3D Reconstruction with Depth Camera

Carleton University

COMP5115

Prof. Oliver Van Kaick

Faubert, Joël
jfaub011@uottawa.ca

December 9, 2019

1 Objective

This project investigates techniques for producing 3d models using consumer grade depth sensors. The goal is to use the Intel RealSense D435 camera (the D435), the Software Development Kit (the RealSense SDK) and the Viewer to capture depth images and point clouds.

State of the art techniques pioneered by Kinect Fusion [5] build models incrementally by merging (or fusing) a series of depth images into a common frame of reference. These techniques require careful memory management, shared data structures enabling fast searches and parallel programming to exploit graphics processors.

In order to better understand these techniques, I simulate the algorithm by manually collecting, cleaning and merging point clouds, then attempting to reconstruct polygonal meshes. I have dubbed my two reconstruction projects Cube Man and Joe Head.

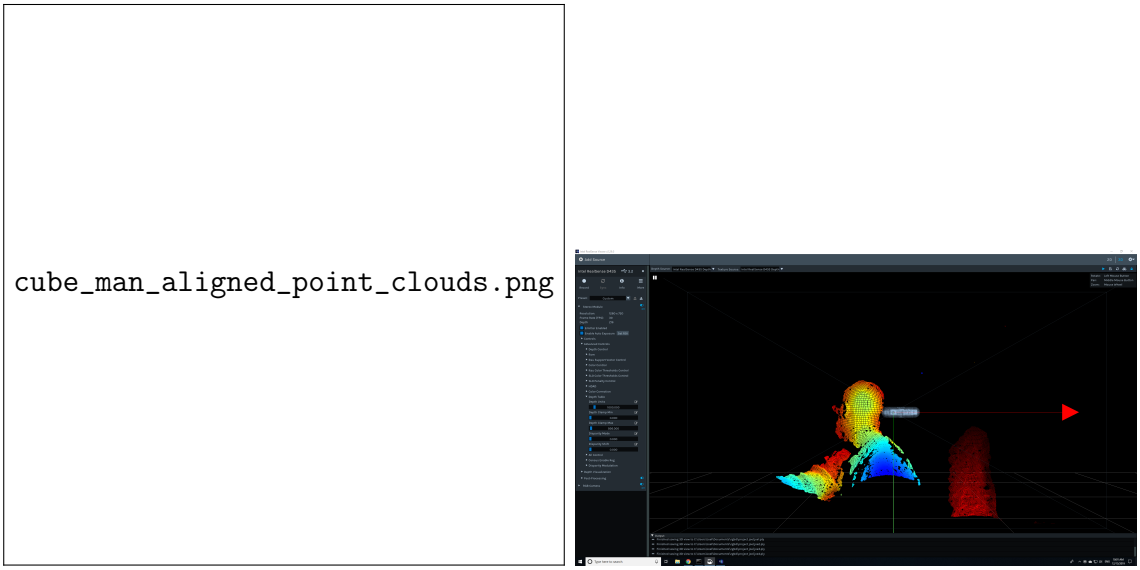

Figure 1: Cube Man (left) and Joe Head (right)

2 Methodology

After taking a quick look at the stereoscopic depth camera's inner workings, this report details the capturing and aligning of point clouds for model reconstruction. This process consists of a few steps:

- the use of the RealSense SDK and Viewer to produce depth images and point clouds [11],
- the use of MeshLab's registration (or align) methods to merge point clouds,
- a survey of some techniques available for smoothing, eliminating noise, and
- reconstructing a mesh from the merged point clouds.

3 Implementation

The following section outlines the key steps for capturing depth images, converting them into point clouds and using these point clouds for 3D reconstruction.

3.1 Depth Images

The workings of the previous generation of RealSense cameras are detailed in [4]. The algorithm described therein has not significantly changed (I believe) in the 400 series. The new features seem to be additional on-device hardware post-processing.

These cameras produce four-channel image composed of the standard RGB colour channels together with a fourth depth channel. The depth channel associates to each pixel an estimate of the distance from the camera to the surface.

The D435 (and similar available devices) use a stereoscopic technique inspired by human binocular vision. It is possible to compute a pixel's depth by comparing the images produced by two cameras placed at known distances and pointing in generally the same direction. After projecting the images onto a common plane (the cameras produce images which are slightly "angled"), the camera computes the *disparity* d between the two images.

