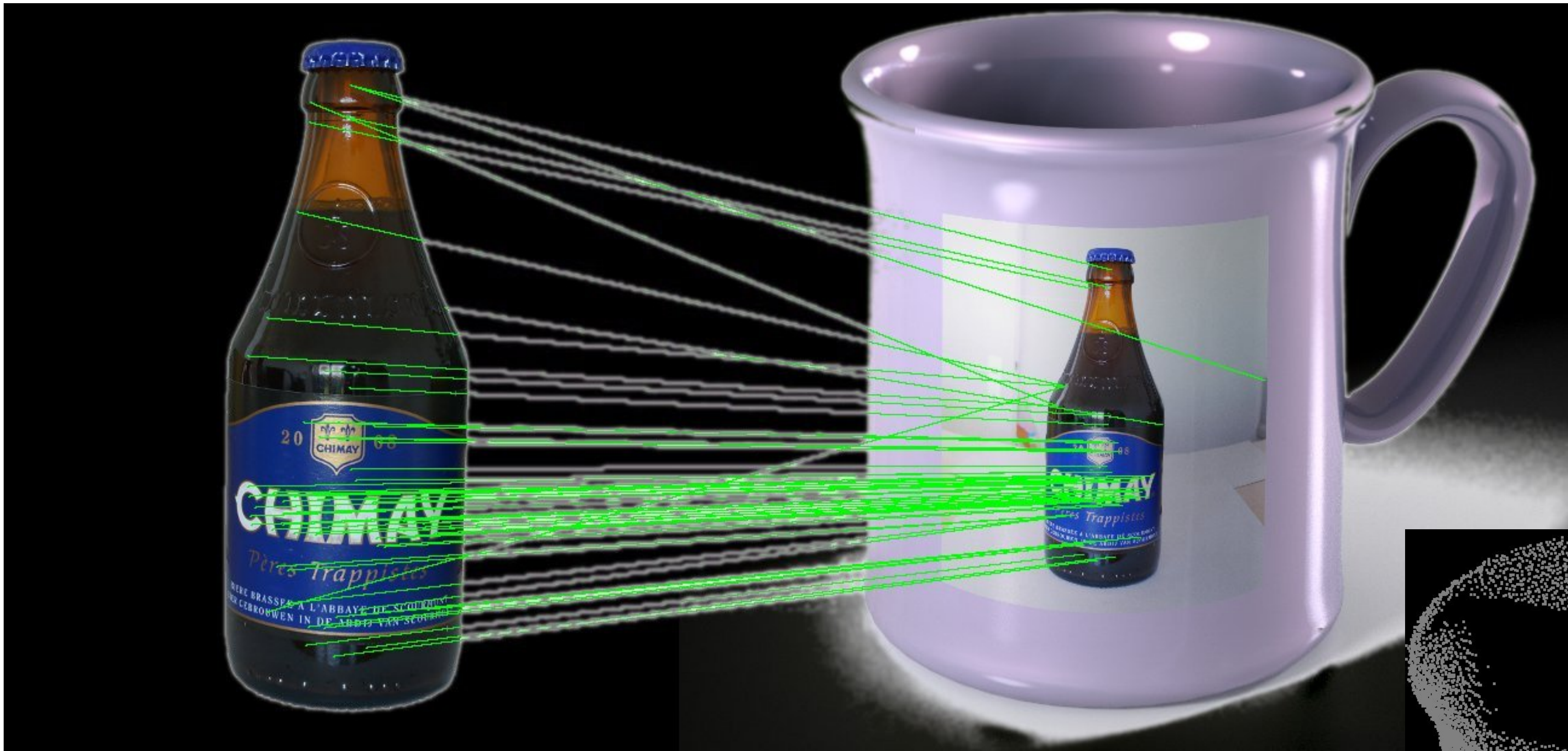
- segmentation data

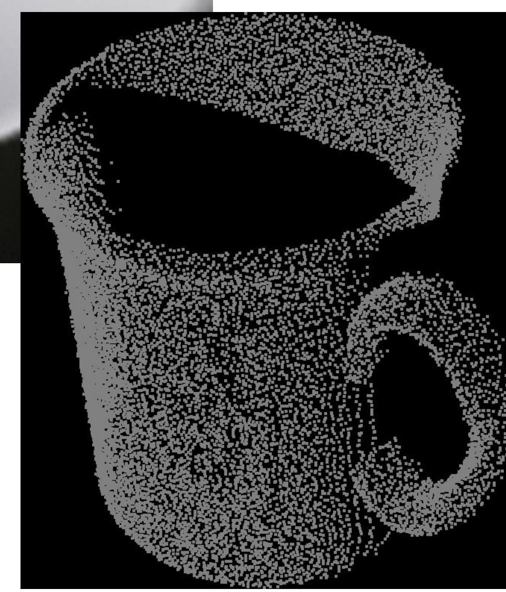
Why are Point Clouds important?





