UNIVERSITY OF CALIFORNIA

SANTA CRUZ

Face Tracking using Commodity Depth Cameras

A project submitted in partial satisfaction

of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Sasidharan Mahalingam

June 2019

The Project of Your Frist Name Last Name
is approved:

_____

Roberto Manduchi, Chair

_____

James Davis

Face Tracking using commodity depth cameras

Sasidharan Mahalingam
samahali@ucsc.edu

ABSTRACT

The aim of this project is to construct a method to track a user's face using commodity depth cameras. A precise tracking algorithm allows a person to easily generate a high quality 3D model of one's face. It also can be used to get the pose of a person's head with respect to the starting frame of reference. A main obstacle that prevents one to write a generic tracking algorithm is that different commodity depth cameras have their own SDKs. Developing a generic program that works on almost every camera is challenging. This project aims to solve the above problems by developing an algorithm that finds the pose of a person's head from RGBD video using an open-source framework that can be ported to work on all widely used commodity depth cameras.

1 MOTIVATION AND OBJECTIVE

Due to the availability of low cost consumer depth cameras, there has been a lot of research done using 3D image data. These cameras allow easy recording and analysis of the 3D world. With the recent advancements in convolutional neural networks and processing capabilities of computers, the size of having to deal with 3D datasets is no longer a bottleneck. However the number of well labelled 3D data-sets are limited. In the case of 3D pose estimation, there are only two well-labelled reliable datasets available, namely, the BIWI Kinect Dataset[30] and Nvidia's SynHead dataset[31]. Also these datasets suffer from drawbacks, the BIWI Kinect Dataset[30] has missing frames, so using this as a dataset for algorithms that need sequential data is ruled out. The other dataset, Nvidia's SynHead[31], suffers from the fact that it is not real-world data. Although well modelled it cannot perfectly replace real-world data.

My project is to implement an accurate face tracking algorithm, using an open source framework named Point Cloud Library (PCL)[29]. This will allow the program to be executed with compile time options to target it towards a particular commodity depth camera. The most widely used depth cameras are Intel's RealSense cameras and Microsot's Kinect cameras. PCL[29] has compile time flags to work with both of the cameras. I use a Hough Voting Mechanism for improved initialization [3], Generalized ICP[2] and an added check to avoid the drift between previous frames from cascading on to the later frames.

2 RELATED WORK

Various methods and algorithms have been devised to find the head pose of the person. These methods have been divided into three different categories: using features, using pose specific classifiers and by registration to previous frames or 3D head models. Yi Sun and Lijun Yin proposed a method to find the face pose by first finding the corners of the eyes and then the tip of the nose by using curvature properties of the face and then estimate frontal view based on that [11], Michael D. Breitenstein used the nose orientation as an initial estimate to narrow down the pose estimates and then compared head range images rendered on different similar poses to arrive that the best pose estimation [12]. Papazov et al., introduce a triangular surface patch (TSP) descriptor to match facial point clouds to a gallery of synthetic faces and to infer their pose [13]. Although these feature based approaches are are simple, efficient, they fail when the features cannot be detected, e.g. in the case of extreme rotations or partial occlusions. Coming to the classifier-based techniques, Seemann et al., propose a method where they detect faces in RGB images and estimate the head pose from the disparity map of a stereo camera using a neural network for each rotation [14]. Fanelli et al., train random classification and regression forests with range image patches for head detection and pose estimation [15]. Tulyakov et al., [16] use cascaded tree classifiers and achieve higher accuracies than [14]. Classifier-based techniques require extensive training with large datasets. Also, classifiers trained on one 3D sensor do not generalize well and the number of well labelled 3D data available also do not generalize well.

The other method widely used in pose estimation involves registering the 3D data to a 3D model using rigid or non-rigid ICP[3,5]. Various methods suggested employ a deformable model fitting to create person-specific models for head pose estimation. However, these methods do not work well when the quality of the observed data has significant noise, involve offline initialization and a lot of interaction from the users.

To register the reference model to the measured data, ICP and its variants are often used. However, ICP fails to converge to the correct solution when it is initialized poorly. To overcome this, Padeleris et al. employ the stochastic PSO algorithm [29] to register facial surfaces [17]. However, PSO also suffers from slow and/or premature convergence to a local optimum.

Face Detection is essential to face pose estimation, a lot of methods have been proposed to solve this problem [18, 19, 20, 21, 22, 23]. For my project, I assume that the captured frame has a properly segmented face.

3 MODELS AND ALGORITHMS

3.1 Data Collection

The data collected for this project was using an Intel RealSense Depth Camera (D435). PCL [29] has a RealSense Grabber routine that captures the x, y and z co-ordinates of each point in space. The captured data is organized in the form of a structured data structure named pcl::point cloud. Every individual entry in this container of type PointT. This makes it easy to run any algorithm on the point cloud as all the x, y and z co-ordinates of a point can be quickly accessed. Point clouds again can be organized or unorganized. organized point clouds have a structured ordering of the points captured. The RealSense D435 camera used provides a structured point cloud. Therefore the entire projects works with .pcd files and uses algorithms that relies on organized point clouds.

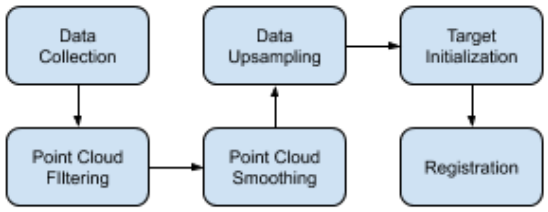

Figure 1: Registration Pipeline
Registration Pipeline

This ability to store and access organized point clouds is of extreme importance for real time applications like point cloud registration, virtual reality and robotics.

3.2 Point Cloud Filtering

The captured data has a lot of unwanted details of the environment and noise. In order to throw away all the unnecessary data, a volume of interest is defined. The near and far values of the x, y and z axis are determined. After filtering only the face region is left.

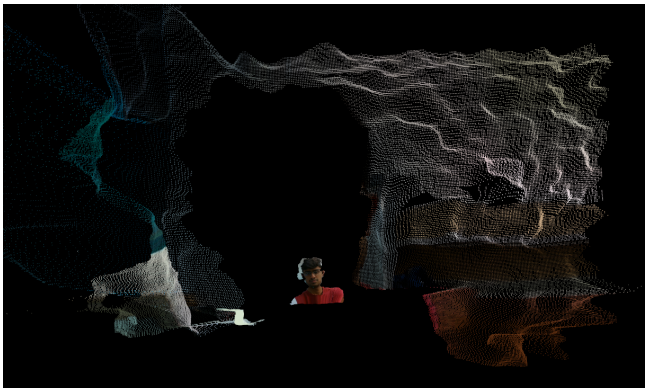

Figure 2: Captured point cloud
Captured point cloud

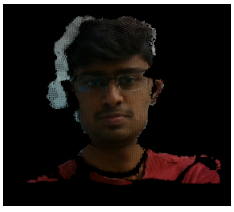

Figure 3: Filtered point cloud
Filtered point cloud

3.3 Point Cloud Smoothing

The filtered point cloud still has noise and a lot of sharp changes in contours. Since the registration uses an Iterative Closest Point (ICP) algorithm, having more grippable smooth surface ensures an easier convergence. In order to do this, we smooth the surface using Moving Least Squares Algorithm [24]. This gives us a smooth point cloud. However the obtained point cloud might contain a lot of holes in it.