Disparity is the distance (in pixels) of a cluster of pixels from one image to the other. Objects closer to the camera will have greater disparity between the two images, distant objects less so. With distance B between the two cameras and the focal length f, the disparity allows the camera to infer the depth z using the formula:

$$z = \frac{f \cdot B}{d}$$

However, this simple stereoscopic approach struggles in many different environments. When materials are transparent, reflective or patterned, or occluded, the algorithm can

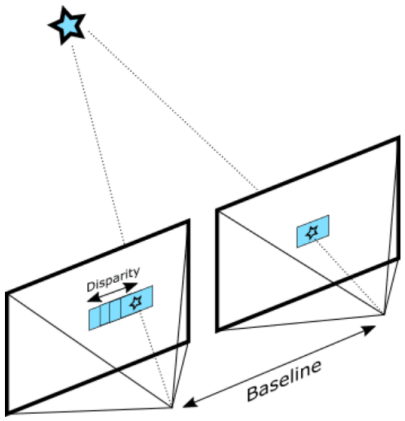

Figure 2: Binocular Vision

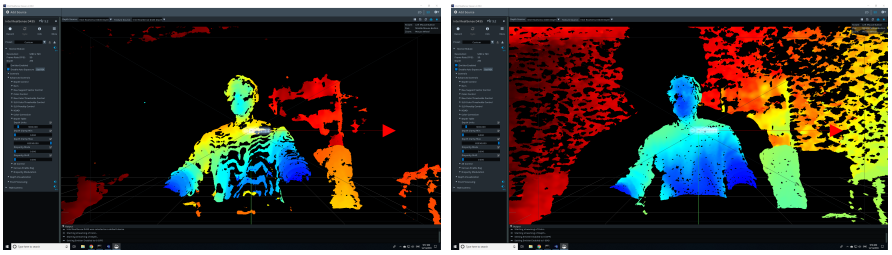

Figure 3: Infrared emitter on (left) and off (right)

struggle to find meaningful clusters of disparity. To improve the robustness of the depth calculation on a variety of textures and surfaces, the devices additionally include an infrared projector.

In the pictures above, you can see that without the emitter, the camera is unable to detect the uniformly white wall behind me. The emitter allows the camera to make measurements, but notice how the unstructured infrared light adds a rippling distortion to the measurements.

All of the complexity of these calculations, of course, are abstracted in the Viewer and SDK bundled with the camera. Using the Intel software, it is straightforward to take depth-snapshots or videos and export them to mesh files.

3.2 Depth Pixel to Point Cloud Transformation

With the x and y coordinates of a pixel and a corresponding depth measurement z, it is possible to apply a transformation to obtain a point in a 3d frame of reference, say, "in front of the camera".

This transformation depends on the configuration and specifications of the camera(s). Suffice it to say that the RealSense SDK and Viewer offer functions which can convert from 2d pixel space to 3d space.

Note that this conversion may involve a tradeoff. Each pixel's frustum is a thin but expanding shape so closer pixels are smaller than distant ones. Sophisticated transformations take this into consideration when converting depth values into voxels or points.

It may be of interest that the RealSense SDK offers a CUDA implementation at:

`github.com/IntelRealSense/librealsense/blob/master/src/cuda/cuda-pointcloud.cu`

This version runs the per-pixel conversion on the GPU. The *device* function

```
deproject_pixel_to_point_cuda(float points[3], rs2_intrinsics,
  float pixel[2], float depth)
```

performs the per-point computation and the *global* function

```
kernel_deproject_depth_cuda(float * points, rs2_intrinsics*,
  uint16_t * depth, float depth_scale)
```

can be used to run a calculation on the GPU after allocating memory on the graphics device and defining the number of cores/thread blocks to use. Both share an *intrisic* object which stores the specifications of the camera and current frame.

3.3 Background subtraction

In order to align the different point clouds, it is necessary to remove noise and outliers. We do not want error points to be considered when attempting align clouds. When scanning an object in a room, the background, floor, and other features of the room are noise.

