The Visual Computer manuscript No.
(will be inserted by the editor)

Kevin Kreiser · Jingyi Yu

Real-time Projector Depixelation for Videos

Abstract The screen-door effect or projector pixelation is a visual artifact produced by many digital projectors. In this paper, we present a real-time projector depixelation framework for displaying high resolution videos. To attenuate pixelation, we use the common defocusing approach of setting the projection a little out-of-focus. Using a camera-projector pair, our system efficiently measures the spatially-varying defocus kernel and stores it as a texture on the graphics hardware. We explore two novel techniques to compensate for the defocusing blur in real-time. First, we develop a steepest descent algorithm on the GPU for estimating the optimally deblurred video frames based on the measured defocus kernel. Second, we present a novel optical flow algorithm that uses an illumination ratio map (IRM) to model illumination transformations between consecutive frames. We store both the IRM and optical flow as textures. To process a frame at runtime, we use the textures to warp the optimized previous frame into an initialization for the iterative GPU optimization. We show that our GPU optimization achieves one magnitude acceleration using this warping. Experiments on various video clips show that our framework can realize real-time video playback with significantly reduced pixelation and defocus while preserving fine details and temporal coherence.

Keywords Pixelation · Defocus · Deblur · GPU · Projector Display

Kevin Kreiser
University of Delaware
Newark, DE 19711
E-mail: kevk@udel.edu

Jingyi Yu
University of Delaware
Newark, DE 19711
E-mail: yu@eecis.udel.edu

1 Introduction

The screen-door effect or projection pixelation is a visual artifact produced by most digital projectors, in which the gaps (called dead zones) separating the projector's pixels become visible in the projected image. Although the latest generation of DLP projectors provide a closer spacing of the mirror elements to reduce pixelation artifacts, some space is still required along the mirrors' edges to provide control circuit pathways. The pixelation artifact is more apparent when one projects images with a resolution higher than the projector's. The spatial digitization creates jaggy boundaries due to undersampling and magnifies the pixelation.

A common approach to mitigate the pixelation artifacts is projector defocus, wherein the projector is deliberately set a little out-of-focus [27]. By defocusing the projector, a small amount of light leaks into the dead zones and blurs the boundaries between pixels. However, it also introduces blurriness in the projection. Many efforts have been proposed to compensate for the defocus blur, ranging from the classical Wiener filters [11] and the Richardson-Lucy algorithm [20], to the recently proposed graph cuts [18] and belief propagation [25]. Other techniques specific to projector deblur such as image preconditioning [5] and focal pre-correction [17] have been proposed. Although these algorithms are effective in deblurring still images, they are also computationally expensive and are therefore used to preprocess the images before projection.

In this paper, we present a real-time projector depixelation framework for displaying high resolution videos. Our framework also uses the defocus approach. Using a camera-projector pair, our system efficiently measures the spatially-varying defocus kernel and stores it as a texture on the graphics hardware. We explore two novel techniques to compensate for the defocusing blur in real-time. First, we develop an iterative optimization algorithm on the GPU for estimating the optimally deblurred frames based on the measured defocus kernel. Second, we present a novel optical flow algorithm that uses an illumination ratio map (IRM) to model illumination transformations between consecutive frames. We precompute both the IRM and the optical flow for each frame and load them as textures at runtime to warp the optimized previous frame into an initialization for the iterative GPU optimization. We show that our GPU optimization achieves one magnitude acceleration using this warping. We test our framework on various video clips and show that we are able to achieve real-time video playback with significantly reduced pixelation and defocus while preserving fine details and temporal coherence. The complete system pipeline is shown in Figure 1.

The specific contributions of this work are:

– A real-time projector depixelation framework based on defocusing for displaying high resolution videos
– A GPU-based steepest descent algorithm for deblurring individual frames of video
– An optical flow algorithm that handles illumination variations across frames using an illumination ratio map (IRM)
– An optical flow/IRM based warping method to accelerate the iterative GPU optimization

2 Previous Work

The recent introduction and rapid adoption of consumer digital projectors has redefined the landscape for displaying images and videos. High resolution and high contrast projectors are increasingly used in a wide array of commercial and scientific applications, ranging from shape acquisition [28,7], to virtual environments [19] and IMax theaters [13]. Many of these applications require the projected images maintain high sharpness, low aliasing [27], and low brightness variation [8].

The most commonly observed visual artifact produced by digital projectors is projector pixelation or the screen door effect. The fine lines separating the projector's pixels become visible in the projected image. The screen door effect can be mitigated by defocusing the projector a little in front of (or behind) the projection screen to allow a slight amount of light to leak into the pixel gaps [27,5]. However, defocusing the projector results in a blurry image, which can make viewing strenuous since one's eyes will constantly try to bring the projection into focus. Although multiple projectors could be combined to fill in the pixel gaps [3,14], multiple projection approaches require pixel-level-accurate geometric and radiometric calibrations.

