DEPTH BASED INPAINTING

W. G. C. W. Kumara[1], Sanjeewa Gamaralalage[2]

[1] Department of Computer Science and Information Engineering, National Central University, Taiwan (R. O. C),
Email: chinthakawk@gmail.com
[2] Department of Industrial and Systems Engineering, Ohio University, Athens, Ohio, USA,
Email: jayasinghegss@gmail.com

ABSTRACT

An example depth based inpainting for the multi view camera projection is presented. We used the color and depth frames of the Balloons dataset for cameras 1 and 5. Reference frames are then projected to the virtual camera position 3. Next, projected views at the virtual camera position are merged based on the depth values. Criminisi's inpainting method was enhanced with a level term incorporating the depth values. Finally depth based inpainting is applied to fill the occluded holes in the virtual view.

***Key words*:** Inpainting, Depth, 3DTV

1. INTRODUCTION

Multi view TV provides a location dependable 3D view for the users. 3D TV should be able to receive multiple streams to replicate a real world scene, which indeed increase the bandwidth requirement during the transmission. Hence researches are conducted to transmit only a selected number of streams and then to reproduce the intermediate views at the receiving TV [1]. Then it is required to project a frame from the reference image plane to the target image plane. Based on the camera intrinsic and extrinsic matrices, one camera view can be projected to world coordinates, and can then be re-projected to the second view. Relevant equations for this projection are detailed in section 2.

Due to the difference in depth between foreground and background in the real world scene, voids created in virtual views after the projection. These voids can be filled using inpainting. Criminisi first introduced the inpainting algorithm [2] then followed by several other researchers [3],[4],[5]. Following texts explain basics of Criminisi's algorithm referring to Figure 1.

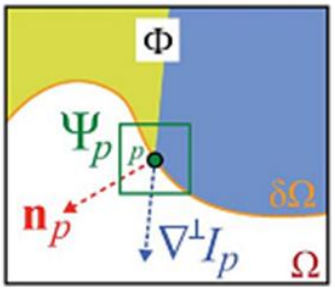

Figure 1: Notation diagram [2].

We use the input image I, missing region Ω, and source region Φ is defined as $\Phi = I - \Omega$. A best-first filling strategy depends on priority values that are assigned to each patch on boundary $\delta\Omega$. Given a patch ψ_p centered at point p for some $p \in \delta\Omega$, its priority is defined as the product of two terms, $P(p) = C(p) \cdot D(p)$, where, $C(p)$ is the confidence term which indicates reliability of the current patch, and $D(p)$ is the data term. $C(p)$ and $D(p)$ is expressed as,

$$C(p) = \frac{1}{|\Psi_p|} \sum_{q \in \Psi_p \cap \Phi} , \qquad (01)$$

$$D(p) = \frac{\left\langle \nabla^{\perp} I_p, \mathbf{n}_p \right\rangle}{\alpha} , \qquad (02)$$

where, $|\psi_p|$ is the area of ψ_p (number of pixels within the patch ψ_p), α is the normalization factor (ex: $\alpha = 255$ for a gray image), $\mathbf{n}_p$ is a unit vector which is orthogonal to $\delta\Omega$ at p, and $\nabla^{\perp} = (-\partial_y, \partial_x)$ is the direction of the isophote. $C(p)$ is then the percentage of non-missing pixels in patch ψ_p, which is set at initialization to $C(q) = 0$ for missing pixels in Ω, and $C(q) = 1$ everywhere else. Once all priorities on $\delta\Omega$ are computed, a block-matching algorithm derives best exemplar $\Psi_{\hat{q}}$ to fill in missing pixels under highest priority patch $\Psi_{\hat{q}}$, previously selected as,

$$\Psi_{\hat{q}} = \arg\min_{\Psi_q \in \Phi} \left\{ d\left(\Psi_{\hat{p}}, \Psi_q\right) \right\}, \qquad (03)$$

where, $d(.,.)$ is the distance between two patches, defined as sum of squared difference (SSD).