Background subtraction is a well-studied problem in image computation [6]. A more sophisticated approach would adopt some of those techniques, such as first taking an "empty" snapshot and subtracting it from subsequent ones to isolate the model. More complicated approaches construct models of the background to accomodate for change over time and so on.

For our purpose of capturing toy examples, it suffices to set a maximum depth and discard the back wall of a fixed scene. The RealSense Viewer allows the user to pause a 3d scene and export a point cloud to ".ply" format. It's also possible to do this programmatically using the SDK: a modified version of the librealsense pointcloud example takes a number of delayed snapshots (included below in Appendix).

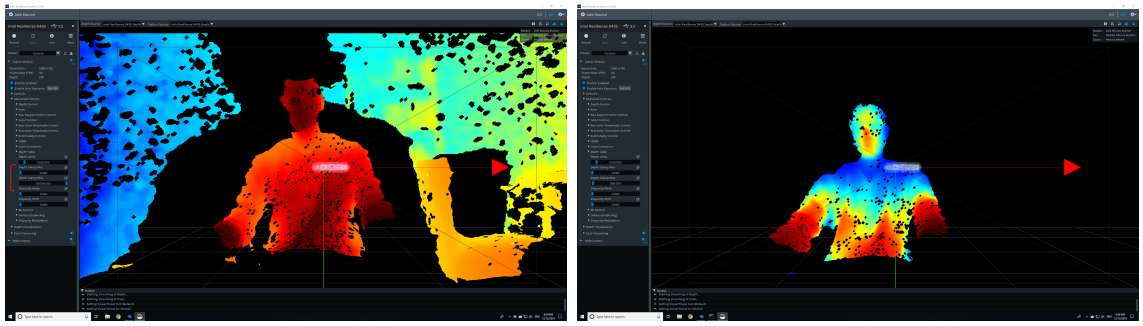

Figure 4: Culling the back wall with max depth clamp

3.4 Filters

The SDK and Viewer offer several post-processing filters to improve the quality of the captured depth data. The spatial filter compares each pixel to its neighbours in a frame to ensure that there are no improbable measurements. Temporal filters compare new frames to some average or distribution of previous ones. The temporal filter in particular reduces the flickering and warping of depth streams.

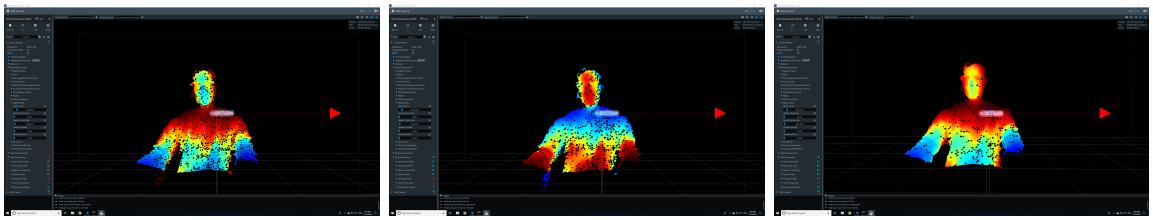

Figure 5: Depth images captured without post processing (left), with spatial filtering (middle) and using all available options (right)

Large-scale version of these images are included in an appendix.

3.5 Removing Outliers and Errors

Even with the background subtraction and filters, there are often still outliers and errors that should not be included in a quality model. Worse still, these garbage vertices make it harder to perform the registration in the next phase. Luckily, MeshLab [7] makes it quite easy to select, move and delete vertices.

One way to remove extraneous points is to use the connected component selection tool to grab the object of interest, then use the selection inversion to select every point except the model. All these vertices and/or faces can then be deleted.

This process needs to be repeated for each point cloud which will constitute a “layer” in the model construction.

