

Kinfu – an open source implementation of Kinect Fusion + case study: implementing a 3D scanner with PCL

Michele Pirovano
PhD student in Computer Science at POLIMI
Research fellow at UNIMI

Introduction

With this project, we aim to explore solutions for the reconstruction of real objects by analyzing the state-of-the-art algorithm Kinect Fusion that allows real-time reconstruction and rendering of a real world scene and one implementation of it. We use the point clouds obtained from Kinect Fusion and the PCL libraries to create a 3D scanner application and show the results of our work.

This document is divided in four parts:

- Part one introduces the Kinect Fusion algorithm with a detailed explanation of its inner workings.
- Part two introduces the Point Cloud Libraries (PCL) and KinFu, the implementation of the Kinect Fusion algorithm that is contained in PCL.
- In part three, the 3D scanner algorithm implemented using PCL is explained.
- In part four, example results obtained with the 3D scanner application are shown.

Kinect Fusion

Kinect Fusion is an algorithm developed by Microsoft Research in 2011¹². The algorithm allows a user to reconstruct a 3D scene in real-time and robustly by moving the Microsoft Kinect sensor³ around the real scene. Where SLAM techniques provide efficient camera tracking but only rudimentary reconstructions, Kinect Fusion's results possess both a high degree of robustness and detail (see Figure 1 and Figure 2). A detailed explanation of Kinect Fusion's components and algorithm follows.

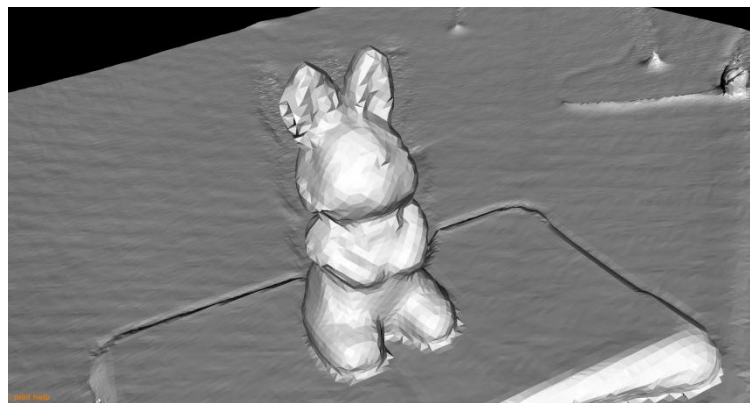


Figure 1 - A rabbit-like statue reconstructed with Kinect Fusion

¹ *KinectFusion: Real-Time Dense Surface Mapping and Tracking*

Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon - October 2011 - IEEE ISMAR

² *KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera*

Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon - October 2011 - ACM Symposium on User Interface Software and Technology

³ <http://en.wikipedia.org/wiki/Kinect>

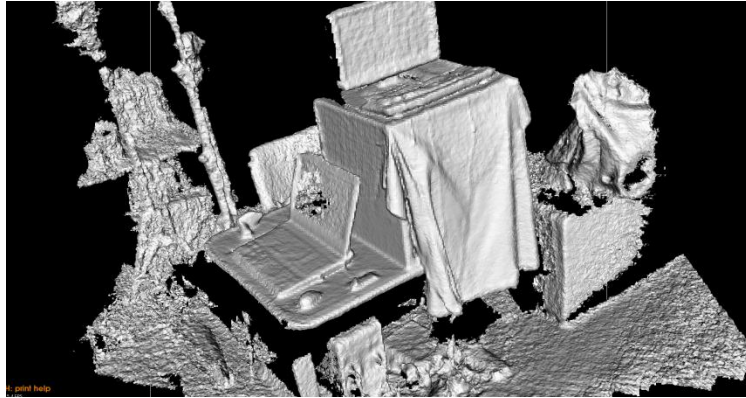


Figure 2 - Part of the AIS-lab at the University of Milan, reconstructed with Kinect Fusion

Components

Kinect Fusion requires a Microsoft Kinect sensor. The Kinect has been originally created for use with the Xbox 360 console as a natural input device for gaming. The sensor is equipped with a RGB camera (640x480 pixels at 30 FPS), an infrared projector and an infrared sensor that, combined, provide a depth map of the scene (640x480 pixels at 30 FPS), a four-microphone array with built-in noise suppression and directionality estimation and a tilt motor (see Figure 3). Internally, the Kinect has a processor for data acquisition, manipulation and transmission.

The Kinect has been rising in popularity lately due to its release as an input device for the Windows platform, allowing a range of diverse NUI (Natural-User Interface) applications to spawn.

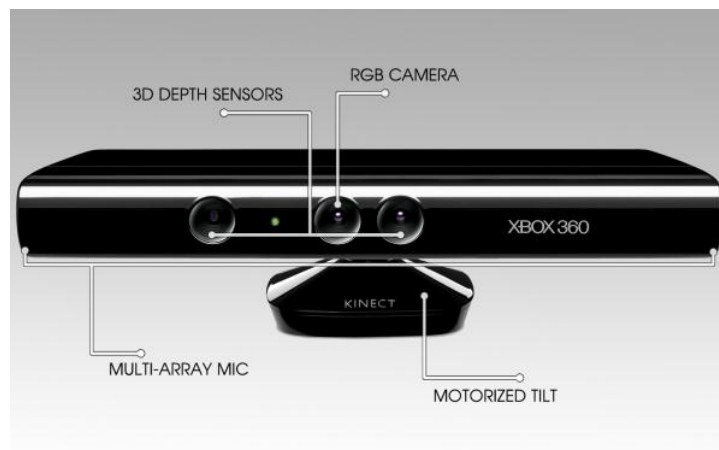


Figure 3 - The Microsoft Kinect sensor

Other than the Kinect, for Kinect Fusion to work a PC equipped with a CUDA-enabled Nvidia GPU is required, used for fast parallel processing. The processor is also required to be of high power, as the approximations done by Kinect Fusion are acceptable only in the case the algorithm can run at least at 30 frames per second.

Algorithm

The input of the Kinect Fusion algorithm is a temporal sequence of depth maps returned by the Kinect. Since the algorithm is using just the depth maps and no color information, lighting conditions do not interfere with it, allowing Kinect Fusion to function even in complete darkness. The algorithm runs in real-time, so it proceeds by using one input depth frame after the other as it is fed from the sensor. A surface representation is extracted from the current depth frame and a global model is refined by first aligning and then merging the new surface with it. The global model is obtained as a prediction of the global surface that is being reconstructed and refined at each new step.

Surface extraction

At each new frame, a new depth map obtained from the Kinect sensor is used as input.

A depth map is a pixel image that, instead of holding color values, holds depth values, i.e. distances from the camera to the 3D scene points. Kinect computes the depth map by emitting a non-uniform infrared pattern on the scene through its infrared projector (see Figure 4) and then acquiring the same pattern through an infrared sensor, measuring its time-of-flight (TOF)⁴. Holding hard-coded images of the light pattern, the processor inside the Kinect is able to match the incoming pattern to the hard-coded images, it finds correlations and thus computes the 3D positions of the points in the scene, building the depth map⁵.



Figure 4 - The infrared pattern emitted by Kinect

The depth map returned by Kinect is bound to be noisy, thus a bilateral filter⁶ is applied to it that smooths the image and removes the noise from the depth values through neighbor averaging of values around a single pixel, weighted by the difference in depth intensities from the target pixel to its neighbors. The result is a smoother depth that preserves sharp edges (see Figure 5 for an example).



Figure 5 - A bilateral filter applied to a 2D image. The left picture is the original image. The right picture has the filter applied to it, resulting in noise removal.

In order for the Kinect Fusion algorithm to work, the depth map must be converted into a 3D point cloud with vertex and normal information. This is done at different, layered resolutions, resulting in a number of images with different levels of detail, which for the purpose of the standard algorithm is set to 3

⁴ <http://www.wired.com/gadgetlab/2010/11/tonights-release-xbox-kinect-how-does-it-work/>

⁵ <http://courses.engr.illinois.edu/cs498dh/fa2011/lectures/Lecture%2025%20-%20How%20the%20Kinect%20Works%20-%20CP%20Fall%202011.pdf>

⁶ <http://scien.stanford.edu/pages/labsite/2006/psych221/projects/06/imagescaling/bilati.html>

(see Figure 6). This is also called a *multi-resolution pyramid*. The lower resolution layers are obtained through sub-sampling.

