3D Scanning On the Cheap

A Senior Comprehensive Paper

presented to
the Faculty of Carleton College
Department of Computer Science
Jadrian Miles, Advisor

by

Alex Calamaro Yawen Chen Karen Halls Jonathan Liou Sam Watson

March, 2015

1. Introduction

3D scanning is an important emerging technology, but there are not many affordable or practical choices on the market yet. The technical goal of this project is to examine a variety of theories related to 3D object reconstruction and implement 3D scanning using affordable hardware under the guidance of related theory. We present a framework for 3D object reconstruction in which the user moves a hand-held scanner around the object of interest, obtaining color and depth "snapshots" from various perspectives. The rigid transformations needed to align these snapshots into a single, cohesive underlying model are calculated and the transformed images are combined to reconstruct the object.

In this work, we present three separate approaches to 3D object reconstruction. First, we present a simplistic naïve approach where different perspectives of an object are aligned according to a known rotation. We also present two other more sophisticated implementations where the rigid transformations needed for alignment are unknown. We go on to explore the factors that contribute to the overall accuracy of alignment.

Through this project, we realize that 3D alignment remains a challenging technical issue despite the fact that the field of computer vision has advanced tremendously in recent years. Therefore, we also propose some potential work that might help improve the alignment accuracy through combining information obtained through various technologies.

2. Theory and Background

2.1 Motivation

Three-dimensional scanning has conceptually existed for quite some time (many of the algorithms we reference were developed in the 1990s.) However, it has largely remained in theory since then, with the exception of a few notable industry examples such as rapid prototyping. The consumer space, on the other hand, has been nearly void of 3D-scanning tools, and those which do exist can be prohibitively expensive for a hobbyist or casual user. This is in sharp contrast with the recent increase in availability and affordability of virtual-reality and augmented-reality related hardware. These concepts are all intertwined in that they provide new ways of interfacing the physical and virtual worlds, and as they proliferate the consumer space and become staples of modern computing, it will be increasingly important that individuals are able to represent their environments in digital space. Thus, we chose to examine some of the cheapest available 3D-scanning hardware to explore how powerful the low-end market offerings can be.

2.2 Point Clouds

For the purpose of three-dimensional scanning, a point cloud is a set of x, y, z coordinate points in three-dimensional space that represents the external surface of an object. Most three-dimensional scanners use a depth sensor to capture data as point clouds by emitting an array of infrared beams which intersect and reflect off encountered surfaces at varying points; these points are then assigned the appropriate width, height, and depth coordinates at the point of contact. Given the numerous infrared beams emitted by the depth sensor, a cohesive topological representation of the space can be represented by these points, forming a point cloud. A surface represented by a point cloud can be colored by assigning each point an RGB value captured by the color camera; each point is represented by a six-dimensional vector $< x, y, z, r, g, b >$.[9]

2.3 Rigid Transformation

One of the challenges of generating a 3D model from multiple perspectives of the same object is to determine how positions of pairs of points transform from one perspective to another. The type of transformation involved in the above process is able to map each point $(x_1.x_2, x_3)$ to its corresponding position in another space (y_1, y_2, y_3) without violating the distance and angel between pairs of points.

This operation is called a Rigid transformation, which preserves distances between pairs of points and angles between triplets of points. In section 2.2, we introduce the system of representing a point p in any given point cloud with six coordinates: x, y, z, r, g, b. Specifically, x, y, z refers to the three coordinates in terms of position and r, g, b corresponds to color coordinates. While color information might assist calculating rigid transformation between pairs of points by providing valuable information on colors,it is not taken into consideration in our approach.

Rigid body transformation consist of only rotations and translations,which can be expressed using the following formula:

$$T(v) = Rv + t \tag{2.1}$$

where v is a vector representing a point, the 3×3 rotation matrix R determines the rotation, and a three dimensional vector t determines the translation. Given a series of point clouds representing the same object, the 3D reconstruction of the object requires aligning pairs of points appropriately, which is to find transformation - both rotation and translation - between pairs of points that correspond to each other. Here we will illustrate how to apply translation and rotation on a point separately. To translate a point p by t

units, we denote the position of p after the translation as q, from equation (2.1) we have:

$$q = p + t \tag{2.2}$$

which is alternatively:

$$\begin{pmatrix} q_x \\ q_y \\ q_z \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} = \begin{pmatrix} p_x + t_1 \\ p_y + t_2 \\ p_z + t_3 \end{pmatrix} \tag{2.3}$$

Understanding how rotations work in three dimensional space is slightly more complicated, as an object can rotate in three orthogonal planes along the axes. Let's denote the three axes as $x-$axis, $y-$axis and $z-$axis. The rotation matrix R_x representing the rotation of θ radians counterclockwise for a point about $x-$axis is:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{bmatrix} \tag{2.4}$$

Similarly, the rotation matrix R_y representing the rotation of α radians counterclockwise for a point about $y-$axis is:

$$\begin{bmatrix} cos(\alpha) & 0 & sin(\alpha) \\ 0 & 1 & 0 \\ -sin(\alpha) & 0 & cos(\alpha) \end{bmatrix} \tag{2.5}$$

Similarly, the rotation matrix R_z representing the rotation of β radians counterclockwise for a point about $z-$axis is:

$$\begin{bmatrix} cos(\beta) & -sin(\beta) & 0 \\ sin(\beta) & cos(\beta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

The rotation matrix R in equation (2.1) combines rotations of a point in both three axes, and thus can be computed by multiplying the three matrices in the appropriate order which is further determined by the specific order of rotations.

Based on the discussion above, we can apply the equation for rigid transformation to any point P in a given point cloud A and get its corresponding x, y, z position in another point cloud B if the rotation matrix and translation vector are both known. This is exactly how we are able to align neighboring point clouds in our naïve Approach, which will be discussed in detail section 3.2

2.4 Singular Value Decomposition

While it is relatively straightforward to perform rigid transformation to a given point, finding a consistent pair-wise rotation matrix and translation vector from any point in a set of points to it corresponding point in another set of points is challenging and complicated. Specifically, it is not easy to tell which points should be pairs. As a result, we aim to find pair-wise transformation that minimize the sum of Euclidean square distance between all pairs under such transformation. We denote the sum of pair-wise Euclidean square distance as the Least Square Error E^2, and from definition we know that:

$$E^2 = \sum_{i=1}^{n} ||(Rq_i + t) - p_i||^2 \tag{2.7}$$

where $Rq_i + t$ is the position of point Q after the pair-wise rigid transformation from point in point cloud Q to its paired point in point cloud P. Since p_i and q_i are the pairs under such transformation, meaning that they are regarded as the same point in two coordinate systems, the Euclidean Square distance between the position

of P after applying the rigid transformation and Q should ideally be zero, but in practice may be small. That is why we want the sum of all such Euclidean Squared distance difference: $(Rq_i+t)-p_i)^2$ to be minimized.