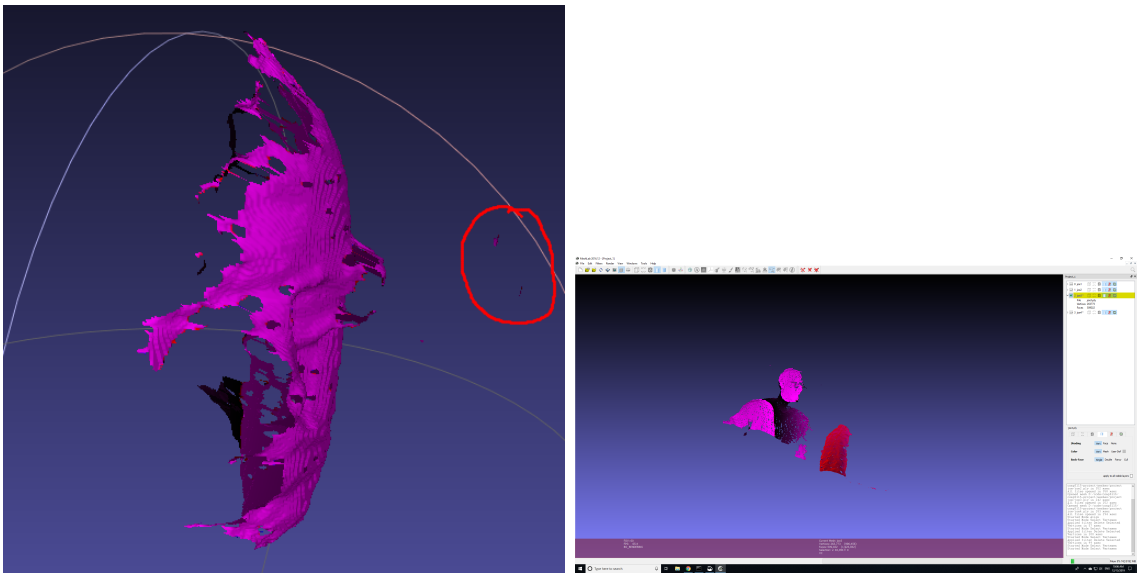

Figure 6: Outliers caused by erroneous readings (left), removing chair from capture

3.6 Aligning Point Clouds

Once cleaned, MeshLab [7] can be used to align the point clouds. Each point cloud mesh is imported as its own layed, which can be moved and rotated so that they lign up closely. This problem is also known as point cloud registration [10] and the Iterative Closest Points (or ICP) is another well-studied algorithm. MeshLab provides a user-assisted method for giving the ICP a head start by choosing an initial alignment.

In MeshLab, open the Alignment tool and Glue one of the layers as a base. Now choose a different layer to align and select Point Based Gluing. It opens a new window which allows the user to move and rotate the two clouds to better view common areas. Then, the user selects (at least) four points corresponding points on both of the models to find an initial alignment. Often, this initial alignment is not correct and the user must try again with different points.

Once an initial alignment is found, presssing Process in the Alignment window runs Iterative Closest Points to refine the alignment. It is often helpful to run the algorithm several times and/or increase the number of iterations in the ICP parameters.

It can be particularly tricky to align two models with little in common. For a quality point cloud, it would be necessary to take dozens if not hundreds of captures so that the transition from each frame is smooth and common surface between each one is maximized, especially if there are sharp edges as in the Cube Man project.

Figure 7: Point-based Initial Transform

After aligning two (or more) layers, you can then flatten them together by making the aligned layers visible, right clicking in the layers window, and selecting Flatten Visible Layers. Flattening the layers arguably makes it easier align additional point clouds, since the known surface has hopefully expanded.

3.7 Simplifications, Sampling and Smoothing

Attempting reconstruction directly on the merged point clouds leads to fairly messy objects. There are apparently non-manifold vertices and incorrect normals which produce strange objects.

4 Results

An initial goal of this project was to create a model of my head for 3d printing, but it was difficult to keep a consistent posture for multiple angles. Photography skill (and a longer USB cable!) would be helpful for capturing quality depth images. When the object is not rigid across different frames the point clouds do not quite fit together.

For the Cube Man project, the final reconstructions are somewhat disappointing. None of the reconstruction methods tried reconstruct a smooth but detailed model. Even after carefully aligning multiple clouds the reconstruction obtained from the merged point clouds is jagged and full of artifacts.

The reality is that these devices do not have very good accuracy at small scales. The infrared emitter mitigates some of the problems of stereoscopic depth but also seem to introduce some noise.

