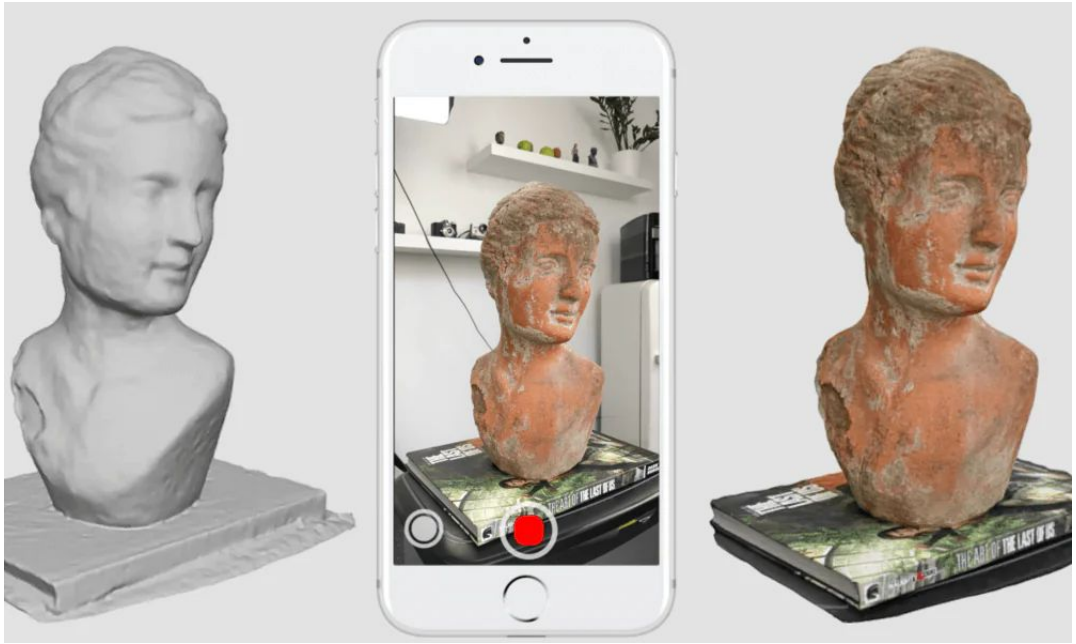
A Software Pipeline for Visual Object Mesh Creation

Author: Evtim Kostadinov
Supervisor: Prof. Dr.-Ing. Darius Burschka
Advisor: Andrei Costinescu

Motivation

Obtain geometry of real-time objects



Basics Overview

- Header files files are located under the namespace ScannerLib
- Everything necessary is packed inside the Scanner class
- It can be constructed only using a json file
  (if empty -> default settings used)

```
#include <ScannerLib/Scanner.h>

int main()
{
        Scanner sc("SomeJson.json");
}
```

Basics Overview

- The library uses a third party library called Point Cloud Library (PCL).
  It provides a convenient way to handle point clouds.
- Each point in the pointcloud can be either stored as pcl::PointXYZ for a point storing only the position coordinates or as pcl::PointXYZRGB for position + color.
- When ScannerLib is used we can omit pcl:: for those data types.

```
#include <ScannerLib/Scanner.h>

int main()
{
	//point containing XYZ coordinates
	//same as using pcl::PointXYZ
	PointXYZ p1(0.f,0.f,0.f);

	//point containing XYZ coordinates + RGB values
	//same as using pcl::PointXYZRGB
	PointXYZRGB p2(0.f,0.f,0.f,255,255,255);
}
```

Basics Overview

- Point clouds are stored in pcl::PointCloud<PointT> container object
- However this object is mostly used through a pointer using pcl::PointCloud<PointT>::Ptr
- When using ScannerLib the following abbreviations are provided:

  PointCloudXYZ -> pcl::PointCloudXYZ<PointXYZ>

  PointCloudXYZRGB -> pcl::PointCloudXYZ<PointXYZRGB>

  PointCloudXYZPtr -> pcl::PointCloudXYZ<PointXYZ>::Ptr

  PointCloudXYZRGBPtr -> pcl::PointCloudXYZ<PointXYZRGB>::Ptr

```
#include <ScannerLib/Scanner.h>

int main()
{
	//same as using pcl::PointCloud<PointXYZ>
	//empty pointcloud
	PointCloudXYZ cloud;

	//to access points at index 0 stored in std::vector
	cloud.points[0];

	//same as using pcl::PointCloud<PointXYZ>::Ptr
	//empty pointcloud pointer
	PointCloudXYZPtr cloud (new PointCloudXYZ) ;

	//to access points at index 0
	cloud->points[0];
}
```

Basics Overview

- We can catch a single frame using the member functions:

  PointCloudXYZPtr capture_FrameXYZ(bool);

  PointCloudXYZRGBPtr capture_FrameXYZRGB(bool);

- And also multiple frames with:

  std::vector<PointCloudXYZPtr> capture_FramesXYZ(bool,int count);

  std::vector<PointCloudXYZRGBPtr> capture_FramesXYZRGB(bool,int count);

- Works with Intel Realsense cameras.

