1 Kinect Depth Inpainting and Filtering

I. Problem

Figure 1. A typical depth map retreived from the Kinect sensor.

The Kinect sensor is a technology that recently made its way into the field of computer vision. In many ways the sensor is revolutionary, because it provides unprecedented ease of access to depth data. In practice, however, the data collected from the sensor is unusable for many computer vision applications. The Kinect RGB camera, IR camera, and IR projector are positioned on different parts of the device, so when one aligns the depth information with the point of view of the camera, the result is an image similar to the one above. The black regions are either occluded from the point of view of the IR camera, or absorb IR light and so contain no depth information. Furthermore, surfaces on object boundaries are often perpendicular to the IR camera which results in noise when trying to observe the projected light. The result is that object boundaries are very unstable, and the alignment of depth and color edges is poor. In our project we will implement a package that will make use of the underlying color and temporal information to inpaint holes and stabilize edges, thus preparing the depth information for further applications in computer vision. We believe this is an important and useful step due to the important role object boundaries and edges play in many computer vision applications.

II. Related Work

Our project is concerned with inpainting and noise filtering.

We base our inpainting approach on a paper by Alexandru Telea [1]. This is a commonly used algorithm because it is simple to implement, fast, and produces results of comparable quality to much more complicated and time demanding algorithms. Like most existing inpainting algorithms, the Telea method is designed to fill cracks and removed objects in color images, and does so by propogating the grey levels, intensity gradients, and details like lines from a narrow band of known pixels around the hole into the center. Because the algorithm is designed for generic hole-filling in colored images, it does poorly when applied to our problem out of the box. A result typical of any naive application of an inpainting algorithm to our raw image from Figure 1 is shown at the top of Figure 3. As can be seen, the inpainting extended object boundaries and created some arbitrary shapes that simply do not exist in reality, thus making the image even less useful than in the raw case.

Similar to inpainting, generic filtering approaches fail to take advantage of all of the information available in our specific problem: depth, color, and time. We consider two filtration approaches. The first is based on Joint Bilateral Upsampling by Kopf et. al. [2]. This algorithm takes advantage of the color in a high resolution image to improve the upsampling of a low resolution labeling. Labels from the neighborhood of a pixel that are close in color space have more influence on its label. We apply a similar methodology to take advantage of the available time and color information when filtering the depth image. The second approach is based on median filtering, which is well established in the vision community as an edge-preserving filter. A naive application of this filter is shown in Figure 4. Again, we note that we can make the filter much more effective by taking advantage of the available color information.

III. Approach

Our approach centers on leveraging knowlege about hole and noise formation, as well as additional sources of information like color and time to create better performing algorithms.

A. Fast Marching Method

A popular inpainting algorithm is the Fast Marching Method (FMM) by Alexander Telea [1].

This algorithm initializes a mask over the image, where pixels that are known are labeled with 0, and pixels that are unknown are labeled with ∞. It then finds a narrow band of pixels between the known and unknown regions, placing these pixels into a heap, which is organized according to the function $H(p) = d(p)$, where H is the heaping function and d is the distance of this pixel to the edge of the hole. At each iteration, the algorithm picks the minimum unknown pixel off of the heap, inpaints it with a first-order approximation of the local gradient (for details consult [1]), and updates its neighbors, possibly placing them into the heap. The resulting $H(p)$ function is illustrated in Figure 2.

This algorithm does something that we do not desire - namely, it gives the same priority to "marching" far-away pixels as it does to foreground object pixels. To fix this, we modify the heaping function: $H'(p) = \alpha d(p) - (1-\alpha)z(p_n)$.

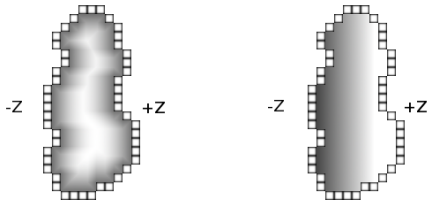

Figure 2. Visualization of our modification of FMM. The grey values inside the hole are representative of the values of $H(p)$, which dictates the order in which the pixels will be filled (darker pixels get picked first). The original approach (left) evenly fills the hole from all directions. Our approach(right) modifies the ordering by giving the pixels closer to a deeper edge (-Z) more priority, thus marching the background across the hole.

Here, $d(p)$ is the distance to the edge of the hole, as before. The other term, $z(p_n)$ is the (possibly inpainted) depth of a neighbor pixel. Finally, α is a constant in the interval $[0, 1]$ that is a user-specified parameter for how much we value these two relative terms. The result of this modification is that pixels which are further into the image (have larger depth) will have smaller heap values, and so will be given priority when choosing the next pixel to inpaint. This is illustrated in the right hand side of Figure 2.

B. Joint Bilateral Filter

While the above algorithm addresses the issue of filling holes in Kinect depth images, we still have the problem of noise around object boundaries. In order to address this, we consider the Joint Bilateral Upsampling filter by Kopf et. al. [2].

A bilateral filter is a combination of a domain kernel, which gives priority to pixels that are close to the target pixel in the image plane, with a range kernel, which gives priority to the pixels which have similar labels as the target pixel. This filter is often useful when one wants to preserve edge information because of the range kernel advantages. The Kopf approach modified the domain kernel by also considering color distances in the domain filter, thus weighing similar-looking pixels more when computing a new label.