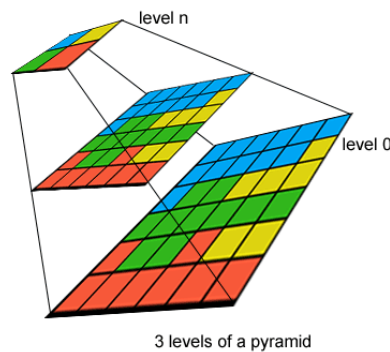


Figure 6 - A 3-level multi-resolution pyramid

The depth map is converted into a 3D point cloud through back-projection. To do so, the Kinect's internally stored calibration matrix is used, that matches depth pixels to actual 3D coordinates. The normal of each point is estimated through cross-product of two vectors: the vector joining the chosen point and the one above it, and the vector joining the chosen point and the one to its right. Since the points are arranged as the depth pixels, choosing point (x,y,z) we can take the point $(x,y+1,z)$ and $(x+1,y,z)$ for normal estimation, with x and y being the image coordinates and z being the depth at those coordinates.

The result of this step is a point cloud with vertex and normal data for each point at three different levels of detail. Note that the cloud is considered ordered, as the points are arranged as the depth map's pixels.

Alignment

At the first time step of the algorithm, this point cloud is regarded as our model. At later steps, the new point cloud is aligned to the model we have (see the next section) and then merged with it to produce a new, more refined model through an iterative method. To perform alignment, the Iterative Closest Point algorithm (ICP) is used⁷⁸⁹.

The ICP algorithm performs alignment of two point clouds, called source and target. After ICP executes successfully, the source will be aligned to the target. The algorithm requires the source cloud to be already close to the correct match, as such ICP is used for refinement, while other methods (such as SAC-IA, as explained later) can be used to perform an initial alignment.

The ICP algorithm works by estimating how well the two current point clouds match. This can be done in different ways, but the preferred and simpler method is to use point-to-point errors: for each point in the first cloud, the algorithm searches for the closest point in the second cloud and computes its distance (the search is conducted inside a user-defined radius). The transform that aligns the source cloud to the target cloud is then the one that minimizes the total error between all point pairs (see Figure 7 for an example).

This alignment step is performed iteratively until a good match is found. The standard ICP algorithm has three alternative termination conditions:

- The number of iterations has reached the maximum user imposed number.
- The difference between the alignment transform at the last iteration and the current estimated alignment transform is smaller than an user imposed value.

⁷ P. Besl and N. McKay. A method for registration of 3D shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 14(2):239–256, 1992

⁸ http://en.wikipedia.org/wiki/Iterative_closest_point

⁹ http://pointclouds.org/documentation/tutorials/iterative_closest_point.php

- The sum of Euclidean squared errors is smaller than a user defined threshold.

The algorithm is thus $O(N*M*S)$ where N is the number of points in the target cloud, M is the number of points considered in the closest-point search and S is the number of iterations.

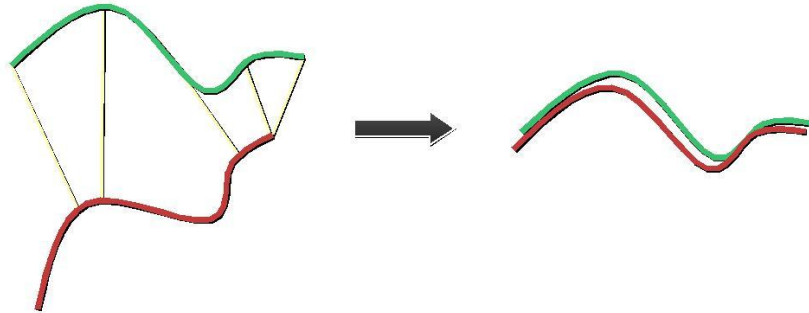


Figure 7 – How an ICP step works in 2D

Kinect Fusion does not use the standard ICP algorithm, since it is too slow for its real-time purposes. Instead, it makes the assumption that the changes between the current cloud and the previous one are small. The assumption is reasonable if the algorithm runs in real-time (at 30 frames per second) and if the Kinect is moved around the scene at a slow pace. If the processor, however, is not able to cope with the speed, the ICP algorithm will fail and the new cloud will be discarded. The modified ICP then back-projects the two clouds onto the camera image frame of the model and considers two points to be a match if they fall on the same pixel. This allows the algorithm to be parallelized on GPU. To further speed up the computation, the ICP iterations are performed at the three resolutions, starting from the coarser one, and the transformation matrix is computed in the GPU by taking advantage of the incremental nature of the transform (due to the small difference between two consecutive frames). Thanks to this, the modified ICP algorithm presents a complexity of only $O(S)$, with S is the number of iteration steps, which is kept small by the use of the multi-resolution pyramid. In addition, the assumption of small changes between two subsequent frames allows ICP to be used without resorting to another algorithm for initial alignment.

After a match is found, the modified ICP algorithm computes the error between the two points in a match with a point-to-plane metric, instead of a point-to-point metric. This means that the error is computed as the distance between the plane parallel to the surface at the first cloud point and the position of the second cloud point. This metric has been shown to converge faster than the point-to-point metric¹⁰.

ICP, after a set of iterations, generates a six degrees of freedom transformation matrix that aligns the source point cloud to the target point cloud with a rotation and a translation.

Surface reconstruction

Once the alignment transform is found, and thus the current pose of the camera is estimated, the new cloud can be merged with the current model. The raw depth data is used for merging instead of the filtered cloud to avoid losing details. A Truncated Signed Distance Function (TSDF) is used for this step. This function basically extracts the surface of the objects in the scene and assigns negative numbers to those pixels that are either inside objects or inside an area we have not yet measured, positive numbers to pixels that are outside the surface, increasing the further they are from it, and zero to the pixels that are on the surface (see Figure 8). A TSDF is computed for the new cloud and merged to the current model's TSDF through a weighted running average to compute the new surface.

This representation has been chosen by the authors mainly due to the ease of merging different TSDFs just by weight averaging and due to the ease of parallelization, which allowed the authors to achieve real-time performance.

¹⁰ http://www.comp.nus.edu.sg/~lowkl/publications/lowk_point-to-plane_icp_techrep.pdf

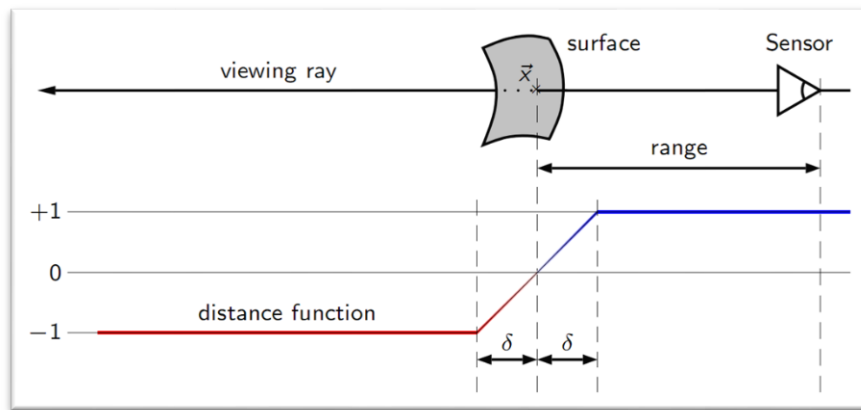


Figure 8 - How a TSDF is created

The resulting TSDF is used to reconstruct the surface of the current, refined model. To do so, ray casting is performed.

Ray casting is a surface visualization method used in computer graphics¹¹. It consists of choosing a point of view, the camera's focal point, and casting rays in the scene from it. A ray is basically a traveling line, it starts from the point of view with zero length and travels in a chosen direction until it intersects something. When the ray intersects a surface, the pixel it corresponds to in the image space of the camera is assigned to the 3D intersection point. This makes ray casting, with its output-centric focus, different from typical object-centered visualization methods such as vertex and segment rendering. Ray casting is often used in the visualization of translucent and semi-transparent objects, since it is easy to perform reflection and refraction by changing the direction of the traveling rays. It is also considered one of the better 3D visualization methods, resulting in high-quality images. Ray casting is also used for physics checks in virtual reality environments, as well for other distance-based computations such as 3D user-picking.

In Kinect Fusion ray casting is used for this surface reconstruction step. Ray casting is performed from the global camera focal point by intersecting the zero level set of the TSDF. A prediction of the current global surface is obtained, with vertex data and estimated normal data. The refined, ray-casted model is the one that will be used in the next ICP step for alignment. By doing so instead of using just the last frame point cloud as the source for alignment, a less noisy model is obtained, placing the focus on finding a smooth surface. The process is sped up by using ray skipping, that is by advancing the rays at discrete steps. Ray casting is also used for the final visualization step. The end result is a 3D surface representing the acquired scene (see Figure 2).