Why are Point Clouds important?





Data representation

- a point ***p*** is a n-tuple, e.g.

  $\boldsymbol{p}_i = \{x_i, y_i, z_i, r_i, g_i, b_i, \ldots\}$

- a Point Cloud ***P*** is represented as a collection of points $\boldsymbol{p}_i$, e.g. $\boldsymbol{P} = \{p_1, p_2, \ldots, p_n\}$
- in terms of data structures, an XYZ point can be represented as: *float32 x, float32 y, float32 z*
- an n-dimensional point is then: *float32[] point*
- therefore a Point Cloud **P** is *Point[] points*

Outline

- Lasers and 3D sensing
- Visualizing Laser Scans

- From LaserScan to PointCloud
- What are Point Clouds?
- Data representation
- Visualizing PointCloud messages

- Preview :: ROS C-Turtle (latest)

Preview :: ROS C-Turtle (latest)

Preview

- PointCloud2 message types: compact, aligned, efficient representations for point clouds
- PCL (Point Cloud Library): a full package for 3D processing

PointCloud**2**

- Point Clouds are big (!)
  - Operation on them are typically slower
  - They are expensive to store (float/double)
- Solutions:
  - Store each dimension data in different (the most appropriate) formats, e.g., *rgb* – 24bits, instead of 3x4 (sizeof float)
  - Group data together, and keep it aligned (SSE) to speed up computations
  - Support organized data – *nD images*

PointCloud**2**

- The ROS PointCloud2 data format:

  Header header

  uint32 height

  uint32 width

  PointField[] fields

  bool is_bigendian

  uint32 point_step

  uint32 row_step

  uint8[] data

  bool is_dense

PointField

- Where PointField is:

  uint8 INT8=1, UINT8=2, INT16=3,
  UINT16=4, INT32=5, UINT32=6,
  FLOAT32=7, FLOAT64=8
  string name
  uint32 offset
  uint8 datatype
  uint32 count

- Examples:

  “x”, 0, 7, 1
  “y”, 4, 7, 1
  “z”, 8, 7, 1
  “rgba”, 12, 6, 1
  “normal_x”, 16, 8, 1
  “normal_y”, 20, 8, 1
  “normal_z”, 24, 8, 1
  “fpfh”, 32, 7, 33

pcl::PointCloud<T>

- Binary blobs are hard(er) to work with
- We provide converters, Publishers / Subscribers, filters, etc, similar to images
- **PointCloud2 → PointCloud**<**T**>
- Examples of T:

```
struct PointXYZ
{
    float x;
    float y;
    float z;
}
```

```
struct Normal
{
  float normal[3];
  float curvature;
}
```

