WYSIWYG Computational Photography via Viewfinder Editing

Jongmin Baek[1], Dawid Pająk[2], Kihwan Kim[2], Kari Pulli[2], and Marc Levoy[1]

[1]Stanford University [2]NVIDIA Research

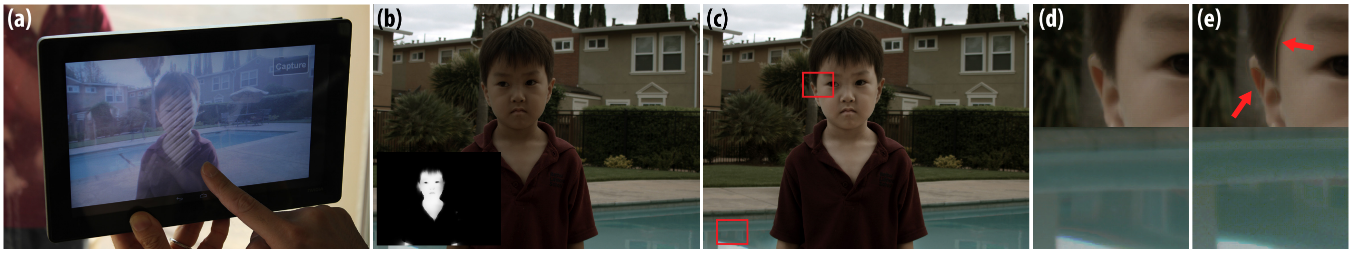


Figure 1: *Viewfinder editing and appearance-based metering.* **(a)** *In the proposed system, the user interacts with the camera viewfinder as if it were a live canvas, by making sparse strokes to select regions and to specify edits. The edits propagate spatiotemporally as the camera and the scene objects move, and the edits are applied in the subsequent frames of the viewfinder, which are tonemapped HDR images, to provide WYSIWYG experience.* **(b)** *After making a local tonal edit, the user presses the shutter to trigger a high-resolution capture. Here we show the resulting tonemapped HDR composite without any edits applied, for reference. The edit mask computed from the user strokes is shown in the inset.* **(c)** *The result with edits applied is shown. This approximately corresponds to what the user sees on the screen just as he presses the shutter. Our appearance-based metering acquired an exposure stack at (0.645 ms, 5.555 ms, 11.101 ms) by optimizing the quality of the final result based on the user's global and local edits.* **(d)** *The regions indicated in (c), magnified.* **(e)** *We captured another stack with a histogram-based HDR metering at (0.579 ms, 9.958 ms, 23.879 ms) and applied the same post-processing pipeline. As the standard HDR metering considers equally all the pixels in the scene, it allocates too much effort to capture the dark areas that were not as important to the user, leading to a longer capture times that invite ghosting (top) and higher noise in mid-tones (bottom).*

Abstract

Digital cameras with electronic viewfinders provide a relatively faithful depiction of the final image, providing a WYSIWYG experience. If, however, the image is created from a burst of differently captured images, or non-linear interactive edits significantly alter the final outcome, then the photographer cannot directly see the results, but instead must imagine the post-processing effects. This paper explores the notion of *viewfinder editing*, which makes the viewfinder more accurately reflect the final image the user intends to create. We allow the user to alter the local or global appearance (tone, color, saturation, or focus) via stroke-based input, and propagate the edits spatiotemporally. The system then delivers a real-time visualization of these modifications to the user, and drives the camera control routines to select better capture parameters.

Keywords: image editing, burst photography, HDR metering

Links: DL PDF

1 Introduction

Early photographers could not directly see their photographs as they were taking them, but had to imagine the results as a function of various imaging parameters such as exposure, focus, even choice of film and paper. Digital cameras with real-time digital displays, which show a good preview image, have made photography much easier in this respect, providing a WYSIWYG (what you see is what you get) interface for the camera. In particular, framing the image and choosing the timing of capture is made easier and more fun as the user sees right away what the outcome will be. However, when applying many computational photography techniques, the user still cannot see an approximation of the result, but needs to imagine the end result. One example is combining an exposure burst into an HDR image and tonemapping it back to LDR for display, In addition, many photographers edit their photographs after capture, using tools such as Photoshop or Lightroom. However, users must compose the shot in the field *before* knowing the effect such edits might have on the result. The capture process thus remains separated from the image editing process, potentially leading to inadequate data acquisition (wrong composition or insufficient SNR) or excessive data acquisition (longer capture time, exacerbated handshake or motion blur, and increased storage and transmission cost).