To find such rotation matrix R and and translation vector t that produce pair-wise transformation between two sets of points P and Q with Least Square Error, Arun et al. (1987) proposed a method that computes the rotation matrix through finding the singular value decomposition (SVD) of a covariance matrix derived from the standard representation of rigid transformation.[8] We will briefly examine the concept of SVD before introducing this method in detail.

Any matrix has a SVD. The SVD of a $m \times n$ matrix A is:

$$A_{m\times n} = U_{m\times r}\Sigma_{r\times r}V^T_{r\times n} \tag{2.8}$$

$$= \left(\begin{array}{c|c|c} & & \\ u_0 & ... & u_{r-1} \\ & & \end{array}\right) \begin{pmatrix} \sigma_0 & & \\ & ... & \\ & & \sigma_{r-1} \end{pmatrix} \begin{pmatrix} v_0{}^T \\ . \\ . \\ . \\ v^T_{r-1} \end{pmatrix} \tag{2.9}$$

where r is the rank of A, which is equal or smaller than $min(m, n)$, depending on if some of the columns in matrix A are linearly independent. The SVD of a matrix is deterministic, and the matrices U and V are orthonormal, meaning that $U^TU = I$ and that $V_TV = I$. The matrix Σ is a diagonal matrix with singular values σ_i as its entry for each row. These singular values are all non-negative and are ordered in increasing order in the SVD:

$$\sigma_0 \leq \sigma_1 \leq \sigma_2 \leq ... \leq \sigma_{r-1} \tag{2.10}$$

The increasing order of singular values ensures the uniqueness of decomposition and determines the order of u_i and v_i in the matrices U and V^T.

To compute a rotation matrix with SVD, we first identify the centroids of the two set of points as a corresponding pair for rigid transformation. Such choice is based on the assumption that the centroids of two sets of points representing the same object and scene should approximately correspond to each other in the two coordinates.The centroid of P and Q, each containing n points, are defined to be $\overline{p}$ and $\overline{q}$ respectively in the following way:

$$\overline{p} = \frac{1}{n}\sum_{i=1}^{n} p_i$$

$$\overline{q} = \frac{1}{n}\sum_{i=1}^{n} q_i$$

Each point p_i in P can then be translated to $p'_i = p_i - \overline{p}$, and similarly each point q_i in Q can be translated to $q'_i = q_i - \overline{q}$. Applying the above translations eliminates the effect of pair-wise translation between the two sets of points. If we denote the $3 \times n$ matrix consists of all vectors p'_i as P^{prime} and the $3 \times n$ matrix consists of all q'_i as Q^{prime}, we can compute the covariance matrix H Arun et al. (1987) proposed as:

$$H = Q'P'^T \tag{2.11}$$

Since Q' and P' are both $3 \times n$ matrix, H is a 3×3 matrix. Arun et al. (1987) found that the singular decomposition the covariance matrix H: USV^T can be used to compute the rotation matrix R that minimize the Euclidean Square Errors described in equation (2.7). Specifically:

$$R = Udiag(1, 1, det(UV^T))V^T \tag{2.12}$$

[11]

2.5 Iterative Closest Point

Iterative Closest Point (ICP) is a commonly used algorithm for point cloud alignment. The classic implementation of ICP works by iteratively finding corresponding points between two point clouds and estimating the rigid transformation necessary to minimize the overall distance between these point pairs [4]. Specifically, in order to align two point clouds A and B, ICP works in the following way:

1. For each point A_i in A, find the point B_j in B which has the nearest Euclidean distance to A_i. Associate these points as a pair.
2. Estimate the rigid transformation T which minimizes the overall distance between point pairs.
3. Apply T to A.
4. Repeat steps 1-3 until T is sufficiently small.

Since the original proposal of ICP there have been numerous extensions and variations to the algorithm. One such improvement is to incorporate the concept of surface normals in order to more accurately align corresponding regions in clouds. This approach, which we will refer to as Iterative Closest Point with Normals (ICPN) (see section 3.4), differs from the traditional implementation in the following way: given two corresponding points $p \in A$ and $q \in B$, compute the distance from a surface normal at point p to its corresponding tangent plane at q, and attempt to iteratively minimize the sum of these distances [5] [3]. In practice this tends to provide better alignment, because surface normals can describe surface geometry in richer detail than points alone. It also removes a degree of freedom from the barebones ICP algorithm, which accelerates its convergence.

2.6 Downsampling

Downsampling is used to decrease the number of points in a data set while still preserving the object and scene it represents. Three major motivations contribute to applying downsampling method in our project:

1. To decrease the influence of noise: depth sensor are subject to noise, which leads to errors in regards to depth, boundaries and surface representations and increases pixel mismatch.
2. To reduce computation time: it is computationally expensive to find optimal pair-wise rigid transformation if there exist too many points to pair.The point clouds we acquired through the camera easily contain hundreds of thousands of points.
3. To create a uniform density on the point cloud: the alignment should not be dominated completely by dense part of the cloud, and downsampling can decrease the number of points in each area the point cloud represents to a similar one.

Conceptually, we achieve the goal of downsampling by dividing the 3D point cloud into small 3D cubes of the same size and replace the points within each cube with their centroid. This method is named Voxel Grid filter[7],and the length of each voxel is called leaf size and is usually relatively small in comparison to the size of the object we scan, in order to preserve the integrity. Since all points within a voxel is replaced by their centroid, and the voxel is uniformly distributed across the point cloud, we are able to obtain a relatively uniform point cloud containing less points but represents the same object.

2.7 Pipeline Approach

One possible approach for aligning point clouds is what we refer to as the Pipeline Approach.[1] Each step that is needed in order to calculate a rigid transformation is done by a different algorithm. While our attention was ultimately focused on ICP and its variants, this approach also holds promise for cloud-pair alignment.

1. **Keypoints:** Keypoints are unique points within a point cloud. Keypoints should be sparse, since every point within a point cloud should not be a keypoint.[6] A good key point will be:

- Distinctive: A keypoint that is distinctive is a point that is significantly different from other points in its own point cloud. More specifically, good distinctive keypoints are corners or tips of an object, but edges in general are acceptable.[6]
- Repeatable: A keypoint that is repeatable is a point that can be seen in multiple point clouds.[6] In other words, it is a point in a scene that can be seen from multiple perspectives

These keypoints are then fed into the algorithm that calculates feature descriptors.

