17th International Conference on Machine Vision Applications (MVA)
Fully Online, July 25–27, 2021.

Synthetically Generating Motion Blur in a Depth Map from Time-of-Flight Sensors

Bryan Rodriguez, Xinxiang Zhang, and Dinesh Rajan
Department of Electrical and Computer Engineering
Southern Methodist University, Dallas, TX 75205, USA
brodrigu@smu.edu, xinxiang@smu.edu, rajand@smu.edu

Abstract

Motion blur is a common artifact that affects imaging systems. Synthetically creating motion blur in two-dimensional (2D) images is a well-understood process and has been used to develop deblurring systems. However, there are no well-established techniques for synthetically generating motion blur within three-dimensional (3D) images since the behavior of motion blur within 3D images, such as depth maps and point clouds, is not as well-understood. In this work, we develop a simple and accurate framework to synthetically generate motion blur within depth maps that accurately captures the behavior of the real motion blur that is encountered using a Time-of-Flight (ToF) sensor. We develop a probabilistic model that predicts the location of invalid pixels that are typically present within depth maps that contain real motion blur. We also introduce a method to quantify the performance of a synthetic motion blur filter for depth maps based on a comparison between a depth map with synthetic motion blur and a depth map with real motion blur. Our results indicate that our framework is able to achieve an average Boundary F1 (BF) score of 0.8864 for invalid pixels for synthetic radial motion blur and a BF score of 0.8401 for synthetic linear motion blur.

1. Introduction

With the growth in the number of 3D sensor technologies in the market, understanding their behavior is critical to widespread adoption. This work presents a methodology for synthetically generating motion blur within a depth map that mimics the real motion blur observed when using a ToF sensor. The proposed models can be used by researchers to develop more effective and adaptive deblurring techniques that work in applications including autonomous vehicles, drones, robotics, and logistics. In these applications, a ToF sensor or objects within the field of view of the ToF sensor may be moving with respect to each other. The relative movement between the ToF sensor and other objects can create a motion blur effect which increases the number of invalid pixels and flying pixels that are present in a depth map and distorts the appearance of objects in the depth map captured by a ToF sensor [1].

Synthetically creating motion blur in 2D images is a well-understood process [2] [3]. In traditional 2D images, motion blur appears as a softening of edges within a 2D image along a motion path. The motion blur in depth maps is distinct from 2D images because of the presence of invalid pixel values and flying pixels which do not exist in 2D image motion blur. To our knowledge, there are no well-established techniques for synthetically generating motion blur within 3D images such as depth maps and point clouds. In previous works, motion blur in depth maps has been observed as a baseline for comparing different 3D sensing technologies or for evaluating the performance of various deblurring algorithms [1][4]. However, these works typically focus on how to minimize the effects of motion blur and they do not provide any insight about how to synthetically create motion blur. For example, radial motion blur is captured using both a Kinect v1 sensor that uses structured light and a Kinect v2 sensor that uses ToF [4]. For each Kinect sensor, a depth map is generated for a flat fan blade, a Siemens Star, while it is static and while it is rotating and the distortion between the static and rotating depth maps is computed. The focus of reference [4] is simply to compare the amount of distortion that is present in the depth maps generated by the different Kinect sensors.

In this work, we introduce a probabilistic model that is used to predict the location of invalid pixels, also referred to as zero-value pixels, that are present when motion blur occurs. Several types of motion blur exist such as radial motion, linear motion, out-of-focus blur, or a combination of blur types [5]. Capturing 3D images of motion blur in a real-world environment is challenging because the motion and the speed of objects are not always easily controllable. Using our framework, a synthetic radial or linear motion blur can be applied to depth maps of static objects. If desired, the blurred depth map can then also be converted into a point cloud that will include the applied motion blur. The ability to synthetically generate motion blur in 3D images enables future works to create test benches for evaluating algorithms for deblurring this motion blur.

In summary, this work contributes to the state of the art by: 1) Demonstrating a framework that can be used to synthetically generate motion blur in depth maps that mimics the appearance and behavior of real motion blur that can be observed using a ToF sensor, 2) Developing a probabilistic model that predicts the location of invalid pixels that are typically present in depth maps that contain a real motion blur, and 3) Quantifying the performance of a motion blur filter for depth maps based on a comparison between a depth map with synthetic motion blur and a depth map with real motion blur.

This paper is organized as follows. Section 2 discusses our methodology for synthetically generating motion blur within a depth map. Section 3 discusses our experimental setup and results. Section 4 provides concluding remarks.