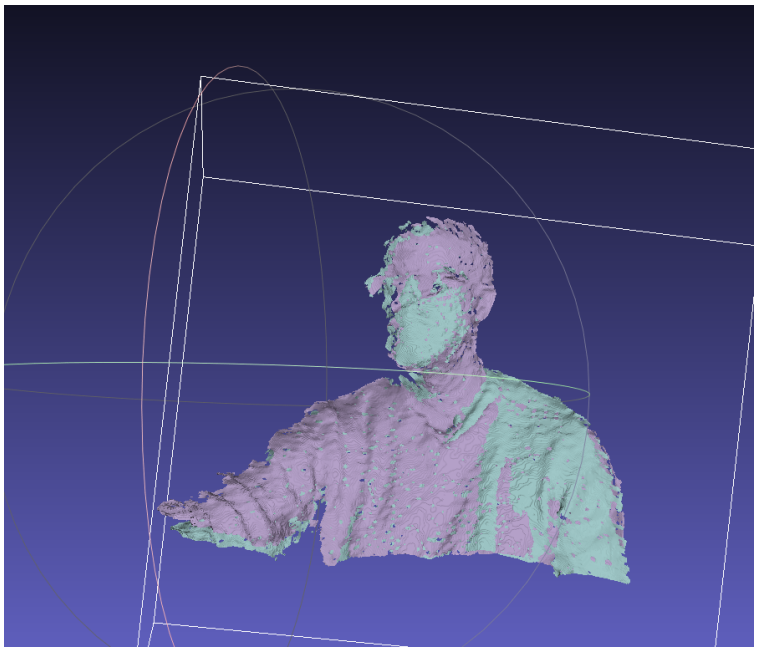

Figure 8: Two point clouds aligned using ICP

However, the D435 and like can produce a stream of decent depth images at high speed. Obviously, the Fusion method [5] is superior to this artisanal approach. With a big data streaming approach, it is possible to identify noise, discard erroneous values and fill in holes and details.

