Recovering Missing Depth Information from Microsoft's Kinect

Abdul Dakkak **Ammar Husain**

Abstract

This paper presents a technique to estimate missing depth information obtained from the Microsoft Kinect. Due to the limitations of the depth sensing technology used by the Kinect, a significant loss and noise incurs when capturing depth information. The paper describes the steps needed to go from raw depth, as captured by Kinect, to a point cloud representation of the recovered missing data. Our main contribution in this paper is the use of an iterative diffusion method that accounts for both the known depth values and RGBD segmentation results to recover missing depth information. The algorithm proposed is highly parallelizable and is capable of being implemented for real-time processing by utilizing the Graphical Processing Unit.

1 Introduction

The Microsoft Kinect is an inexpensive device to obtain both depth and RGB data. The Kinect captures the depth map by projecting infra-red patterns on the scene and measuring their displacement. Unlike depth measuring technologies, like laser based LIDAR or range finding cameras, the Kinect contains many missing data and is incapable of measuring depth for shiny objects or objects on certain orientation. On the flip side, traditional depth capturing devices cost in the order of a several thousand dollars, while the Kinect costs around $150. This makes the Kinect affordable for both recreational use. Using the processing step detailed in this paper, it can achieve similar noise free measurements and robustness as the costly solutions.

Microsoft's main purpose for the Kinect is to obtain human pose recognition for the Xbox 360 gaming console. With depth information, this becomes a trivial problem of binary segmentation followed by a more complicated problem of pose recognition [1]. Since the Kinect was conceived as an addendum to the Xbox, and is operated in an environment with a conspicuous foreground (human) and background (room), a clean and smooth depth map is not necessary. However, if the Kinect is to be extended to a more general setting such as the inclusion in consumer laptops for human-computer interaction, robotics for robot motion planning, or graphics for environment reconstruction, a color and texture invariant multi-object segmentation is essential. This therefore requires the depth to be both smooth and complete.

1.1 Objective

The goal of this paper is to develop a real-time algorithm for both smoothing known depth and recovering missing depth information. In this paper, we present an novel algorithm that estimates the missing depth pixels by utilizing the RGBD segmentation and diffusing data from known depth values to unknown ones. The paper is divided as follows: in (2) we introduce our algorithm, how the data is captured and applying the homography matrix to account for the intrinsics of the depth

camera (2.1), using a mean filter to recover small missing depth information (2.2), modifying the Parzen density function for Quickshift to get an augmented segmentation that accounts for the depth values (2.3), using both the segmentation and known depth values to recover missing data (2.4), and, finally, visualizing the resulting point cloud (2.5). We end the paper by discussing possible application (3) and future work (4).

1.2 Previous Work

Missing depth map information for the Kinect or other time-of-flight devices is a new problem in computer vision, with no published work attempting depth recovery. However, there have been multiple attempts at missing data estimation in various fields. Kokaram, A.C et al [2] present an interpolation technique to fill obscured patches in image sequences. Similarly, in the field of Bioinformatics, Shigeyuki Oba et al [3] propose Bayesian Principal Component analysis to estimate missing data in gene expression profiling. In the same field, a study by Hyunsoo Kim[4] et al attempts to estimate gene expression data using imputation methods based on least squares formulation.

In this paper, we propose a fusion of both RGBD segmentation, known depth values, and the Hough transform voting scheme. Angela Yao et al [5] have incorporated similar voting in their work on action recognition. The research at the University of Michigan by Min Sun et al [6] also proposes a Depth Encoded Hough Voting for object detection. But, to our knowledge no one has used segmentation along with voting and depth, and to that effect our algorithm is original. The implication of our algorithm is described in Section 4 of [7], which discusses at length Surface Based Segmentation and highlights techniques to optimize segmentation using depth data.

2 Algorithm

The project is divided into 5 stages. First, we discuss the calibration process for getting the depth field to align with the RGB image. We then discuss the filtering step used to recover small missing depth information, and to smooth the depth map. Then, we discuss the RGBD segmentation process, and how we incorporated the depth information into the Quickshift algorithm. We then discuss the recovery phase, a novel algorithm which performs quite well. Finally, we show how we visualized our results.

2.1 Capturing

The Kinect captures both image and depth of the world. It captures depth by projecting infra-red patterns into the world and measuring the displacement of the pattern. While this is less accurate than laser based depth capturing devices (like LIDAR), it does make the Kinect more affordable. Since the infra-red and RGB camera are not in the same location, and almost no processing of the depth is performed on the Kinect device, the depth and RGB data suffer from a parallax effect. This must be corrected before processing the image.