3.4 Data Upsampling

In order to fill the holes in the captured point cloud, after smoothing the curves, Moving Least Square upsampling is employed, followed by Poission upsampling[25]. Poission upsamping provides a water tight fit to all the holes in the pipeline, thereby making it easier to converge to the right solution.

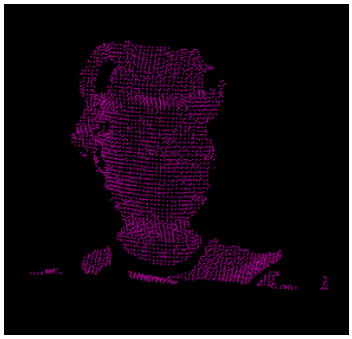

Figure 4: input point cloud

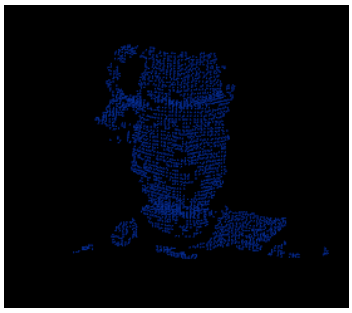

Figure 5: smoothed point cloud

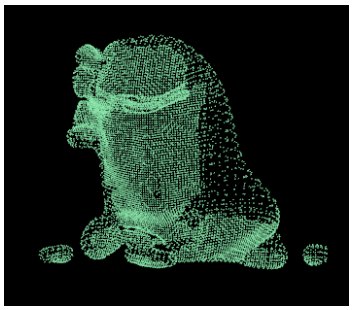

Figure 6: upsampled point cloud

3.5 Target Initialization and Registration

Problem Statement

A point cloud is a data structure P used to represent a collection of multi-dimensional points $p \epsilon P^n$. In a 3D point cloud, the elements usually contain the X, Y and the Z co-ordinates. Sometimes point clouds contains additional details like the local surface normal n or the curvature κ. Given a source cloud p ϵ P and a target point cloud q ϵ Q, the problem of registration relies on finding correspondences between P and Q, and estimating a transformation T that, when applied to P , aligns all pairs of corresponding points $p \epsilon P$, $q \epsilon Q$. One fundamental problem of registration is that these correspondences are usually not known and need to be determined by the registration algorithm. Given correct correspondences, there are different ways of computing the optimal transformation with respect to the used error metric.

Iterative Registration of Closest Points

The algorithm of ICP can be condensed to two steps:
1) Compute correspondences between the two point clouds
2) Compute a transformation that minimizes distance between corresponding points

Iteratively repeating these two steps typically results in convergence. Because we are violating the assumption of full overlap, a maximum matching threshold d_{max} is set. This threshold accounts for the fact that some points in the target point cloud will not have any match in the source point cloud and vice versa. The choice of d_{max} represents a tradeoff between convergence and accuracy.

Input:
Two point clouds: $P = p_i$ and $Q = q_i$
Starting Transformation T_0

Output:
Correct Transformation T, that aligns the two pointclouds P and Q.

$T = T_0$
while *not converged* **do**
  for $i = 1$ *to* N **do**
    m_i = FindClosestPointinP($T.q_i$)
    if $\|m_i - T.b_i\| \leq d_{max}$ **then**
      $w_i = 1$
    else
      $w_i = 0$
    end
  end
  T = $\arg\min_T \sum_i w_i ||T.b_i - m_i||^2$
end

Algorithm 1: ICP Algorithm

Point-to-plane

The point-to-plane variant of ICP improves performance by taking advantage of surface normal information. Originally introduced by Chen and Medioni[26], the technique has come into widespread use as a more robust and accurate variant of standard ICP when presented with 2.5D range data. Instead of minimizing $\sum \|(T.b_i - m_i)\|^2$, the point-to-plane algorithm minimizes error along the surface normal (i.e. the projection of $(Tb_i - m_i)$ onto the sub-space spanned by the surface normal)

$T = T_0$
while *not converged* **do**
  for $i = 1$ *to* N **do**
    m_i = FindClosestPointinP($T.b_i$)
    if $\|m_i - T.b_i\| \leq d_{max}$ **then**
      $w_i = 1$
    else
      $w_i = 0$
    end
  end
  T = $\arg\min_T \sum_i w_i ||\eta_i (T.b_i - m_i)||^2$
end

Algorithm 2: Point to Plane Algorithm

Generalized ICP

Generalized-ICP is based on attaching a probabilistic model to the minimization step on line 11, ($T = \arg\min_T \sum_i w_i ||\eta_i (T.b_i - m_i)||^2$) of Alg. 1. The technique keeps the rest of the algorithm unchanged so as to reduce complexity and maintain speed. Notably, correspondences are still computed with the standard Euclidean distance rather then a probabilistic measure. This is done to allow for the use of kd-trees in the look up of closest points and hence maintain the principle advantages of ICP over other fully probabilistic techniques, speed and simplicity.

In order to derive the update step of the Generalized ICP algorithm, let's start with the assumption that the closest point lookup has been performed on the two point clouds, A = $\hat{a_i}_{\{1,\dots N\}}$ and B = $\hat{b_i}_{\{1,\dots N\}}$ are indexed according to their correspondences. Assume that all correspondences with $||m_i - T.b_i|| > d_{max}$ have been removed. In the probabilistic model, assume that the existence of an underlying set of points, $\hat{A} = \hat{a_i}$ and $\hat{B} = \hat{b_i}$, which generate A and B according to $a_i \sim N(\hat{a}, C_i^A)$ and $b_i \sim N(\hat{b}, C_i^B)$. In this case, C_i^A and C_i^B are the covarience matrices associated with the measured points. If we assume perfect corresponding points and the correct transformation $T*$, we know that,

$$\hat{b_i} = \boldsymbol{T} * \hat{a_i} \tag{1}$$

For an arbitrary rigid transformation, $\boldsymbol{T}$, we define $d_i^{(\boldsymbol{T})} = b_i - \boldsymbol{T} a_i$, and consider the distribution from which d_i^{T*} is drawn. Since a_i and b_i are assumed to be drawn from independent Gaussians,

$$\begin{aligned} d_i^T &\sim N(\hat{b_i} - (\boldsymbol{T}*)\hat{a_i}, C_i^B + (\boldsymbol{T} * C_i^A (\boldsymbol{T}*)^T)) \\ d_i^T &= N(0, C_i^B + (\boldsymbol{T}*) C_i^A (\boldsymbol{T}*)^T) \end{aligned} \tag{2}$$

Now we use MLE to iteratively compute $\boldsymbol{T}$ by setting

$$\begin{aligned} \boldsymbol{T} &= \arg\max_T (\Pi_i p(d_i^T)) \\ \boldsymbol{T} &= \arg\max_T (\sum_i log(p(d_i^T)) \end{aligned} \tag{3}$$

3.6 Drift Correction

Initialization using Hough Voting

Although Generalized-ICP is fast and simple, it suffers from a problem that every existing variant of ICP suffers, i.e., bad initialization. In order to solve this problem, an initialization using SHOT3D features [27] is used. Using the Hough Voting scheme described in [10], we get a set of transformations. For every transformation that obtains a high score, the average and standard deviation of the distances from each point int the source point cloud to it's closest point in the target point cloud is calculated. Then the transformation with the least weighted average and standard deviation is selected. This transformation constitutes the initial transformation.