Image deblurring algorithms have been widely used to compensate for defocus blur. However, the problem of deblurring (deconvolution) is inherently ill-posed in that multiple distinct solutions exist for the same convolution kernel. Several methods have been proposed for approximating a solution. The classical Wiener filter [11] attempts to statistically estimate the image's blur kernel and uses regularization to compute the inverse kernel. Computer vision methods such as graph cuts [18] and belief propagation [25] have also been used to recover nearly optimal deblurred images. Mathematically, image deblurring can be formulated as a bound-constrained quadratic programming problem [16]. Iterative methods based on steepest descent [27] and conjugate gradient [11] have been used to efficiently locate a local optimal solution. However, most of these approaches are computationally expensive and cannot achieve real-time performance.

Our approach focuses on real-time video deblurring and uses the programmable graphics hardware (GPU). The recent addition of programmable graphics chipsets has led to a myriad of efforts in exploiting them for general-purpose computations, in particular, iterative linear algorithms including the sparse matrix conjugate gradient [4], Gauss-Seidel [12], and fast fourier transformation [15]. The stream-based architecture and large texture size of the graphics hardware have also enabled new computational video algorithms [6,2,1]. In this paper, we combine both GPU-based linear algorithms and computational video methods for real-time video deblur.

A key factor that determines the performance of iterative linear optimization is the initial condition. We present a new optical flow estimation method for warping the optimized previous frame into an initialization for the GPU-based optimization. Classical optical flow methods [9] assume that the scene objects maintain consistent appearance in neighboring frames. Tremendous efforts have been focused on how to robustly handle textures, noise, and occlusion boundaries [21,23]. However, these algorithms are sensitive to illumination changes such as varying shadows or lighting, which commonly appear in videos. In this paper, we propose finding the illumination transformation between two frames by calculating the ratio of intensity for each scene point in the frames. The resulting ratio image is called the illumination ratio map (IRM). Jacob et al. [10] have shown that illumination ratio images tend to exhibit spatial smoothness, although abrupt changes may occur across the shadow and occlusion boundaries. We develop efficient GPU algorithms based on the Bilateral Filter [26] to simultaneously recover the optical flow and the IRM and use the two maps as initialization for the GPU-based optimization.

Before proceeding, we clarify our notation. F_{in} represents the source video and F_{opt} represents the optimized video in our framework. $F_{in}(n)$ and $F_{opt}(n)$ represent the n^{th} frame of the input and optimized output respectively. $OF(n)$ and $IRM(n)$ represent the optical flow field and illumination ratio map between frame $F_{in}(n-1)$ and $F_{in}(n)$. Superscripts such as $F^{i}_{opt}(n)$ represent the i^{th} iteration result when estimating $F_{opt}(n)$.

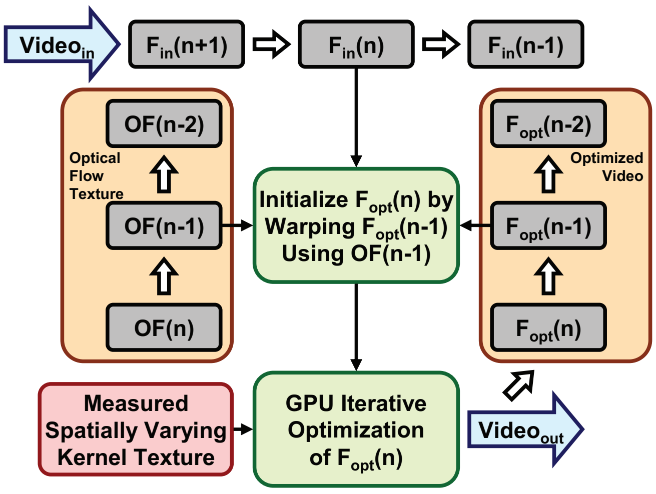
Video_in
F_in(n+1)
F_in(n)
F_in(n-1)
OF(n-2)
Optical Flow Texture
OF(n-1)
OF(n)
Initialize F_opt(n) by Warping F_opt(n-1) Using OF(n-1)
F_opt(n-2)
Optimized Video
F_opt(n-1)
F_opt(n)
Measured Spatially Varying Kernel Texture
GPU Iterative Optimization of F_opt(n)
Video_out

Fig. 1 Our real-time projector depixelation framework. Our system first defocuses the projector and measures the blur kernel. At runtime, to process frame $F_{in}(n)$, we warp the optimized previous frame $F_{opt}(n-1)$ using the optical flow and then correct it using an Illumination Ratio Map (IRM). Finally, we use the warped $F_{opt}(n-1)$ as the initial guess to find the optimal deblurred frame on the GPU.

3 Kernel Estimation

To attenuate pixelation, we use the common defocusing approach of setting the projection a little out-of-focus. Defocusing the projector, however, blurs the projected image as shown in Figure 5. To measure the blur kernel, we project a binary dot-pattern onto the screen as shown in Figure 2(a) and 2(b). We then use a camera to capture the projected pattern Γ. A similar setup has been proposed by Zhang and Nayar [27], where a beam splitter was used to form a coaxial camera-projector system. The coaxial camera-projector system avoids rectifying Γ but requires accurate geometric calibrations.

We do not require the camera and projector share the same optical center. Instead, we assume the projection surface is planar and we use homography to warp the captured pattern Γ. Our system automatically detects the center of each blurred dot and uses Singular Value Decomposition (SVD) to compute the homography.