5 Documentation

6 Conclusion

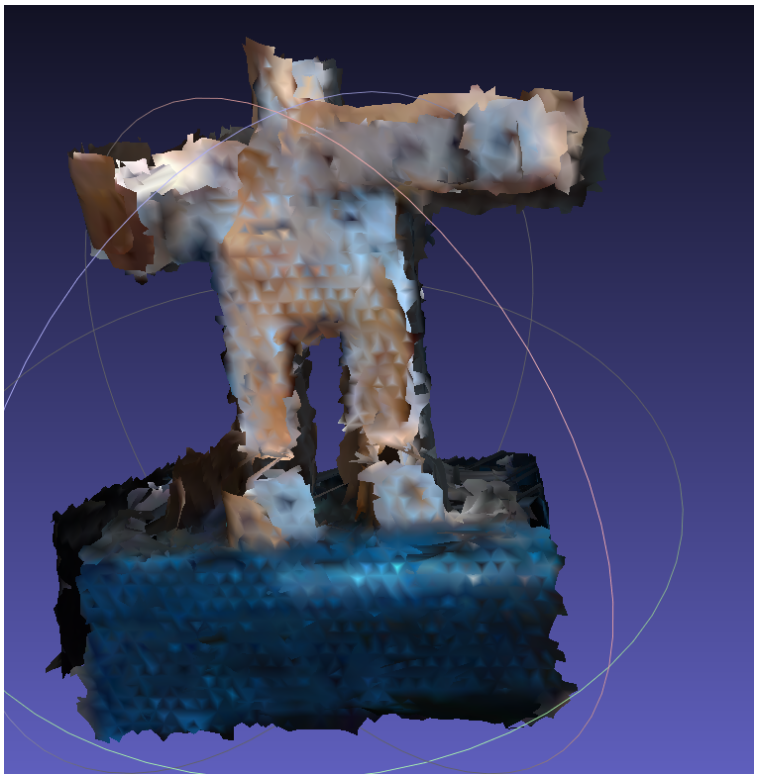

Figure 9: Simplifying merged cloud with clustering decimation

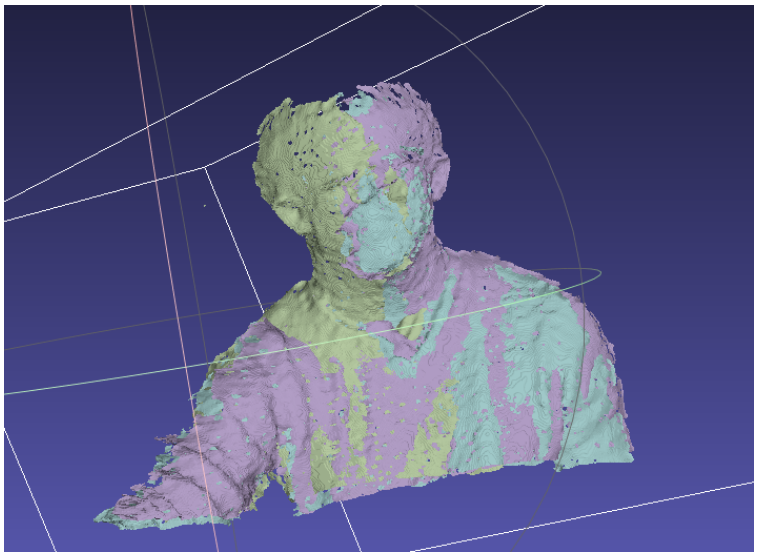

Figure 10: Difficult to take depth selfies from different angles without moving

References

[1] G. K. Tam, Z.-Q. Cheng, Y.-K. Lai, F. C. Langbein, Y. Liu, D. Marshall, R. R. Martin, X.-F. Sun, and P. L. Rosin, "Registration of 3d point clouds and meshes: a survey from rigid to nonrigid," *IEEE transactions on visualization and computer graphics*, vol. 19, no. 7, pp. 1199–1217, 2012.

[2] B. Bellekens, V. Spruyt, R. Berkvens, and M. Weyn, "A survey of rigid 3d pointcloud registration algorithms," in *AMBIENT 2014: the Fourth International Conference on Ambient Computing, Applications, Services and Technologies, August 24-28, 2014, Rome, Italy*, 2014, pp. 8–13.

[3] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3d reconstruction at scale using voxel hashing," *ACM Transactions on Graphics (ToG)*, vol. 32, no. 6, p. 169, 2013.

[4] L. Keselman, J. Iselin Woodfill, A. Grunnet-Jepsen, and A. Bhowmik, "Intel realsense stereoscopic depth cameras," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 1–10.

[5] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. W. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking." in *ISMAR*, vol. 11, no. 2011, 2011, pp. 127–136.

[6] M. Piccardi, "Background subtraction techniques: a review," in *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No. 04CH37583)*, vol. 4. IEEE, 2004, pp. 3099–3104.

[7] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, "Meshlab: an open-source mesh processing tool." in *Eurographics Italian chapter conference*, vol. 2008, 2008, pp. 129–136.

[8] D. Chetverikov, D. Svirko, D. Stepanov, and P. Krsek, "The trimmed iterative closest point algorithm," in *Object recognition supported by user interaction for service robots*, vol. 3. IEEE, 2002, pp. 545–548.

[9] N. Mellado, D. Aiger, and N. J. Mitra, "Super 4pcs fast global pointcloud registration via smart indexing," in *Computer Graphics Forum*, vol. 33, no. 5. Wiley Online Library, 2014, pp. 205–215.

[10] F. Pomerleau, F. Colas, R. Siegwart *et al.*, "A review of point cloud registration algorithms for mobile robotics," *Foundations and Trends® in Robotics*, vol. 4, no. 1, pp. 1–104, 2015.

[11] I. C. Vision, "RealSense SDK 2," https://github.com/IntelRealSense/librealsense, 2019, [Online; accessed 11-December-2019].