Conclusion and notes

Kinect Fusion allows reconstruction of 3D scenes in real-time with ease thanks to its assumptions and the heavy parallelization. However, it still presents a few issues.

The algorithm requires high computational power and a powerful GPU to work, due to the assumption of small movements between two frames. In addition, even at a high frame-rate, the motion of the Kinect sensor must be kept in check, as a sudden tilt or translation may break the assumption. The consequence of breaking the assumption is that the ICP alignment may not converge to a correct match. In this case, the algorithm will discard the new cloud and fall back to an older pose.

However, as we have experienced with the KinFu implementation (see the next section), a different problem may arise. Due to sudden movements and thus due to the breaking of the assumption, the ICP alignment step may erroneously converge to a wrong match. In this case, the algorithm is not able to detect that tracking has been lost and instead will merge two different clouds together, breaking both the tracking and the reconstruction. We found that the major reason of loss of tracking in the scene was the

¹¹ http://en.wikipedia.org/wiki/Ray_casting

ambiguity of the scene itself. A scene with many details, such as presenting scattered objects on the floor or having objects with distinct features, results in the pose being correctly tracked. This is due to how ICP works, requiring specific features in the scene to be present in order to find the correct alignment pose.

It must be noted, however, that the Kinect Fusion algorithm is robust to the presence of dynamic objects in the scene, which are removed from the point cloud model. This means that someone may walk into the Kinect's field of view without contaminating the result, given enough iterations of the algorithm (see Figure 9). The robustness to dynamic objects presence is due to the weighted running average used during the merging of the TSDFs.

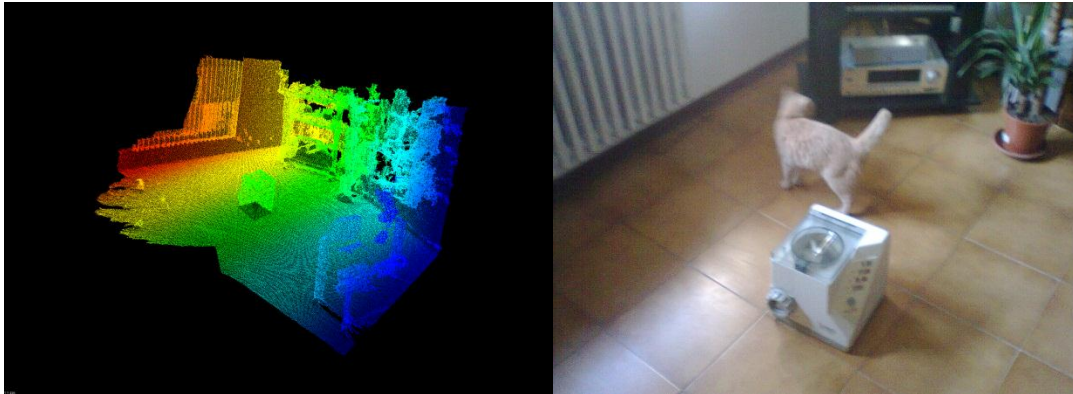


Figure 9 - Right: the point cloud reconstructed using Kinect Fusion (KinFu). Left: the actual scene, with a cat intruder passing by. There is no sign of the moving intruder in the reconstructed scene.

KinFu – an open source implementation of Kinect Fusion

Point cloud library

The Point Cloud Library¹² (PCL) is a stand-alone, large scale, open project for 2D/3D image and point cloud processing. The library is freely available under a BSD license and the project is actively maintained through the collaboration of many universities and the industry. PCL can be effectively used to work at a high level with point clouds (as well as, at a minor degree, other 3D representations). What the library brings to the table is a set of high-technology easy-to-use modules that allow the developer to focus on the results of image manipulation and not on the need to re-invent the wheel.

The library is divided in several separate library modules, each with its own focus, such as the *filters* library or the *visualization* library. These libraries allow the developer to perform filtering, feature extraction, keypoint and model detection, registration, neighbor search, segmentation (see Figure 10), surface reconstruction, input/output handling and visualization.

We use this library both for its implementation of Kinect Fusion (see next section) and for the many classes and methods it exposes for manipulation of point clouds, used for our 3D scanner.

¹² <http://pointclouds.org/>

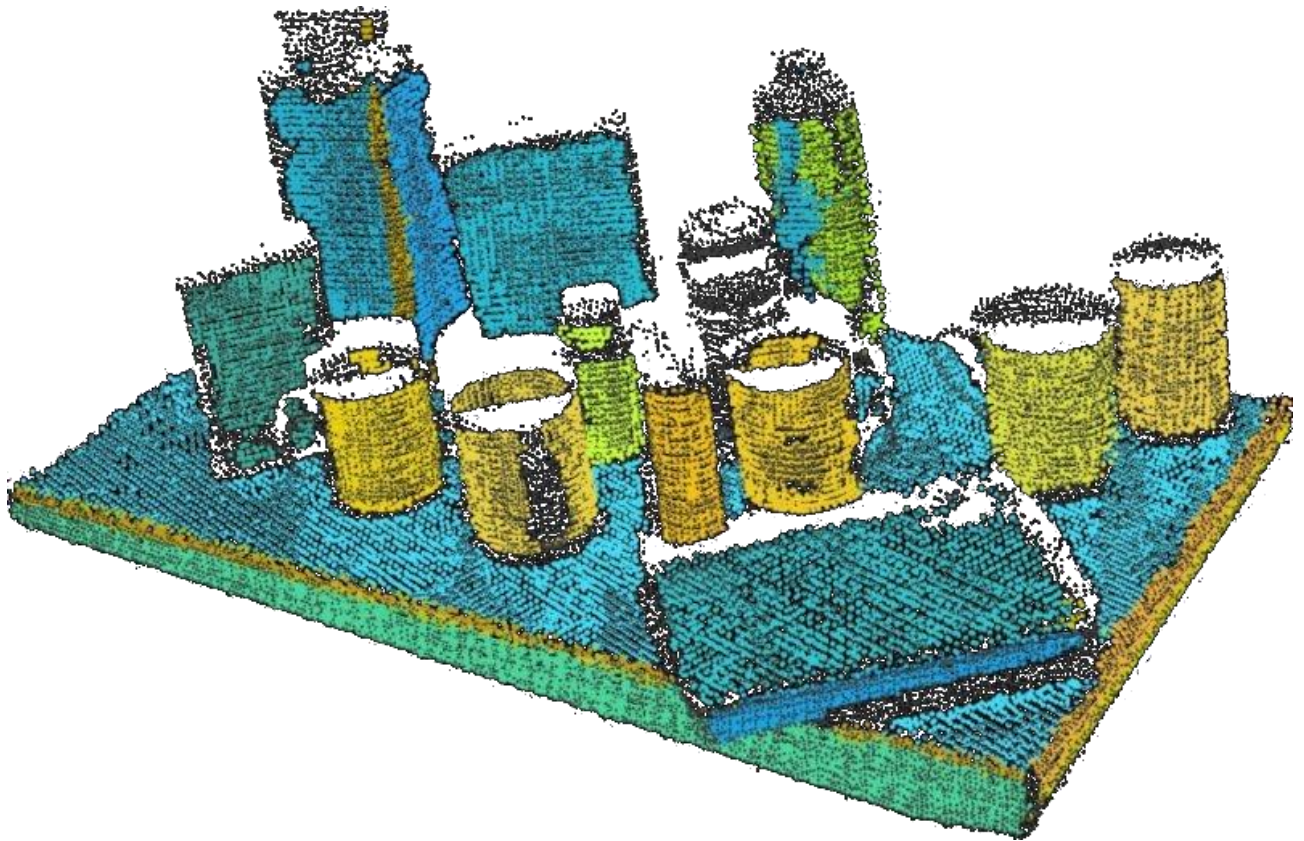


Figure 10 - The result of sample consensus to segment and detect shapes in a point cloud using PCL.

KINFU

KinFu is an open source implementation of Kinect Fusion and is part of the PCL library¹³. It is currently in a refinement phase and thus it is not in the stable version of PCL (as of version 1.4). KinFu can be currently found in the development version of PCL and can be freely downloaded as part of the trunk of PCL from the project's SVN repository, hosted on SourceForge¹⁴. The code is written in C++ and uses the PCL libraries.