To calibrate the RGB and depth information, we use projective geometry. Little information is found on the calibration process, since most people use libraries that do this for them. Since we opted to only use the basic raw USB library, we had to perform the correction ourselves.

The raw depth information received from the Kinect is an 11-bit value in the range of $[0, 2047]$. We use equation (1) to convert the raw depth information into world meters (with $rd_{i,j}$ being the raw depth value at the i and j position).

$$Z_{i,j} = \frac{1}{3.32788\, rd_{i,j}} \tag{1}$$

We then project the depth information into world space, by using equation (2)

$$W_{i,j} = f\left(\begin{pmatrix} i \\ j \end{pmatrix} Z_{i,j} - c\right) \tag{2}$$

With $c = \begin{pmatrix} 339.308 \\ 242.739 \end{pmatrix}$ and $f = (0.00168289 \;\; 0.00169193)$ being the depth cameras' intrinsics.

We then transform the world data measured by depth sensor into world data measured by RGB sensor, this requires multiplication by the rotation and translation matrix to transform between sensors as done in equation (3)

$$X(i,j) = A \begin{pmatrix} W_{i,j} \\ - \\ Z_{i,j} \\ 1 \end{pmatrix} \tag{3}$$

With A being the affine transformation

$$\left(\begin{array}{ccc|c} 0.99984 & -0.00147 & -0.0174 & 0.0199 \\ 0.00126 & 0.99992 & -0.0122 & -0.0007 \\ 0.01748 & 0.01225 & 0.09990 & -0.0011 \\ - & - & - & - \\ 0 & 0 & 0 & 1 \end{array}\right) \tag{4}$$

The point is projected into RGB coordinates from homogeneous coordinates by using Equation (5).

$$RGBZ_{i,j} = f \begin{pmatrix} \frac{X(i,j).x}{Z_{i,j}^{-1}} \\ \frac{X(i,j).y}{Z_{i,j}^{-1}} \end{pmatrix} + c \tag{5}$$

With $Z_{i,j}^{-1} = \frac{1}{X(i,j).z}$, $c = \begin{pmatrix} 328.942 \\ 267.4806 \end{pmatrix}$ and $f = (529.215 \;\; 525.563)$ being the intrinsics.

In more sophisticated software, such as the found on the Xbox, the calibration is performed by having a user place a supplied card in front of the Kinect, with the property of the card is that keypoints on it can be detected by both the infra-red and RGB sensor. A low cost solution to this is to print a checker board pattern and place it under a lamp. The dark boxes get hot and can be picked up by the infra-red sensor and those form lines that are matched with the RGB image via a hough transform and RANSAC. Since calibration is not the objective of this project, we initially found the values for the camera matrices on the internet and tweaked to better our results.

Figure (2) shows that without calibration, the depth pixels do not correspond to the RGB pixels. When projected in RGB space, there is no depth pixel values for some of the RGB pixels. Essentially what the homography is performing is projecting the depth value into world space and then projecting back to RGB space.

2.2 Filtering

The captured depth information contains many small missing values. This is due to the inherit inaccuracy resultant from the way the Kinect captures images. The raw captured data contains both noise and missing depth values – due to occlusions. To fill in those gaps with use a fast mean filter (borrowed from the graphics shaders' world[8]).

The idea of the fast mean filter is to recognize that that performing the mean filter on a $(2r+1) \times 1$ window and then a $1 \times (2r+1)$ window approximates the full mean filter for a $(2r+1) \times (2r+1)$ window for small values of r – i.e. mean filter is separable for small r which was 5 in our case.

Figure 1: Capture image with and without calibration. The white lines denote the edges of the depth capture. Note that in the figure on the right that the edges match with the objects, but one looses information for many pixels during the projection (purple pixels).

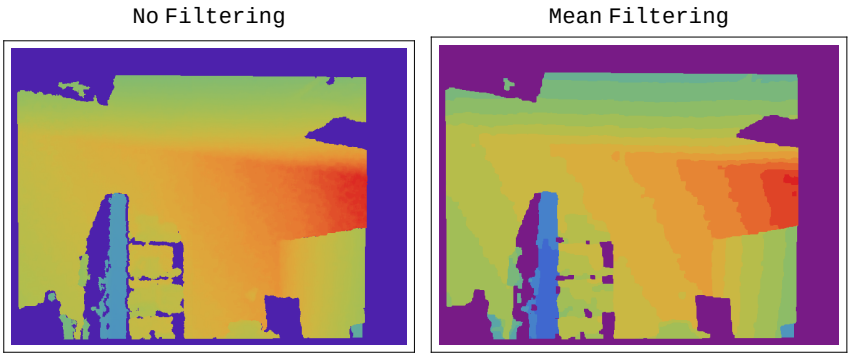

Figure 2: Capture image contains many small holes. We use a fast mean filter and then quantization to remove the small holes while preserving the edges.