Having found source exemplar $\Psi_{\hat{q}}$, value of each pixel-to-be-filled $p' \in \Psi_{\hat{p}\cap\Omega}$ is copied from its corresponding pixel in $\Psi_{\hat{q}}$. After patch $\Psi_{\hat{p}}$ has been filled, confidence term $C(p)$ is updated as,

$$C(p) = C(\hat{p}), \qquad \forall p \in \Psi_{\hat{p}} \cap \Omega. \tag{04}$$

Depth aided priority computation was introduced in [3]. Given associated depth patch Z in targeted image plane, priority computation is done by adding a third multiplicative term, $P(p) = C(p) \cdot D(p) \cdot L(p)$, where, $L(p)$ is the level regularity term, defined as inverse variance of depth patch as,

$$L(p) = \frac{|Z_p|}{|Z_p| + \sum_{q \in \Psi_p \cap \Phi} \left(Z_p - \overline{Z_p}\right)^2}, \tag{05}$$

where, $|Z_p|$ is the area of Z_p (in terms of the number of pixels) and $\overline{Z_p}$ is the mean value.

2. IMAGE TO WORLD PROJECTION

We used the raw video files from [6]. The video files are in the uncoded YUV4:2:0 format. We used FFmpeg [7] to extract the frames as jpeg image files. Each data set contains color and depth data files in raw YUV format. Video files are 24 bits RGB, whereas depth files are 24 bits but all the 3 fields are the same, hence considered as 8 bit depth values. 3D world point in non-homogeneous coordinates can be found as [8],[3],

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \boldsymbol{R}_1^{-1} K_1^{-1} \begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix} \lambda_1 - \boldsymbol{R}_1^{-1} \mathbf{t}_1, \tag{06}$$

where, $\boldsymbol{R}_1$, is the rotation matrix, $\mathbf{t}$ is the translation matrix, K is the camera parameter matrix, λ_1 is a positive scaling factor, (u_1, v_1) is a point in image plane, and (x, y, z) is the corresponding 3D world point. λ can be found using following equations.

$$z_k(u_k, v_k) = \frac{1}{d_k(u_k, v_k) \cdot \left(\left(1/z_{k,\text{near}}\right) - \left(1/z_{k,\text{far}}\right)\right) + \left(1/z_{k,\text{far}}\right)}, \tag{07}$$

$$\lambda_1 = \frac{z}{c}, \qquad \text{where,} \qquad \begin{pmatrix} a \\ b \\ c \end{pmatrix} = K_1^{-1} \begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix}, \tag{08}$$

where, z_{far} and z_{near} values are given in the dataset. Depth data $d_k(u_k,v_k)$ was originally normalized to the range $[0\cdots1]$ and $z_{k,\text{near}}$ and $z_{k,\text{far}}$ are the minimum and maximum depth values of the 3D scene, respectively.

3. RESULTS

We present our results only for the Balloons dataset here.

Figure 2 shows example color and depth frames of the Balloons dataset for the camera numbers 1 and 5.

First, the above given camera views 1 and 5 are projected to the virtual camera view 3. Results of this projection are shown in Figure 3.

We can now notice the empty regions (or holes) created in each projected view. Left column shows the projected view of camera 1 to camera location 3. Hence holes are in the right side of each object. Right column shows the projected view of camera location 5 to location 3. In contrast to left column, now the holes are in the left side of each object.

Both the left and right images in Figure 3 are the images of the same view. We can notice most of the content duplicates. We select the most closest depth values to the front as the correct value and two projected images are merged as shown in Figure 4.

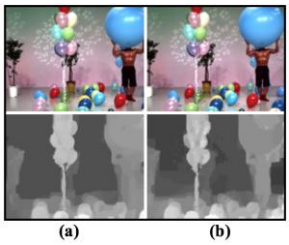

Figure 2: Color and depth images of Balloons dataset (1st frame), (a) Camera no. 1, (b) camera no. 5.

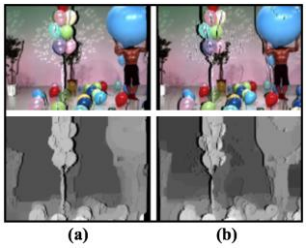

Figure 3: Projected color (top) and depth (bottom) frames for the Balloons dataset, (a) projection from camera 1 to camera 3 position, (b) projection from camera 5 to camera 3 position.

Figure 4: Depth based merged color and depth frames of the Balloons dataset at virtual camera location 3.

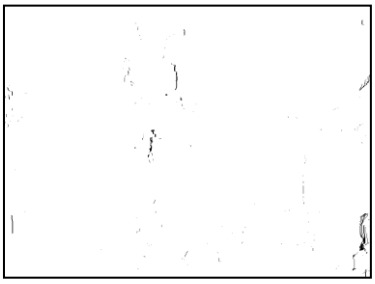

Figure 5: Occluded holes in the merged projected views for the Balloons dataset.