KinFu introduces a few modifications to the published Kinect Fusion algorithm, as listed here. A few of the modifications can be found by reading the implementation's code. Where the original Kinect Fusion algorithm worked only with a Microsoft Kinect, KinFu can instead work with any OpenNI-compatible depth camera, due to its open-source nature. The normal estimation step is done through eigenvalue estimation instead of simple cross-vector computation. Rendering is done using a marching cubes algorithm instead of ray casting. A few PCL contributors also merged the color information into the results of KinFu, resulting in the reconstructed surface being textured with the color input of the scene. This has however also been shown by the authors of Kinect Fusion in a video of their implementation. In addition, KinFu allows the surface mesh output to be saved to .vtk files and it also provides a point cloud representation of the global scene.

The choice of using different methods than those used in the original Kinect Fusion such as the eigenvalue normal estimation and the marching cubes rendering algorithm is probably due to the fact that KinFu will be later merged with the other PCL libraries, thus already existing and proven implementations of such methods would be easily available to KinFu. Such methods are, as of now, implemented into the KinFu code, probably due to their PCL development implementation being unstable, as we were sadly able to experience while trying to use the marching cubes implementation found in the PCL trunk. It is safe to

¹³ <http://pointclouds.org/news/kinectfusion-open-source.html>

¹⁴ <http://svn.pointclouds.org/pcl/>

assume that the existing PCL implementations will be used instead when the KinFu code is published in the stable trunk.

In addition to these modifications, the contributors of PCL have created a large-scale version of the Kinect Fusion implementation (KinFu large-scale) that can be used to reconstruct large scenes, such as full rooms¹⁵ (see Figure 11). This version however needs an even larger computational power than standard KinFu due to additional operations needed to manage the larger area and due to missing optimizations, with the authors reporting reaching 20 fps using a GTX480 and 4GB of RAM.

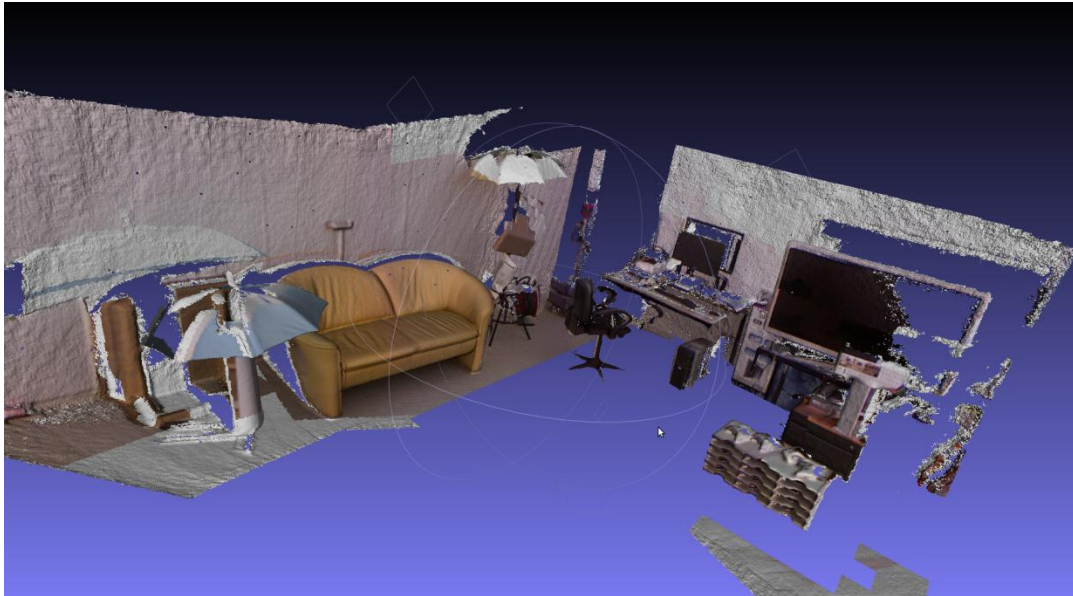


Figure 11 – A scene reconstructed using KinFu large scale

We compiled the PCL trunk as well as KinFu, tested its functionalities and used its output for our 3D scanner application. Examples of our output using the program can be seen in Figure 1 and Figure 2 at the start of this document.

3D scanner

Using PCL we created a 3D scanner that takes as input point clouds created using KinFu and produces as output the 3D mesh of a desired object in the scene, previously positioned accordingly. The clouds are filtered, aligned and merged to obtain a single cloud, representing the object, that is then used for mesh reconstruction. See the results section for an example of application.

The algorithm

The algorithm requires two point clouds, cloud A and B, that contain on a planar surface the desired object placed in different orientations. The multiple point clouds are used to reconstruct the 3D object in its entirety, since the area that lies on the ground (the base of the object) is occluded by the ground itself.

The algorithm goes through three phases: filtering, alignment and reconstruction.

Filtering

The two clouds, A and B, are first down-sampled using a voxel grid filter to reduce the number of points in the cloud for further computation. The voxel grid filter works by dividing the 3D scene into voxels and

¹⁵ http://pointclouds.org/documentation/tutorials/using_kinfu_large_scale.php

approximating all points in a voxel with their centroid¹⁶. This has the advantage of reducing the computation needs in the next steps and of obtaining a more structured cloud, since the density of points is kept in check.

Cloud A and B are then cropped using a crop-box filter, that is a high pass filter on the three spatial dimensions. The result is that the points outside a chosen 3D box get discarded. Special care must be taken to extract only the desired object from the scene through this cropping phase.

However, the floor is still present in the resulting clouds, hence we perform a segmentation of the clouds to remove it. We define a planar model and apply Random Sample Consensus (RANSAC) to the cloud in order to find it. RANSAC¹⁷ is an iterative method for estimating model parameters given a set of points as input containing both inliers (points that belong to the model) and outliers (points that do not belong to the model). The method proceeds by iteratively selecting a random subset of the input data, fitting the model to it and keeping the model that fits the majority of the total input data. After finding the points that define the plane, we remove them and obtain a cloud representation of the desired object.

The cloud, at this stage, still possesses noisy outliers. We perform a final filtering phase to remove the noise from the cloud, using a radius outlier removal method, which removes points which do not have enough neighbors in a sphere of chosen radius around them.

What we obtain is two noiseless cloud representations of the desired object, although with different positions and orientations in the 3D space and missing the side that is contact with the floor, disappeared due to floor plane removal.

Alignment

After filtering is complete, the two clouds need to be aligned, so that they can be merged together.

A preliminary step should be performed, which consists in the removal of a side from one of the two clouds (for instance, cloud A) through an additional pass of the crop-box filter. This has been deemed necessary to obtain a correct alignment (see the results section). The resulting cloud is called A°.

We perform a first alignment using the Sampled Consensus – Initial Alignment algorithm (SAC-IA)^{18,19,20}. This algorithm performs the alignment of a source cloud to a target cloud with an idea similar to the RANSAC method. A sub-set of points is randomly selected from the input data and a set of closest matches is acquired. A transform is then estimated using these matches and point-to-point error metrics (just as with the ICP algorithm). This is repeated a number of times and the best transform is kept. This results in an initial alignment that must however be refined due to its correspondence to only a sub-set of points.

SAC-IA can be modified, as is our case, to use point features instead of just raw point data. Point features are point representations which encode alongside the point's position additional information regarding the neighboring geometric area. The simplest example of a point feature is the surface normal estimated at a 3D position. We compute Fast Point Feature Histograms²¹ (FPFHs) for the input cloud and use these features for the SAC-IA alignment. Point Feature Histograms (PFHs) attempt to capture the variations in the curvature of the surface in a point by taking into account the estimated normal of the point and its neighbors, resulting in 6 degree-of-freedom invariant features that cope very well with noise and different sampling densities. FPFHs are similar, but trade some accuracy for speed.

In our case, cloud B is the source and cloud A° is the target, hence the result is a newly oriented cloud B: cloud B'.

¹⁶ http://pointclouds.org/documentation/tutorials/voxel_grid.php

¹⁷ <http://en.wikipedia.org/wiki/RANSAC>

¹⁸ http://pointclouds.org/documentation/tutorials/template_alignment.php

¹⁹ Fast Point Feature Histograms (FPFH) for 3D Registration - Rusu, Radu Bogdan., Blodow, Nico., and Beetz, Michael – 2009 - ICRA

²⁰ <http://www.pointclouds.org/assets/icra2012/registration.pdf>

²¹ http://pointclouds.org/documentation/tutorials/fpfh_estimation.php

The alignment is then refined using ICP, resulting in an almost perfect match of the two clouds. Applying ICP using cloud B' as the source and cloud A° as the target, we obtain cloud B''. The choice of two alignment algorithms lies on the fact that ICP performs well when the its input clouds are relatively already close enough, while SAC-IA is less accurate but more robust to large translations and rotations. At last, cloud A (not A°, since we want to use all the points of the cloud in this phase) and cloud B'' are merged into one: cloud C.