Since the captured pattern Γ consists of an ambient component, we capture an ambient image $\Gamma_{ambient}$ by turning off the projector and then subtract $\Gamma_{ambient}$ from Γ such that $\Phi = \Gamma - \Gamma_{ambient}$.

To measure the spatially-varying blur kernel K, we warp Φ using the estimated homography and detect the center c of each pattern. We then place a window w centered at c of size $t \times t$ for each blurred dot. For every pixel $[u, v]$ with respect to window w, the kernel $k_c[u, v]$ is normalized as:

$$k_c[u,v] = \frac{I[u,v]}{\sum_{u,v \in w} I[u,v]} \quad (1)$$

where $I[u, v]$ represents the radiance received at pixel $[u, v]$. Since we only measure a sparse sampling of the

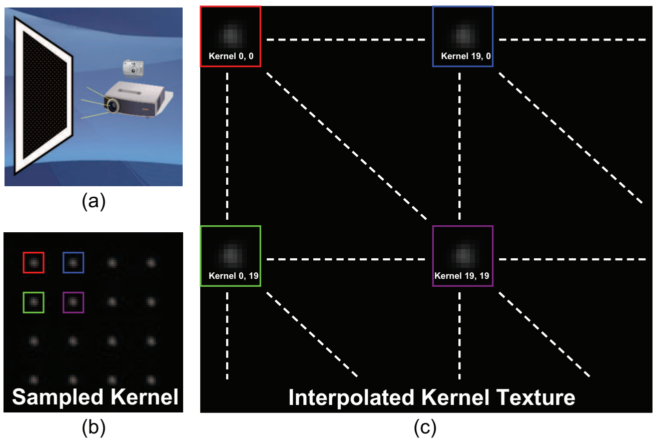

Fig. 2 Measuring the defocus kernel. (a) We use a camera to capture the projected binary pattern and use homography to warp the pattern. (b) The warped camera image with samples of the blur kernel. (c) We interpolate the spatially varying kernel and pack them into a single texture.

spatially-varying kernels, we bilinearly interpolate between the neighboring kernels to estimate the deblur kernel $k_{p,q}$ for every pixel $[p, q]$ in the image Φ. We then pack the kernels into a large spatially-varying kernel texture K that will be used by our GPU-based deblur algorithm, as shown in Figure 2(c).

We benefit from the large texture memory and memory bandwidth available in today's graphics cards for efficient storage and querying of the kernels. On an nVidia 8800GTS card that supports a texture size of 8Kx8K, we are able to store a blur kernel up to 16x16 pixels for 512x512 resolution video clips. In our experiments, a defocus blur of size 5x5 pixels is usually sufficient to remove most pixelation artifacts.

4 GPU-Based Defocusing

We formulate the problem of image deblurring as a constrained quadratic programming problem [16, 27]. Given the measured spatially-varying kernel K and frame $F_{in}(n)$, our goal is to compute the optimal image $F_{opt}(n)$ that is as close as possible to $F_{in}(n)$ after it is blurred by kernel K, i.e.:

$$F_{opt}(n) = \arg\min_{D} \left\{ \begin{array}{l} \|K \otimes D - F_{in}(n)\|^2 \\ \forall p,q \; 0 \le D[p,q] \le 255 \end{array} \right\} \quad (2)$$

where $\|\cdot\|$ is an image distance metric.

Notice that each pixel $[p, q]$ in image D has an associated kernel $k_{p,q}$ in the kernel texture K, therefore $\otimes$ represents spatially-varying convolution for each pixel. In our implementation, we use the sum of squared pixel differences as the image distance metric.

4.1 Image and Kernel Representation

To solve for Equation 2, linear optimization algorithms such as steepest descent and conjugate gradient can be used to approximate a nearly optimal solution. For an image of size $m \times m$ pixels, traditional CPU-based linear optimization methods pack all pixels in the image into a single column vector of size m^2 and represent the blur kernel as a sparse matrix of size $m^2 \times m^2$. Using this representation, convolving an image with the kernel can be written as matrix multiplication [27].

Our goal is to use the GPU to solve for Equation 2. This requires storing both the image and convolution kernel as textures on the graphics card. For high resolution videos, packing all the pixels of a frame into a single vector can easily result in a matrix that exceeds the largest texture size allowed on the graphics card. Therefore, we choose to represent both the image and the spatially varying blur kernel K as 2D textures. We then implement the convolution operation using a fragment shader which multiplies the two textures (see supplemental material).

4.2 Steepest Descent Optimization

We use the steepest descent method for computing the optimal D in Equation 2. We start with some initial guess to D as D^0. A common choice to D^0 is to directly use the input frame $F_{in}(n)$. In Section 5.1, we show that a better initialization is obtained by warping the optimized previous frame $F_{opt}(n-1)$.

Once we initialize D^0, we then repeat the following three steps.

In step 1, we compute the residue R as:

$$R = \tilde{K} \otimes (F_{in}(n) - K \otimes D^i) \tag{3}$$

where $\tilde{K}$ is a spatially varying kernel satisfying $\tilde{k}[u,v] = k[t-u, t-v]$. The derivation of $\tilde{K}$ follows from formulating convolution as matrix multiplication. If we pack the pixels in image D into a single vector V and represent K and $\tilde{K}$ as two large sparse matrices M and $\tilde{M}$ respectively, it is easy to see that $\tilde{K}$ is the kernel corresponding to the transpose of the sparse matrix form M of kernel K (i.e. $\tilde{M} = M^T$).