We then perform a quantization which is done using integer arithmetic by noting that the depth values are in $(0, 5)$ meters. We use the following formula to quantize the depth into 20 bins

$$Q(Z_{i,j}) = \frac{\lfloor 4Z_{i,j} \rfloor}{20}$$

2.3 Augmented QuickShift

Once the values are quantized, we use QuickShift[9,10] to segment. QuickShift is a mode seeking optimization algorithm, similar to MeanShift, it computes the Parzen density function. Unlike MeanShift, which performs a gradient descent to seek the modes of the distribution, QuickShift uses nearest neighbours. Pixels that converge to the same mode are in the same segment.

The Parzen density function for Quickshift is defined for RGB images only, therefore we augment it to incorporate the depth information. This is seen in Equation (6).

$$P(x) = \frac{\sum_{i=-r}^{r} \sum_{j=-r}^{r} e^{-\frac{|I - I_{ij}|^2}{4\sigma^2}} + e^{\frac{-Max\left[D_{ij}^2\right]}{2\sigma^2}}}{2\pi(2r+1)^2\sigma^2} \tag{6}$$

with $r = 3$ being our window radius, $I_{i,j}$ being the RGB pixel value at i, j, and $D_{i,j}$ being the depth at coordinate i, j. The RGB distance between pixels account not only for the pixel values, but also the i, j coordinates.

The reason for choosing the maximum depth is because some of the depth is missing and have 0 value, but the adjacent depth values would be valid. Furthermore, if a patch does not have a depth value, then the segmentation is not affected – the location of the modes of a distribution do not change if you multiply a scalar by the distribution.

A by-product of the augmented Quickshift is getting larger clusters as seen in Figure (3). This allows us to have much bigger superpixels that do not cross depth boundaries (a requirement to have a better estimate of missing depth information).

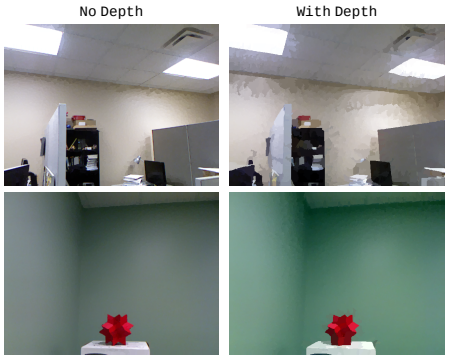

Figure 3: On the left is the result of segmentation without augmenting depth information, the right shows the segmentation with augmented depth. The augmented Quickshift gives larger clusters allowing us to better estimate missing depth information in the next step.

Initially, when picking the segmentation algorithms we looked at Watershed and Meanshift. While both would have provided us with the correct output, the complexity of the algorithm and inability to perform in real time made us look elsewhere. Quickshift was chosen because it was a new algorithm, had a GPU implementation[10], and was small enough (around 400LOC) to understand and modify. Furthermore, in our measurements, Quickshift was regularly $5\times$ faster than Meanshift. We further improved our performance by implementing the visualization in C and shifting the algorithm to use the GPU — this was done based on code from [10].

2.4 Recovering Depth

The depth recovery is our main contribution in this project. We use an iterative process that propagate values from known depth values to unknown depth values in the same segmented cluster. To speed-up the algorithm for large windows, we use a MonteCarlo inspired method of sampling K values. Furthermore, we use a simple Hough voting scheme to estimate the missing depth, with depth values in and out of the segmented cluster voting (with appropriate weight to the vote). So far as we can tell, this algorithm is original.

The algorithm is detailed in Figure (5). A bin is considered missing if its depth value is bellow $tol = 0.0001$, and, after experimenting with a few values, we chose $M = 50$, $MAXITER = 100$, and $K = 40$.