Basics Overview

```
#include <ScannerLib/Scanner.h>

int main()
{
	Scanner sc(“SomeJson.json”);

	//true show it in the viewer or false capture and do not show
	//capture single frame
	PointCloudXYZPtr one_frame = sc.capture_FrameXYZ(true);

	//capture multiple frames (5 frames here)
	std::vector<PointCloudXYZPtr> one_frame = sc.capture_FramesXYZ(true,5);


}
```

Basics Overview

- We can view a single point cloud or multiple point clouds using the view function of the Scanner class:

```
#include <ScannerLib/Scanner.h>

int main()
{
        Scanner sc("SomeJson.json");
        PointCloudXYZPtr cloud (new PointCloudXYZ) ;
        ...
        sc.view(cloud);
}
```

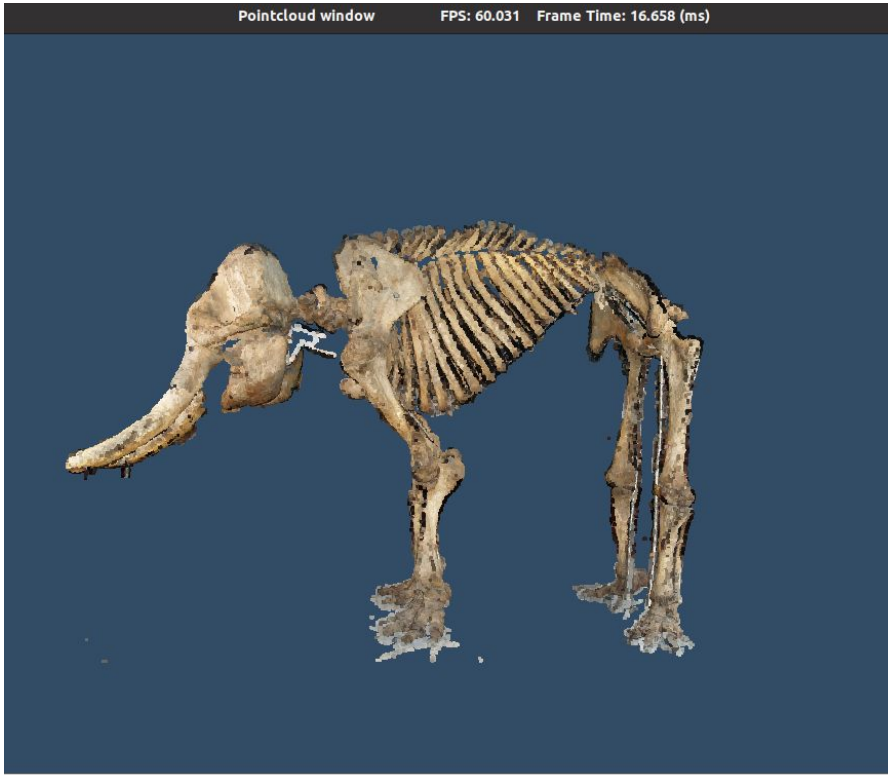
- Switching between frames when viewing multiple point clouds can be done by pressing the key “N”

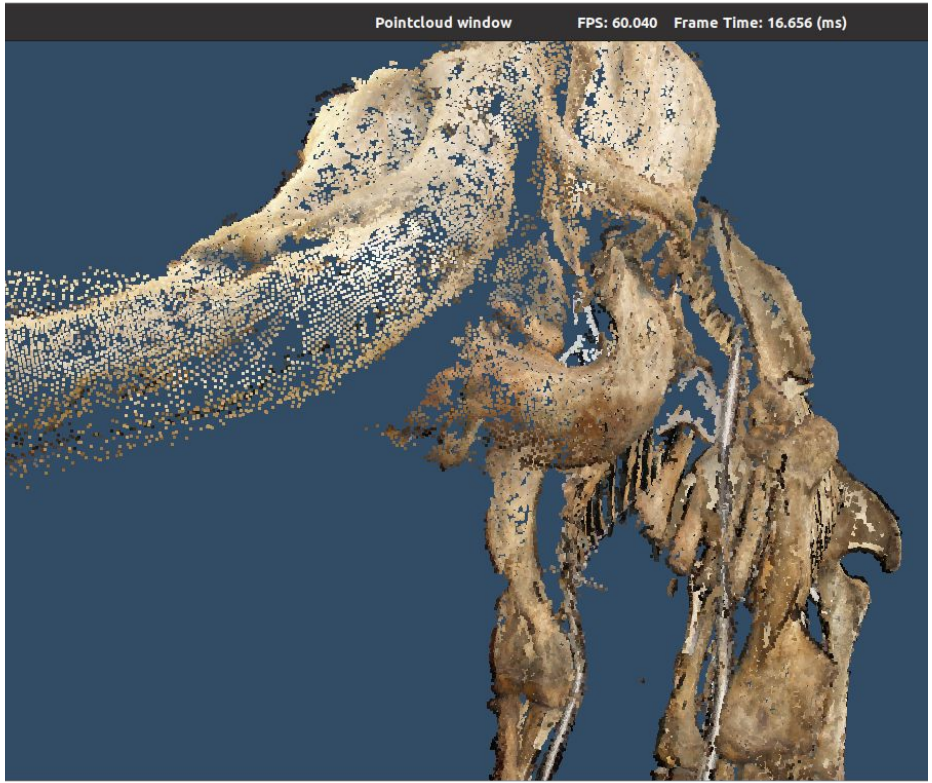
Basics Overview

- Save and load functions are provided supporting “.pcd”, “.ply” and “.obj” file formats.