2. **Feature Descriptors:** Once we have the keypoints, we need to figure out their feature descriptors. Feature descriptors are extra information that can be collected about the point, most notabl. Surface curvature information is calculated by looking at the point's neighboring points.[10]

3. **RANSAC:** To align separate point clouds with one another, a set of inlying (as opposed to outlying) features in both clouds must be found before they are transformed and overlayed. The Random Sample Consensus algorithm (or RANSAC) is used to eliminate outlying features by first randomly selecting a subset of features from one cloud, creating a hypothetical model from this subset (the consensus), then testing random samples of features of the other cloud against this consensus. This process is performed repeatedly until a set of inlying features is converged upon.[2]

3. Methods

3.1 Materials

For data collection, we used the Asus Xtion PRO LIVE. The Xtion PRO contains two sensors, a 1280 x 1024 resolution color camera and a separate 640 x 480 resolution depth camera. These data can be used to model real-world objects as point clouds. The relatively low resolution and noisiness of the depth camera make it difficult to gather fine-grained surface detail, particularly at ranges closer than one meter or further than 3 meters due to non-adjustable focal depth. Therefore, our implementation is geared towards scanning larger objects or environments, with less precision than most commercially available 3D laser scanners. In order to interface with the Xtion PRO and process point clouds, we used the Point Cloud Library (PCL). PCL is an open source library containing tools for point cloud collection, manipulation, and visualization. PCL is designed specifically to work in conjunction with devices similar to the Xtion PRO. We used PCL version 1.7.2 installed on a Macintosh computer running Windows 7. All our code was written in C++ with Microsoft Visual Studio 2010 as our development environment.

3.2 Naïve Approach

A naïve approach was taken as a proof of concept - naïve in the sense that no algorithms specific to point cloud manipulation or otherwise related to computer vision were utilized - at this stage. Instead, a rather elementary method was devised to reconstruct a three-dimensional surface, in which we had complete quantitative control over all variables in our setup. To do so, we required the object in question to be placed on a turntable, and to have each successive scan be performed at rotations of equally defined increments. As such, aligning each captured point cloud of the object simply required rotating subsequent point clouds by a known degree to fit the orientation of the first point cloud, and this known degree is the increment of rotation multiplied by the number of scans subsequent to the first scan.

3.3 Dynamic Alignment

The ultimate goal for our implementation is to combine multiple views of an object or scene into a single underlying 3D model accurately describing the surface geometry and coloration. This is accomplished by having the user move the camera around the object of interest and taking a series of point cloud "snapshots" in rapid succession. Next, the rigid transformations needed to align these snapshots are estimated, and the transformed clouds are combined into a single model representing the scanned object. Specifically, we take a pairwise alignment approach as follows:

1. For each point cloud P_i in a sequence of clouds $P_1 \dots P_n$, estimate the rigid transformation T_i needed to align P_i to P_{i-1}
2. Add T_i to a cumulative transformation matrix T, which contains the combined rigid transformations $T_{2 \dots i-1}$
3. Apply T_i to P_i
4. Concatenate P_i to a point cloud representing the cumulative model M, which already contains each transformed point cloud $P_1 \dots P_{i-1}$

A more intuitive approach would be to simply align each point cloud directly into the cumulative model. That is, align each P_i directly into M, which already contains the previously aligned clouds $P_{1 \dots i-1}$. The pairwise approach poses several advantages over this framework. First, it is algorithmically easier to align two clouds in similar poses than two clouds in disparate poses. As i grows larger, the difference in poses

between P_i and P_{i-1} is much less than the difference between P_i and P_1. Since each point cloud is ultimately aligned to match the starting pose of P_1, it becomes increasingly difficult to align P_i to this starting point. Second, it is easier for most algorithms to align clouds of similar sizes. Since M is the accumulation of all point clouds in the sequence, M contains approximately i times more points than P_i. This creates a greater opportunity for alignment error. Thus, it is more effective to align clouds in a pairwise fashion and build a model by applying accumulated transformations to each cloud.

3.4 Revised Approach with ICPN

We decided to pursue an approach using ICPN instead of the Pipeline Approach. We thought focusing on one algorithm would make our project more successful. We were inspired by the Pipeline Approach to incorporate downsampling and normals.

1. **Apply Cropping Box:** Once we have all of the point clouds, we apply a cropping box to them. This removes all of the points outside of a certain distance from the camera, and keeps the points within the specified distance. This reduces noise and keeps the focus on the object(s) of importance.

2. **Downsample Point Clouds:** Once the point clouds have a cropping box applied to them, we downsample the first two point clouds (see section 2.7).

3. **Calculate Normals:** Calculate the normals of each downsampled point within the point clouds. Once these are calculated, we can run ICPN on the pair of point clouds.

4. **ICPN:** ICPN runs on these two point clouds and approximates the rigid transformation between the two point clouds.

We apply the rigid transformation that ICPN gives us to the point cloud that was getting aligned. Then, the cumulative transformation gets applied to the point cloud as well, so it is oriented in the same way as the rest of the 3D model we are building up. Therefore, the pose of the point cloud is updated. This point cloud then gets added to the 3D model we are accumulating, and the cumulative transformation gets updated. Steps 2 through 4 are repeated until we have gone through all of the point clouds.

4. Results

Figure 4.1 is an image of our successful scan of a boot, which came from our naïve approach. Figures 4.2 and 4.3 show scans from our revised approach. In the next section we will evaluate why these scans were or were not successful.