In step 2, we update D^i using the steepest descent as $D^{i+1} = D^i + \alpha R$ where α is computed using steepest descent algorithm as:

$$\alpha = \frac{\mid R \mid^2}{\mid K \otimes R \mid^2} \tag{4}$$

Notice that after we update D^i, some pixel intensities may become negative or greater than 255. Therefore, we clamp D^{i+1} after step 2.

Finally, in step 3, we compute the sum of absolute differences for all pixels $[p,q]$ in consecutive iterations of D by:

$$\sum_{p,q} |D[p,q]^{i+1} - D[p,q]^i| \tag{5}$$

We repeat steps 1, 2, and 3 until the difference in Equation 5 is less than a predefined threshold. We set our threshold to be βm^2, where m^2 is the number of pixels in each frame $F_{in}(n)$ and β is a user definable constant between 0 and 1 exclusive. For scenes with higher detail, we choose a smaller β (and hence execute more iterations) to preserve high frequencies.

There are many advantages to implementing the steepest descent deblur algorithm on the GPU. First, by using a fragment shader, we efficiently parallelize convolution for all pixels. Second, our GPU scale-addition shader implicitly clamps D^i to values between 0 and 255, by rendering it directly to an 8 byte texture. To efficiently compute the image difference metric, we employ the classic MIPMAP technique; we generate a MIPMAP of image $|D[p,q]^{i+1} - D[p,q]^i|$ and multiply its lowest level MIPMAP value by m^2 to compute Equation 5.

5 Illumination Coherent Optical Flow Estimation

Notice, like many iterative linear optimization methods, the performance of our GPU-based deblur algorithm depends heavily on the initial condition D^0. The most straightforward initialization is to use the original frame $F_{in}(n)$ as D^0 [27]. A better choice is to take advantage of the temporal coherence between the consecutive frames and warp the optimized previous frame $F_{opt}(n-1)$ as D^0 using the optical flow.

However, two fundamental problems still remain. First, classical optical flow approaches [9] assume the corresponding pixels have consistent intensities in consecutive frames. In the presence of illumination inconsistencies such as varying shadows or lighting, which frequently occur in videos, state-of-art optical flow methods break down. Second, directly warping $F_{opt}(n-1)$ using $OF(n)$ without compensating for the illumination inconsistencies results in limited improvement in the number of iterations.

5.1 Illumination Ratio Map

We present a new approach to account for illumination variations between consecutive video frames using an Illumination Ratio Map (IRM). An IRM computes the intensity ratio of corresponding points in an image pair. If two consecutive frames $F_{in}(n)$ and $F_{in}(n-1)$ are captured at the same viewpoint, and the scene is static, then the IRM γ simply corresponds to $\frac{F_{in}(n)}{F_{in}(n-1)}$ [10]. If the camera position changes or any part of the scene moves, we need to find the corresponding pixel in $F_{in}(n)$, i.e.:

$$\gamma(i,j) = \frac{F_{in}(n)[i + OF(n).x, j + OF(n).y]}{F_{in}(n-1)[i,j]} \tag{6}$$

where $OF[x,y]$ corresponds to the optical flow field.

Our goal is to simultaneously recover γ and OF. Notice that the IRM maintains spatial smoothness in general. Discontinuities in the IRM appear in near scene occlusion boundaries, shadow boundaries, or moving objects. For instance, when the illumination direction changes across the two images, shadows abutting the occlusion boundaries may appear or disappear. As a result, sharp edges can appear near depth edges in the IRM. Shadow boundaries also change due to illumination variations. However, since most of the shadows are soft in real scenes, they often appear in the IRM as smooth transitions instead of discontinuities.

The characteristics of spatial smoothness and occlusion discontinuity in the IRM are very similar to those seen in disparity maps from stereo matching. Previous researchers [24,23] have shown that fields satisfying such properties can be modeled as probabilistic graphical models. We propose a new two-step optimization algorithm to iteratively estimate the optical flow and the IRM as follows:

In step 1 we detect and match a sparse set of feature points between frame $F_{in}(n)$ and $F_{in}(n-1)$ using the Shi-Tomasi method [22]. The feature points are then triangulated and interpolated to form a dense optical flow field $OF(n)$.

In step 2 we compute $\gamma(n)$ using Equation 6, apply a bilateral filter to $\gamma(n)$, and de-illuminate $F_{in}(n)$ using $\gamma(n)$. We repeat these steps until satisfactory.

We choose to use the bilateral filter [26] to smooth the IRM as it preserves occlusion boundaries, reduces noise, and can be implemented on the GPU [6]. To further accelerate the optical flow estimation, our algorithm only detects a sparse set of features and uses Delaunay triangulation to create an interpolant. Each vertex is then associated with x and y directional motion components. We then rasterize the triangles' x and y vector components using the GPU, as shown in Figure 3. After rasterization, all pixels have floating point x and y directional motion components. Thus, when warping using our IRM-OF shader, we use bilinear interpolation for fetching floating point texture locations.