2. Methodology

Motion blur appears in depth maps as an increase in the number of zero-value pixels (i.e. invalid pixels) and flying pixels that are present near depth discontinuities within a scene [1]. In general, ToF sensors determine depth values based on the amount of time it takes for light that is emitted from the ToF sensor to return to the ToF sensor after reflecting off a surface within a scene. As an object moves with respect to the ToF sensor, the light that is reflected near the edges of the object may result in erroneous depth values that do not accurately represent the surface of the object. The increase in the number of zero-value pixels and flying pixels results in fewer pixels on the surface of the object which tends to distort the appearance of the object within depth maps. For example, the surface area of an object may reduce and the size of openings may increase.

In this work, we propose an approach for synthetically applying radial motion blur and linear motion blur to the depth map of a static object. After synthetically applying the motion blur to the depth map, the resulting depth map mimics the appearance and behavior of a depth map that would be observed if the object were moving. The key steps to the proposed methodology are as follows. 1) The ToF sensor is configured to generate a depth map for an object. In this configuration, both the ToF sensor and the object are static. 2) After generating the depth map, we apply either a radial blur filter or a linear blur filter to the depth map. The radial blur filter and the linear blur filter are both implemented using 2D spatial filters that are applied to the depth map similar to how motion blur filters are traditionally applied to 2D images. In our experiments, the motion blur filter is applied to a user-defined region of interest within the depth map. 3) We then use our probabilistic model to predict the locations of new zero-value pixels within the depth map. The probabilistic model predicts whether a pixel location within the depth map should have a zero value after the motion blur filter is applied based on its neighboring pixels. 4) The motion blurred depth map is then updated to include the zero-value pixels predicted using the probabilistic model.

2.1. Motion blur filters

In our work, the initial depth map, $I(x, y)$, is first obtained from the TOF sensor where (x, y) identifies pixel locations within the depth map. A region-of-interest is then defined within the depth map. In the case of the radial motion blur filter, a circular region-of-interest is defined around our object. The depth map that is obtained from the TOF sensor includes a combination of zero-value and non-zero value pixels. The zero-value pixels correspond to invalid pixels where the TOF sensor was unable to determine a depth value. The non-zero pixels correspond to locations where the TOF sensor was able to determine depth value. Before we apply our motion blur filter, we first perform a bilinear interpolation process to assign the zero-value pixels a depth value based on their neighboring pixels as defined in Section 2.2. This interpolation process allows the motion blur filter to be applied more accurately to the depth map.

After interpolation, we apply a 2D spatial filter to the depth map to simulate the appearance of a radial motion blur. In this work, we use a 2D spatial filter from JH Labs [5] to average neighboring pixels to apply a radial blur filter to our depth map. The radial motion blur filter blurs the depth map by blurring each pixel of the depth map along a tangential direction to the path of the target pixel. The input parameters for the radial motion blur filter are a spinning center, a spinning radius, and a rotation angle. In this experiment, the filtering center is denoted as the pre-defined center location of the spinning object. The spinning radius is set as the diagonal length of the region-of-interest, where the depth map pixels are only filtered (blurred) within the region-of-interest. The rotation angle is proportional to the spinning speed of the object and can be selected based on the desired level of simulated speed. A linear motion blur filter can be applied to the depth map using a process similar to the process described in this section. The linear motion blur filter input parameters are the linear blur direction and a blur distance. The linear blur direction is the motion path of the object within the depth map. In our experiments, the direction is set to a horizontal direction that simulates the object moving horizontally across a surface. The blur distance is proportional to the moving speed of the object and can be selected adaptively.

2.2. Probabilistic model for predicting zero-value pixels

In this section, we develop a probabilistic model that predicts the location of zero-value pixels in the motion-blurred depth map based on values of neighboring pixels. When real motion blur occurs in the depth map from a TOF sensor, there is an increase in the number of zero-value pixels that are present near discontinuities and edges in the depth map. As the speed of the object increases, the number of zero-value pixels also increases. To predict the locations of new zero-value pixels our probabilistic model uses three assumptions. 1) The number of neighboring pixels that are in the window for predicting the final state of a pixel is proportional to the speed of the object. 2) For each pixel, the neighboring pixels in the window with the same state (i.e. 0 or 1) will contribute more to the final state of the pixel. The final state of the pixel is also more likely to be affected by neighboring pixels with a state of zero because zero-value pixels tend to appear in clusters. 3) The pixels with an initial state of zero are likely to remain in state zero.