Point Cloud Data (PCD) format

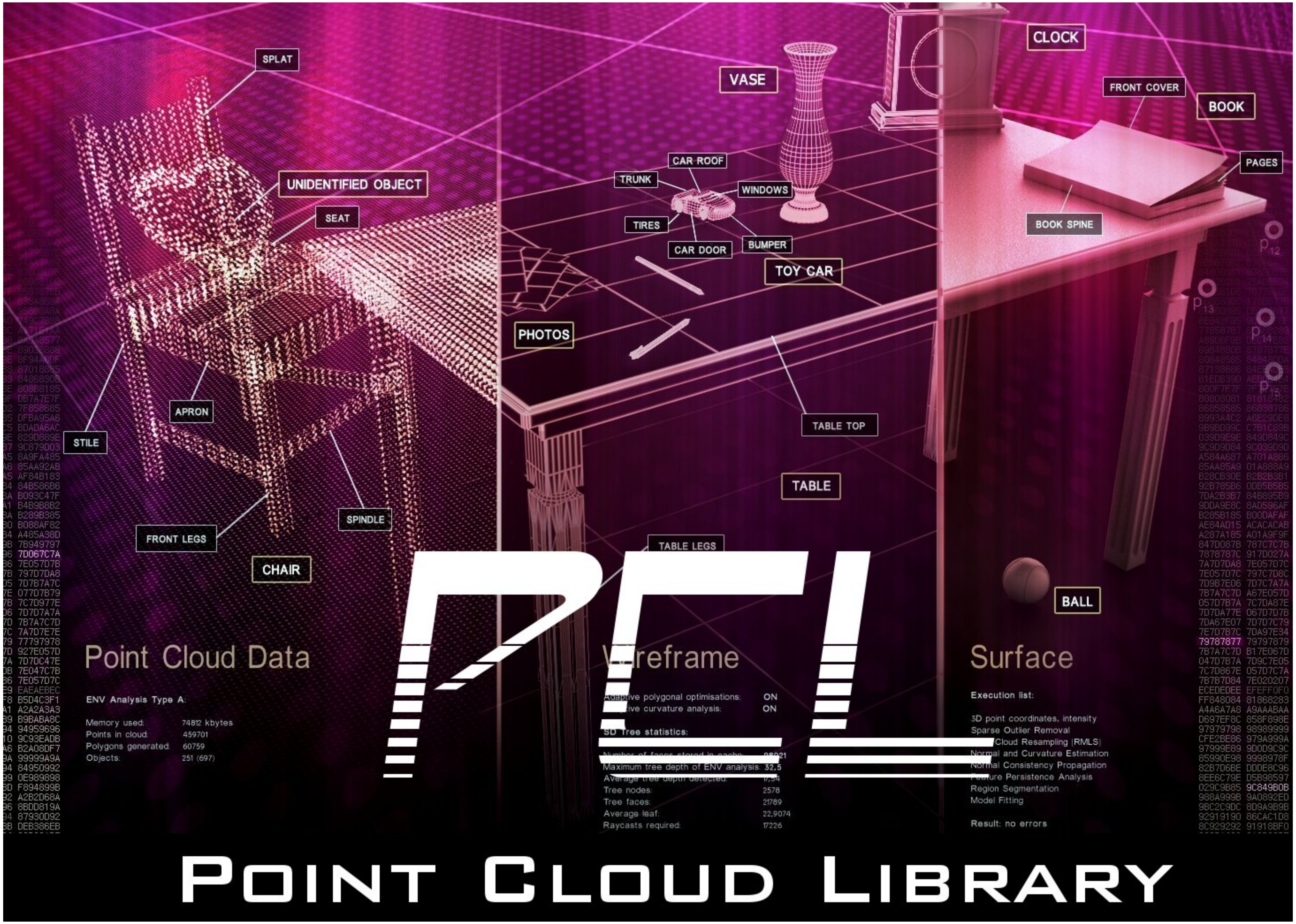
- In addition, point clouds can be stored to disk as files, into the PCD format:

  FIELDS x y z rgba
  SIZE 4 4 4 4
  TYPE F F F U
  WIDTH 307200
  HEIGHT 1
  POINTS 307200
  DATA binary
  ...

- DATA can be either **binary** or **ascii**:

  DATA ascii
  0.0054216 0.11349 0.040749
  -0.0017447 0.11425 0.041273

Point Cloud Library (PCL)



What is P(oint) C(loud) L(ibrary)

- PCL is:
  - fully **templated** modern C++ library for 3D point cloud processing
  - uses **SSE** optimizations (Eigen backend) for fast computations on modern CPUs
  - uses **OpenMP** and Intel **TBB** for parallelization
  - passes data between modules (e.g., algorithms) using **Boost shared pointers**

What is P(oint) C(loud) L(ibrary)

- collection of smaller, modular C++ libraries:
  - **libpcl_features**: many 3D features (e.g., normals and curvatures, boundary points, moment invariants, principal curvatures, Point Feature Histograms (PFH), Fast PFH, ...)
  - **libpcl_surface**: surface reconstruction techniques (e.g., meshing, convex hulls, Moving Least Squares, ...)
  - **libpcl_filters**: point cloud data filters (e.g., downsampling, outlier removal, indices extraction, projections, ...)