Reconstruction

Surface reconstruction tries to extract a mesh from a point cloud, ordered or unordered. Several algorithms have been devised for this purpose.

For this project, we use a greedy triangulation algorithm for fast reconstruction, which creates polygons by linking points and thus transforming them into vertices. Another possibility is to use a marching cubes algorithm, which tied to smooth normal rendering would result into a smoother mesh (used by KinFu itself), or a ray-casting method similar to the one used in the Kinect Fusion algorithm.

The merged cloud C is the point cloud we want to reconstruct a mesh around. However, the cloud will be noisy partly due to the structure of the input clouds and partly due to the non-perfect alignment of the two clouds. If we were to apply the greedy triangulation algorithm to this cloud, we would obtain a really jagged mesh. We thus apply a Least Moving Square (LMS) algorithm to smooth the point cloud before reconstruction, obtaining cloud C'. The LMS algorithm allows resampling of a point cloud by recreating the missing parts of the surface through higher order polynomial interpolations between the surrounding data points²².

Cloud C' is at last reconstructed using our surface extraction method of choice. The output is a 3D mesh of the object, which can be saved for use with other software.

Implementation

The described algorithm has been implemented in C++ with an extensive use of the PCL libraries. Choices taken when designing the algorithm previously described are due to what PCL has to offer.

Details on the parameters used by the implementations are listed in the results section alongside the values we used for those parameters.

We take advantage of the following modules:

Io

From the *input/output* module, we use methods to load the point clouds for use with PCL from .pcd files (which stands for "point cloud data"). We also use methods to save the result mesh into a .vtk file.

Features

From the *features* module, we use normal estimation methods, used both by the alignment phase and reconstruction, and feature extraction methods such as the Fast Point Hystogram Feature extraction implementation.

Registration

From the *registration* module, we use the implementations of the SAC-IA and ICP algorithms.

Filters

From the *filters* module, we use the voxel grid filter, the crop box filter and the radius outlier removal filter.

²² <http://pointclouds.org/documentation/tutorials/resampling.php>

Surface

From the *surface* module, we use the MLS implementation and the greedy triangulation implementation.

Kdtree

From the *kdtree* module, we use the tree search implementation, used for registration and triangulation.

Visualization

From the visualization module, we use the PCLVisualizer class to visualize our result mesh.

Due to the not complete stability of the development trunk of PCL, a few methods we wanted to use have not been inserted into the final algorithm, such as the marching cubes surface reconstruction method.

Results

We used the 3D scanner application to reconstruct an object placed on the floor, specifically the kitchen machine of Figure 12. This object is a good candidate for reconstruction as it has no symmetry plane and because it is rigid. Notice however that the top part has a semi-transparent cover, which will result in a more noisy reconstruction.



Figure 12 - The machine we want to reconstruct

Using KinFu, we extracted two point clouds. One of the two clouds (cloud A) was extracted for the object in a normal position resting on the floor (Figure 12). The resulting cloud, saved as a .pcd file from KinFu and viewed using PCL's cloud viewer program, can be seen in Figure 13.

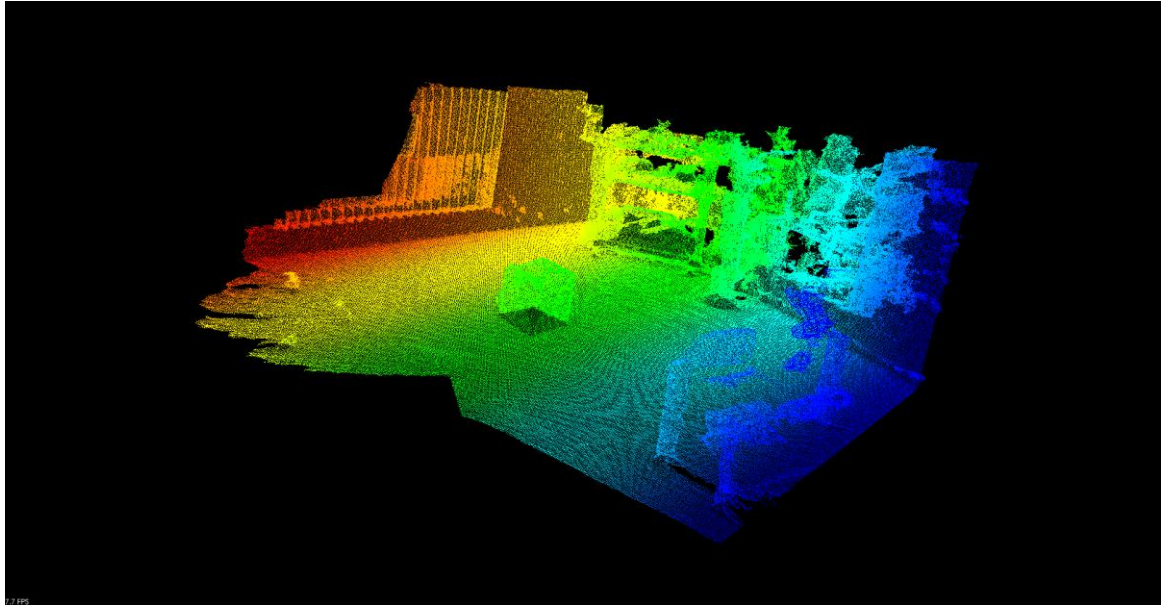


Figure 13 - Cloud A. The object is resting on its base.

The second cloud (cloud B) was instead extracted after rotating the object and placing it on one of its sides. The extracted cloud can be seen in Figure 14.

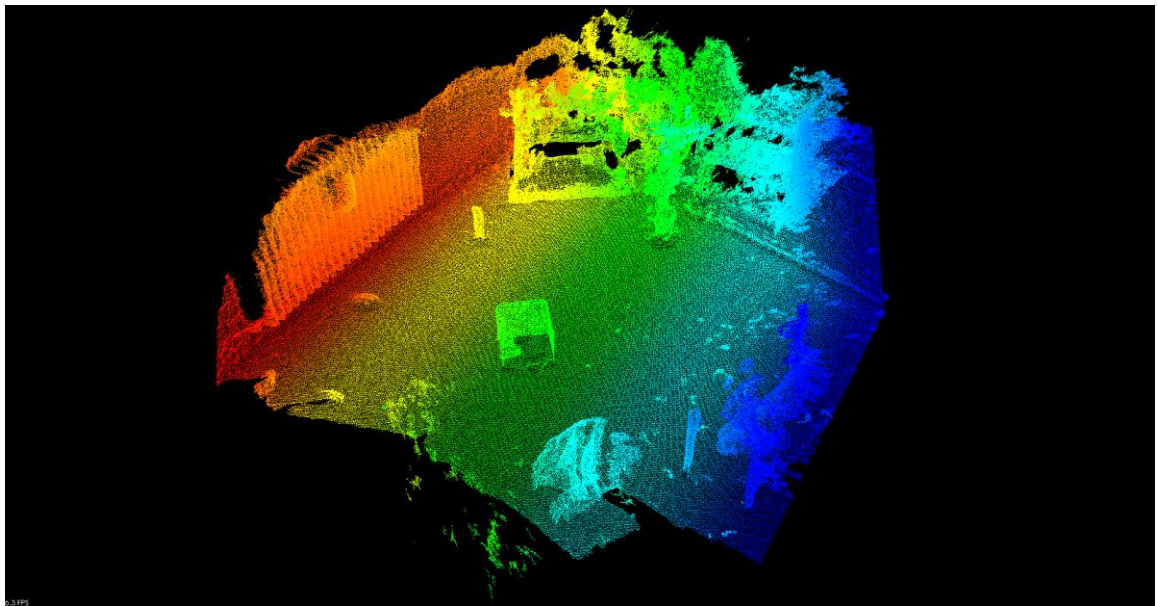


Figure 14 - Cloud B. The object is resting on one of its sides.

The use of these two clouds allowed us to reconstruct the object in almost its entirety, since by using the first cloud we would not have been able to reconstruct the base of the object (it cannot be seen by the sensor). One edge, however, could not be reconstructed (see the final result in Figure 24) due to it being in contact with the floor in both clouds. This problem would be easily solved by using three different clouds, or by placing the object upside-down for the second step of extraction.

Following the algorithm we designed and using the PCL-enabled implementation, the two clouds were filtered until achieving the results in Figure 15 and Figure 16.