At the same time, mobile devices with a digital camera, display, adequate computational power, and touch interface are becoming increasingly commonplace. More photographs are captured by mobile devices now than ever, and many of them are edited directly on device and shared from that device, without ever being uploaded to PCs. This phenomenon is reflected in the recent focus on camera control and image processing on mobile platforms [Adams et al. 2010b; Ragan-Kelley et al. 2012] and in the popularity of photography apps on smartphones. The trend so far has been to harness this increased processing power to enable desktop-like workflow on mobile devices. We instead use these processing capabilities to introduce the notion of *viewfinder editing*, which allows the users to perform edits live on the viewfinder prior to capture. We bring the WYSIWYG interface to computational photography applications, allowing the users to directly see the effects of interactive edits or stack photography on the viewfinder. Using this interface we also gather information on which aspects of the image are important to the user, which in turn affects capture parameters such as the number of images to capture, the values for exposure, focus, white balance, et cetera. This WYSIWYG interface serves particularly well those casual shooters that desire further interactivity in their photography yet would like to avoid offline work.

To realize this philosophy, we propose a unified framework in which the user provides sparse, stroke-based input to control the local (or global) tone, color, saturation, and focus, and receives immediate feedback on the viewfinder, as shown in Figures 1 and 3. The selections the user provides are affinity-based [An and Pellacini 2008] and are stored as a sparsely sampled function over the image-patch space [Adams et al. 2010a]. The selections are then propagated to subsequent viewfinder frames by matching image patches, and the edits are applied to both the viewfinder image and eventually the high-resolution image(s) the user captures.

Also, the edits are internally used to drive the camera control routines that determine the appropriate exposure and/or focus value(s). In our framework, the user expresses a modification of the image, enabling the camera control algorithms to deduce the noise threshold, depth of field, and dynamic range necessary to support the transform. For instance, if the user wishes to locally brighten a dark region of the scene, it ought to lead to a different metering decision; or, if the user is happy to let the sky saturate on the display, a full exposure stack should not be necessary. The user can even provide inconsistent cues: e.g., the user wishes two objects at the same depth to exhibit different levels of defocus. Such cases can be satisfied by taking two images with different settings and combining them. Therefore, stack-based computational photography merges seamlessly with traditional photography, kicking in when and only when necessary.

The technical contributions of this paper are a fast edit propagation algorithm running on the viewfinder stream; a viewfinder interface that visualizes the edits, tonemapping, and multi-exposure blending; and camera control routines to take advantage of the knowledge of the visualization, which altogether form a camera system that can run at an interactive rate on mobile devices.

2 Prior Work

Image processing and editing of captured still photographs are well-explored realms in computer graphics. Many forms of appearance editing exist, such as tonemapping [Reinhard et al. 2002], and detail enhancement or suppression [Farbman et al. 2008; Fattal 2009; Gastal and Oliveira 2012].

As mobile platforms become ubiquitous and computationally powerful, image editing on such platforms is on the rise. For instance, Liang et al. [2010] developed a point-and-swipe interface for edge-aware tone manipulation of still images on a smartphone. Commercial software implementing similar techniques exists in mobile space [Nik Software 2012]. Many computational photography algorithms on the desktop already take sparse gestures as inputs [Agarwala et al. 2004; Levin et al. 2004], and could easily accommodate touch-based interfaces.

Below we survey the prior work on two main challenges addressed in our paper: spatiotemporally propagating edits in a stream of images, and determining capture parameters like exposure and focus. We remark that no prior work in the community has addressed image editing on a live viewfinder, much less feeding back the result of image processing back to capture process.

2.1 Edit Propagation

Edit propagation allows the user to specify a sparse set of edits and automatically propagates them onto the rest of the image or other frames. Image processing on a sequence of frames has been implemented previously for abstraction [Winnemöller et al. 2006], sparse edit propagation [An and Pellacini 2008] and retargeting [Rubinstein et al. 2008]. One family of edit propagation methods is based on spatially diffusing edits to neighboring pixels via edge-aware smoothing. A cost function on the image gradient can be used to build a linear system that diffuses edits [Levin et al. 2004; Lischinski et al. 2006] or to drive graph cuts [Agarwala et al. 2004]. Fattal [2009] performs a wavelet decomposition to propagate edits to neighboring pixels of similar intensity. However, these methods do not handle spatially discontiguous regions and are not robust against occlusions or camera shakes.