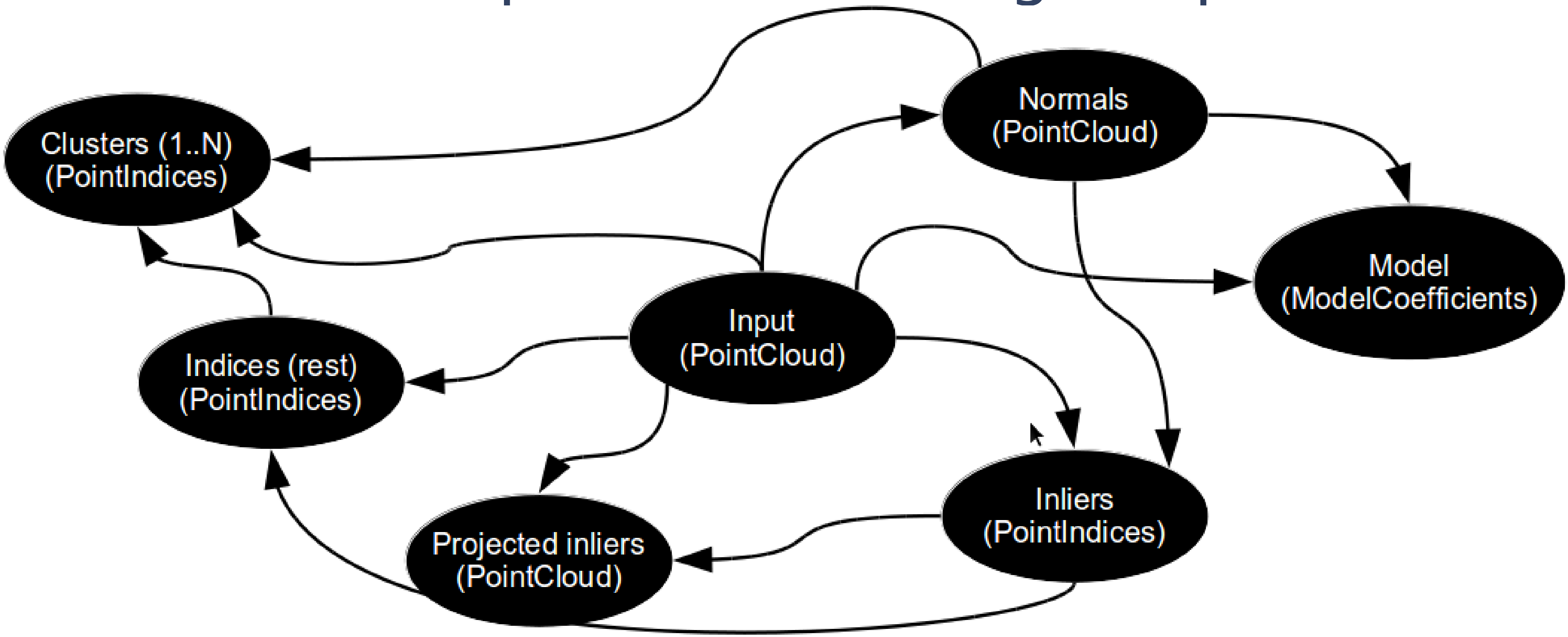
What is P(oint) C(loud) L(ibrary)

- **libpcl_io**: I/O operations (e.g., writing to/reading from PCD (Point Cloud Data) and BAG files)
- **libpcl_segmentation**: segmentation operations (e.g.,cluster extraction, Sample Consensus model fitting, polygonal prism extraction, ...)
- **libpcl_registration**: point cloud registration methods (e.g., Iterative Closest Point (ICP), non linear optimizations, ...)

- unit tests, examples, tutorials
- C++ classes are templated building blocks (**nodelets**!)

PCL Philosophy

- **Philosophy**: *write once, parameterize everywhere*
- **PPG**: **P**erception **P**rocessing **G**raphs