In our experiments, we found that after two to three iterations, the estimated optical flow and IRM were sufficiently denoised to reduce the iterations of the GPU optimization by a magnitude as shown in the table in Figure 4. In Figure 3, we show a frame of the recovered IRM and rasterized optical flow of a video with strong illumination variations. It is also worth noting that more sophisticated algorithms such as [24] can be used to precompute more accurate optical flow fields and IRMs, although at higher computational cost.

To process each frame $F_{in}(n)$ at runtime, we use the estimated optical flow and IRM to warp the optimized previous frame $F_{opt}(n-1)$ to D^0 as:

$$D^0[i,j] = F_{opt}(n-1)[i+OF[i,j].x, i+OF[i,j].y] \cdot \gamma[i,j] \quad (7)$$

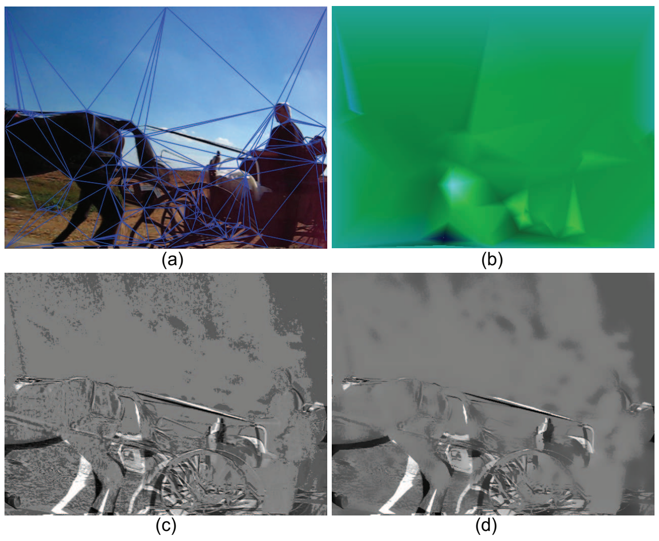

Fig. 3 Recovering the optical flow and the Illumination Ratio Map (IRM). We detect sparse features and find their correspondences in consecutive frames (a). Using our two-step iterative algorithm, we recover both the optical flow field (b) and the IRM (d). (c) shows the initial IRM estimation and (d) shows the final result after 2 iterations.

	512x512 Video	FPS with Warping	FPS w/o Warping	Iterations per Frame with Warping	Iterations per Frame w/o Warping
3x3 Kernel	Clown Fish	67.9	10.6	1.0	13.2
	Grass Hill	33.7	6.4	3.1	22.3
	Roadside	40.1	8.0	2.4	17.8
	Coral Reef	66.3	22.0	1.0	6.3
5x5 Kernel	Clown Fish	31.2	1.5	1.0	34.1
	Grass Hill	18.7	2.4	2.1	21.3
	Roadside	19.0	2.7	2.1	18.5
	Coral Reef	32.3	4.3	1.0	11.6
7x7 Kernel	Clown Fish	16.4	1.0	1.1	25.4
	Grass Hill	10.0	1.1	2.1	21.9
	Roadside	10.1	1.4	2.0	18.0
	Coral Reef	17.2	2.4	1.0	10.2

Fig. 4 Comparing the performance of our GPU framework with and without warping at different blur kernel sizes on an nVidia 8800GTS. All video clips are rendered at 512x512 resolution.

We implement Equation 7 using a fragment shader (see supplemental materials). We then optimize D^0 for frame $F_{in}(n)$ as show in section 4.2.

6 Experimental Results

We have used our framework to display various video clips on an InFocus LP850 projector. We defocus the projector a little in front of the projection screen and use a Canon SD750 camera (7.1M pixels) to measure the defocus kernel and capture the projected video frames.

Quality. Our video depixelation framework is able to significantly reduce screen door artifacts while preserving fine details and high temporal coherence. In our experiments, we found that using a defocus blur kernel of size 5x5 is usually sufficient to reduce most of the pixelation