1 Comparing Post Processing

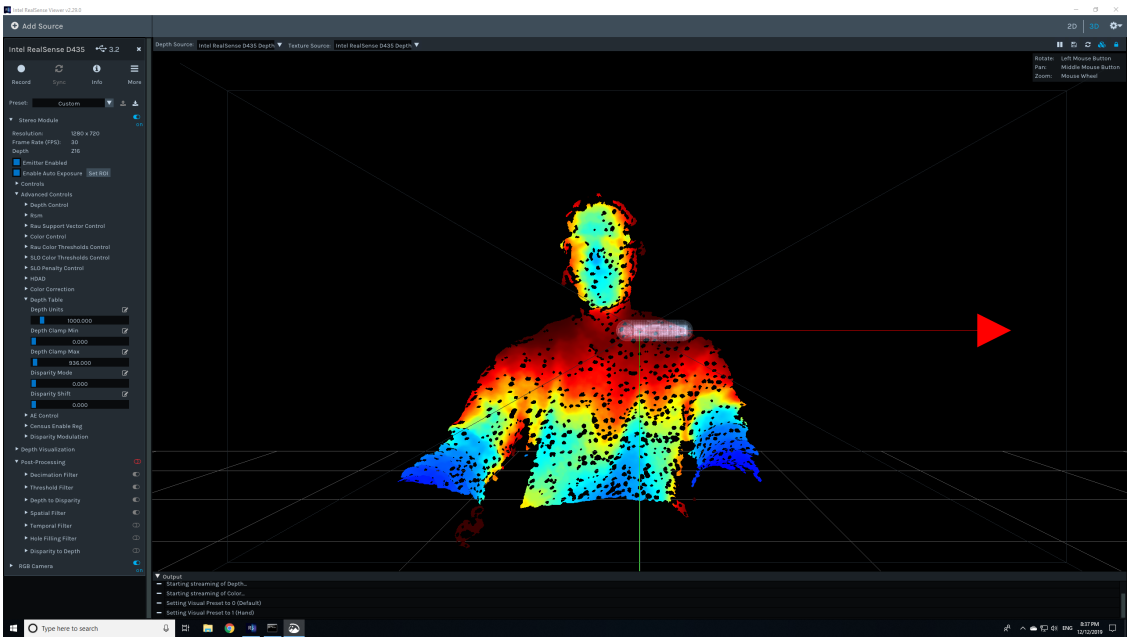

Figure 11: Depth image captured without post processing

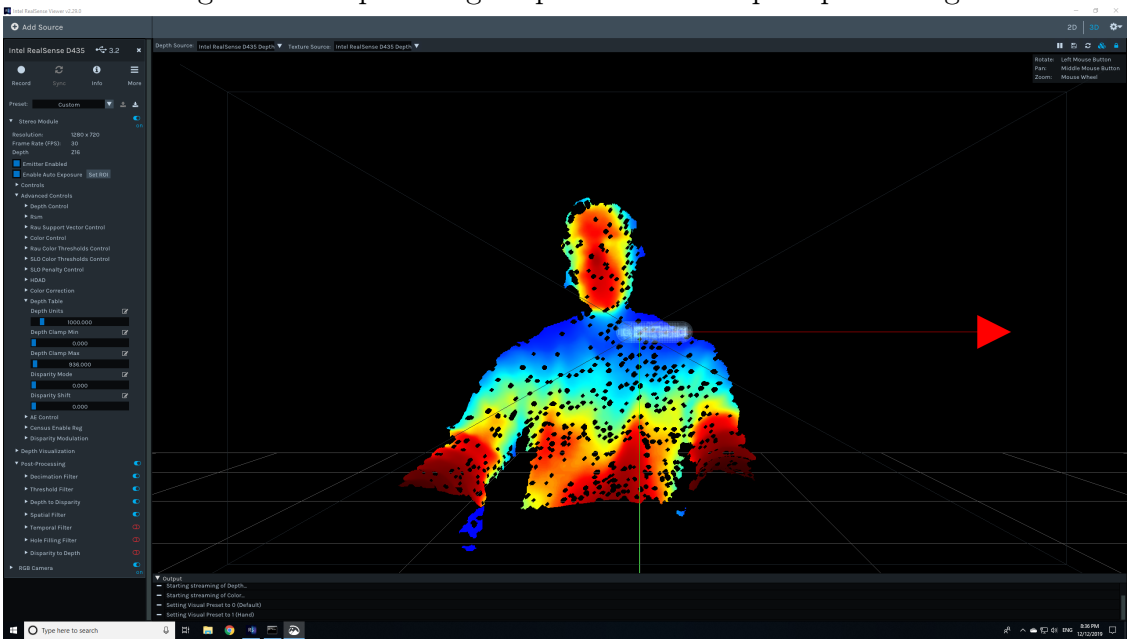

Figure 12: Depth image captured with spatial filtering