We first convert the original depth map into a binary depth map, $I_b(x, y)$, by assigning non-zero value pixels within $I(x, y)$ to a state of one and assigning zero-value pixels within $I(x, y)$ to a state of zero. Using our probabilistic model generally involves iteratively selecting a root pixel, R, where $R \in I_b(x, y)$, within the defined region-of-interest that was defined in Section 2.1 and then using the probabilistic model to predict whether the root pixel should be a zero-value pixel in the motion-blurred depth map based on its neighboring pixels. After generating the binary depth map, we then iteratively select root pixels from within our region-of-interest. For each selected root pixel, we define a neighborhood window whose length is proportional to the speed of the object. Thus, a larger window length is used to simulate faster moving objects. The window is orientated to be along the motion path of the object. When the direction of motion for the object is known, the window is positioned to include neighboring pixels that precede the root pixel in the direction of motion. The window height is set to a fixed value, for simplicity. In this work, the window height was set to a value of one pixel down. When the direction of motion for the object is unknown or random, the window is positioned to include neighboring pixels that both precede and follow the root pixel. The window height does not change based on the direction of motion. Instead, the window is oriented to be tangential to the direction of motion. As an example, for a given root pixel, we use the defined window to identify a group of neighboring pixels, $[N_0, N_1,, \ldots, N_n]$, where $[N_0, N_1,, \ldots, N_n] \in I_b(x, y)$.

After identifying the neighboring pixels for the root pixel, we then apply our probabilistic model to the identified neighboring pixels. We first define a potential function that captures the interaction between the state of the neighboring pixel and the state of the root pixel as:

$$\varphi(s_R, s_{N_i}) = \begin{cases} (1 - |s_{N_i} - s_R|)p_x + |s_{N_i} - s_R|(1 - p_x), s_R = 0 \\ (1 - |s_{N_i} - s_R|)(1 - p_x) + |s_{N_i} - s_R|p_x, s_R = 1 \end{cases} \quad (1)$$

where $\varphi(s_R, s_{N_i})$ is the potential function of the i^{th} neighboring pixel, s_{N_i} is the state of the i^{th} neighboring pixel, and s_R is the state of the root pixel. In this study, p_x is set to 0.9. We also consider the prior knowledge of the root pixel by determining an initial probability as p_R for the root pixel. The probability p_R is determined as:

$$p_R = \begin{cases} p_y, & s_R = 0 \\ 1 - p_{y,} & s_R = 1 \end{cases} \quad (2)$$

where p_R is the initial probability and s_R is the initial state of the root pixel. In our experiments, p_y is set to 0.6. Effectively, this process ensures that a higher probability weight is used when a neighboring pixel has the same state as the root pixel and when the root pixel has an initial state of zero. We then predict a probability for each root pixel based on the modeled potential function for each of its neighboring pixels and the initial probability for the root pixel. The predicted probability $p_{\hat{R}}$ is calculated as:

$$p_{\hat{R}} = \frac{\sum_{i=0}^{n} \frac{\varphi(s_R, s_{N_i}) * p_R}{\rho}}{n} \quad (3)$$

where ρ is the normalization factor. We then assign the probability to the root pixel that corresponds with the likelihood that the root pixel will be a zero-value pixel in the blurred depth map. This process is repeated to assign probabilities for all of the root pixels within the region-of-interest. We then apply a threshold to identify root pixels that will become zero-value pixels in the blurred depth map. The following expression describes how the threshold is applied:

$$s_{\hat{R}} = \begin{cases} 0, & p_{\hat{R}} > t \\ 1, & p_{\hat{R}} \le t \end{cases} \quad (4)$$

where $s_{\hat{R}}$ is the predicted state for root pixel, $p_{\hat{R}}$ is the predicted probability for the root pixel, and t is the threshold value which is set to 0.05. Note that the selection of p_x, p_y, and t is based on a separate validation set. Once the zero-value pixels are identified, the blurred depth map described in Section 2.1 is updated to include the predicted zero-value pixels from the probabilistic model.