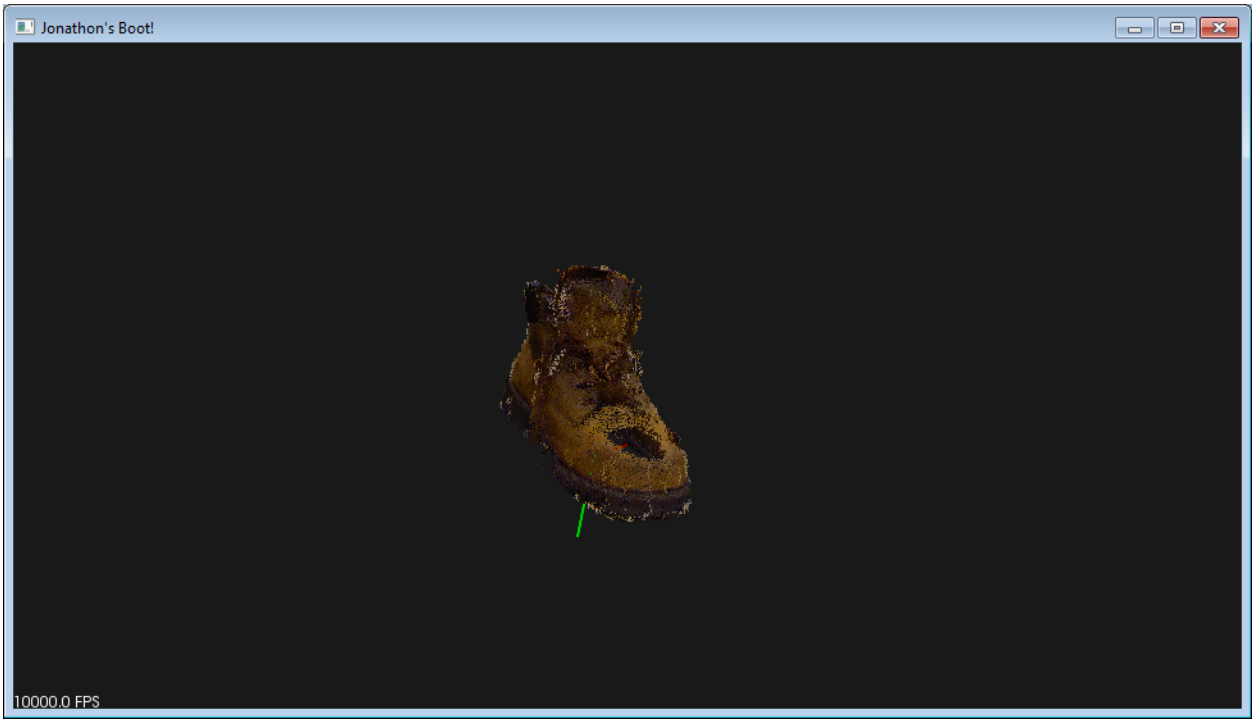

Figure 4.1: Scan from naïve Approach

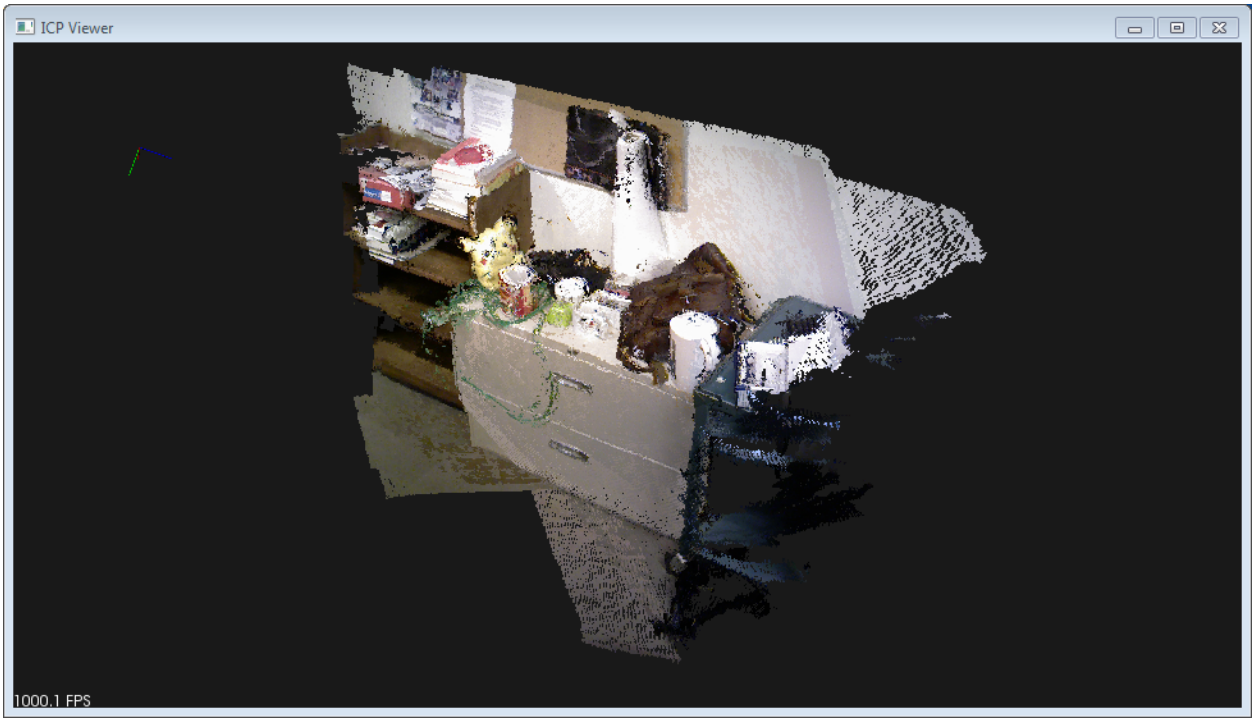

Figure 4.2: Scan from Revised Approach

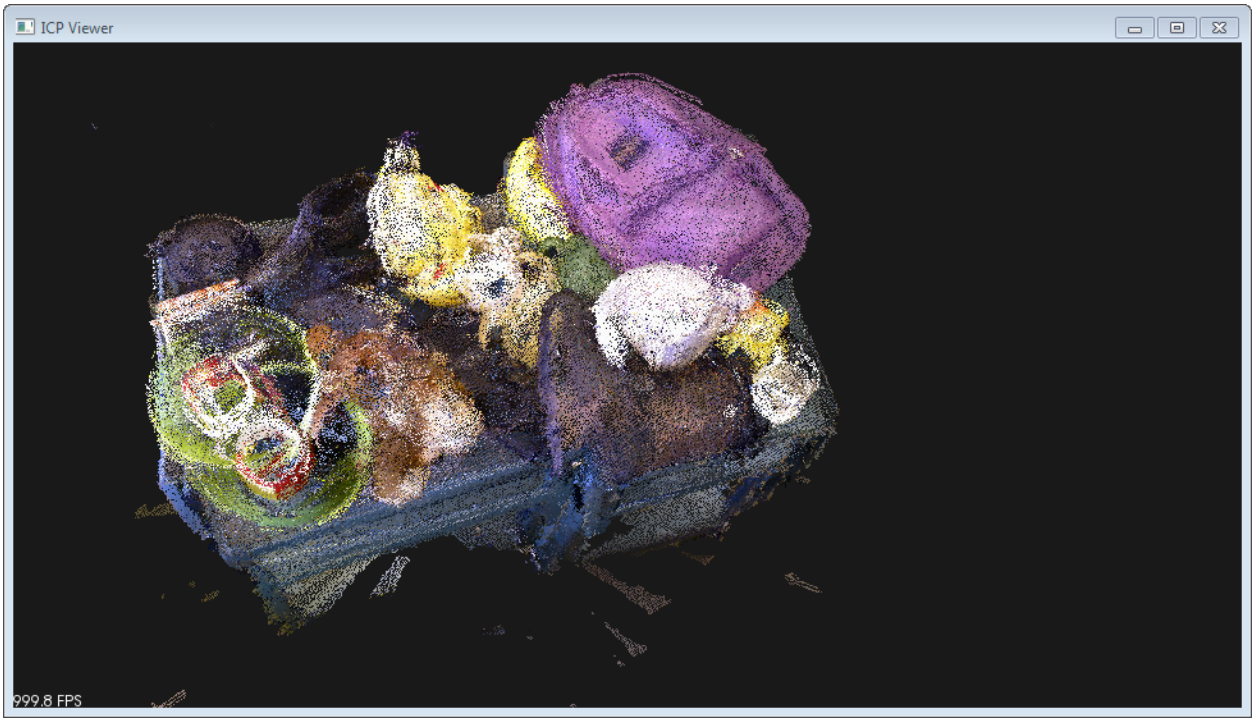

Figure 4.3: Scan from Revised Approach

5. Evaluation

5.1 Determining the Quality of a Scan

In developing our process, we depended largely on subjective analysis of the quality of our scans to make decisions about what variables were well calibrated. While ICP, as implemented by PCL, does have a fitness metric, it is based on the sum of euclidean distances between estimated point pairs and is therefore unreliable. Ostensibly, should a perfect evaluation metric exist, it would be used in the actual alignment process to find a globally optimal solution, so without such an algorithm, we are left to use our best judgment. To illustrate our conclusions about parameter choice, we have included several comparison-scans of which demonstrate the impact of key variables.