The algorithm performed very well, and with the exception of the image corners, it was able to get sensible estimates for depth. Because this processes is related to a diffusion, the results automatically have the depth information interpolated. The result can be seen in Figure (4).

```
For m from 1 to M do:
    For each pixel coordinate [i,j] do:
        If Depth[i,j] is Missing then:
            Let bins = {0, 0, ..., 0}
            Let maxIter = 0
            For k from 1 to K and If maxIter < MAXITER do:
                maxIter++
                Let rx = RandomInteger in {-2*(m+1), 2*(m+1)}
                Let ry = RandomInteger in {-2*(m+1), 2*(m+1)}
                If Depth[i+rx, ry+ry] is Missing then:
                    Continue
                EndIf
                If I(i, j) in same cluster as I(i+rx, j+ry) then:
                    bins[Depth[i,j]] += 1
                Else
                    bins[Depth[i,j]] += 1/(rx^2 + ry^2)
                EndIf
            EndFor
            If maxIter != MAXITER then:
               newDepth[i,j] = ArgMax[bins]
            EndIf
        EndIf
    EndFor

    Depth = newDepth

EndFor
```

Figure 4: The recovery algorithm uses a reverse iterative diffusion process where values from known pixels are propagated to unknown pixels.

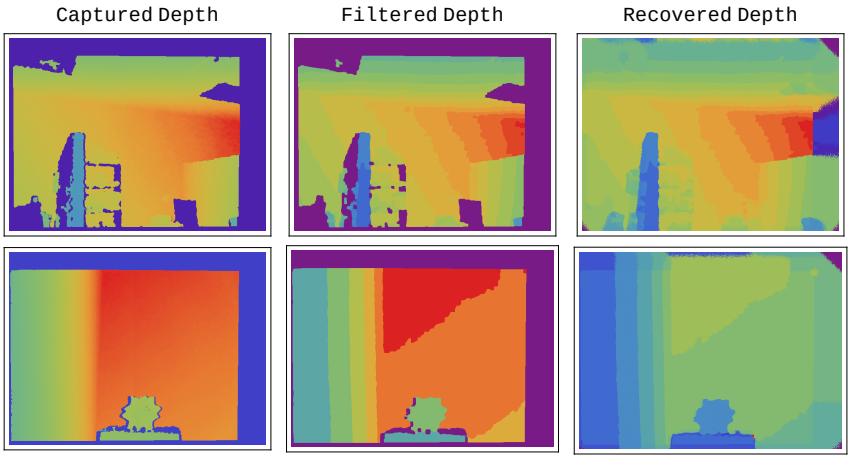

Figure 5: The reverse iterative diffusion process is able to estimate all missing depth information with the exception of the corner values. The figures show the transition for raw captured depth, to filtered depth, to recovered depth.

2.5 Visualizing

While everything was written in C, the visualization is done using *Mathematica*'s graphics language (since it would have been difficult to do it in a few lines in any other language). Given the recovered depth and RGB information, we first convert our depth values into a 3-tuple $(i, j, d_{i,j})$

```
indexedData = Flatten[
    Table[{ii, jj, depth[[ii, jj]]}, {ii, 1, 480}, {jj, 1, 640}], 1];
```

Then, we convert our (r, g, b) color to an `RGBColor` value. Finally, we plot each point with its corresponding vertex color. The result is shown in Figure (6)

```
colors = Map[RGBColor, Flatten[rgb, 1]/255.0];
Graphics3D[Point[indexedData, VertexColors -> colors]]
```

Figure 6: After depth reconstruction, the depth map is plotted with color information from the RGB image using *Mathematica*'s graphics language.

3 Applications

Recovering the missing depth, and getting a smoothed representation of it, from noisy input data is applicable to fields which rely on, make use of, or can be more robust with depth information. These range from object detection for human-computer interaction, to grasping point for robots, topological mapping for graphics, or better segmentation for vision. Certainly, our contribution can be applied to fields outside vision and is not confined to estimating depth.

3.1 Object Detection

For understanding the geometry of an object as a function of its depth values relative to the sensor, it is vital to have both a smooth and accurate estimate of the depth map. An extension of our work would be training a dataset of objects on not only on color data, but also depth data. Even though the depth value would be dependent on the location of the Kinect, it is possible to make the training set depth invariant. Segmenting out the object and then scaling the test depth data to match the training data could do this. This could significantly enhance the accuracy of object detection.

3.2 Grasping Points

Detecting accurate object geometries is a highly researched topic in Robotic Manipulation. Current techniques incorporate silhouette detection, expensive LIDAR technology as well as object detection using a RGB camera. Roboticists are now exploring the Kinect for these applications – making robots more affordable. Since binary segmentation is trivial with the recovered depth (a 2 cluster problem), using our technique would help robots not only maneuver in space, but also interact with objects by accurately identifying grasping points of objects.