In order to avoid the drift from previous frames to accumulate on to the later frames, an additional drift check is included in the algorithm. Also, a new reference frame named anchor frame is introduced to further reduce the accumulation of drift. Anchor frame refers to the frame to which the subsequent target Point Clouds will be registered. The algorithm involves the following steps:

1) Set anchor frame to the first source frame. Also the properly oriented face is assumed to be seen in the first frame, let's call this check frame.

2) Align the current frame to the anchor frame.

3) Check if the angle of rotation from current frame to the check frame is small. If yes, find the cascaded transformation from the current frame to the check frame. Check if the angle of rotation of this transformation is also low. (If there is no drift then the two angles must be almost the same).

4) Set check frame to the current frame.

5) If the angle of transformation between the current frame to the anchor frame exceeds a threshold, set the current frame as the anchor to the subsequent frames.

6) If the angle of rotation from the current from to the anchor frame exceeds a threshold, set current frame to anchor frame.

Input:

D - Starting frame from which drift check has not been done (check frame).
P_i - Source point cloud. Corresponds to the one observed in the i^{th} frame.
Q_i - Target point cloud. Corresponds to the one observed in the $(i-1)^{th}$ frame.
A - Anchor frame. Used as the Source Point Cloud for registration.
θ_i - Angle of Rotation between A and Q_i.
a_i - Angle of Rotation between the point cloud at time frame i and the check frame.
T_i - Cascaded Rotation that sends the source at any frame i to the frame of reference of D_i^{th} point cloud

```
D = P_0
A = P_0
anglesum = 0;
for i = 1 to N do
    θ_i = findAngleofRotation(A, Q_i)
    if A > threshold1 then
        A = Q_i
    end
    a_i = findAngleofRotation(D_i,Q_i)
    if a_i < threshold2 then
        if anglesum > threshold3 then
            print("Drift present")
            D = Q_i
            A = Q_i
            angle-sum = 0
        else
            print("No drift")
            D = Q_i
            angle-sum = 0
        end
    else
        angle-sum += θ_i
    end
end
```

Algorithm 3: Algorithm for Drift Correction

4 FUTURE WORK

The described algorithm tracks the person's face without a lot of user interaction and offline processing. It also checks if there is any drift and prevents it from affecting each and every subsequent frame. However, it does not attempt to correct the drift present in the previous frames. Correcting the drift present would make this algorithm more robust to the drift that occurs over time.

REFERENCES

[1] G. P. Meyer, S. Gupta, I. Frosio, D. Reddy and J. Kautz, "Robust Model-Based 3D Head Pose Estimation", 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, 2015, pp. 3649-3657. doi: 10.1109/ICCV.2015.416

[2] T. Baltrusaitis, P. Robinson, and L. Morency. "3d constrained local model for rigid and non-rigid facial tracking". In CVPR,pages 2610âĂŞ2617, 2012

[3] Besl, Paul J.; N.D. McKay (1992). "A Method for Registration of 3-D Shapes". IEEE Transactions on Pattern Analysis and Machine Intelligence. 14 (2): 239âĂŞ256.

[4] Chen, Yang; Gerard Medioni (1991). "Object modelling by registration of multiple range images". Image Vision Comput. 10 (3): 145âĂŞ155

[5] Zhang, Zhengyou (1994). "Iterative point matching for registration of free-form curves and surfaces". International Journal of Computer Vision. 13 (12): 119âĂŞ152.

[6] Rusinkiewicz, Szymon; Marc Levoy (2001). "Efficient Variants of the ICP Algorithm. Proceedings Third International Conference on 3-D Digital Imaging and Modeling". Quebec City, Quebec, Canada. pp. 145âĂŞ152.

[7] Kok-Lim Low (February 2004). "Linear Least-Squares Optimization for Point-to-Plane ICP Surface Registration" (PDF). Comp.nys.edu.sg. Technical Report TR04-004, Department of Computer Science, University of North Carolina at Chapel Hill. Retrieved 2017-02-27.

[8] FranÃğois Pomerleau, Francis Colas, Roland Siegwart, and StÃĺphane Magnenat. Comparing ICP Variants on Real-World Data Sets. In Autonomous Robots, 34(3), pages 133âĂŞ148.

[9] Holz, Dirk; Ichim, Alexandru E.; Tombari, Federico; Rusu, Radu B.; Behnke, Sven (2015). "Registration with the Point Cloud Library: A Modular Framework for Aligning in 3-D". IEEE Robotics Automation Magazine. 22 (4): 110âĂŞ124.

[10] F. Tombari and L. Di Stefano, "Object Recognition in 3D Scenes with Occlusions and Clutter by Hough Voting", 2010 Fourth Pacific-Rim Symposium on Image and Video Technology, Singapore, 2010, pp. 349-355. doi: 10.1109/PSIVT.2010.65

[11] Y. Sun and L. Yin. "Automatic pose estimation of 3d facial models". In ICPR, pages 1âĂŞ4, 2008
[12] M. D. Breitenstein, D. Kuettel, T. Weise, L. Van Gool, and H. Pfister. "Real-time face pose estimation from single range images". In CVPR, pages 1âĂŞ8, 2008.
[13] C. Papazov, T. K. Marks, and M. Jones. "Real-time 3d head pose and facial landmark estimation from depth images using triangular surface patch features". In CVPR, pages 4722âĂŞ4730, 2015.
[14] E. Seemann, K. Nickel, and R. Stiefelhagen. "Head pose estimation using stereo vision for human-robot interaction". In AFGR, pages 626âĂŞ631, 2004
[15] G. Fanelli, T. Weise, J. Gall, and L. Van Gool. "Real time head pose estimation from consumer depth cameras". In Pattern Recognition, pages 101âĂŞ110. 2011.
[16] S. Tulyakov, R.-L. Vieriu, S. Semeniuta, and N. Sebe. "Robust real-time extreme head pose estimation". In ICPR, pages 2263âĂŞ2268, 2014
[17] P. Padeleris, X. Zabulis, and A. A. Argyros. "Head pose estimation on depth data based on particle swarm optimization". In CVPRW, pages 42âĂŞ49, 2012
[18] P. Viola and M. J. Jones. "Robust real-time face detection". Int. J. Comp. Vision, 57(2):137âĂŞ154, 2004
[19] Q. Cai, D. Gallup, C. Zhang, and Z. Zhang. "3d deformable face tracking with a commodity depth camera". In ECCV, pages 229âĂŞ242. Springer, 2010
[20] A. Rekik, A. Ben-Hamadou, and W. Mahdi. "3d face pose tracking using low quality depth cameras". In VISAPP, pages 223âĂŞ228, 2013
[21] M. Martin, F. Van De Camp, and R. Stiefelhagen. "Real time head model creation and head pose estimation on consumer depth cameras". In 3DV, volume 1, pages 641âĂŞ648, 2014
[22] E. Seemann, K. Nickel, and R. Stiefelhagen. "Head pose estimation using stereo vision for human-robot interaction". In AFGR, pages 626âĂŞ631, 2004
[23] T. Baltrusaitis, P. Robinson, and L. Morency. "3d constrained local model for rigid and non-rigid facial tracking". In CVPR, pages 2610âĂŞ2617, 2012
[24] S. Fleishman, D. Cohen-Or, C. T. Silva. "Robust moving least-squares fitting with sharp features". In Siggrpah2005.
[25] M. Kazhdan, M. Bolitho, H. Hoppe. "Poisson surface reconstruction". In Siggraph 2006.
[26] Y. Chen, G. Medioni. âĂİObject Modeling by Registration of Multiple Range Images,âĂİ Proc. of the 1992 IEEE Intl. Conf. on Robotics and Automation, pp. 2724-2729, 1991.
[27] S. Salti, F. Tombari, L. Di Stefano, "SHOT: Unique Signatures of Histograms for Surface and Texture Description", Computer Vision and Image Understanding, May, 2014.
[28] P. J. Angeline. "Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences". In Evolutionary Programming VII, pages 601âĂŞ610, 1998
[29] Radu Bogdan Rusu and Steve Cousins. "Rusu ICRA2011 PCL,3D is here: Point Cloud Library (PCL), IEEE International Conference on Robotics and Automation (ICRA)", May 9-13, 2011.
[30] G. Fanelli, T. Weise, J. Gall, L. Van Gool, "Real Time Head Pose Estimation from Consumer Depth Cameras", 33rd Annual Symposium of the German Association for Pattern Recognition (DAGM'11).
[31] J. Gu, X. Yang, S. De Mello, and J. Kautz, âĂIJDynamic Facial Analysis: From Bayesian Filtering to Recurrent Neural NetworksâĂİ, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2017, July 2017.