6. Analysis

6.1 Scan Overlap

Since the conception of ICP, it has been understood that it fares poorly with highly disparate scans [5]. In general terms, this can be distilled to the difference in angle between each scan. For example, our "cluttered desk" scan uses 13 scans over a 180°range. Thus, each scan differs by approximately 14°. In our testing, we found that approximately 15°intervals were optimal. Greater angles quickly caused a deterioration of scan quality, while lesser angles provided no discernible improvement
Figures 6.1 and 6.2 illustrate the deterioration in alignment quality when the angle between each capture point cloud is higher. Notice the second handle on the mug–a highly undesirable artifact.

6.2 Tuning Parameters

Simply by changing some parameters, we are able to significantly improve our results.

- **Apply cropping box:** By applying a cropping box were able to improve alignment. The cropping box gets rid of extraneous points, which in turn gets rid of noise. Figure 6.3 shows a poor alignment between two point clouds when there is no cropping box applied to the point clouds. Figure 6.4 shows a much more successful alignment between the same two point clouds - the only difference is that there is a cropping box applied to them. Again, the alignment on the point clouds with the cropping box is successful because we have gotten rid of more noise.
- **Adjust amount of downsampling:** By adjusting the leaf size, we can change the amount of downsampling that occurs on the point clouds. If there is not enough downsampling, then there is too much noise, and it is harder to align the point clouds, which leads to a poor alignment (Figure 6.5). So, by increasing the leaf size, which increases the amount of downsampling, we are able to get a much more accurate alignment, as shown in Figure 6.6.
- **Change what is scanned:** We also found that by simply changing what we scanned, we could improve our results. A scene or object with more faces facing multiple directions is more likely to have a good alignment than a scene or object that does not. This can be shown in Figures 6.7, 6.8 and 6.9. Figures 6.7 and 6.8 are scans of a cart with a floor and some counter. There are not many faces pointing different directions, so it is harder to distinguish the different points from each other. This leads to a very poor alignment. Figure 6.7 has the same cropping box and leaf size as Figure 6.9. And Figure 6.8 has about the same number of points in its clouds after they have been downsampled as Figure 6.9. In Figure 6.9, there are many more unique points, so it is easier to distinguish points from each other. This leads to a much better alignment. This conclusion can be made because we know that we want the normals to be different for the different downsampled points. In Figures 6.7 and 6.8, the normals are all facing the same direction, so it is hard to distinguish them from each other. Whereas with Figure 6.9, the normals will be pointing in all different directions, so it is easier to match them up.

A notable aspect of ICP in general is that there is no "magic formula" for a good alignment. Variable selection depends on the object you wish to scan as well as the environment which surrounds it. Our scans were all taken in the same environment with similar objects, thus we found that the same set of variables to be optimal for most of our scenes.