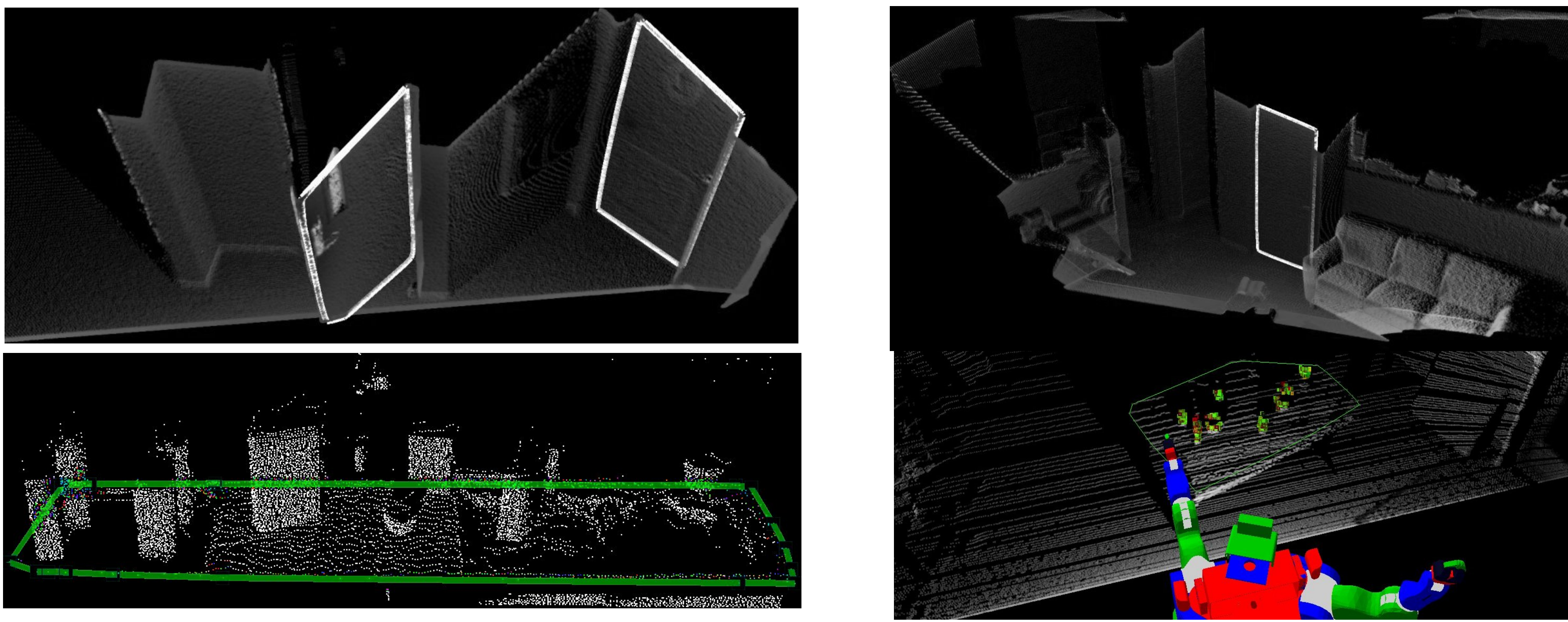


Why PPG? Example

- Algorithmically:

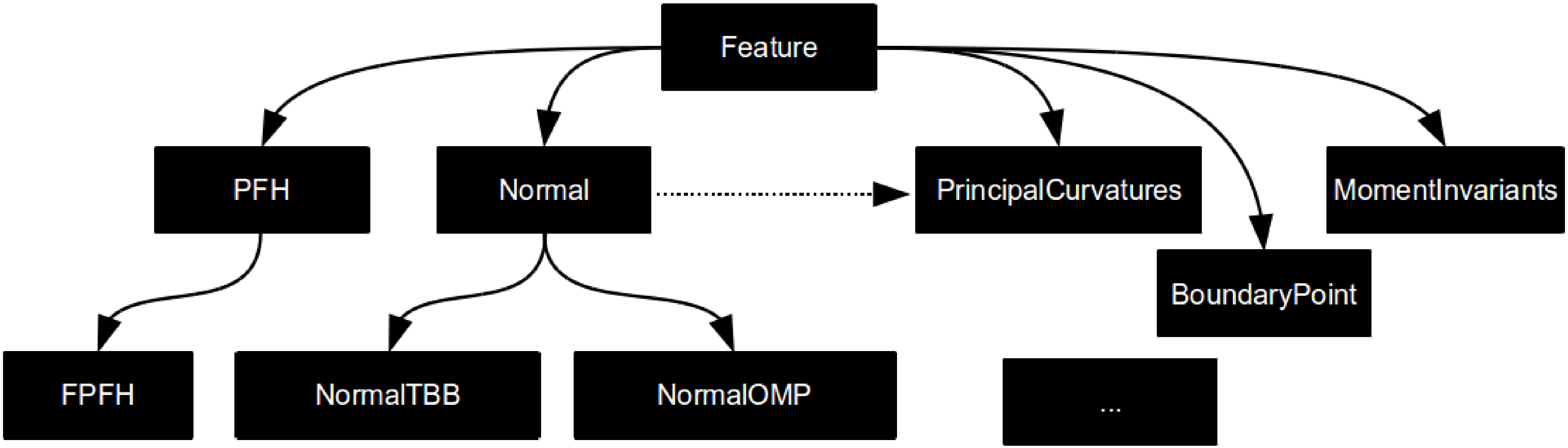
    door = table = wall detection =...

  - the only thing that changes is: parameters (constraints)!