5 VISUALIZATION OF THE POINT CLOUDS FOR THE FIRST TWO REGISTRATION LOOPS:

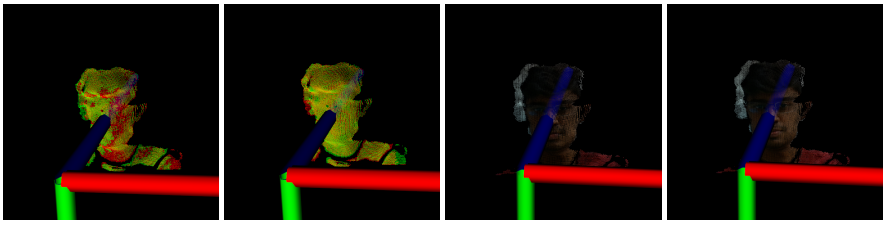

Figure 7: Point Cloud Vizualizations - from left, first before registration (rendered without texture), second after registration (rendered without texture), third before registration (rendered with texture) and fourth after registration (rendered with texture)

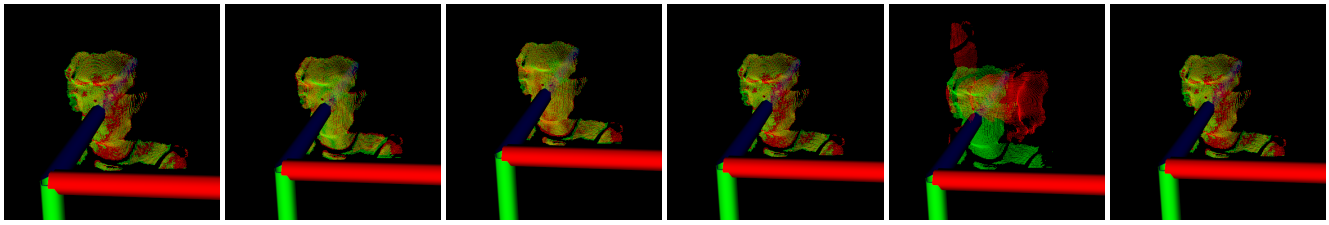

Figure 8: Transformations obtained by Initial Hough Voting

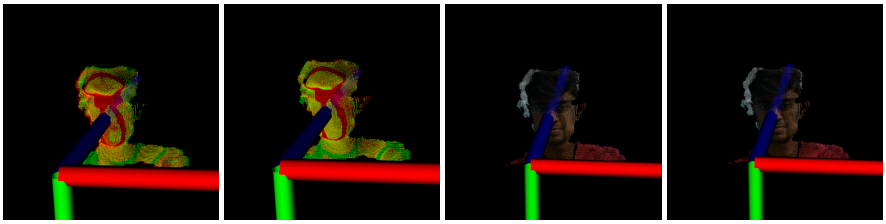

Figure 9: Point Cloud Vizualizations - from left, first before registration (rendered without texture), second after registration (rendered without texture), third before registration (rendered with texture) and fourth after registration (rendered with texture)