3.3 Topological Mapping

Most research on robot mapping and exploration makes use of expensive sensors, such as scanning laser range finders and powerful computers. However, now the Kinect has turned out to be an invaluable alternative. A study at the University of California at Berkeley places a Kinect on a

quadrotor and traverses the environment to map out obstacles in the world. By obtaining smoother depth information we can create accurate maps and optimize obstacle avoidance.

3.4 Better Segmentation with Augmented QuickShift

Even without using the depth recovering step, our augmented QuickShift performs better at segmentation that regular QuickShift or MeanShift. This is because we enforce same clusters to be on the boundary, and thus are able to generate larger clusters. A result of this better segmentation is shown in Figure (7). As can be seen, the checker board is much better clustered than the original QuickShift – at about the same computational cost.

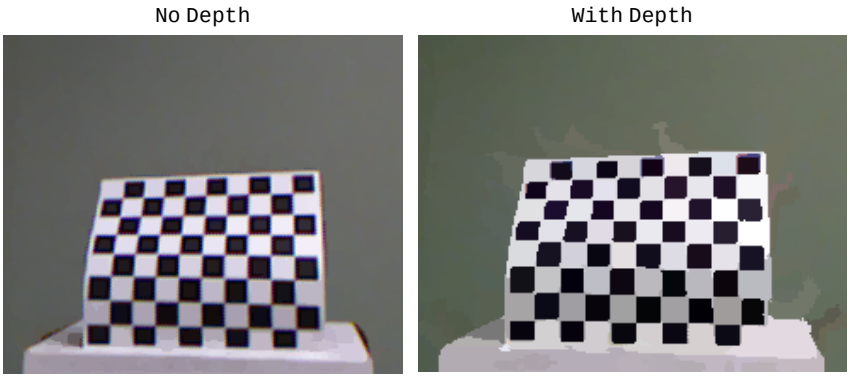

Figure 7: By using our augmented QuickShift, we are capable to get much bigger and accurate segmentations.

4 Future Work

While we use the GPU for QuickShift, we do not utilize it for the other parts of the algorithm. Yet since memory bandwidth is the bottle neck in GPU computation, one needs to migrate all the steps to use the GPU (keeping data on the GPU and reusing it). Both regularly binning and mean filtering are simple to implement on the GPU – and, the techniques were borrowed from the graphics shader field. Our depth recovery phase is also highly parallizable both due to its iterative nature and our use of MonteCarlo based methods for sampling data. The algorithm can be implemented on textures to minimize the cost of bank conflicts and coalescing associated with random memory access on the GPU.

One undesired by-product of our method is that the recovered depth is quantized. A post-processing step that uses the recovered depth data and re-projects it onto the original filtered data can be done. It is likely that this will also involve borrowing techniques from graphics for estimating the local surface for a small set of points in a point cloud.

Finally, since our algorithm is original, we would like to compare it against out missing value estimation algorithms found in other fields – such as machine learning. Based on our research, none were applied to depth recovery.

Citations

[1] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, Andrew Blake. "Real-Time Human Pose Recognition in Parts from Single Depth Images". Microsoft Research Cambridge & Xbox Incubation.

[2] Kokaram, A.C., Morris, R.D., Fitzgerald, W.J., Rayner, P.J.W.; "Interpolation of missing data in image sequences"

[3] Shigeyuki Oba, Masa-aki Sato, Ichiro Takemasa, Morito Monden, Ken-ichi Matsubara and Shin Ishii. ”A Bayesian missing value estimation method for gene expression profile data”

[4] Hyunsoo Kim, Gene H. Golub and Haesun Park. ”Missing value estimation for DNA microarray gene expression data: local least squares imputation”

[5] Angela Yao, Juergen Gall & Luc Van Gool. ”A hough transform-based voting framework for action recognition”.

[6] Min Sun, Gary Bradski, Bing-Xin Xu, Silvio Savarese. Depth-Encoded Hough Voting for Joint Object Detection and Shape Recovery

[7] Nikhil R Pal, Sankar K Pal, A review on image segmentation techniques, Pattern Recognition, Volume 26, Issue 9, September 1993, Pages 1277-1294,

[8] Morgan McGuire. (2008) A fast, small-radius GPU median filter. In ShaderX6.

[9] A. Vedaldi and S. Soatto. Quick Shift and Kernel Methods for Mode Seeking, in Proc. ECCV, 2008.

[10] Really quick shift: Image segmentation on a GPU, B. Fulkerson and S. Soatto. In Proceedings of the Workshop on Computer Vision using GPUs, held with the European Conference on Computer Vision, September 2010.