More on PCL Architecture

- Inheritance simplifies development:

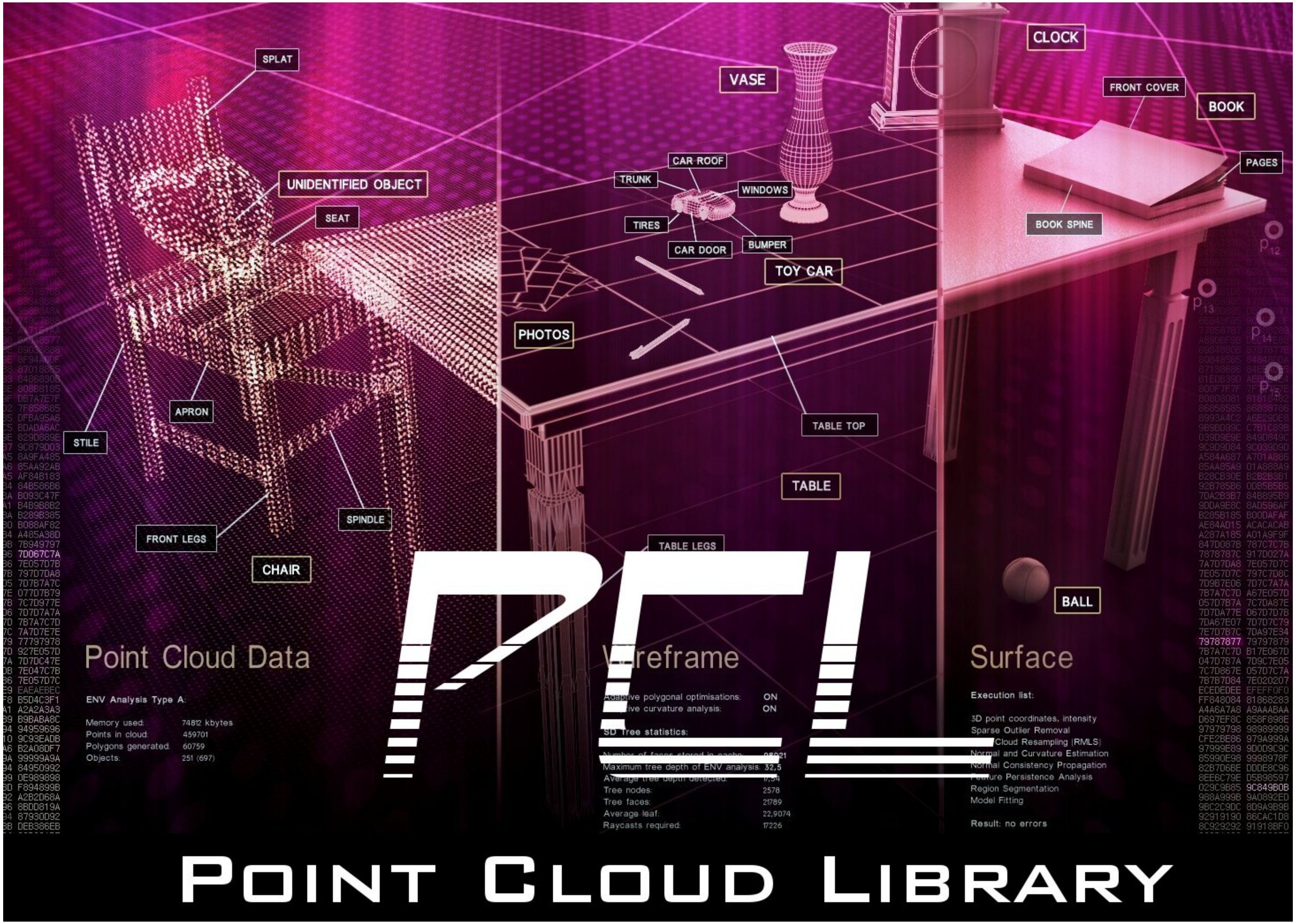



```
pcl :: Feature < PointT > feat ;
feat = pcl :: Normal < PointT > ( input );
feat = pcl :: FPFH < PointT > ( input );
feat = pcl :: BoundaryPoint < PointT > ( input );
feat . compute (& output );
```

PCL Statistics

- Misc, stats:
  - 9 releases so far (*latest: 0.1.8*)
  - over 100 classes
  - over 25k lines of code
  - external dependencies (for now) on eigen, cminpack, ANN, FLANN, TBB
  - internal dependencies (excluding the obvious) on dynamic_reconfigure, message_filters
- tf_pcl package for TF integration