Another family of methods is based on detecting regions that have appearance similar to that of the user-selected area. Edits are modeled as a function over the space of feature descriptors for the pixels, with the underlying assumption that pixels with similar feature descriptors should have similar function values. The function is typically recovered from an energy minimization framework [An and Pellacini 2008; Chen et al. 2012; Farbman et al. 2010], or an explicit clustering of known values provided by the user [Bie et al. 2011; Li et al. 2008; Li et al. 2010; Xu et al. 2009].

One caveat is that propagating edits on the viewfinder in real time is distinct from the problem of propagating edits on an image sequence as an offline process. Most existing methods for the latter rely on a preprocessing step that analyses the whole sequence and builds expensive classifiers whose costs are amortized over the multiple frames to be processed or the multiple edits to be applied [An and Pellacini 2008; Bie et al. 2011; Farbman et al. 2010; Li et al. 2008; Xu et al. 2009] for acceleration. Our proposed method is designed to be a truly online method that handles streaming data.

2.2 Optimizing Capture Parameters

Many computational photography algorithms make use of stack photography, requiring the determination of capture parameters such as exposure, gain, and focus.

High-dynamic-range (HDR) photography captures multiple photos with varying exposure levels, which are then blended to form the HDR image. Hasinoff et al. [2010] and Gallo et al. [2012] solve for the exposures that maximize the signal-to-noise ratio of the HDR image. Such HDR image typically undergoes tonemapping for display, boosting some signal and suppressing the rest. This aspect is not modeled and accounted for by metering algorithms because HDR photography is currently not WYSIWYG—the transform is generally not known a priori, and neither is the type and extent of any local edits the user will make.

Focus bracketing can be used to digitally extend the depth of field [Hasinoff and Kutulakos 2008] or reduce it [Jacobs et al. 2012]. These algorithms register and blend images taken at various focus settings to mimic a photograph with a different aperture setting. Vaquero et al. [2011] acquires the minimal focus stack necessary for all-in-focus imaging by preliminarily sweeping the scene and identifying focal settings that provide sharp pixels.

One capture parameter that is difficult to algorithmically optimize is composition, as it is subjective. The difficulty is compounded by the fact that images are often edited or tonemapped after capture, altering its feel significantly. Photos could be cropped to obey standard composition rules [Liu et al. 2010] after capture, but cropping discards data and is inherently hamstrung by the composition of the

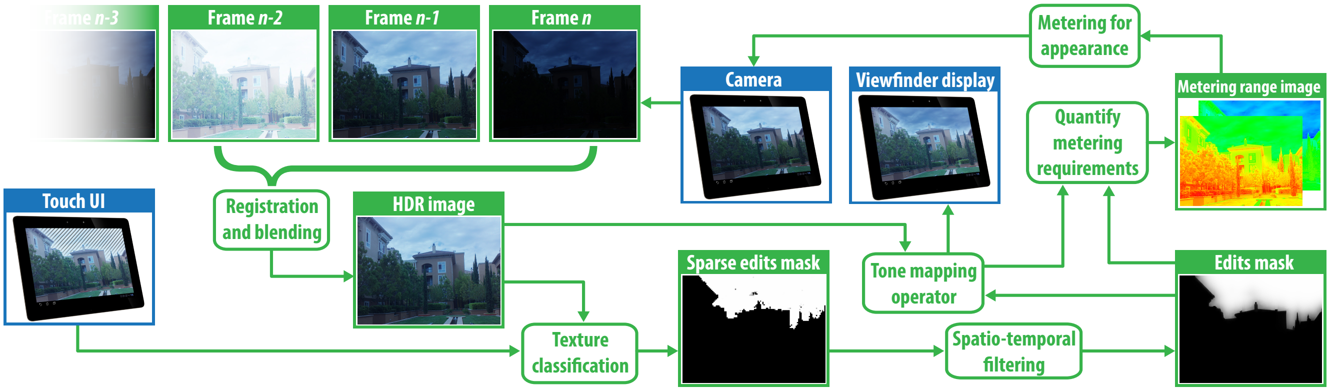


Figure 2: *System overview for the exposure-stack photography application. The camera streams frames with alternating exposures, which are registered and merged into an HDR composite, which is displayed for the user. The user can specify edits via touch interface by marking textures, creating a sparse edit mask, which is spatio-temporally filtered to make consistent and to avoid flickering. The selected areas can be modifed, for example by brightening or darkening them, and this user input is taken into account when metering for the best exposure values for the next frames. The tonemapping edits are continually applied for every frame and displayed for the user.*

original, so the decision on composition is best made at the time of capture. A WYSIWYG viewfinder facilitates better composition as a faithful estimate of the final image is shown in real time.