We manually inserted the grid size for voxel-grid filtering (0.01 units), the search radius (0.02 units) and the number of neighbors (10) for the radius outlier noise removal filter and the two points for defining the

crop-box filter parallelepipeds ([1,1,0] and [1.8,1.8,1.6] for the first cloud, [1.2,1.5,0.1] and [1.8,2.0,1.2] for the second cloud). The chosen values were found by first computing the dimensions in cloud-space of the object. Its side measured 0.4 units (that are, with all probability, meters).

The resulting clouds do not present noise and the floor plane has been segmented out, as well as all the other objects that were present in the scene. Cloud A has just 3931 points out of the initial 774490 points, while Cloud B has 4130 points out of the initial 1035783 points.

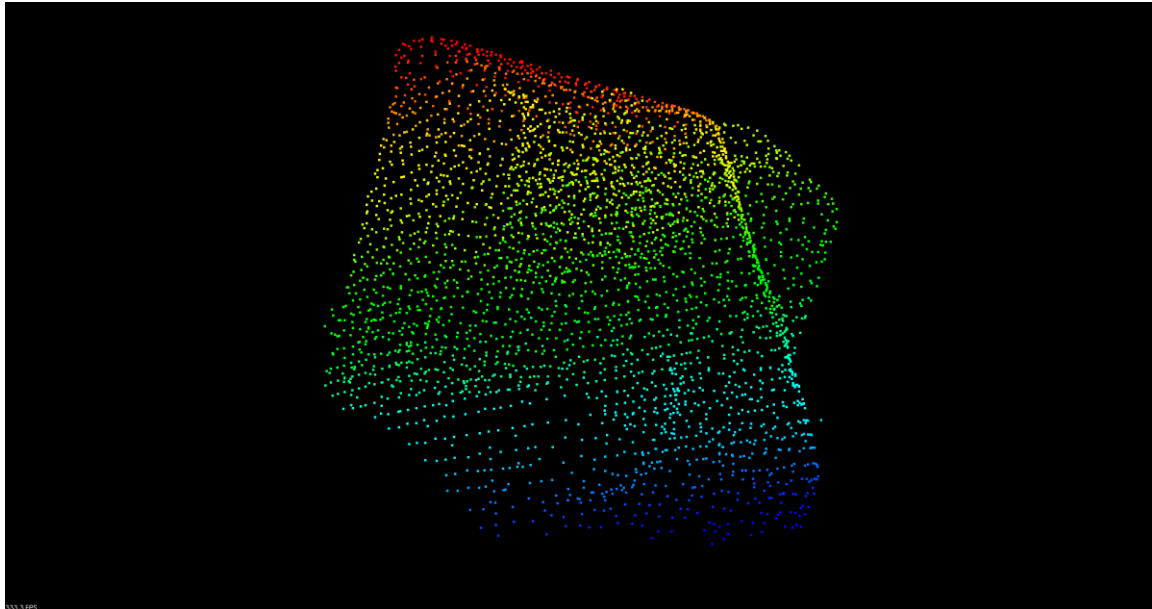


Figure 15 - Cloud A after the filtering phase. The base is empty.

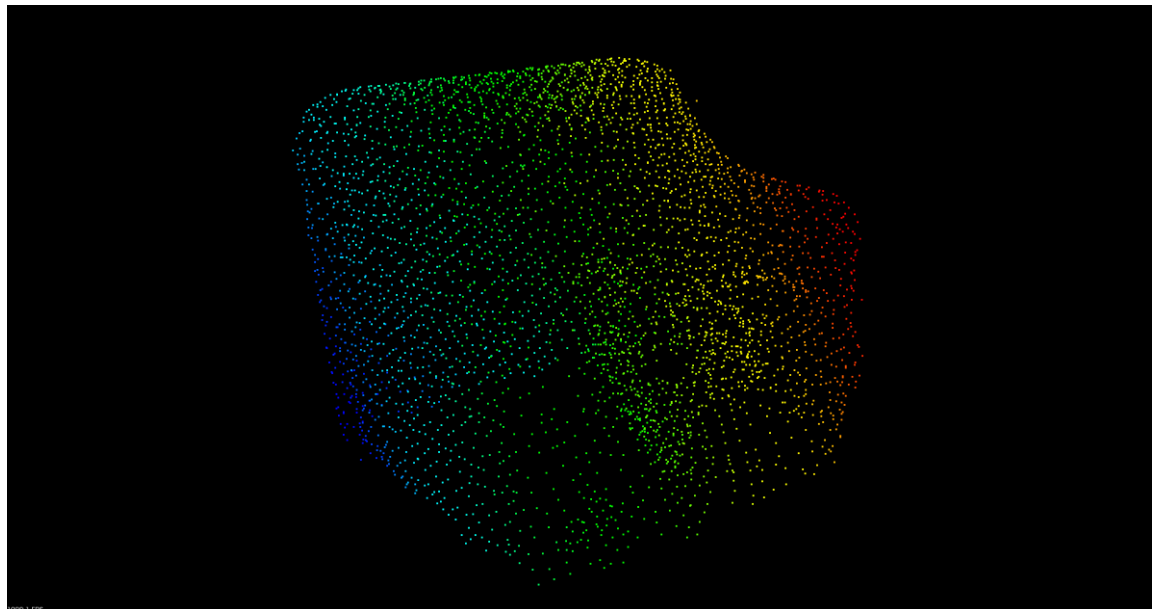


Figure 16 - Cloud B after the filtering phase. The object is leaning on its side and thus that side is empty.

We then removed one side from cloud A, corresponding to the side the object is resting on in the second scene, through an additional pass of the crop box filter, obtaining cloud A° (the parameters for the box filter were points [1.32,1,0] and [1.8,1.8,1.6]). The result is the cloud seen in Figure 17.

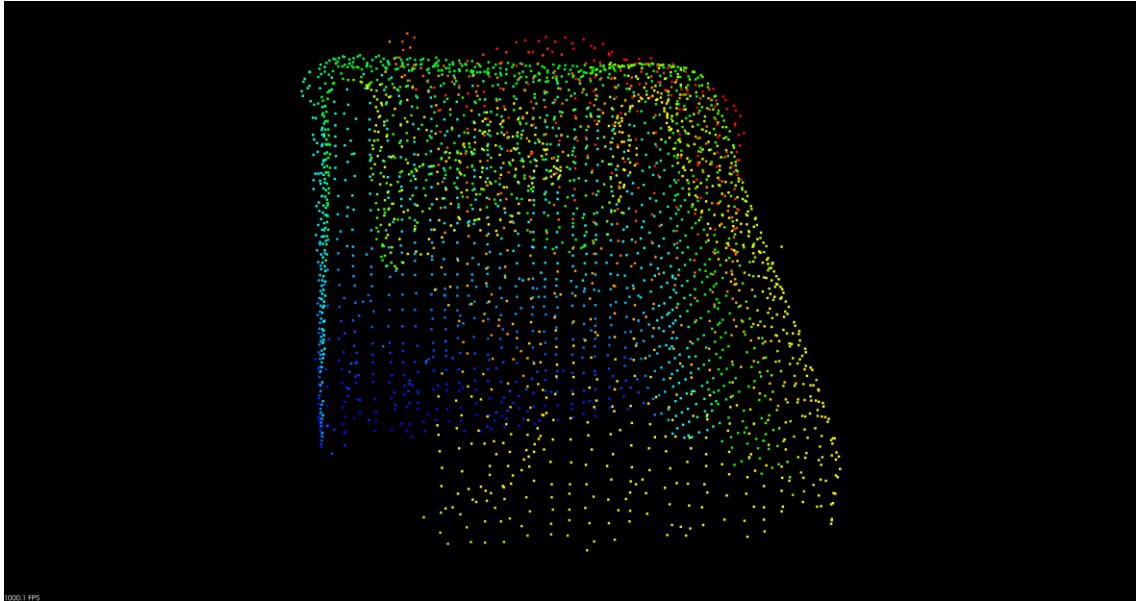


Figure 17 - Cloud A^o, obtained from cloud A by removing the side on the left of the image. Notice that the bottom base is empty as well.

As explained in the details on the algorithm, this step was necessary to obtain a correct alignment. When we tried to align cloud A and B without any additional step, both the SAC-IA algorithm and the ICP algorithm would apply a wrong alignment, due to the fact that both clouds had an empty side (the bottom for cloud A and one of the lateral sides for cloud B). The empty side is considered by the algorithm to be a major feature and the alignment tries to match the two empty sides, as seen in Figure 18.