3. Experiments

3.1. Hardware configuration

In our experiments, we use a Kinect v2 [6] sensor to generate depth maps for our experiments. This ToF sensor has a resolution of 512x424 pixels with a framerate of 30 frames per second [7]. We used OpenKinect libraries [8] to capture depth information and MATLAB 2020 [11] for implementing our synthetic motion blur process. In our experiments, we disabled both the Bilateral filter and the Edge-aware filter for the Kinect v2 while capturing depth information, to ensure raw depth information is captured [9]. In our experiments, the Kinect v2 sensor is positioned in a fronto-parallel configuration at a distance of 0.78m away from the front surface of our object.

For our experiments with radial motion blur, we used a custom radial motion device. Our radial motion device includes a 381mm diameter Siemens Star with six flat fan blades. The Siemens Star is attached to a 5v DC motor that is controlled using an Arduino Uno. We collected data while rotating the Siemens star between 60-135 RPM. For our experiments with linear motion blur, we used a custom linear motion device. Our linear motion device includes a box that is attached to a linear rail system. The front surface of the box includes three vertical openings that are each about 76mm by 254mm in size.

In future works, these experiments can be extended to include more complex scenarios with motion that is not parallel to the image plane or with multiple objects.

3.2. Evaluating synthetic motion blur performance

Comparing a depth map with synthetic motion blur to a depth map with real motion blur is challenging because the position of objects may be different in both depth maps. Hence, traditional direct comparisons between two depth maps cannot be made without additional considerations to ensure that the position of objects within the depth maps is the same. In this work, we demonstrate a framework that can be used to compare a depth map with a synthetic motion blur to a depth map with a real motion blur. This process involves capturing a series of depth maps of a static object in different positions that would occur when the object is in motion. We then generate a depth map of the object in motion with real motion blur. The depth maps of the static object are then compared to the depth map with the real motion blur to identify the closest match based on the position of the object. After identifying the depth map of the static object that best matches the depth map of the object in motion, we then apply our synthetic motion blur to the depth map. The depth maps can then be directly compared to quantify the similarity between the two depth maps. This process enables a direct comparison between a depth map with synthetic motion blur and a depth map with real motion blur since we are able to align the position of an object before making the comparison. The ability to compare a depth map with synthetic motion blur and a depth map with real motion blur enables future works to evaluate the performance of other motion blur algorithms.

To evaluate the performance of the zero-value pixel predictions from the probabilistic model described in Section 2.2, we use the Boundary F1 (BF) score with an error tolerance of 2 pixels [10]. Also, to evaluate the performance of non-zero value pixels from Section 2.1 after updating the blurred depth map with the predicted zero-value pixels, we are using a root mean-square error (RMSE) between the synthetic motion blur depth map and the real motion blur depth map and an RMSE ratio between their difference and the real motion blur depth map.

3.3. Motion blur experiment results

Fig. 1 illustrates binary depth maps that show the locations of zero-value pixels and non-zero value pixels. In Fig. 1, the zero-value pixels are shown in white and the non-zero value pixels are shown in black. The first row of Fig. 1 shows a depth map of the Siemens Star in different static positions without motion. In our experiments, one of the fan blades of the Siemens star is marked with an Infrared (IR) reflective marker that artificially creates a hole in the fane blade. Our results show that when motion blur occurs, the number of zero-value pixels that are present in the IR reflective marker and that are near the edges of the fan blades increases. In addition, our results also show that when motion occurs, the surface area of the fan blades appeared reduced due to the increase in the number of zero-value pixels and flying pixels. The reason for the reduction in surface area is because the motion of the fan blades causes pixels near the edges of the fan blades to become zero-value pixels or flying pixels that are no longer on the surface of the fan blade. As the speed of the Siemens Star increases, the amount of motion blur increases which also increases the number of zero-value pixels and flying pixels that are present near the edges of the fan blades in the depth map. Fig. 1 also shows examples of real motion blur at various speeds and their corresponding synthetic motion blur. As seen in Fig. 1, as the speed of the Siemens Star increases (shown from left to right), the number of zero-value pixels that are present in the IR reflective marker and that are near the edges of the fan blades increases.

Table I shows the performance of the proposed synthetically generated motion blur process. For each speed, we evaluate the performance of our framework by comparing depth maps of the Siemens star in thirty-six different positions. As shown in Table I, as the speed of the Siemens star increases, the BF score decreases and the RMSE increases. Our results show that our framework for synthetically generating radial motion blur is able to achieve an average BF score of 0.8864, an average RMSE of 8.7135, and an average RMSE ratio of 0.0090 over a range of speeds between 60 RPM and 135 RPM.