In summary, existing work on computational image capture aims to acquire as much information about the scene, for maximum flexibility later. We propose an alternate philosophy in which only the information necessary or desired by the user is actually acquired, provided that the user can communicate what is necessary and desired via a real-time interface.

3 Overview

Figure 2 summarizes the major components in our framework for the specific case of exposure stack photography. The algorithmic details are explained in the subsequent sections.

Image acquisition. We stream the raw image data from the sensor into a stack that caches the N most recent frames. We internally acquire a full exposure or focus stack for viewfinding, because our edit propagation mechanism works better on an HDR image than an LDR image with saturated, blurry, or underexposed regions.

The capture parameters are updated on a per-frame basis as follows: for exposure stacks, we examine the log-luminance histogram of the scene and meter for the bins not covered by the last $N-1$ frames. For focal stacks, we iterate from the minimum to the maximum focus distance in fixed increments.

Edit propagation on the viewfinder. The processing thread fetches the N most recent frames (typically $N = 3$ for exposure stacks and $N = 4$ for focal stacks), and merges them into an HDR radiance map or an all-focus image. The resulting scene estimate is stored in a LogLUV format [Reinhard et al. 2005], which accounts for nonlinearity of human visual system.

Each pixel has a context based on an 8×8 patch around it, from which we extract a descriptor. The selection and edits are modeled as functions over the space of these image patch descriptors. For every frame, we compute these functions over each image patch in the scene and generate edit masks (Section 4.2), apply the edit masks onto the LogLUV data, tonemap the result, and display it on the screen. In case of focus edits, we recompose the images from the focal stack appropriately. Because the masks are quickly recomputed on each frame based on image patches, the user edits are robust against scene and camera motion.

Lastly, we recompute the optimal set of exposure and/or focus values (Section 5) taking all the user edits and post-processing steps (such as tonemapping for the display) into account. These parameters are used when the user presses the shutter. The frame(s) captured with these settings can be processed by the same pipeline, or an offline method for higher quality (Section 4.3).

User interface. The user is presented with a seemingly normal viewfinder, though internally the camera is constantly acquiring an exposure or focal stack. The user begins by selecting a region via stroke gestures. The user may cancel a selection by tapping outside the selected region, or confirm a selection by tapping within, which triggers an overlay with icons representing various types of edits. Once the user chooses the type of edit to apply to the selected region, the user makes a swiping gesture horizontally left or right to adjust the designated trait, e.g., darker or brighter (Section 4.1).

4 Viewfinder Editing

Image editing on a viewfinder stream of a hand-held camera must accommodate temporally persistent selection of objects through sparse user input. We rely on affinity-based edit propagation [An and Pellacini 2008; Xu et al. 2009], in which edits are modeled as functions residing in the space of local patch descriptors:

$$S_i : \mathbb{R}^d \to [-1, 1], \quad i = 1, 2, \ldots \tag{1}$$

where each of $S_1, S_2, \ldots$ corresponds to a particular type of edit, such as tone, color, saturation or blurriness. We reserve $S_0 : \mathbb{R}^d \to [0, 1]$ for the selection mask. Then, our goal is to compute the vector-valued function $\vec{S} = (S_0, S_1, \ldots)$ for the patches in each viewfinder frame, and to apply corresponding edits in real time. Alternatively, one can think of each image patch having tags S_i, either stored explicitly or interpolated from the tags of nearby patches, indicating the level of selection or the state of an edit.

We use 8-dimensional patch descriptors ($d = 8$), in which the features are the mean and 1st- and 2nd-order derivatives of the log-luminance, plus the mean CIELUV chrominance, each yielding 1, 2, 3, and 2 values, respectively. Note that we leave out the (x, y) coordinate, in order to be robust against scene and camera motion.

Existing affinity-based methods attempt to globally optimize or interpolate $\vec{S}$ based on the user-provided samples. The cost of global optimization or interpolation is often mitigated by preprocessing the dataset to learn the topology of the patch descriptors or training a classifier. While this approach works well for offline processing of video or image sequences, we need to process each viewfinder frame independently in a streaming fashion.