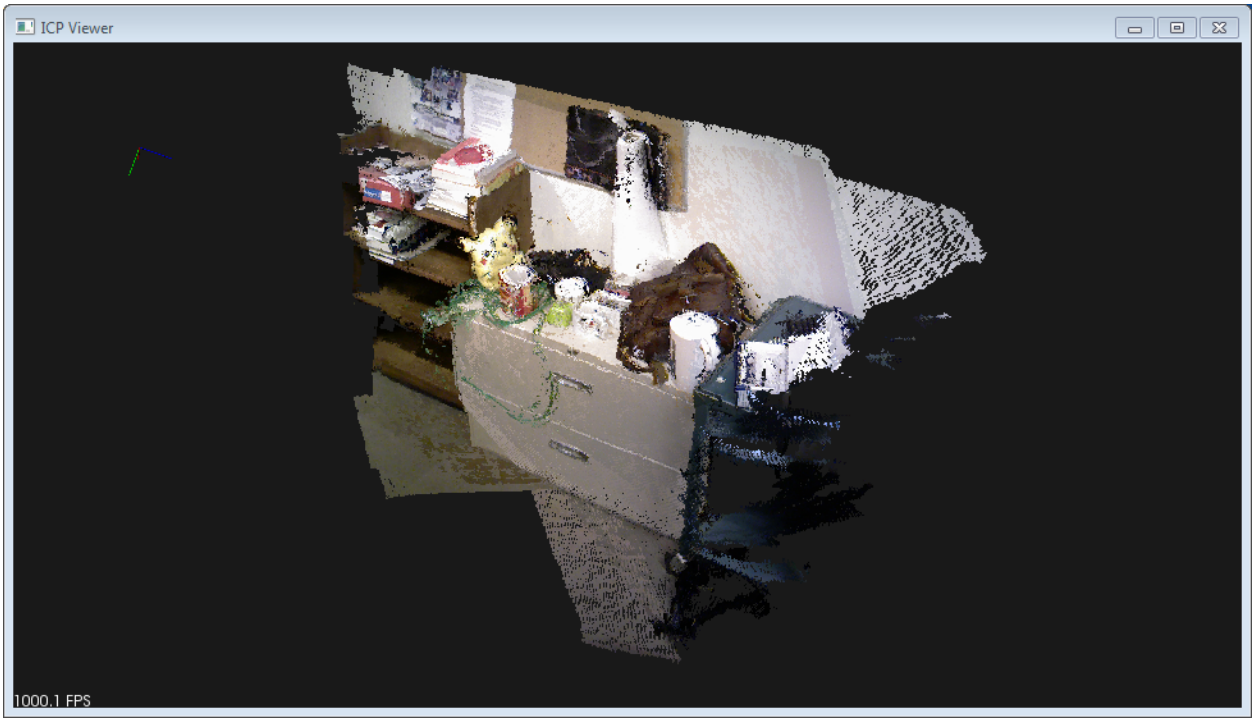

Figure 6.1: Scans at 14°interval

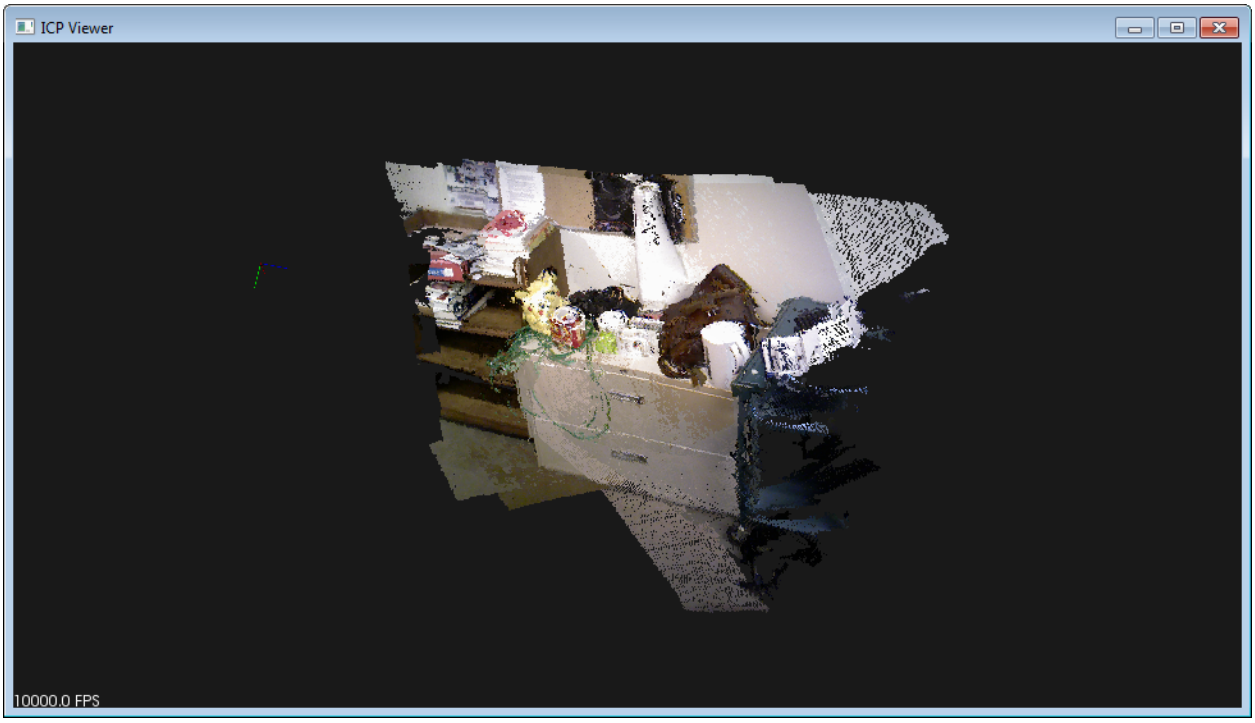

Figure 6.2: Scans at 30°interval

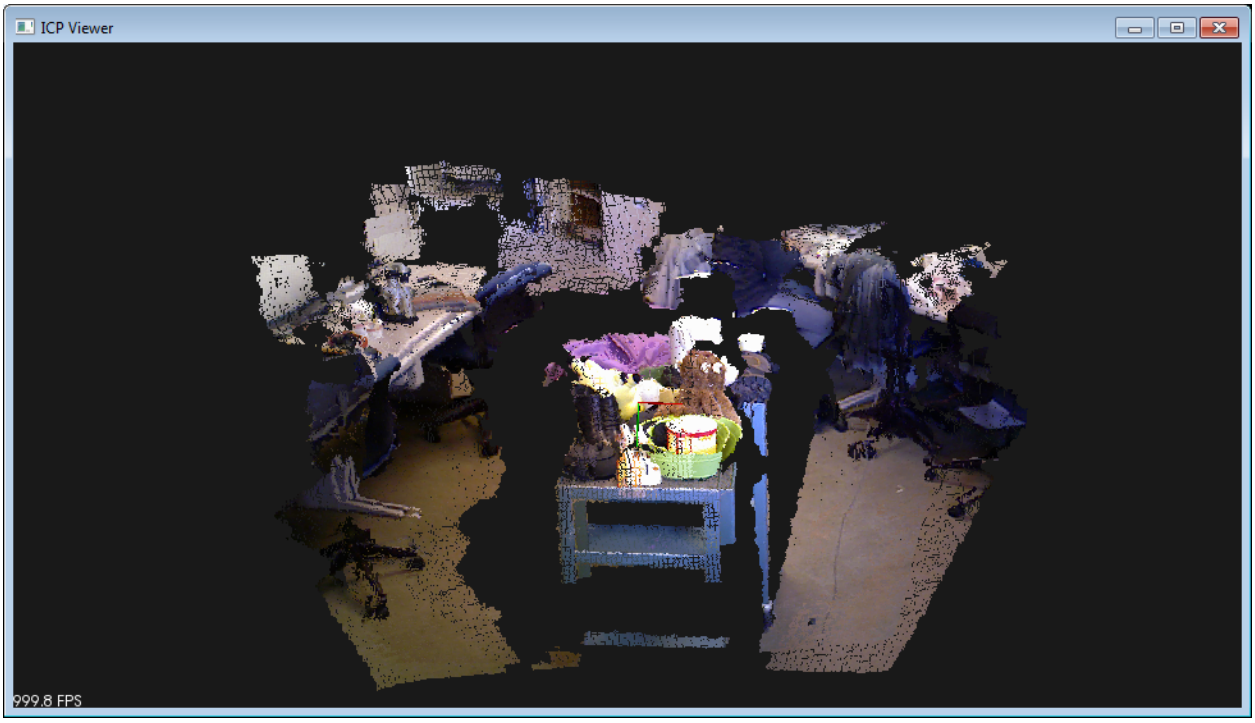

Figure 6.3: Alignment of 2 Point Clouds without Cropping Box

Figure 6.4: Alignment of 2 Point Clouds with Cropping Box

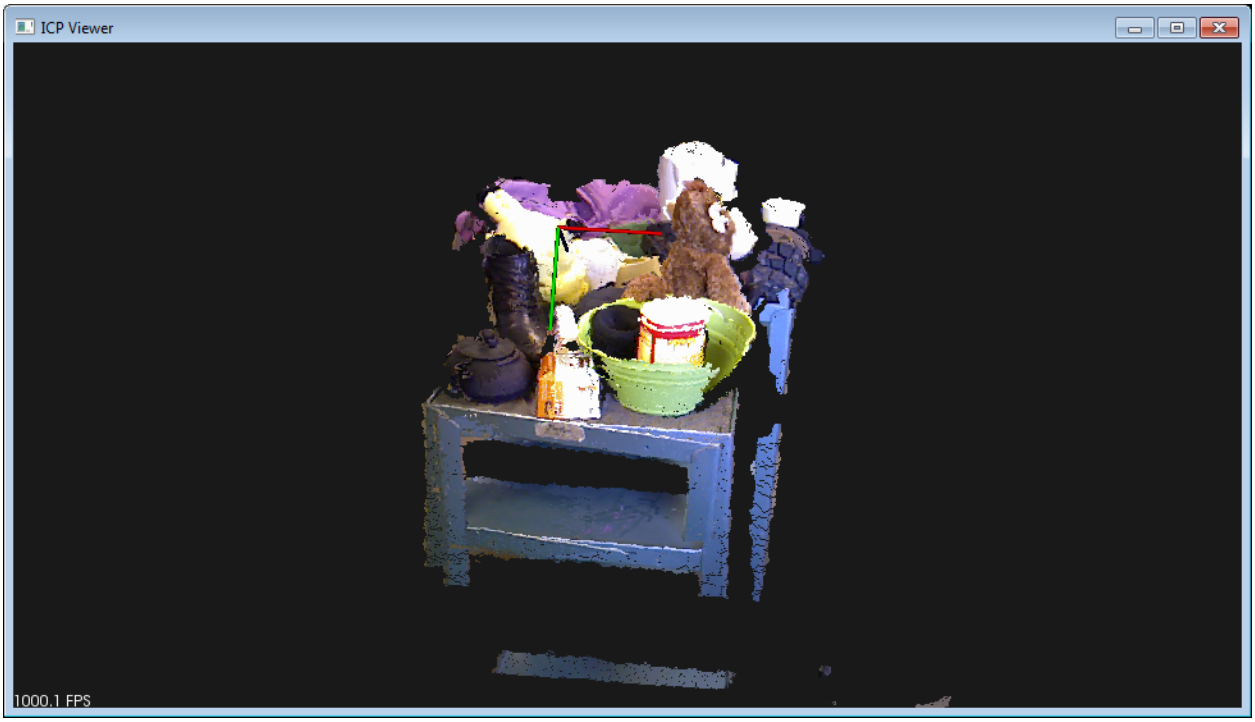

Figure 6.5: Alignment of 2 Point Clouds with Leaf Size of .0015

Figure 6.6: Alignment of 2 Point Clouds with Leaf Size of .005

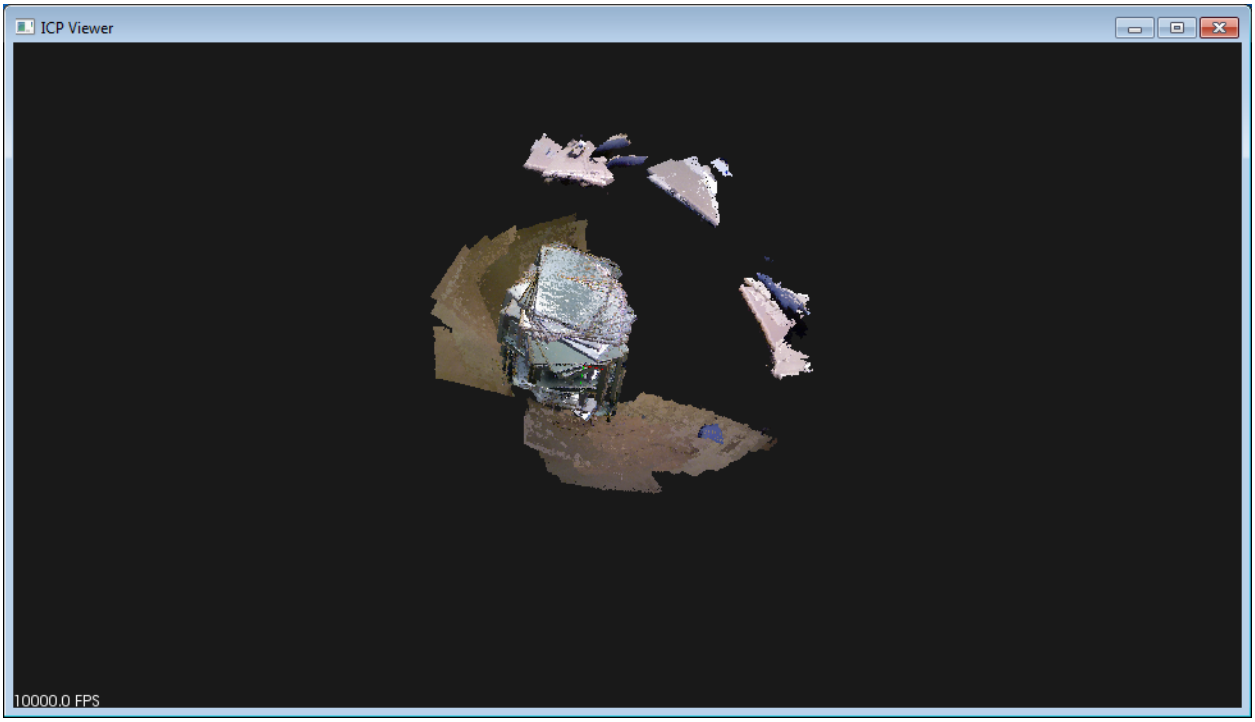

Figure 6.7: Cart With Same Bounding Box and Leaf Size

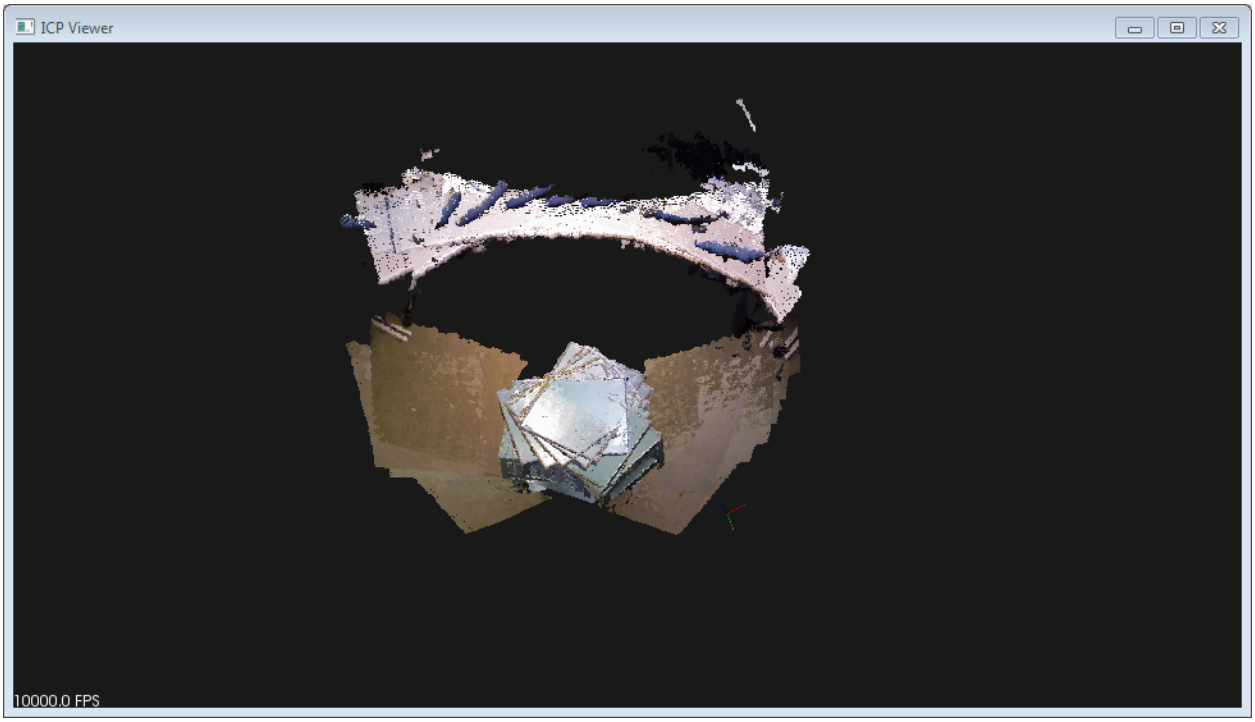

Figure 6.8: Cart With Same Number of Points as Figure 6.9

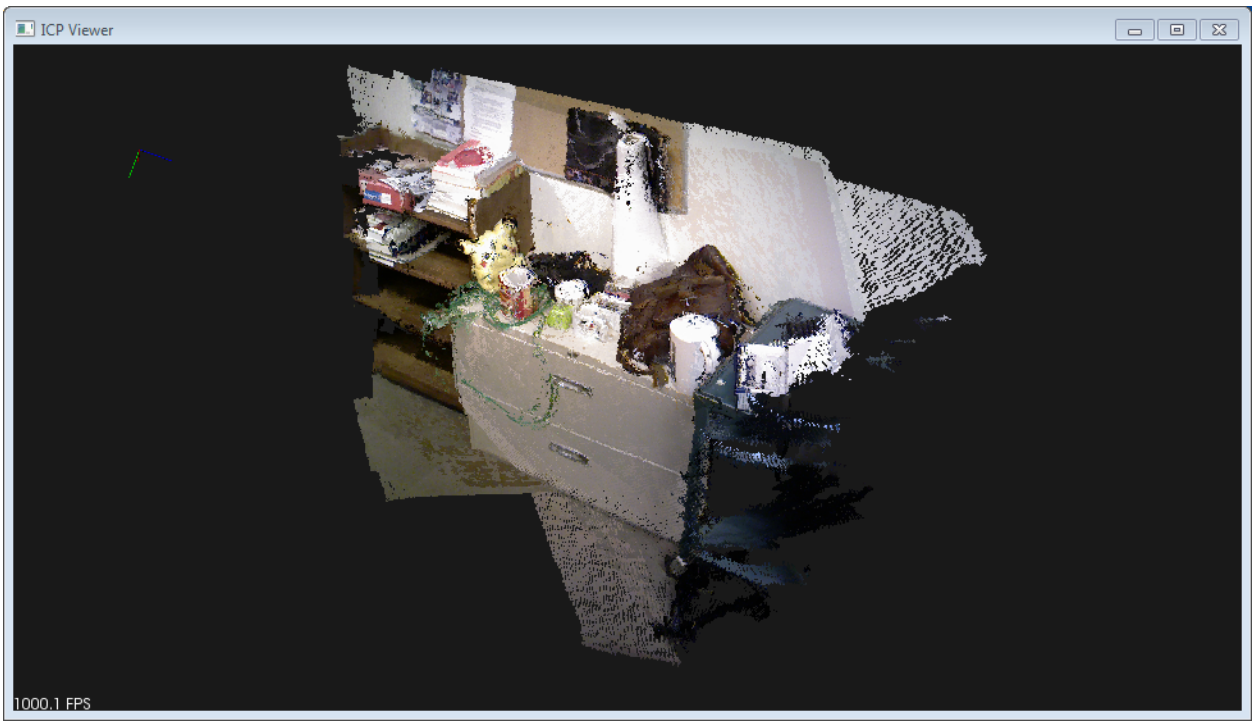

Figure 6.9: Successful Alignment

7. Conclusion

7.1 In Summary

Observing the improvement from our naïve implementation to our adaptation of the Iterative Closest Point algorithm, we can conclude that estimating the transformation of a scene based on the initial position is quite powerful if well constructed, and can be fine-tuned with the tools provided by the Point Cloud Library. Transformation estimation greatly reduced poor alignments in our implementation of ICP, and as such, we can consider improving more complex pipelines by performing this estimation as preprocessing, especially in the case of aligning using RANSAC, which was unsuccessful in our first attempts without initial estimation.