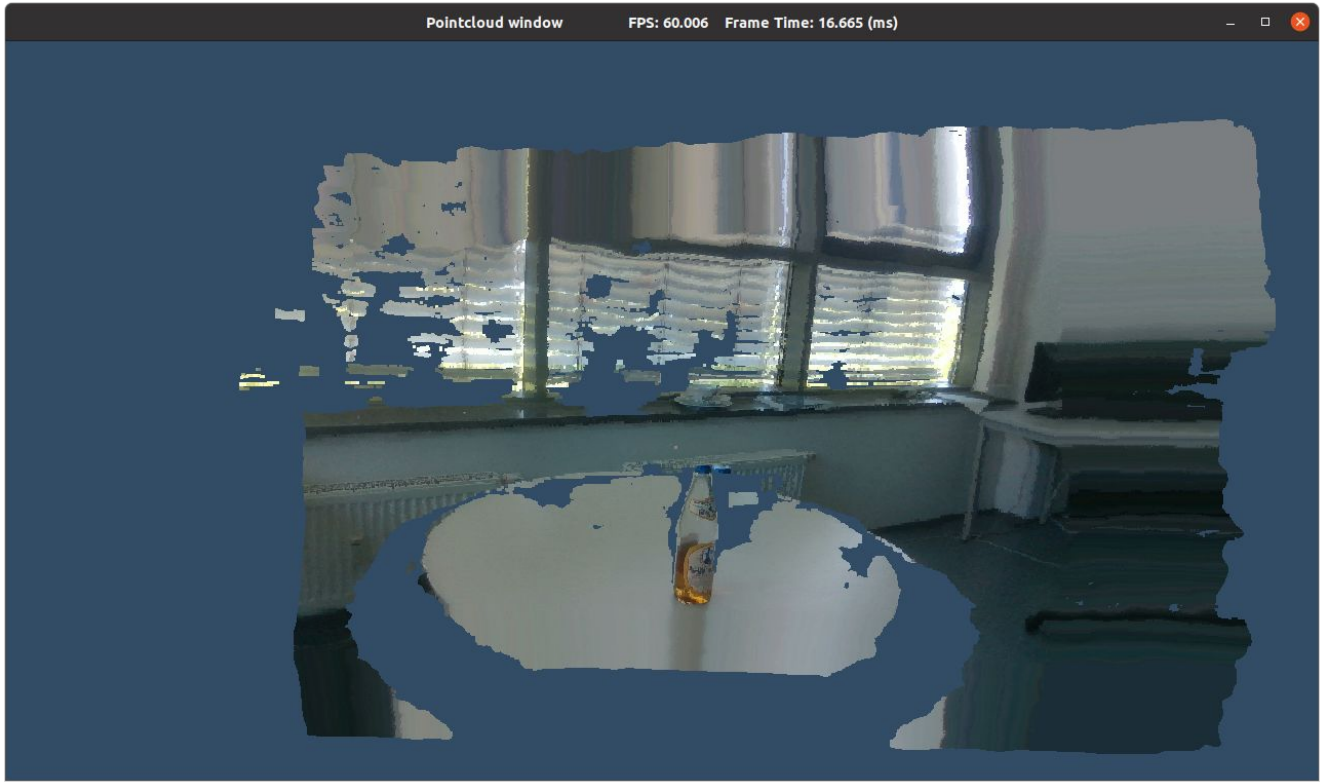
```
#include <ScannerLib/Scanner.h>

int main()
{
        Scanner sc(“SomeJson.json”);
        PointCloudXYZPtr cloud (new PointCloudXYZ) ;

        sc.load_pcd<PointXYZ>(“data.pcd”,cloud);

        sc.view(cloud);
        …
        sc.save_pcd<PointXYZ>(“data_new.pcd”,cloud);
}
```

Visualizer


Pointcloud window
FPS: 60.031
Frame Time: 16.658 (ms)


Pointcloud window
FPS: 60.040
Frame Time: 16.656 (ms)

Visualizer



Methods

Plane Segmentation

Iterative Closest Point (ICP) algorithm

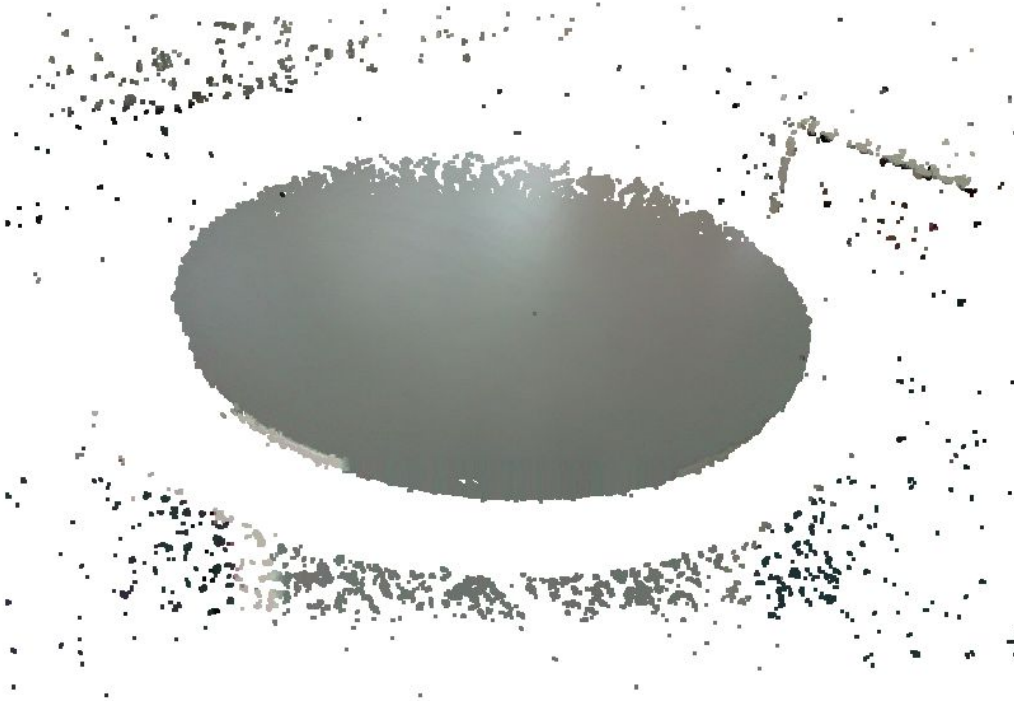
Object Surface Reconstruction

Plane Segmentation

- Object must be placed on a flat surface (e.g table)
- Scans must not contain other objects above the plane
- Problem: Extract object from the surface and remove noise data

Plane Segmentation

- A separate scan from the same distance of the plane only is required for the for the plane extraction method