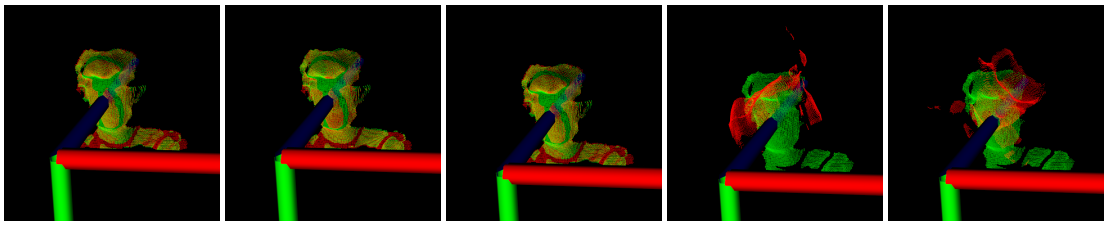

Figure 10: Transformations obtained by Initial Hough Voting

6 VISUALIZATION OF THE POINT CLOUDS FOR THE FIRST THREE DRIFT CORRECTION ITERATIONS:

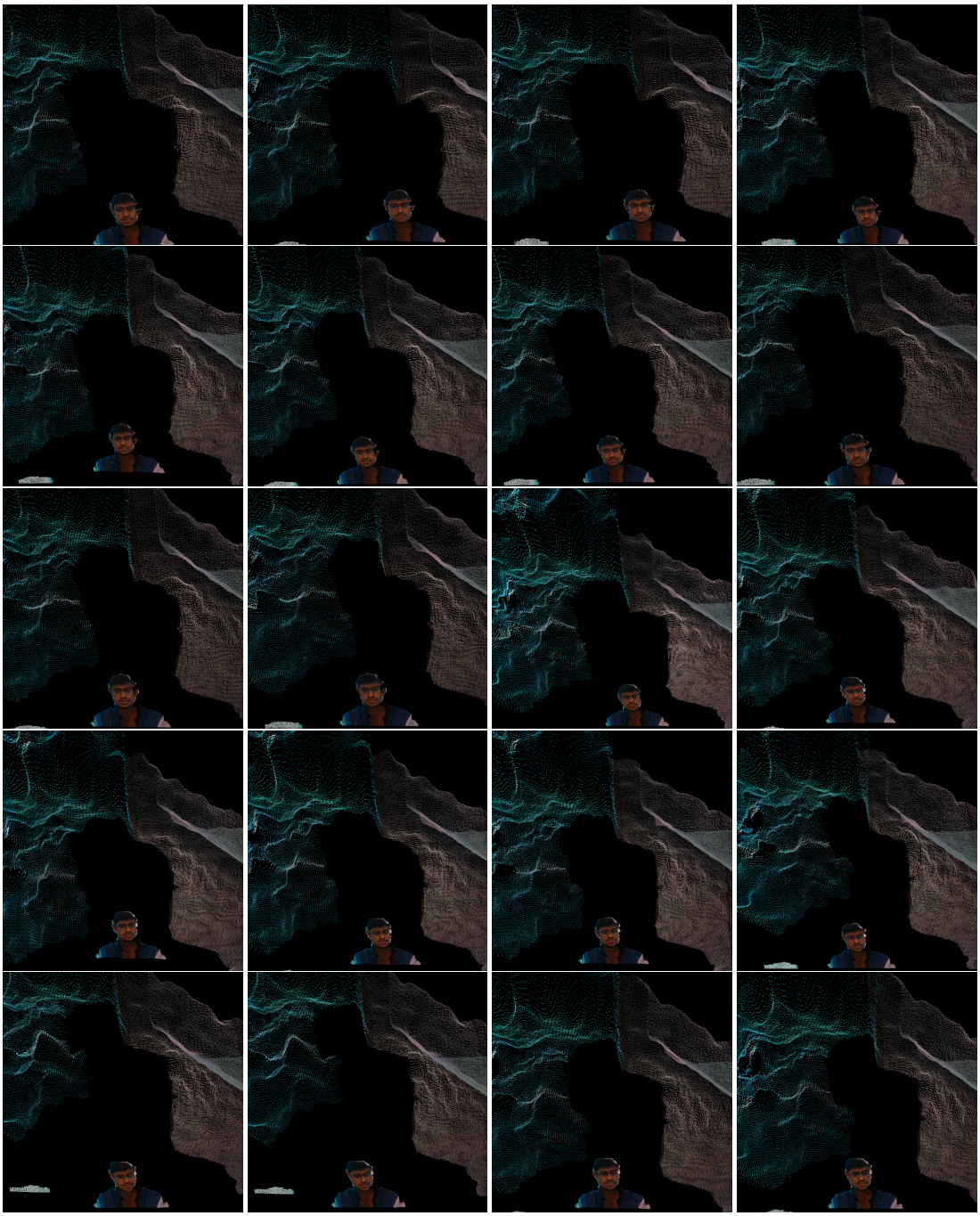

Figure 11: Point Cloud captures for the first loop of drift correction (frames 1,...,10,104,129,...,138 are shown)

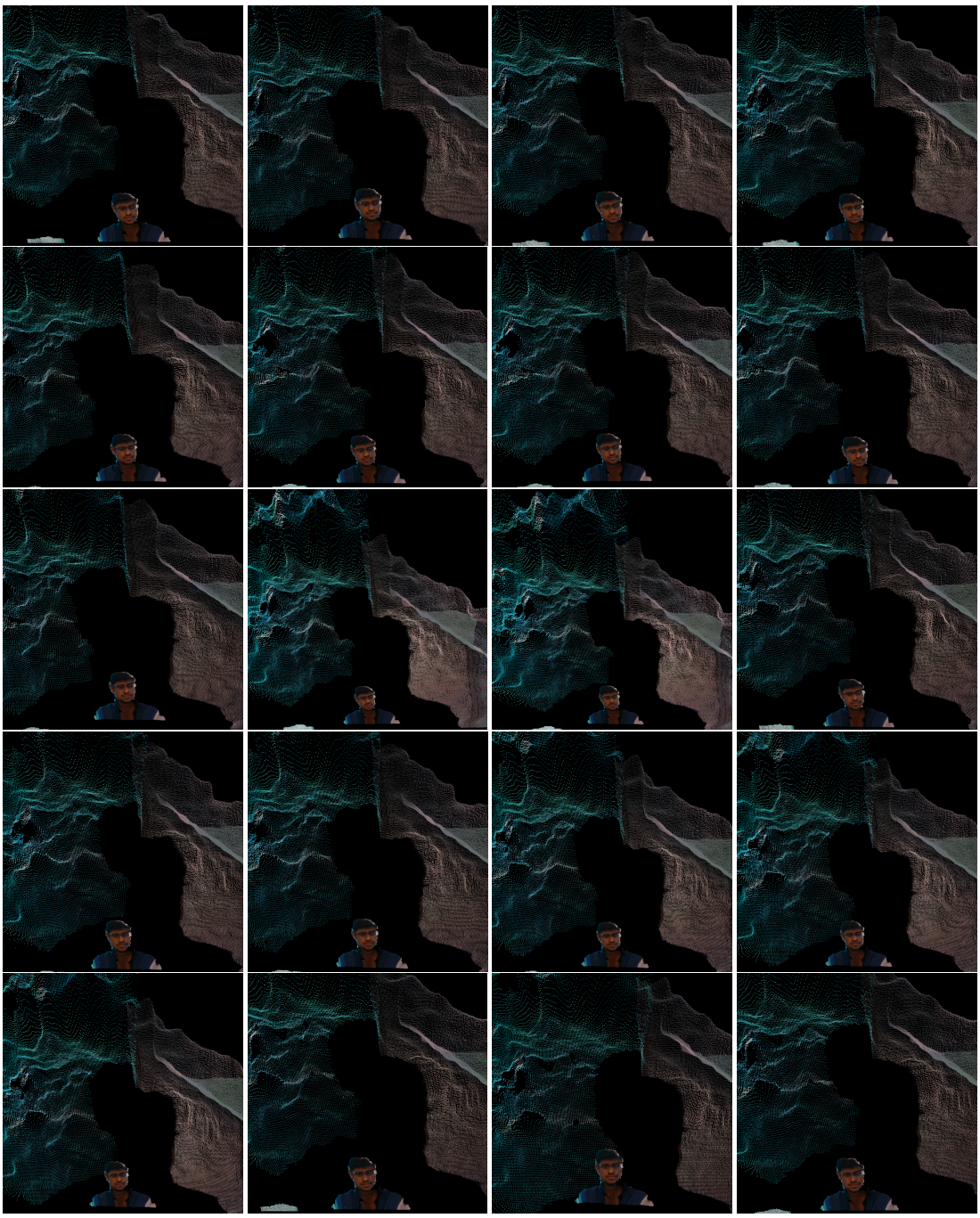

Figure 12: Point Cloud captures for the second loop of drift correction (frames 139,...,150,229,...,236 are shown)