7.2 Future Work

From this conclusion, we can make our current implementation more robust by using another algorithm to find an initial pose estimation and then refining the result with ICP. To further refine our 3D reconstruction, we can incorporate additional algorithms, such as general iterative closest point with color, considering color information when aligning with ICP as well, effectively assessing six dimensions of data, x, y, z, r, g, b for a tighter alignment. We can also revisit using RANSAC as mentioned above, first using estimation to improve our results. Alternatively on the hardware side, we could consider integrating accelerometer and gyroscope information to apply even more accurate transformations to subsequently capture point clouds during alignment. This has the potential to improve the speed and precision of our method as fewer iterations of alignment are needed, and makes environment scanning more palatable.

Bibliography

[1] The pcl registration api. http://pointclouds.org/documentation/tutorials/registration_api.php#registration-api.

[2] Random sample consensus. http://pointclouds.org/documentation/tutorials/random_sample_consensus.php.

[3] Dirk Haehnel Aleksandr Segal and Sebastian Thrun. Generalized-icp. 2009. http://www.robots.ox.ac.uk/~avsegal/resources/papers/Generalized_ICP.pdf.

[4] McKay Neil D. Besl, Paul J. Method for registration of 3-d shapes. 1992. http://www.cvip.uofl.edu/wwwcvip/education/BMEECE643/download/PAMI1992.pdf.

[5] Yang Chen and Gerard Medioni. Object modeling by registration of multiple range images. 1991. http://www.math.zju.edu.cn/cagd/seminar/2007_autumnwinter/2007_autumn_master_liuyu_ref_2.pdf.

[6] Jeff Delmerico. Pcl tutorial: The point cloud library by example, 2013. http://www.cse.buffalo.edu/~jryde/cse673/files/pcl_tutorial.pdf.

[7] Wanliang Wang Xianping Huang Fengjun Hu, Yanwei Zhao. Discrete point cloud filtering and searching based on vgso algorithm.

[8] S.D. Blostein K.S. Arun, T.S. Huang. Least-squares fitting of two 3-d point sets, 1987.

[9] RenderMan. Point clouds. http://renderman.pixar.com/view/point-clouds.

[10] Radu Bogdan Rusu. *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, Computer Science department, Technische Universitaet Muenchen, Germany, October 2009. http://files.rbrusu.com/publications/RusuPhDThesis.pdf.

[11] Inge Soderkvist. Using svd for some fitting problems.