Plane Segmentation

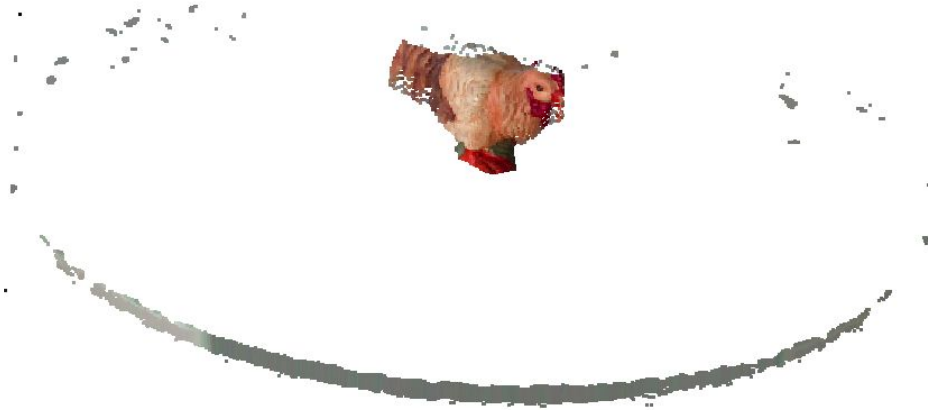
- Apply a preprocessing step with configured RANSAC-based method from PCL

Plane Segmentation

- Apply Singular Value Decomposition to find surface normal ***n***
- Calculate for each point of the given point cloud calculate using the mean point of the plane :

D = (Point - plane_mean) X ***(n)***

If D <= 0 we remove the point.



Plane Segmentation

- Now we apply again the preprocessing function from PCL to clear outliers

Iterative Closest Point (ICP)

- The ICP is used to align two separate point clouds and merge them into one
- Given two sets of points:

$$X = x_1, ..., x_{N_x}$$
$$Y = y_1, ..., y_{N_y}$$

- Goal: find rotation R and translation t to minimize the error:

$$E(R,t) = \frac{1}{N_x} \sum_{i=0}^{N_x} ||y_i - R * x_i - t||^2$$

Iterative Closest Point (ICP)

- Find means of both point clouds:

$$\mu_x = \frac{1}{N_x}\sum_{i=1}^{N_x} x_i \qquad \mu_y = \frac{1}{N_y}\sum_{i=1}^{N_y} y_i$$

- Calculate matrix *H*:

$$H = \sum_{i=1}^{N_x} (y' - \mu_y)(x_i - \mu_x)^T$$

Iterative Closest Point (ICP)

- Apply Singular-Value Decomposition on *H*:

$$svd(H) = UDV^T$$

- Rotation *R*:

$$R = UV^T$$

- Translation *t*:

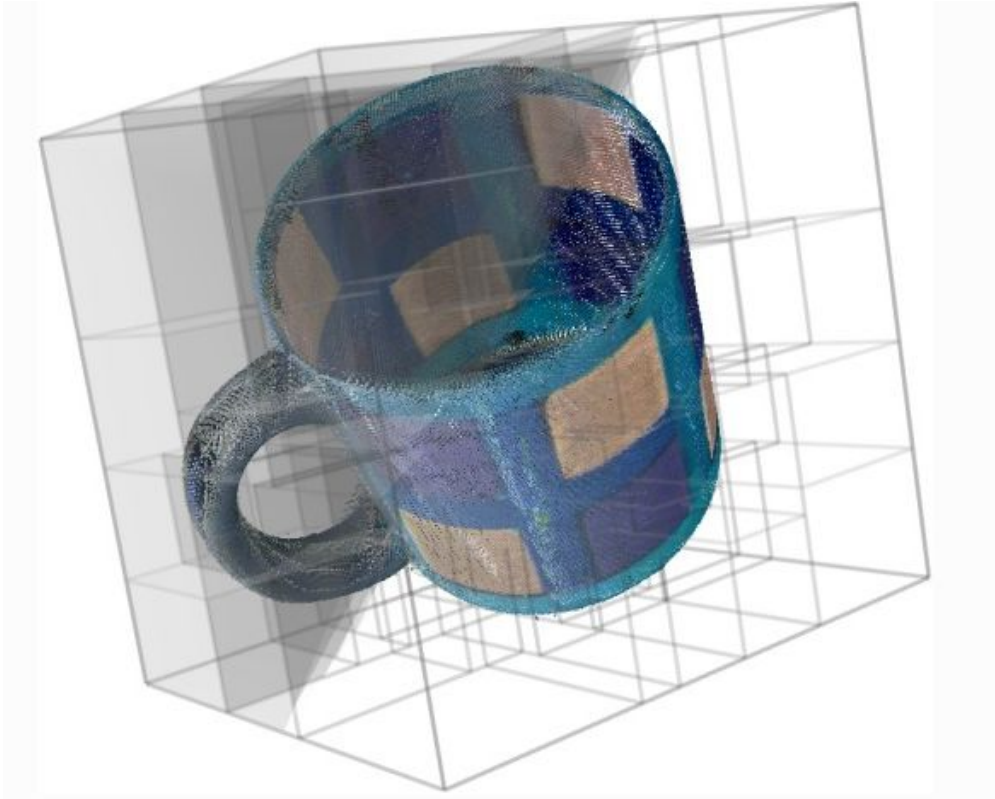
$$t = \mu_y - R\mu_x$$

Iterative Closest Point (ICP)