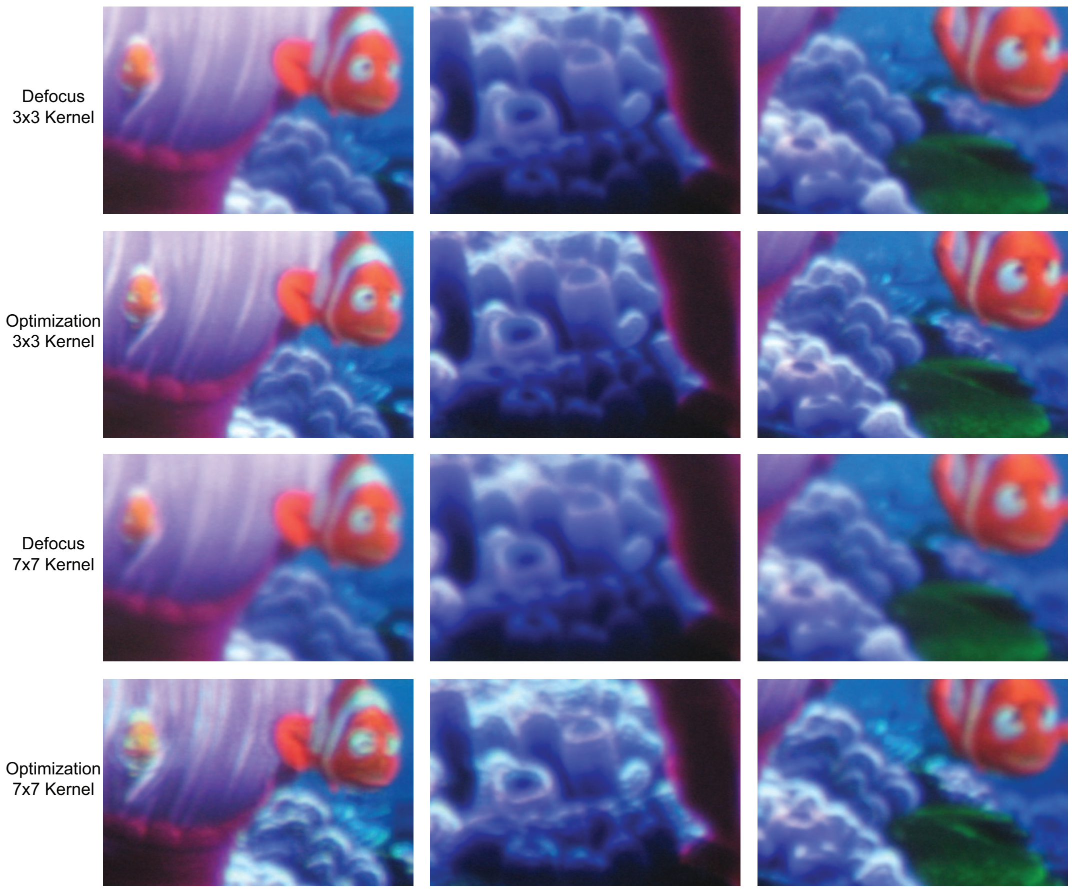

Fig. 5 Results of depixelated video projection using our framework. We use a Canon SD750 camera to capture the projected video clip rendered at 24 fps. The projected video frames with defocusing blur are shown in the first and third rows. Notice the strong blurring artifacts in the defocused projection. The optimized results are for the smaller and larger kernel sizes are shown in the second and fourth rows respectively. Please refer to the companion video for more examples.

artifacts without introducing ringing. Our algorithm also performs well with this kernel size for most video clips. High frequency features are much better preserved using our framework, as is shown on the grass, corn stalks, and coral reef in Figure 6 and in the supplemental video.

The quality of our algorithm also depends on the size of the blurring kernel. We found that our algorithm usually performs better with smaller kernel sizes than with larger ones. Kernels larger than 7x7 pixels (Figure 5 row 4) generally require more optimization and sometimes result in ringing artifacts, as is shown on the clownfish in the left column of Figure 5. This is an inherent problem in many existing deblurring algorithms [27].

Speed. In the table in Figure 4, we compare the processing speed of our GPU-based deblur method with and without the OF/IRM warping. For small kernel sizes, initializing our steepest descent optimization using IRM-OF warping renders at over 30 fps. This allows real-time playback of the video while it is deblurred. The IRM-OF warping also achieves a speedup of 5 times over simply using the input frame as initialization. For even larger kernel sizes, this speedup is more significant. This is intuitive in that large blur kernels product deblurred frames which are very different than the input frames. This leads to a large number of iterations before steepest descent converges. However, consecutive frames in a video are temporally coherent. Therefore, the warped optimized previous frame is more similar to the deblurred target frame and this allows the optimization to quickly converge to the optimal solution with much fewer iterations. In fact, the average number of iterations using the IRM-OF warping is a magnitude less than the one using the nave initialization, as shown in the rightmost two columns of table in Figure 4.

The speed of our algorithm also depends on the quality of the IRM-OF approximation. For highly textured regions such as the grass hill scene shown in the third column of Figure 6, the estimated IRM-OF is less accurate, and hence, the GPU-based optimization takes more iterations. Furthermore, if an input video has many high frequency details, those details will require more optimization. It is also important to note that the optimization is computed globally. Thus videos which are spatially homogenous with respect to the frequency domain, will require a predictable amount of optimization, where as spatially heterogenous videos are less predictable.

Finally, both initialization schemes scale approximately linearly with the kernel size. This can be explained by the large number of convolutions performed in our GPU-based optimization, since the overhead of convolution grows linearly with the kernel size.

7 Conclusions and Future Work

We have presented a real-time projector depixelation framework for displaying high resolution videos. To attenuate pixelation, we have used the common defocusing approach of setting the projection a little out-of-focus. Our system efficiently measures the spatially-varying defocus kernel and stores it as a texture on the graphics hardware. We have explored two novel techniques to compensate for the defocusing blur in real-time. First, we have developed an iterative optimization algorithm on the GPU for estimating the optimally deblurred frames based on the measured defocus kernel. Second, we have presented a novel optical flow algorithm that uses an illumination ratio map (IRM) to model illumination transformations between consecutive frames. We store both the IRM and the optical flow as textures and we use them at runtime to warp the optimized previous frame into an initialization for the GPU optimization. We have shown that our GPU optimization with warping achieves one magnitude speedup. Experiments on various video clips have shown that our framework is able to realize real-time video playback with significantly reduced pixelation and defocus while preserving fine details and temporal coherence.