Even after the merging of the two projected views, we can still notice some holes left as in Figure 5. Our goal is now to fill these holes from the available information. We can use inpainting for this.

For the inpainting of the holes, we used a solution based on the Criminisi's method. As discussed in the related work section, Criminisi used only confidence and data term in calculating the target priority. We used the depth (level) term introduced in [3] in calculating the priority. So the priority is the product of three terms, confidence, data, and level. We implemented our experiment program in Matlab based on the Surahul's original code written in C++. Original code is located at [9]. Our Matlab code is available at https://github.com/chinthakawk/vigilant-fiesta.

Our example frame of the Balloons dataset is 1024×768 and takes a long time to complete the inpainting. Hence, for the experimental purposes, we scaled down the image by 50% and 25%, resulting sizes 384×512 and 192×256. 25% image completed inpainting in 1,602.28 seconds, and 50% image took 25,050.13 seconds to complete. We still need to work on improving the performance of the code.

Source and target patch locations of each iteration for the 50% experiment is shown in Figure 6. Source patch is shown in red, target patch is shown in blue, and the iteration number is shown in the interconnecting line of source to target. 50% experiment was completed in 54 iterations. Progress of the confidence, data, level, and priority values are shown in Figure 8.

We can clearly notice the effect of the level term in the priority value. The selected source patches would be different if the level term was not used. Number of the holes and the fill front count at the beginning of each iteration for both 50% and 25% experiment is shown in Figure 10.

25% experiment was completed in 15 iterations. The change of confidence, data, level, and priority values for the 25% dataset is shown in Figure 9.

The reason for the different values for 50% and 25% experiments are the number of pixels in each image, and the reduced number of holes due to scaling down. Specially the level term for the 25% experiment stays roughly equal. This may due to the fact that scaling down, changes the depth values. An example patch inpainting process is shown in Figure 7 for the iteration 41 of the 50% experiment.

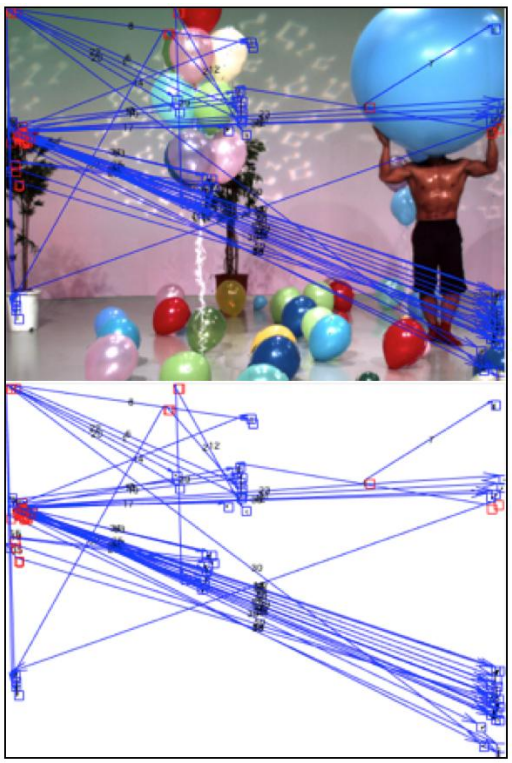

Figure 6: Inpainting progression, red: source path, blue: target patch, iteration number is shown in the interconnecting line of source and target patch.

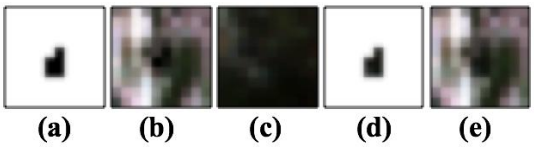

Figure 7: Patch transfer for 50% experiment in iteration 41, (a) hole, (b) original target patch, (c) source patch, (d) copied pixels of source patch, (e) inpainted target patch.