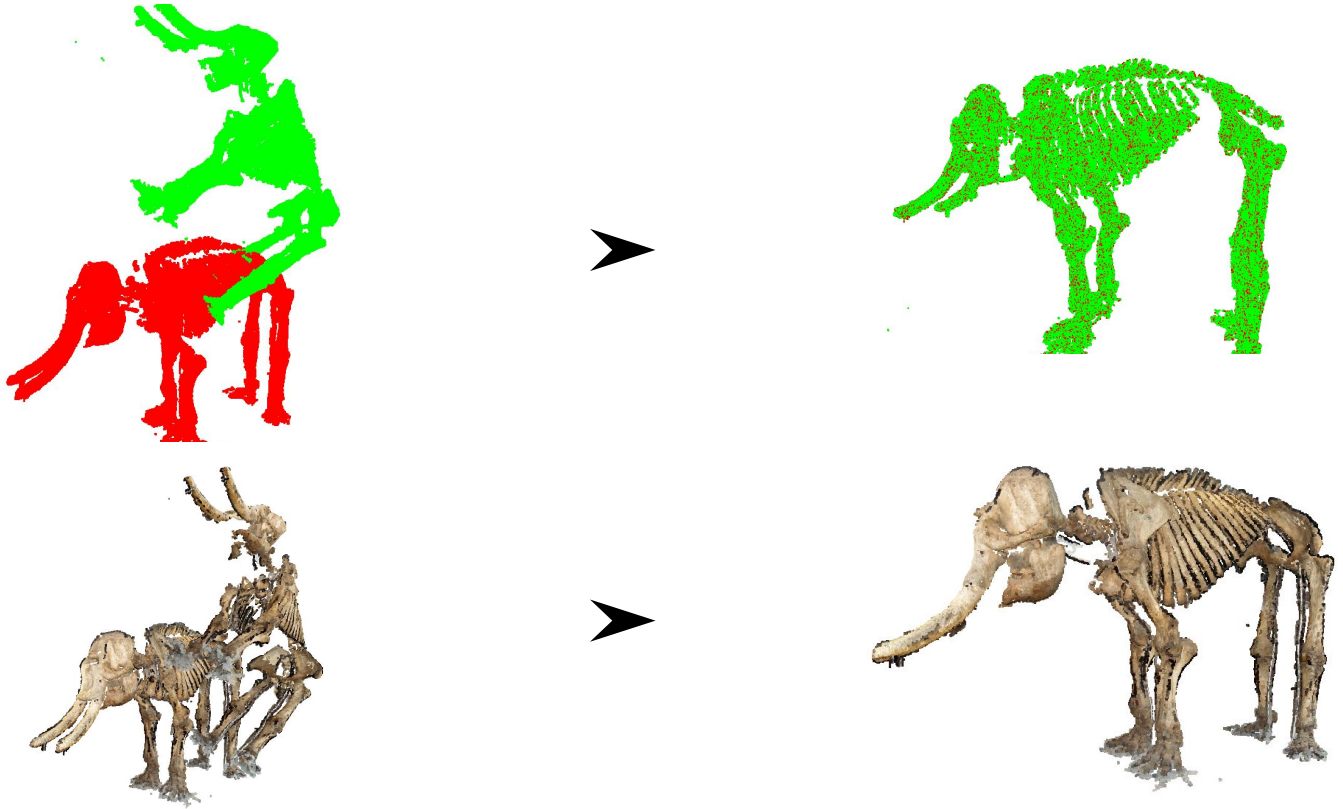
How to find point correspondences?

- Find the closest point using a space-partitioning data structure: K-Dimensional tree (K-d tree)

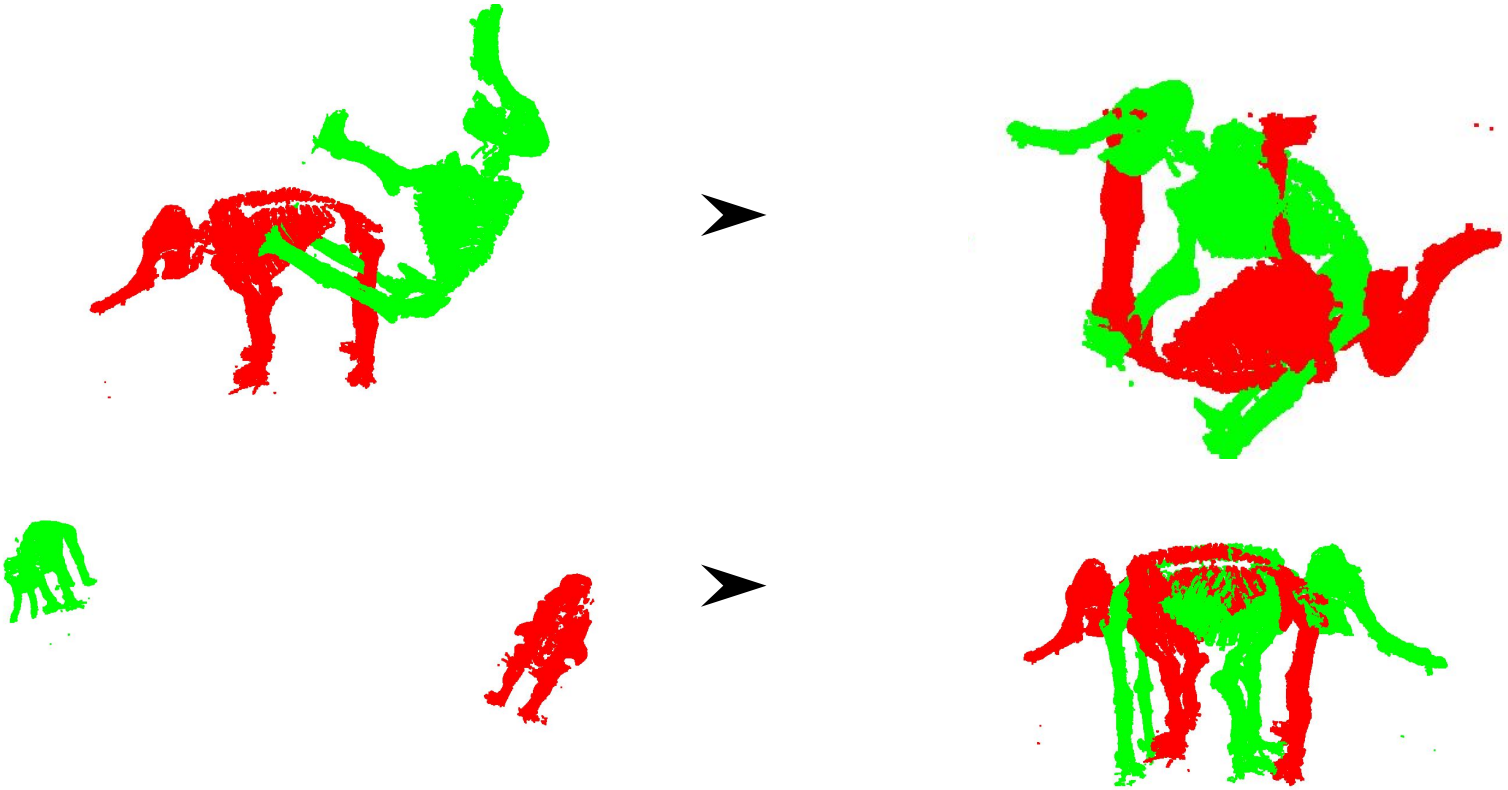
- In this case K = 3 since X, Y, Z dimensions are used for partitioning