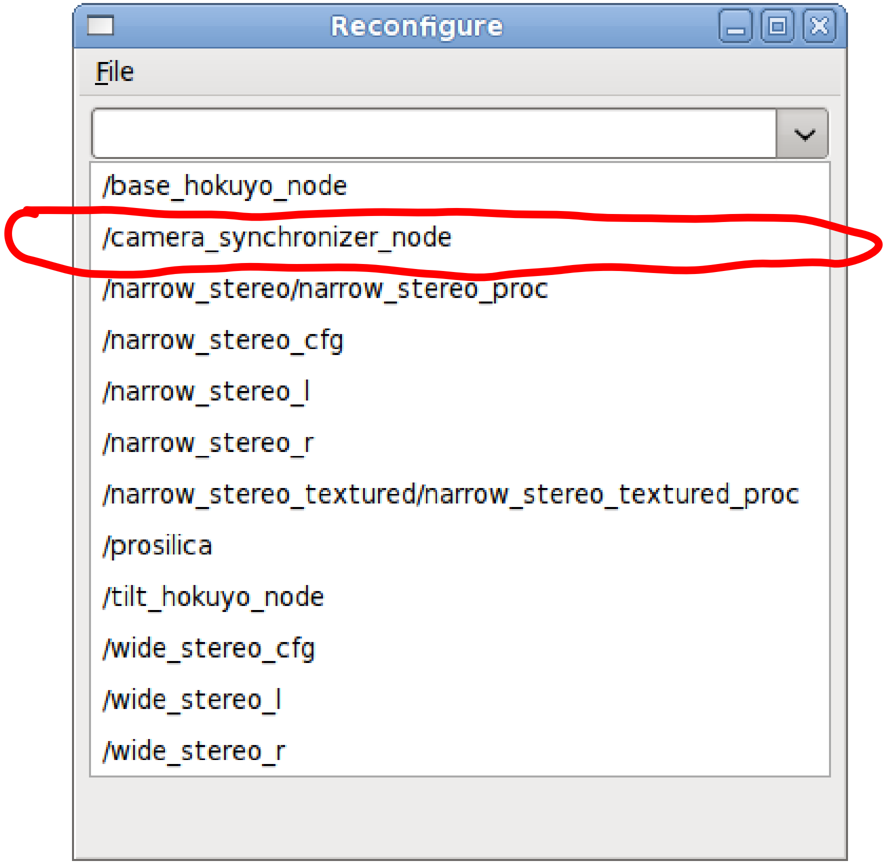
PCL :: Exercise



Re: Using the Texture Projector

$ rosrun dynamic_reconfigure reconfigure_gui

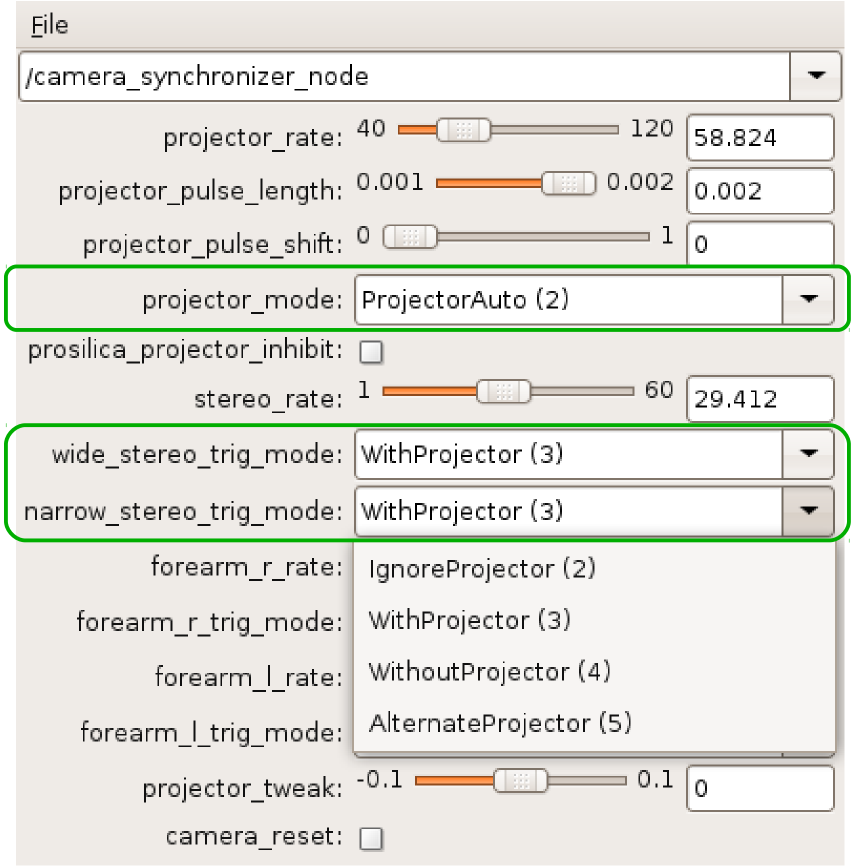
- Turn texture projector on/off



www.ros.org/wiki/dynamic_reconfigure

Re: Using the Texture Projector

- projector_mode – whether projector is turned on
- *_trig_mode – whether the camera syncs with the projector on all, no, or some frames