Our approach leverages not only color information, but also time information. This is perhaps best explained with an equation:

$z'(p) = \Sigma_{q \in N}(z(q) \times w_z(|z(p) - z(q)|, \sigma_z) \times w_d(d(p,q), \sigma_d) \times w_c(c(p,q), \sigma_c))$

Here, N is a time and space neighborhood around the pixel in the video volume of the image. $z(p)$ is the depth(or label) of the pixel. $d(p, q)$ is the distance between p and q in space and time. The functions w_z, w_d, w_c are zero-mean gaussians with standard deviations $\sigma_z, \sigma_d, \sigma_c$, which are user-specified parameters for how much leeway the labeling has in each of those dimensions. Thus, we take a weighted sum of the spatiotemporal neighborhood of the pixel, allowing pixels that are close in space, time, color, and depth to have more influence on the given pixel. With such a weighing, we expect pixels that are close to the edge that might have the wrong depth label, to be more influenced by neighboring pixels with similar appearance, thus restoring the correct label and aligning depth boundaries with color boundaries.

C. Color-Assisted Median Filter

After implementing the bilinear filter, we found out that it was simply too slow to use in any real-time applications. Capturing enough information to correct edge mistakes requires relatively large neighborhoods (at least 7×7). The result is that many lookups, distance computations, and multiplications have to be done for every pixel.

In light of this, we implemented a second filtering method based on median filtering. The median filter assigns to each pixel the depth given by the median of its neighborhood. In naively applying this method, we found that it did smooth out flicker and noise around edges, but did not do a good job in correcting the object boundaries, as seen in Figure 4. To improve performance, we first trimmed the neighborhood by removing a fraction of the pixels that were furthest from the current pixel in color space, and then took the median over the remaining pixels. Similar to the Bilinear approach, this leveraged the color information with the assumption that pixels similar in color will have similar depths.

IV. Evaluation

Since we do not have access to ground truth data, and constructing a ground truth database would be prohibitively time consuming, we evaluate our approach subjectively on a video that we recorded ourselves. We give images of the first frame of this video as reference for the reader.

The figure at the start of the paper is the raw depth image given to us by the Kinect device, and is illustrative of the problems we wish to solve. The dark black areas are the regions that contain no depth information (holes). Also, a lot of noise can be seen around object boundaries, like the top right part of the head.

Figure 3 is a comparison of applying the naive Telea inpainting method, and our modification of that method ($\alpha = .5$). It is clear that our method outperforms the original, the best illustration being the shoulder on the left side of the image which was incorrectly extended by the naive method, but preserved with our method.

Even in the inpainted image, however, we see some uneven edges around object boundaries (which flicker in the video), and note that the depth boundaries are in some cases incorrect - like in the top right part of the head region, where a hole appears in the person's head. In Figure 4 we demonstrate the results of applying different filtering approaches to the depth image. A naive bilinear filter did not have any effect, so we omit it for the sake of space. This is because it relies on the erroneous depth labels, and no additional sources of information, and so cannot recover from the failures of the inpainting method.

Our modification of the bilinear filter can be seen to improve the inpainted image. We see a really crisp edge on the shoulder on the right side of the image, some repair to the hole in the head, and a step towards the cleanup of the outline of the person. Repetitive applications of the filter do

Figure 3. Naive Telea inpainting method (top) compared to our method (bottom).

completely resolve these issues, but cannot be shown due to space constraints.

In the same figure, we also note that our modification to the median filter does a much better job at aquiring correct edges than the naive median filter. Many parts of the outline are smoother and cleaned up, and the hole in the head, again, begins to disappear.

V. Discussion

In our project, we attempted to leverage additional sources of information, and knowlege about the sources of error, to inpaint and filter Kinect depth maps. As can be seen by Figures 3 and 4, our methods for filling in holes and improving object boundaries are far more successful than existing algorithms. The inpainting algorithm correctly marches the background across holes, but does not necessarily end up with correct object boundaries. Our filtering method help improve inpainting by using color information to better establish edges.

Still, our algorithms do not produce a perfect result, and have some limitations of their own. In the current state, our Bilateral filtering method works far too slowly to be applicable in real time. The median filter addresses this problem, but produces less optimal results in the process.

In general, large filters have to be applied for several iterations to produce high-quality depth maps. The reason for this is that the inpainting method does not take advantage of color information, and compunds the error in object boundary

Figure 4. Comparison of different filtering method. From the top: our bilateral filter, naive median filter, our median filter.

alignment. Then, much more computation time has to be used on filtering to repair these errors. A further step in this project would be to incorporate color in the FMM inpaint method, so we produce color edge-aligned inpainted images right away. Then, much cheaper filtering methods can be used to get the same quality of results.

VI. References

1) Telea A. An Image Inpainting Technique Based on the Fast Marching Method. Journal of Graphics Tools. 2003;9(1):25-36.
2) Kopf J, Cohen MF, Lischinski D, Uyttendaele M. Joint Bilateral Upsampling. Acm Transactions On Graphics. 2006.