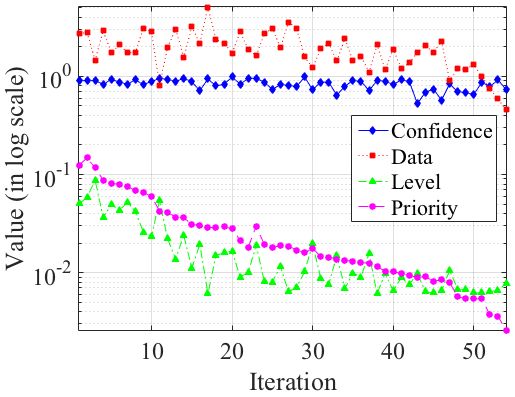

Figure 8: Confidence, data, level, and priority values for the 50% experiment.

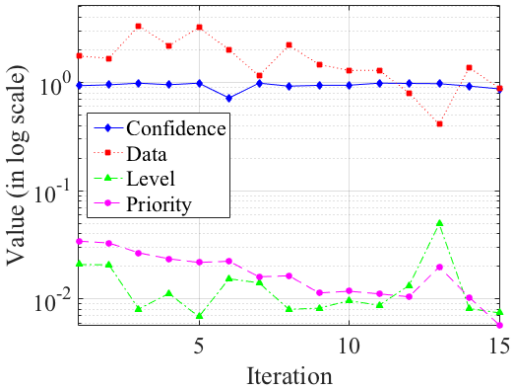

Figure 9: Confidence, data, level, and priority values for the 25% experiment.

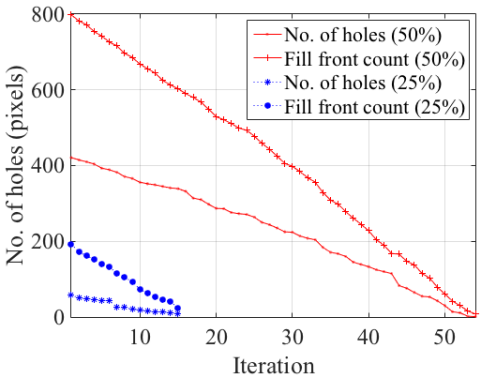

Figure 10: Number of the holes and the fill front count at the beginning of each iteration for both 50% and 25% experiment.

4. CONCLUSION

An example depth based inpainting for multi view projection applications was presented. The color and depth images of the first frame of the Balloons dataset were used. Color and depth images of the camera locations 1 and 5 were then projected to location 3. Then two projected views at the virtual camera position is merged based on z-buffer. Finally a depth term based image inpainting is applied to fill the disoccluded holes in the virtual view.

Further experiments are planned to compare the effect of inpainting results with and without a depth term. Other future works are to increase the performance of the Matlab code, experiment in the full size of the images, extend the experiment for the other available datasets, and how to further improve the quality of the inpainting result.

5. REFERENCES

[1] Z. Tauber, Z.-N. Li, and M. S. Drew, "*Review and preview: Disocclusion by inpainting for image-based rendering*," Syst. Man Cybern. Part C Appl. Rev. IEEE Trans. On, vol. 37, no. 4, pp. 527–540, 2007.

[2] A. Criminisi, P. Pérez, and K. Toyama, "*Region filling and object removal by exemplar-based image inpainting*," Image Process. IEEE Trans. On, vol. 13, no. 9, pp. 1200–1212, 2004.

[3] I. Daribo and H. Saito, "*A novel inpainting-based layered depth video for 3DTV*," Broadcast. IEEE Trans. On, vol. 57, no. 2, pp. 533–541, 2011.

[4] E. J. King, G. Kutyniok, and W.-Q. Lim, "*Image inpainting: theoretical analysis and comparison of algorithms*," in SPIE Optical Engineering+ Applications, 2013, pp. 885802–885802.

[5] O. Le Meur, M. Ebdelli, and C. Guillemot, "*Hierarchical super-resolution-based inpainting*," Image Process. IEEE Trans. On, vol. 22, no. 10, pp. 3779–3790, 2013.

[6] "*MOBILE◦3DTV*." [Online]. Available: http://sp.cs.tut.fi/mobile3dtv/. [Accessed: 23-Apr-2016].

[7] "*FFmpeg*." [Online]. Available: https://www.ffmpeg.org/. [Accessed: 23-Apr-2016].

[8] K. Mueller, A. Smolic, K. Dix, P. Merkle, P. Kauff, and T. Wiegand, "*View synthesis for advanced 3D video systems*," EURASIP J. Image Video Process., 2008.

[9] "*surahul/Inpaint*," GitHub. [Online]. Available: https://github.com/surahul/Inpaint. [Accessed: 23-Apr-2016].