TABLE I. Synthetic motion blur performance

Speed	BF score	RMSE	RMSE Ratio
60 RPM	0.9492	8.0917	0.0079
100 RPM	0.8758	9.3353	0.0088
135 RPM	0.8342	11.2900	0.0104

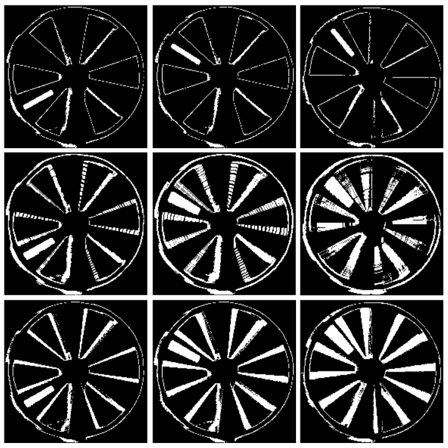
Fig. 1. Siemens Star with without motion (top), synthetic motion blur (middle), real motion blur (bottom) at 60 RPM (left), 100 RPM (center), and 135 RPM (right)

Fig. 2 shows a comparison between the depth map of our box without motion, the depth map of the static box with synthetic motion blur, and the depth map of the box in motion. Our results show that our framework for synthetically generating linear motion blur is able to achieve an average BF score of 0.8401, an average RMSE of 10.5410, and an average RMSE ratio of 0.0101.

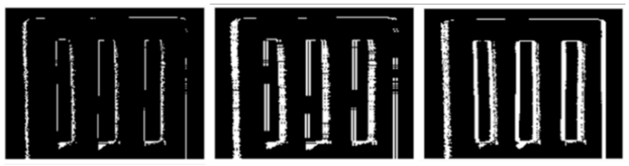
Fig. 2. Static box (left), synthetic motion blur (middle), and real linear motion blur (right)

4. Conclusions

In this work, we present a framework that can be used for synthetically generating motion blur in depth maps that mimics the behavior of real motion blur that is observed using a ToF sensor. This work introduces a probabilistic model to predict the location of zero-value pixels that are present when motion blur occurs. This work also introduces a process for evaluating the performance of a motion blur filter for depth maps by comparing a depth map that has synthetic motion blur to a depth map with real motion blur. One of the limitations in our framework is that our framework relies on the presence of zero-value pixels in the static depth map to create new additional zero-value pixels in our depth map with synthetic motion blur. Future work can work to incorporate more sophisticated models to generate zero-value pixels.

5. REFERENCES

[1] M. Lindner, and A. Kolb. (2009). Compensation of Motion Artifacts for Time-of-Flight Cameras. Lect. Notes Comput. Sci.. 5742. 16-27. 10.1007/978-3-642-03778-8_2.

[2] R. C. Gonzalez and R. E. Woods, *Digital image processing*. New Jersey: Parson, 2008.

[3] R. C. Gonzalez, R. E. Woods, and S. Eddins, *Digital Image Processing Using MATLAB*. New York: Gatesmark, 2010.

[4] H. Sarbolandi, D. Lefloch, and A. Kolb. “Kinect range sensing: Structured-light versus Time-of-Flight Kinect.” Comput. Vis. Image Underst. 139 (2015): 1-20.

[5] Java Image Processing, JH Labs, Accessed on December 11, 2020, Available: http://www.jhlabs.com/ip/blurring.html.

[6] A. Kolb, E. Barth, R. Koch, and R. Larsen. (2010). Time-of-Flight cameras in computer graphics. J. Computer Graphics Forum, 29, 141-159.

[7] J. Jiao, L. Yuan, W. Tang, Z. Deng, and Q. Wu. (2017). A Post-Rectification Approach of Depth Images of Kinect v2 for 3D Reconstruction of Indoor Scenes. ISPRS International Journal of Geo-Information. 6. 349. 10.3390/ijgi6110349.

[8] OpenKinect, OpenKinect Project, Accessed on November 25, 2020, Available: https://openkinect.org/wiki/Main_Page.

[9] A. Cheng and H. Harrison, “Touch Projector,” MIT. Accessed on November 25, 2020, Available: https://tinyurl.com/bx3pfsxt.

[10] G. Csurka, D. Larlus, and F. Perronnin. "What is a good evaluation measure for semantic segmentation?" Proceedings of the British Machine Vision Conference, 2013, pp. 32.1–32.11.

[11] MATLAB. (2020). version 9.9.0. 1467703 (R2020b). Natick, Massachusetts: The MathWorks Inc.