Limitations remain in our approach, however. Our framework can only support blur kernels of relatively small sizes, although they are usually sufficient to reduce pixelation. This is a result of packing the spatially varying kernels into a single texture on the GPU. Since the current generation of graphics hardware can only support texture sizes up to 8Kx8K, we would need to store the kernels as separate textures for larger kernels. Furthermore, larger kernels lead to higher computational overhead and texture fetching when computing the convolution on the GPU. Our warping method also relies on the accuracy of the estimated optical flow and IRM. Our results, in theory, can be further improved using the combined IRM-OF Markov Random Field model using a maximum a posteriori (MAP) optimization [24]. We intend to investigate how to use GPU-based MAP to improve the accuracy of the optical flow and the IRM.

Although initially designed for video deblur, our framework has the potential for real-time motion deblur. In Figure 7, we apply the GPU optimization with known or estimated kernels and initialize it with the blurred input image. Figure 7(c) shows the resolution chart image blurred with a known 7×1 linear kernel. Our GPU algorithm recovers the deblurred image at about 30 fps with an image resolution of 256×256. Figure 7(d) shows a captured image of a moving RC car. We recover the blur kernel using the Wiener Filter and show the deblurred result using our GPU algorithm in Figure 7(e). Compared with the ground truth image of the stationary RC car (Figure 7(f)), our result robustly captures the detail on the Mustang decal and is computed at 14 fps with an image resolution of 256×256. In the future, we plan to explore efficient methods for estimating the blur kernels and integrate them with our GPU deblurring framework.

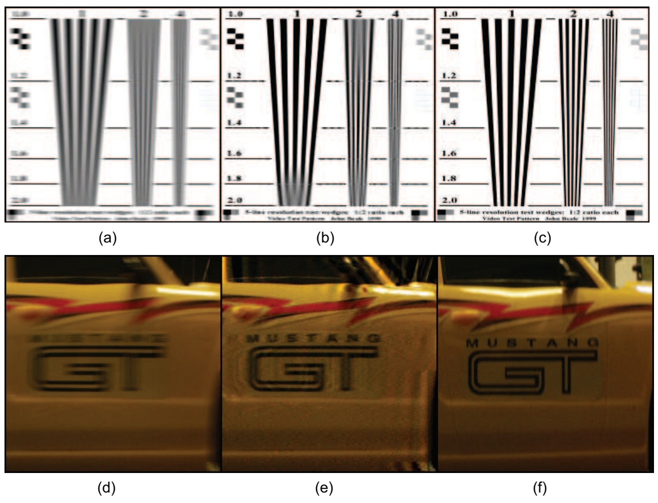

Fig. 7 Motion deblur using our framework. (a) Synthesized motion blur on a resolution chart. (b) Our deblurred result running at 30fps. (c) Ground truth image of (a). (d) Captured motion blur on a Mustang RC car. (e) Our deblurred result running at 14fps. (f) Ground truth image of (d).

References

1. Bennett, E.P., McMillan, L.: Proscenium: a framework for spatio-temporal video editing. In: MULTIMEDIA '03: Proceedings of the eleventh ACM international conference on Multimedia, pp. 177–184 (2003)
2. Bennett, E.P., McMillan, L.: Computational time-lapse video. In: SIGGRAPH '07: ACM SIGGRAPH 2007 papers, p. 102 (2007)
3. Bimber, O., Emmerling, A.:
4. Bolz, J., Farmer, I., Grinspun, E., Schröder, P.: Sparse matrix solvers on the gpu: conjugate gradients and multigrid. In: SIGGRAPH '03: ACM SIGGRAPH 2003 Papers, pp. 917–924 (2003)
5. Brown, M.S., Song, P., Cham, T.J.: Image preconditioning for out-of-focus projector blur. In: CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 1956–1963 (2006)
6. Chen, J., Paris, S., Durand, F.: Real-time edge-aware image processing with the bilateral grid. In: SIGGRAPH '07: ACM SIGGRAPH 2007 papers, p. 103 (2007)
7. Davis, J., Nehab, D., Ramamoorthi, R., Rusinkiewicz, S.:
8. Fujii, K., Grossberg, M.D., Nayar, S.K.: A projector-camera system with real-time photometric adaptation for