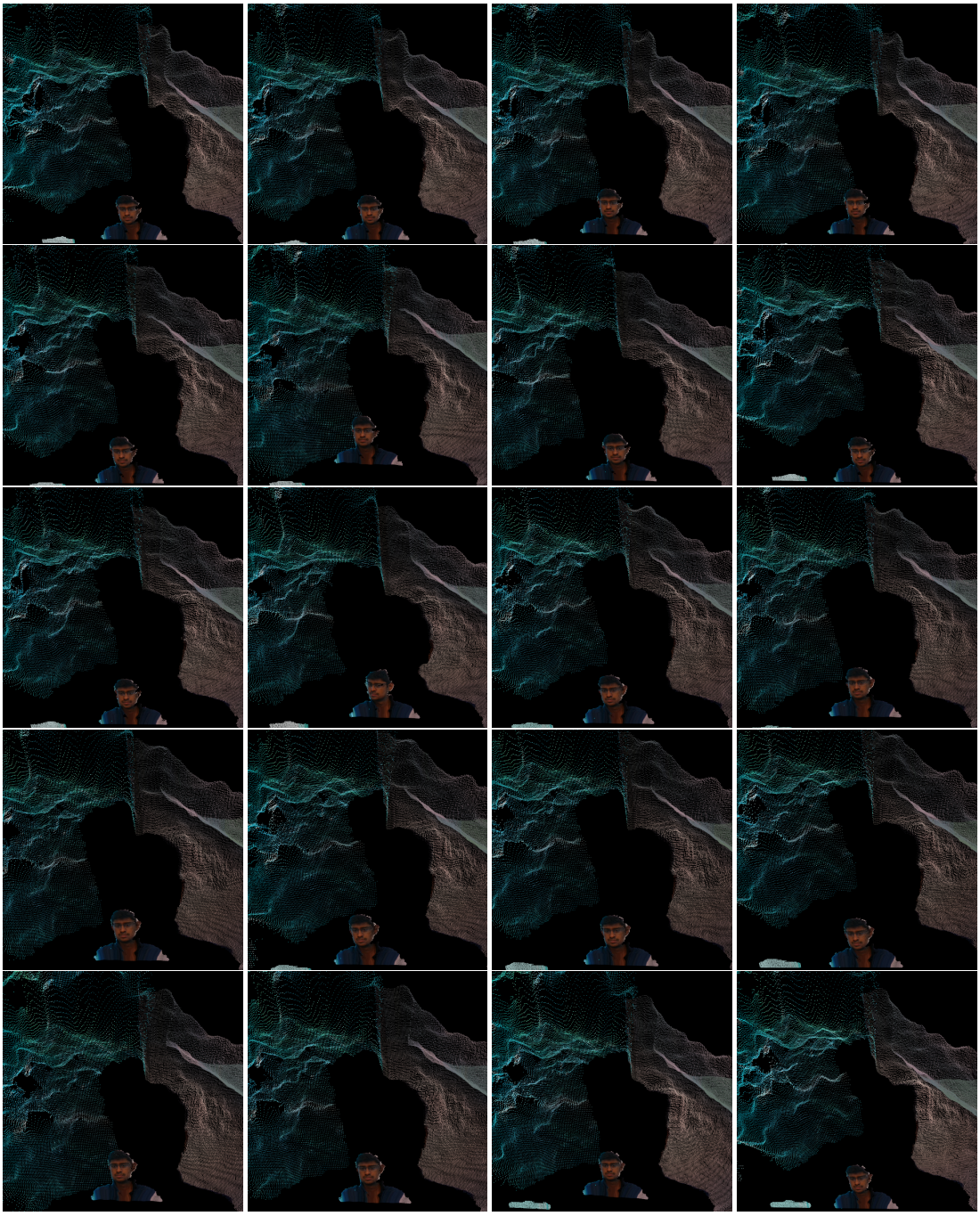

Figure 13: Point Cloud captures for the third loop of drift correction (frames 290,...,298,364,479,...,488 are shown)

7 VISUALIZATION OF THE REGISTERED POINT CLOUD WITH DRIFT CORRECTION:

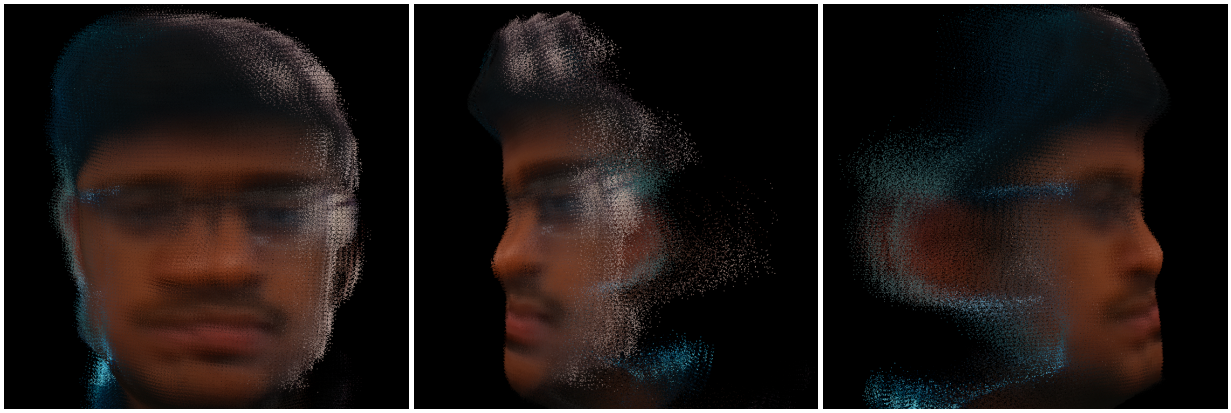

Figure 14: Registered Point Cloud using algorithm with drift correction.

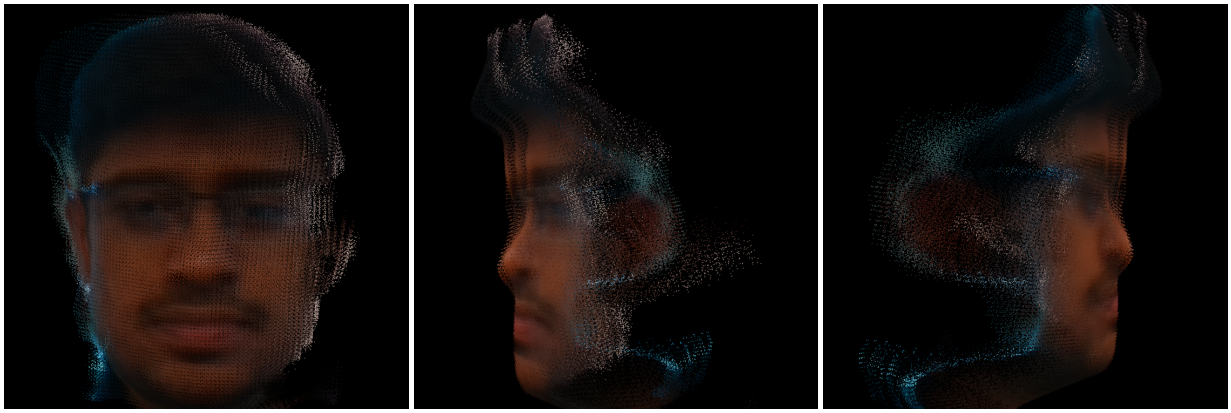

Figure 15: Registered Point Clouds from frames 0 to 138 using algorithm with drift correction.

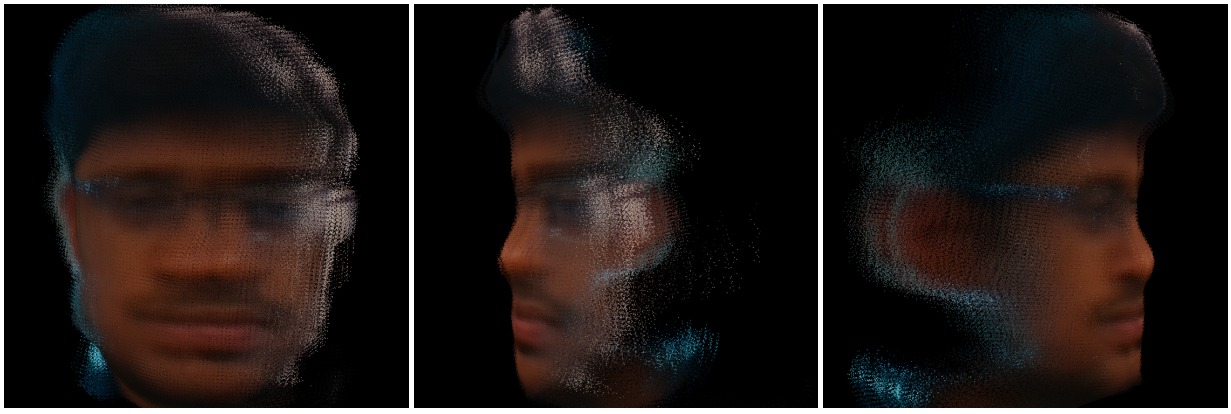

Figure 16: Registered Point Clouds from frames 139 to 236 using algorithm with drift correction.