PCL Tutorials

- http://www.ros.org/wiki/pcl/Tutorials
- Downsampling data (*VoxelGrid*)
- Planar model segmentation (*SACSegmentation*)
- Exercise: build a node that segments a table in front of the robot
- **PROBLEM**: Not all tools support PointCloud2 yet (!)
- **SOLUTION**:
  - $ **rosrun point_cloud_converter point_cloud_converter points2_in:=MYTOPIC**

PCL VoxelGrid - downsampling

```
#include <pcl/filters/voxel_grid.h>
#include <pcl/point_types.h>
int main (int argc, char **argv) {
  ros::init (argc, argv, "pcl_demo");
  ros::NodeHandle nh;
  point_cloud::Publisher<pcl::PointXYZ> pub_downsampled;
  pub_downsampled.advertise (nh, "downsampled", 1);

  pcl::PointCloud<pcl::PointXYZ> cloud, cloud_downsampled;
  pcl::VoxelGrid<pcl::PointXYZ> grid;
  grid.setFilterFieldName ("z");
  grid.setLeafSize (0.01, 0.01, 0.01);
  grid.setFilterLimits (0.4, 1.6);
  while (nh.ok ()) {
    sensor_msgs::PointCloud2ConstPtr cloud2_blob_ptr = ros::topic::waitForMessage<sensor_msgs::PointCloud2>("/narrow_stereo_textured/points2");
    point_cloud::fromMsg (*cloud2_blob_ptr, cloud);
    grid.setInputCloud (boost::make_shared<pcl::PointCloud<pcl::PointXYZ> > (cloud));
    grid.filter (cloud_downsampled);
    pub_downsampled.publish (cloud_downsampled);
  }
}
```

PCL Segmentation - planar

```
#include <pcl/filters/project_inliers.h>
#include <pcl/segmentation/sac_segmentation.h>
#include <pcl/PointIndices.h>
#include <pcl/ModelCoefficients.h>
…
point_cloud::Publisher<pcl::PointXYZ> pub_plane;
pub_plane.advertise (nh, "plane", 1);
…
pcl::PointIndices plane_inliers;
pcl::ModelCoefficients plane_coefficients;
…
pcl::SACSegmentation<pcl::PointXYZ> seg;
seg.setDistanceThreshold (0.05);
seg.setMaxIterations (1000);
seg.setModelType (pcl::SACMODEL_PLANE);
seg.setMethodType (pcl::SAC_RANSAC);

pcl::ProjectInliers<pcl::PointXYZ> proj;
proj.setModelType (pcl::SACMODEL_PLANE);
…
seg.setInputCloud (boost::make_shared<pcl::PointCloud<pcl::PointXYZ> > (cloud_downsampled));
seg.segment (plane_inliers, plane_coefficients);

proj.setInputCloud (boost::make_shared<pcl::PointCloud<pcl::PointXYZ> > (cloud_downsampled));
proj.setIndices (boost::make_shared<pcl::PointIndices> (plane_inliers));
proj.setModelCoefficients (boost::make_shared<pcl::ModelCoefficients> (plane_coefficients));
…
pub_plane.publish (cloud_plane);
```

If time allows: PCL **nodelets**!

- Goal:
  - *write once, parameterize everywhere* ⇒ **modular code**
  - ideally, each algorithm is a "*building block*" that consumes input(s) and produces some output(s)
  - in ROS, this is what we call a ***node***. inter-process data passing however is inefficient. ideally we need shared memory.
- Solution:
  - ***nodelets*** = "nodes in nodes" = single-process, multi-threading

If time allows: PCL **nodelets**!

- Nodelets:
  - same ROS API as nodes (subscribe, advertise, publish)
  - dynamically (un)loadable
  - optimizations for zero-copy Boost shared_ptr passing
  - PCL nodelets use dynamic_reconfigure for on-the-fly parameter setting

PCL VoxelGrid nodelet example

```
<launch>
  <node pkg="nodelet" type="nodelet" name="foo"
  args="load pcl/VoxelGrid pcl_manager">
    <remap from="/foo/input"
    to="/narrow_stereo_textured/points"/>
    <rosparam>
      leaf_size: [0.015 , 0.015 , 0.015]
      filter_field_name: "z"
      filter_limit_min: 0.8
      filter_limit_max: 5.0
    </rosparam>
  </node>
</launch>
```

Questions?

http://www.ros.org/

http://www.ros.org/wiki/pcl

ros-users@code.ros.org