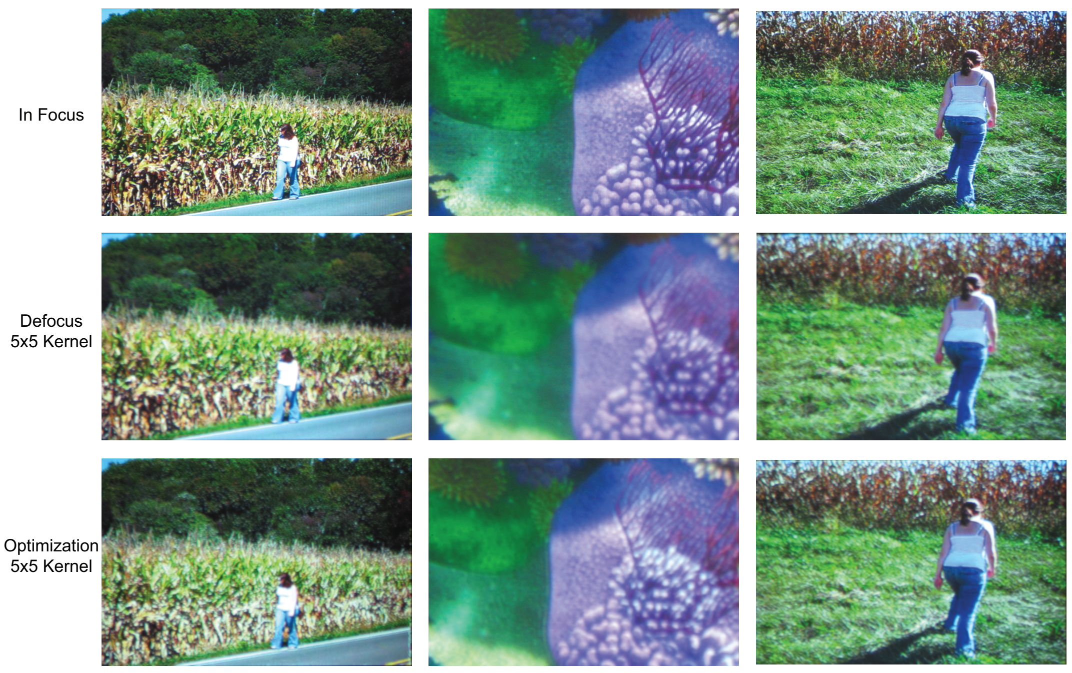

Fig. 6 Comparative results from our framework using highly textured video clips. Notice the high frequency features such as the leaves on the trees (left), the multifaceted coral reef (middle), and the blades of grass (right) retain their detail using our method. All three videos are processed at over 30 frames per second.

dynamic environments. In: CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1, pp. 814–821 (2005)
9. Horn, B.K., Schunck, B.G.: Determining optical flow. Tech. rep. (1980)
10. Jacobs, D.W., Belhumeur, P.N., Basri, R.: Comparing images under variable illumination. In: CVPR '98: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, p. 610. IEEE Computer Society, Washington, DC, USA (1998)
11. Jain, A.K.: Fundamentals of digital image processing. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1989)
12. Krüger, J., Westermann, R.: Linear algebra operators for gpu implementation of numerical algorithms. In: SIGGRAPH '05: ACM SIGGRAPH 2005 Courses, p. 234 (2005)
13. Lantz, E.: A survey of large-scale immersive displays. In: EDT '07: Proceedings of the 2007 workshop on Emerging displays technologies, p. 1 (2007)
14. Majumder, A., GregWelch: COMPUTER GRAPHICS OPTIQUE Optical Superposition of Projected Computer Graphics . pp. 209–218
15. Moreland, K., Angel, E.: The fft on a gpu. In: HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, pp. 112–119. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland (2003)
16. Nocedal, J., Wright, S.: Numerical Optimization. Springer (2000)
17. Oyamada, Y., Saito, H.: Focal pre-correction of projected image for deblurring screen image. In: CVPR. IEEE Computer Society (2007)
18. Raj, A., Zabih, R.: A graph cut algorithm for generalized image deconvolution. In: ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision, pp. 1048–1054 (2005)
19. Raskar, R., Welch, G., Cutts, M., Lake, A., Stesin, L., Fuchs, H.: The office of the future: A unified approach to image-based modeling and spatially immersive displays. In: SIGGRAPH, pp. 179–188 (1998)
20. Richardson, W.H.: Bayesian-based iterative method of image restoration. Journal of the Optical Society of America (1917-1983) **62**, 55–59 (1972)
21. Scharstein, D., Szeliski, R.: A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. International Journal of Computer Vision **47**(1-3), 7–42 (2002)
22. Shi, J., Tomasi, C.: Good features to track. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94). Seattle (1994)
23. Sun, J., Shum, H.Y., Zheng, N.N.: Stereo matching using belief propagation. In: ECCV (2), pp. 510–524 (2002)
24. Tappen, M.F., Freeman, W.T.: Comparison of graph cuts with belief propagation for stereo, using identical mrf parameters. In: ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision, p. 900. IEEE Computer Society, Washington, DC, USA (2003)
25. Tappen, M.F., Russell, B.C., Freeman, W.T.: Efficient graphical models for processing images. In: CVPR (2), pp. 673–680 (2004)
26. Tomasi, C., Manduchi, R.: Bilateral filtering for gray and color images. In: ICCV '98: Proceedings of the Sixth International Conference on Computer Vision, p. 839. IEEE Computer Society, Washington, DC, USA (1998)
27. Zhang, L., Nayar, S.: Projection defocus analysis for scene capture and image display. In: SIGGRAPH '06: ACM SIGGRAPH 2006 Papers, pp. 907–915 (2006)
28. Zhang, L., Snavely, N., Curless, B., Seitz, S.M.: Spacetime faces: high resolution capture for modeling and animation. In: SIGGRAPH '04: ACM SIGGRAPH 2004 Papers, pp. 548–558 (2004)