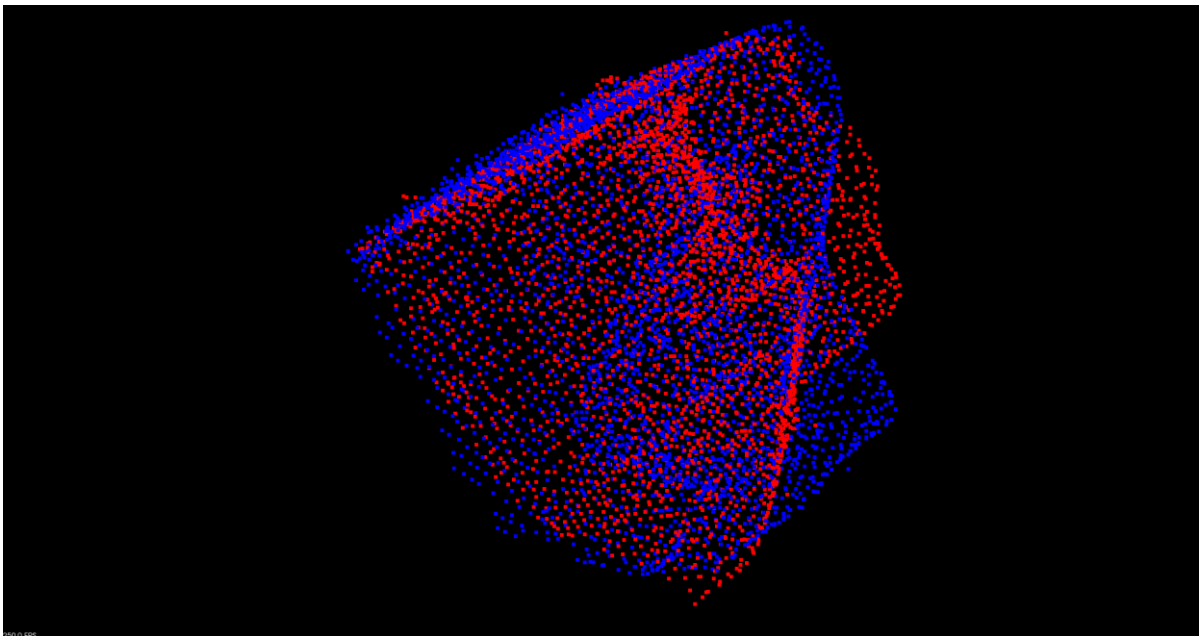


Figure 18 - A wrong alignment is obtained by aligning cloud B'' to cloud A due to the algorithm erroneously aligning the two empty sides of the clouds.

Alignment is then performed first by computing FPFHs for both clouds and then using SAC-IA on these features, obtaining cloud B'. The aligned cloud B' and the target cloud A^o can be seen in Figure 19. From the figure, it is clear that the SAC-IA algorithm has performed just an initial, not accurate repositioning and re-orientation of the cloud. Parameters we used are the number of iterations (500), the FPFH search radius (0.02 units), the minimum sample distance (0.05 units) and the maximum correspondence distance (0.2 units).

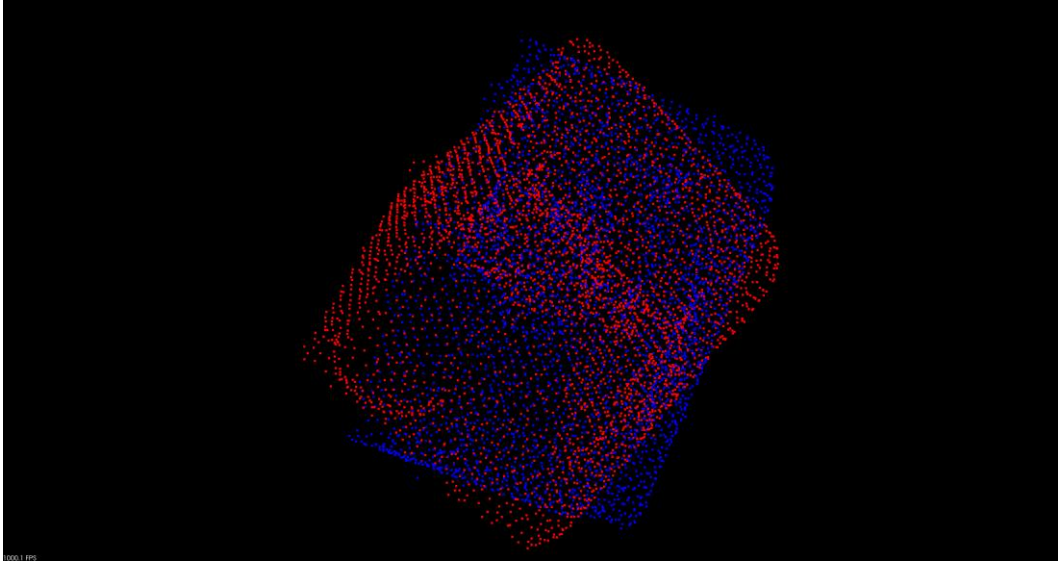


Figure 19 - Cloud A° and cloud B'. An initial alignment is obtained, but a refinement phase is needed.

For a better alignment, we applied the ICP algorithm to B' and obtained B''. The newly aligned cloud B'' and the starting cloud A can be seen in Figure 20. As parameters, we chose to do 400 iterations, without defining any other termination condition. The maximum correspondence distance has been chosen as 0.12 units. The two clouds have been merged to create cloud C, which is composed of 8061 points (Figure 21).

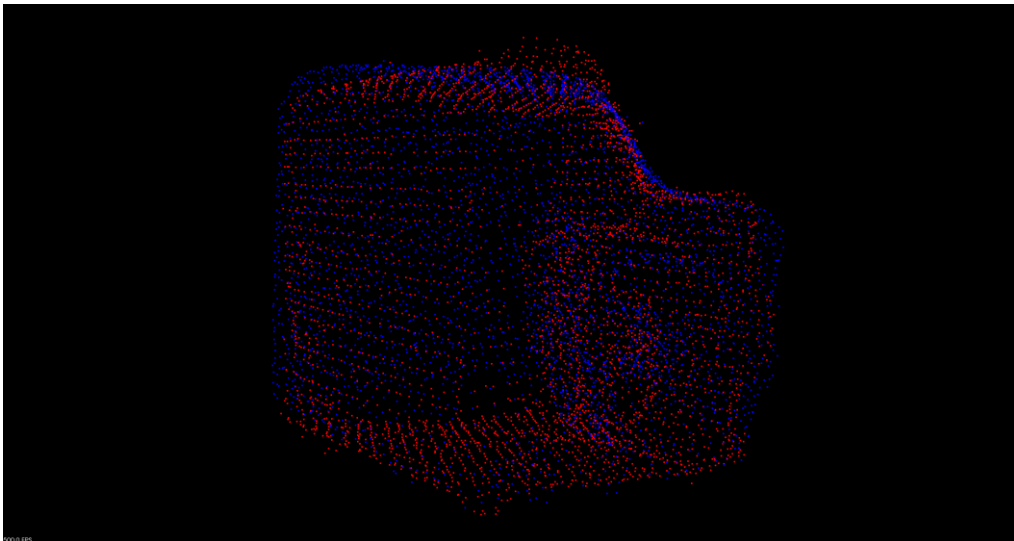


Figure 20 - Cloud A and cloud B''. A refined alignment is obtained.

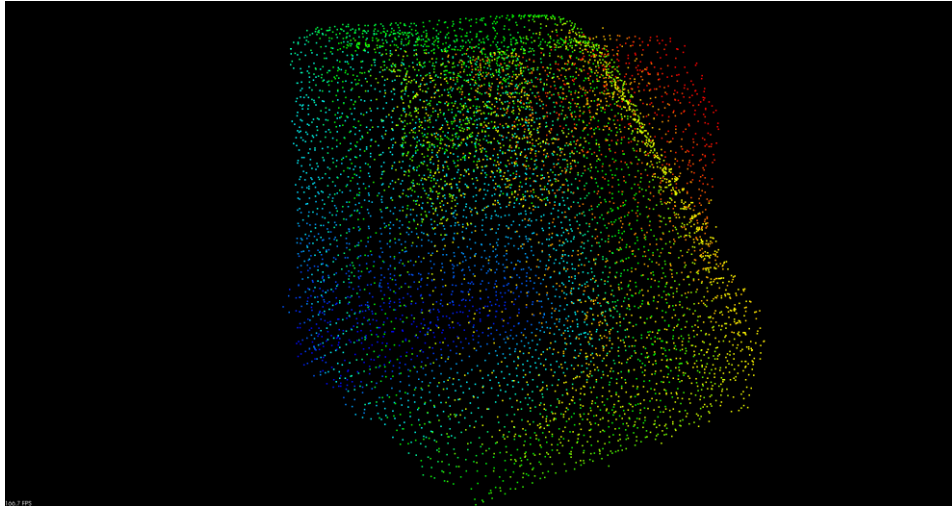


Figure 21 – The resulting cloud C

At last, we reconstruct the 3D mesh by resampling cloud C using LMS into a smoother point cloud (using 0.03 units as the sample size) and then using our surface reconstruction greedy triangulation algorithm. Parameters used for the reconstruction are 0.2 units as maximum edge length, 20 as maximum nearest neighbors considered, 0.25π for the surface angle factor, 0.2π for the minimum angle and 1.6π for the maximum angle. The result using these parameters can be seen in Figure 22, Figure 23 and Figure 24.