Iterative Closest Point (ICP)



Iterative Closest Point (ICP)



Object surface reconstruction

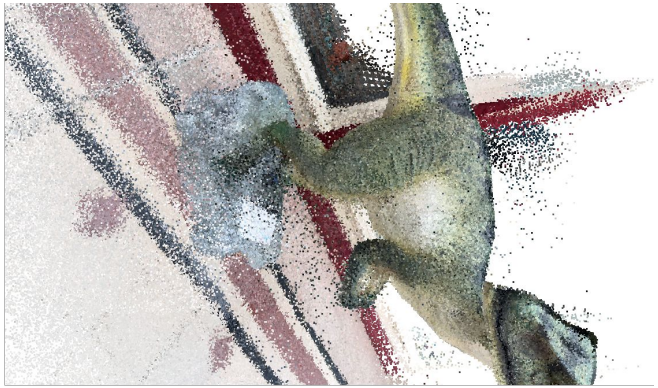
- Two kinds of surface reconstruction algorithms -> one explicit and one implicit method (using Point Cloud Library)

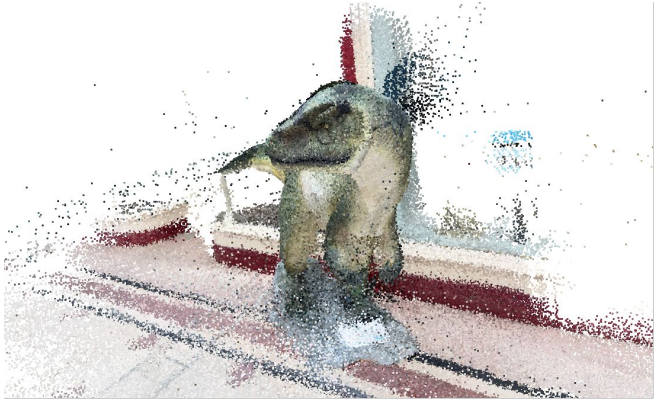
- Explicit -> Greedy Triangulation Reconstruction

- Implicit -> Poisson Surface Reconstruction

Object surface reconstruction

- Explicit -> Greedy Triangulation Reconstruction

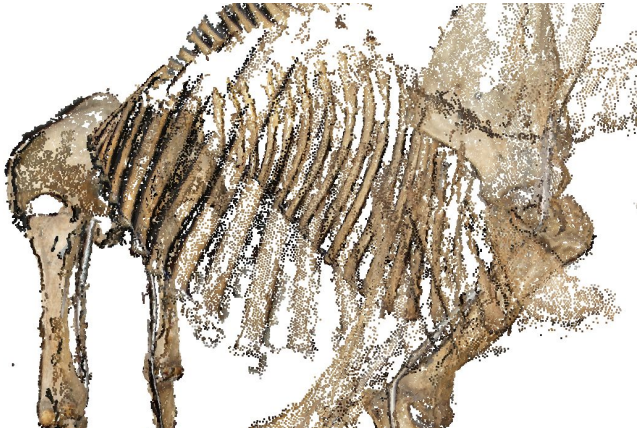




Object surface reconstruction

- Explicit -> Greedy Triangulation Reconstruction
- Using normal estimation (PCL)

Object surface reconstruction



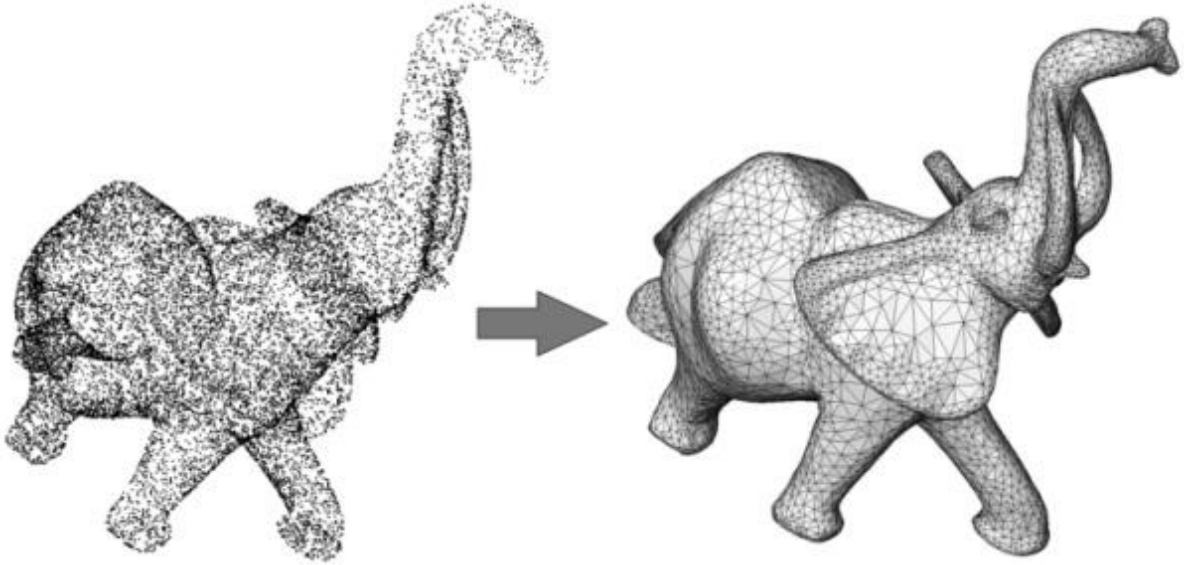
Object surface reconstruction

- Explicit -> Greedy Triangulation Reconstruction
- Using Moving Least Squares (MLS) normal estimation (PCL)