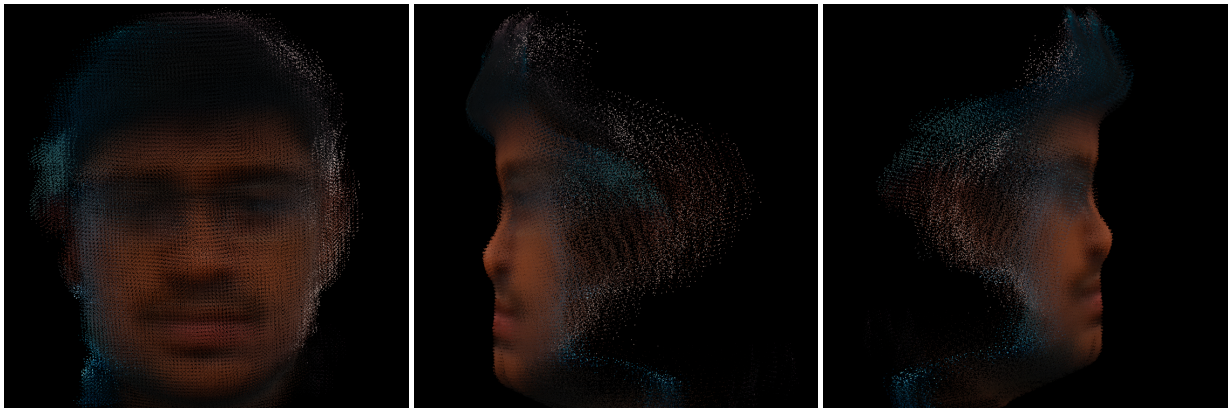

Figure 17: Registered Point Clouds from frames 290 to 488 using algorithm with drift correction.

Figure 18: Registered Point Cloud using algorithm without drift correction.

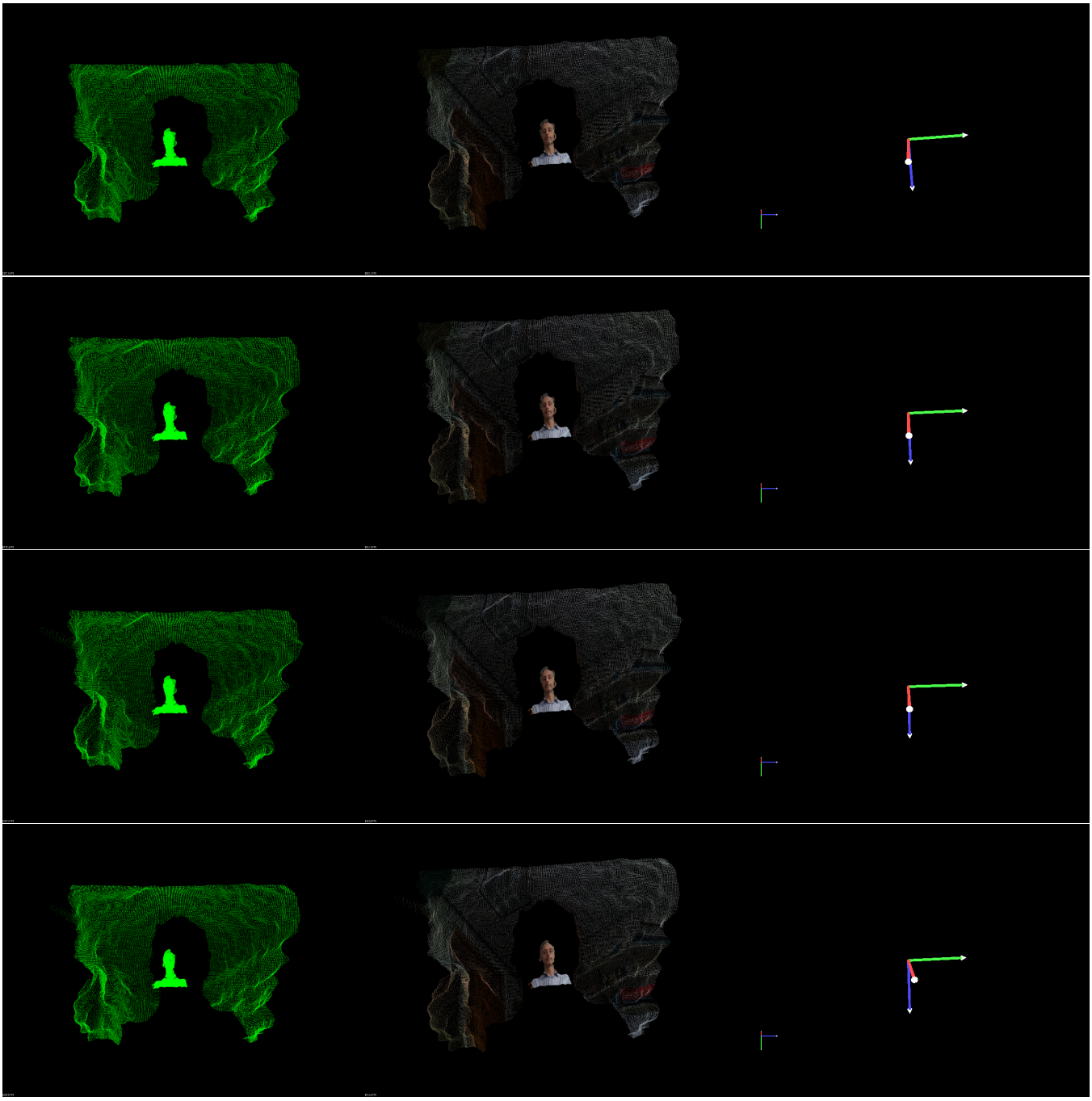

Figure 19: Sample Frames of the Pose Estimated. From left, Captured Point Cloud(rendered without texture), Captured Point Cloud(rendered with texture) and Finally Estimated Pose (small axis represents the world co-ordinate systems and the larger one the calculated face pose)