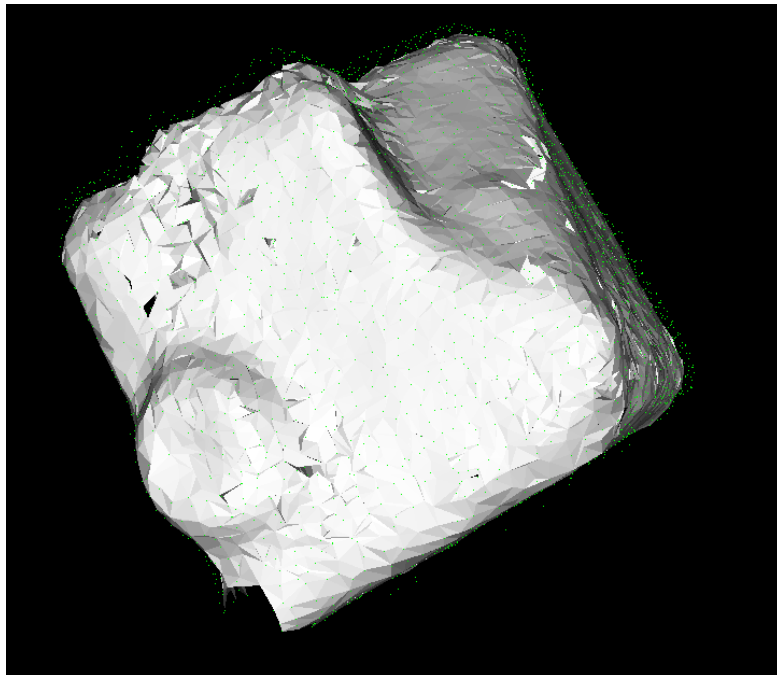


Figure 22 - The reconstructed mesh. The rough shape of the object has been reconstructed.

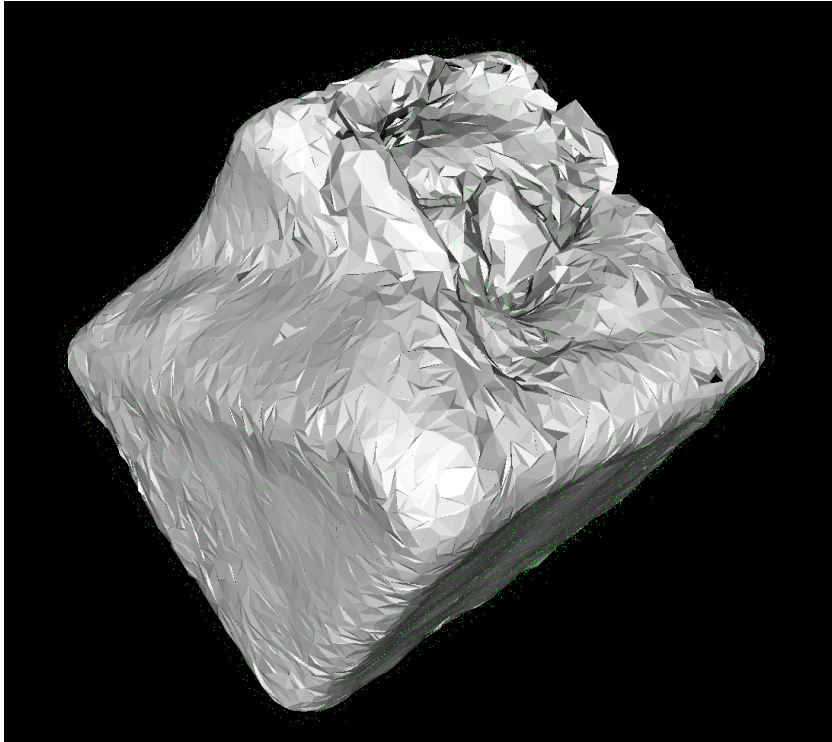


Figure 23 – Although subject to more noise, the inner parts that can be seen through the transparent cover are reconstructed as well.



Figure 24 - The edge that leaned against the floor for both object orientations cannot be reconstructed.

Previous attempts

Previous attempts were made with different objects, such as the plush parrot of Figure 25 or the tea maker of Figure 26. These two objects, however, could not be easily reconstructed.

The plush resulted in a misalignment of the two clouds due to the fact that it is not a rigid body. When placing the plush on its side, it would deform and the resulting extracted point cloud would be different from the one extracted previously (see Figure 27).



Figure 25 - A target for reconstruction, the plush parrot



Figure 26 - Another target for reconstruction, the tea maker

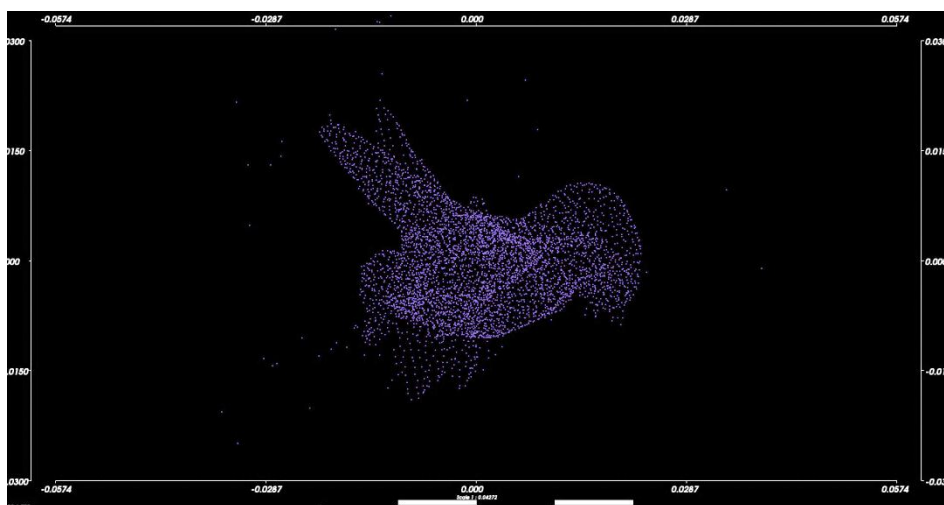


Figure 27 - The two clouds representing the plush parrot are misaligned

The second object showed different problems (that however was a small factor also for the alignment of the plush parrot). The object presents an illusion of radial symmetry that resulted in different problematic effects.

During the use of KinFu, moving around the object while capturing it, the application would often remove the object from the result point cloud model. This would happen because KinFu erroneously assumed that the object was rotating around its vertical axis and thus removed it from the scene, labeling it as a dynamic object. This would happen often when the Kinect was able to see only the object and the floor, so that the KinFu algorithm lost track of its orientation (as explained in the conclusions on the Kinect Fusion algorithm).

This problem has been reduced by placing objects around the scene as points of reference and by making sure that the different features of the object would be seen by the camera, removing the illusion of radial symmetry.

However, the symmetric properties of the object, even on the horizontal plane, presented additional problems during manipulation of the point clouds. Alignment was hard to achieve and prone to error, due to the SAC-IA algorithms and ICP not realizing the correct orientation of the object in absence of the floor plane. For these reason, we switched to the kitchen machine we used in the example results.

Conclusions and future work

In this work we analyzed the Kinect Fusion algorithm for real-time surface reconstruction and exploited its results to develop a 3D scanner application.

The 3D scanner application can be used to reconstruct an object placed on the floor, provided it has no symmetry. The application could be useful for reconstructing 3D meshes to be used in virtual 3D environments, such as virtual museums or videogames. This is just an example of the applications that can be created by taking advantage of Kinect Fusion and, especially, of the PCL libraries.

Improvements can be made to the algorithm, such as by finding a method to reconstruct objects with symmetries (possibly through a better feature extraction procedure), by taking an indefinite number of clouds as input, or by automating the definition of the parameters for some procedures, such as the crop-box filtering.