Object surface reconstruction

Object surface reconstruction

- Implicit -> Poisson Surface Reconstruction



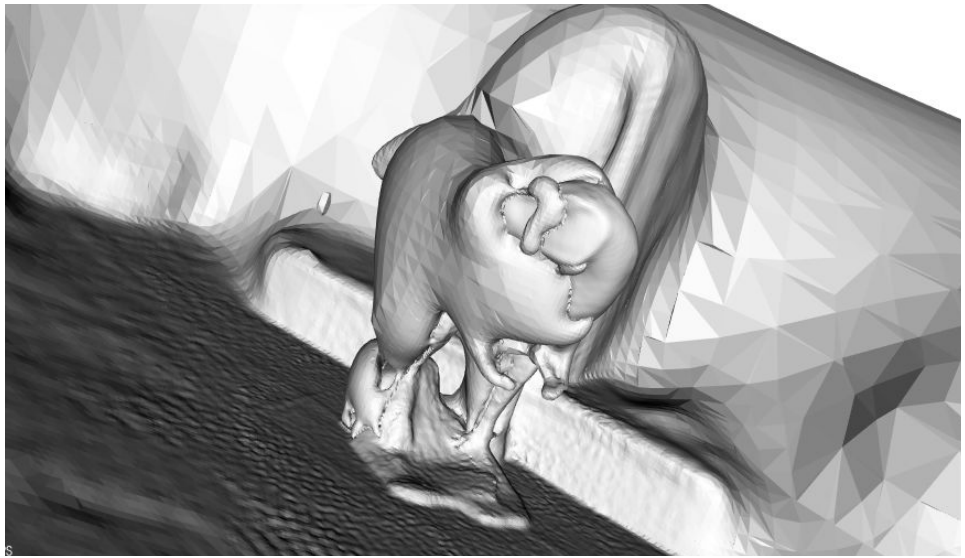
Object surface reconstruction

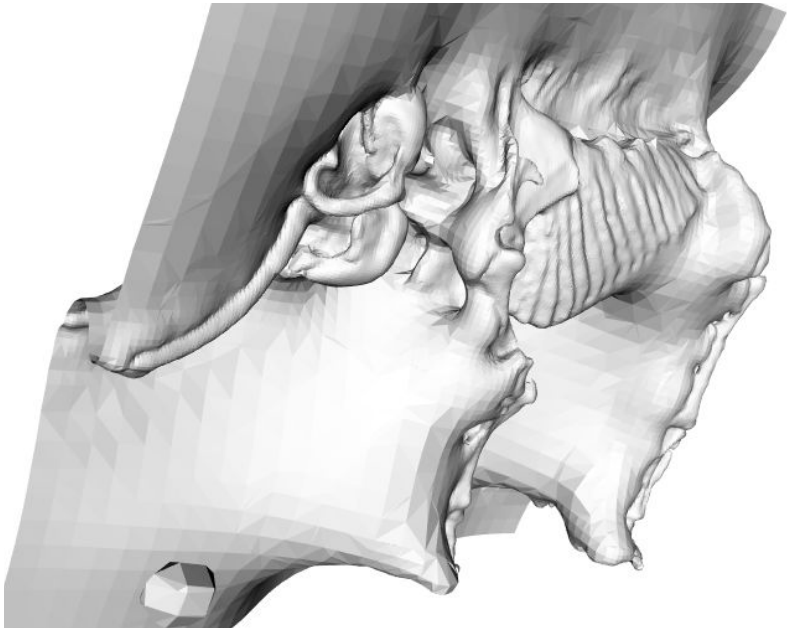
- Implicit -> Poisson Surface Reconstruction

- Neither using normal estimation nor Moving Least Squares (MLS) normal estimation worked properly (PCL)

Object surface reconstruction

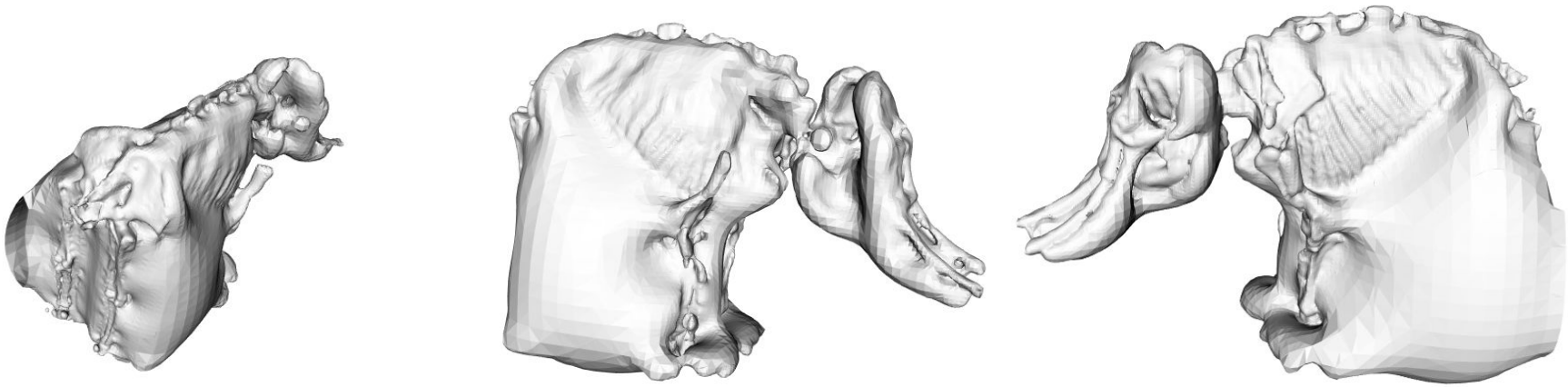
- Moving Least Squares (MLS) normal estimation