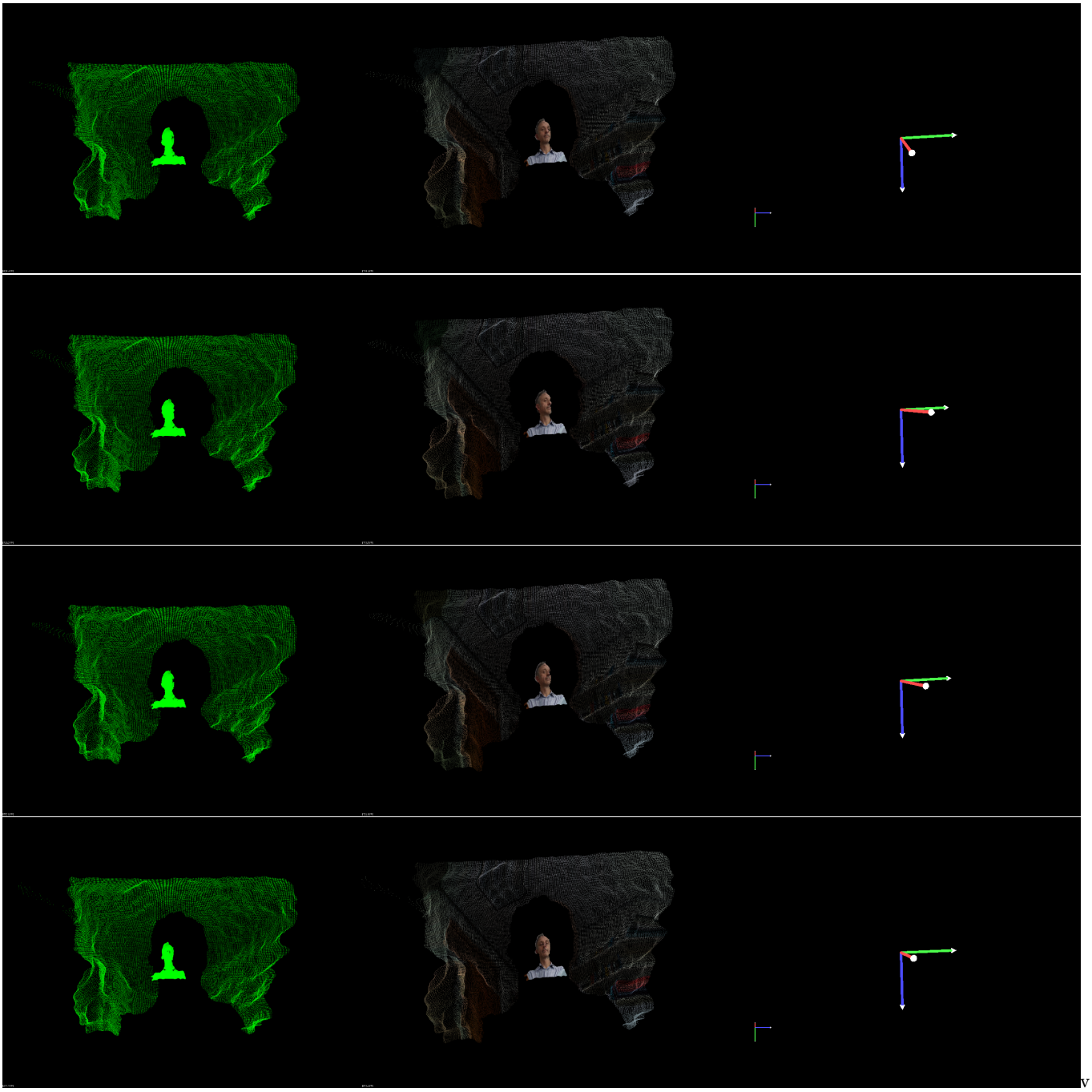

Figure 20: Sample Frames of the Pose Estimated. From left, Captured Point Cloud(rendered without texture), Captured Point Cloud(rendered with texture) and Finally Estimated Pose (small axis represents the world co-ordinate systems and the larger one the calculated face pose)

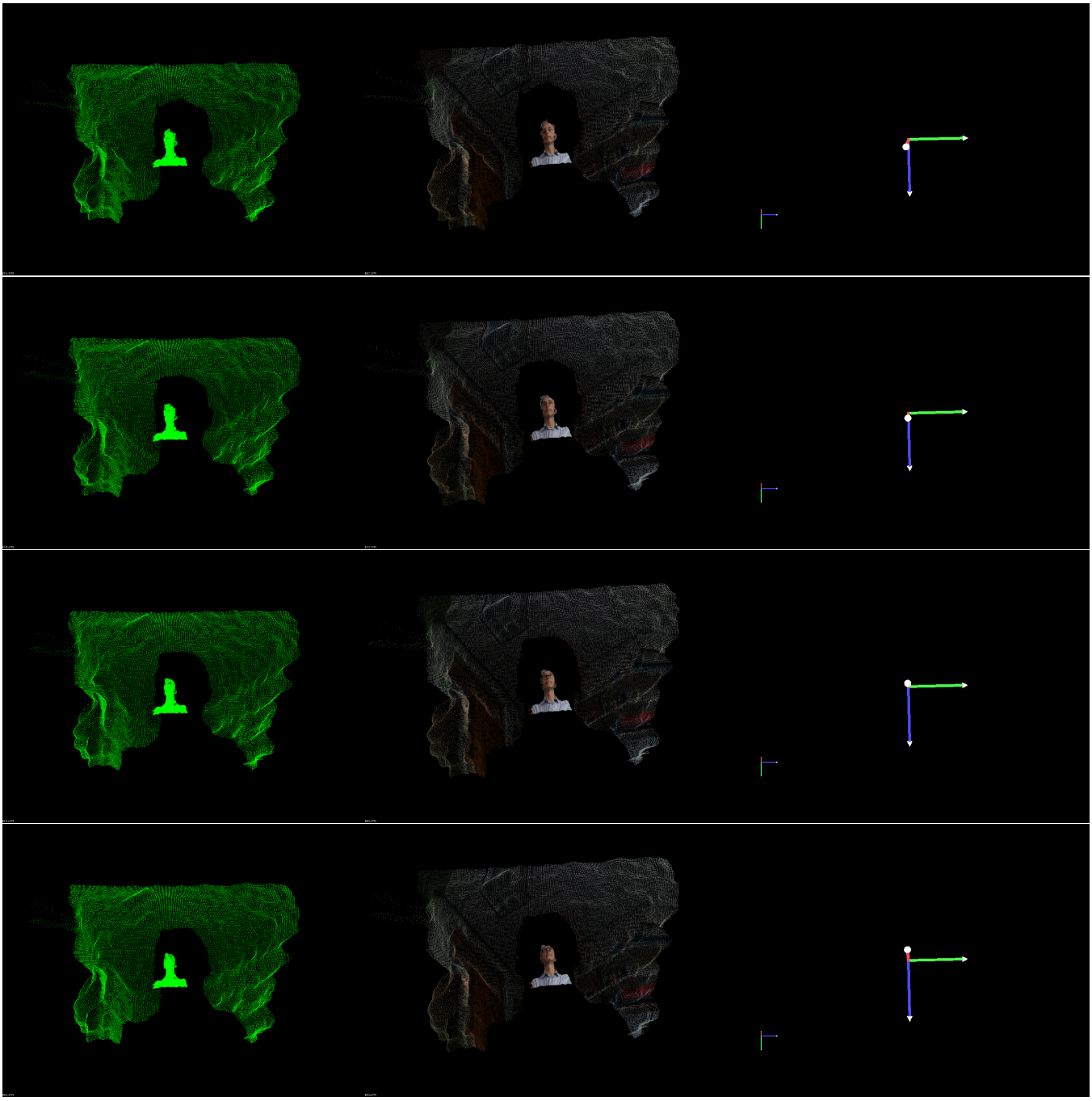

Figure 21: Sample Frames of the Pose Estimated. From left, Captured Point Cloud(rendered without texture), Captured Point Cloud(rendered with texture) and Finally Estimated Pose (small axis represents the world co-ordinate systems and the larger one the calculated face pose)