Object surface reconstruction

- normal estimation



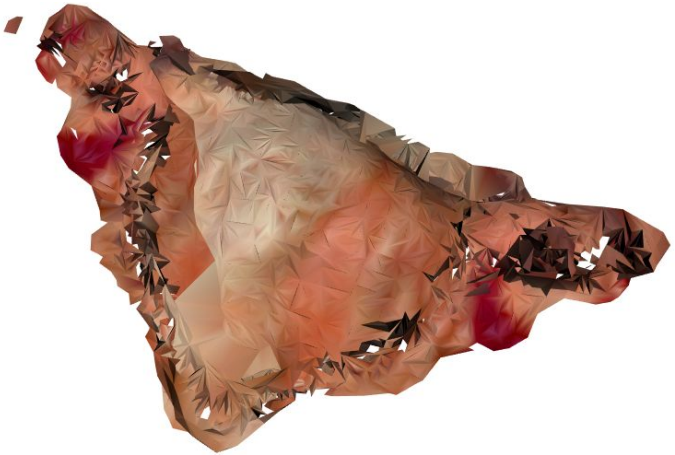
Final Results

Results

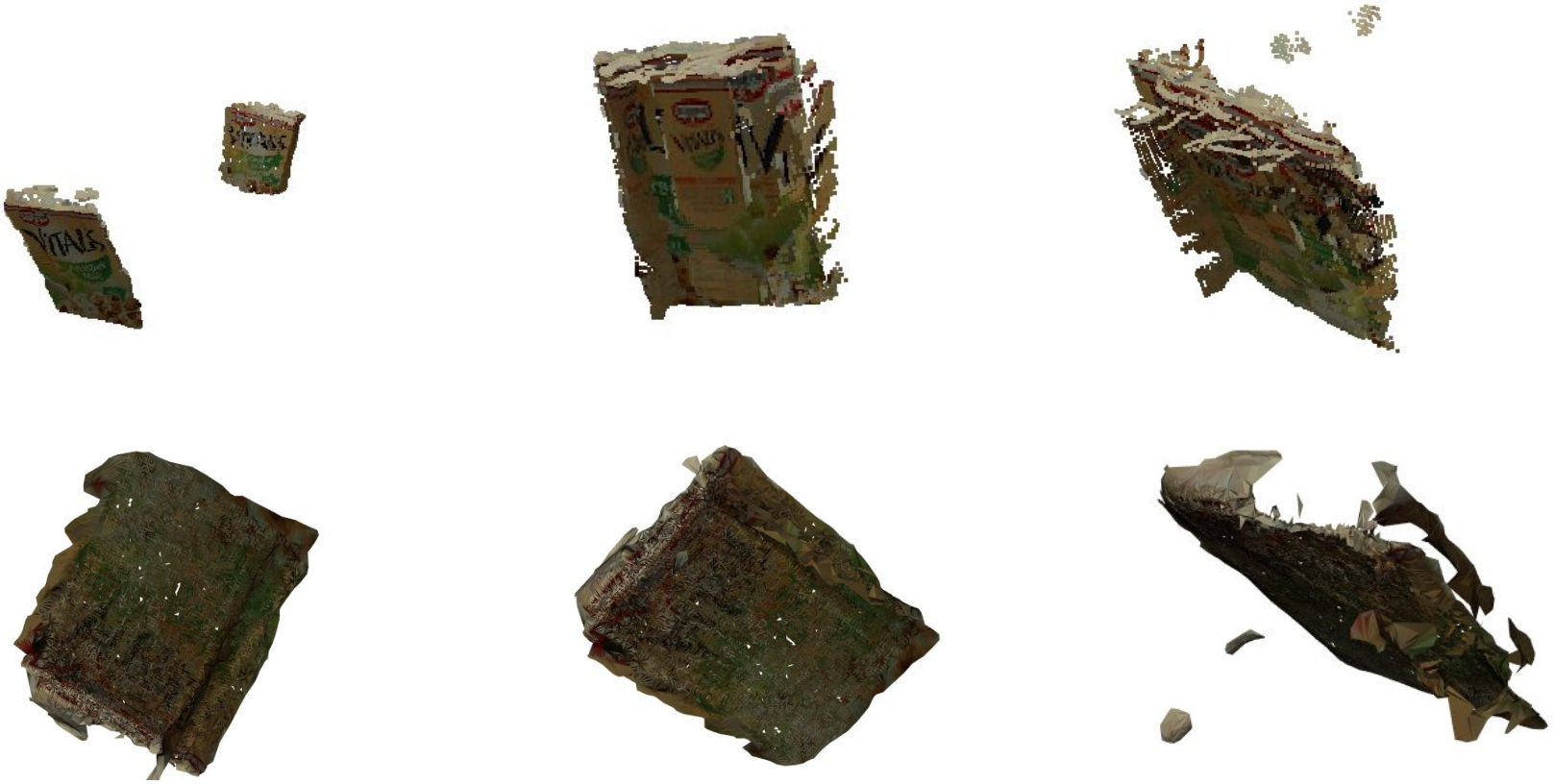
- Unexpected: surprisingly the implemented ICP often “overaligns” the point clouds causing them overlap over their mean points and lose significant geometry details

Results

- Due to the lost geometry information the reconstruction (Greedy Triangulation with MLS) fails to correctly identify the correct triangles that should be constructed



Results



Conclusion

- The proposed plane segmentation algorithm, although intuitive, not particularly practical for segmenting a plane without further improvements (environment restrictions).

- The ICP in practice does not always find the correct alignment.
  Further adjustments needed to enhance the ICP for aligning separate “sliced” object parts.