To achieve this, we store $\vec{S}$ in the permutohedral lattice [Adams et al. 2010a], a sparse data structure designed for rapid high-dimensional filtering. The lattice is suitable since it stores high-dimensional vector-valued function (e.g., $\vec{S}$) with $O(1)$ cost for incremental update. The lattice internally performs resampling in the patch descriptor space, which serves to locally propagate the data.

Because we forgo an explicit optimization or interpolation unlike previous work, edits do not propagate as aggressively, but this issue is mitigated in three ways: first, we apply edge-aware smoothing on S_i with respect to the scene image whenever a viewfinder frame is produced. Second, because the user receives feedback interactively as the strokes are made, it is easy and intuitive to control propagation—the user essentially interactively paints S_i. Third, once the user captures a stack, we rely on existing edit-propagation algorithms in the literature for high-quality offline processing.

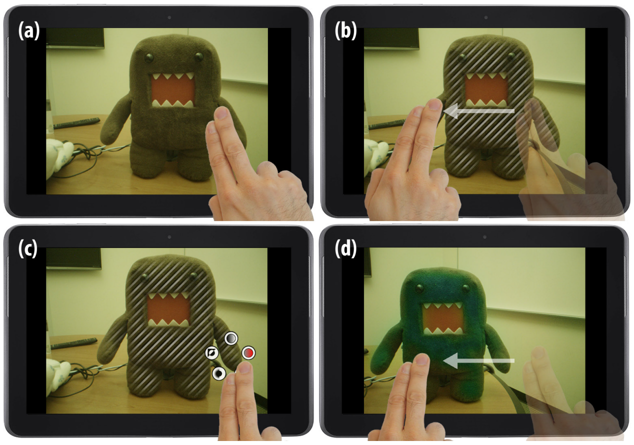


Figure 3: *Interface for viewfinder editing.* **(a):** *The user begins by stroking over the region of interest.* **(b):** *As the user swipes his finger, the selection updates on the screen.* **(c):** *The user confirms the selection by tapping within the selected region, which invokes a UI widget offering various edit operations the user can choose from. The user chooses to edit the hue of the selected region.* **(d):** *As the user swipes his finger horizontally to indicate the sign and the magnitude of the edit, the viewfinder updates.*

4.1 Specifying Edits

Instead of initializing the lattice with all patches present in a given image, we take a streaming approach: as the user strokes over the screen and selects patches, we locate only nodes corresponding to these patches and update their values. Note that unselected patches are never written into the lattice; if a patch lookup fails at any point, a default value is assumed for $\vec{S}$.

As customary in Gaussian filtering, we use 2D homogeneous coordinates to represent each of $S_0, S_1, \ldots$ in the permutohedral lattice. The actual value of $S_0, S_1, \ldots$ is obtained by dehomogenizing the 2D vector. We will denote the homogeneous form as $\tilde{S}_i : \mathbb{R}^d \to \mathbb{R}^2$ for each i. These coordinates express the relative weight of the edit.

Edits are specified in three phases, as illustrated in Figure 3: first, the user strokes over the region of interest, and confirms the selection by tapping on the selected area. Second, the user is shown a widget listing the various types of edits supported, and the user taps on his choice. Third, the user horizontally swipes left or right, in order to specify how much the edited value should be decreased or increased. All of the phases are interactive; as the user moves a finger on the screen, the updated edits are reflected on the viewfinder. We refer the readers to the supplementary video for demonstrations.

Selection. While the user stroke is being registered, image patches whose centers are within a small fixed distance from the touch event are converted to descriptors and are looked up from the lattice. New nodes are created if they are not found. We increment the value of S_0 for these nodes to signify that they are now selected

$$\tilde{S}_0(\vec{p}) := \tilde{S}_0(\vec{p}) + \begin{pmatrix} 1 \\ 1 \end{pmatrix}. \qquad (2)$$

In practice, a vertex lookup in the permutohedral lattice will return several nearby vertices, as a means for resampling. We apply the above equation to these vertices, scaled by the resampling coefficient. If the user passes over the same texture $\vec{p}$ multiple times in a single stroke, the dehomogenized value $\vec{S}_0(\vec{p})$ will still be 1, but the weight of the selection will be larger, so $\vec{p}$ will be more influential in the interpolation performed in the lattice.

The cost of specifying selection is $O(1)$, independent of viewfinder dimensions and the edit history.

Editing. If the user is in the act of specifying an edit $k \in [-1, 1]$ of type j, then for each selected descriptor $\vec{p}$, we adjust $\tilde{S}_j(\vec{p})$ before applying it to the image.

$$\tilde{S}_j(\vec{p}) := \tilde{S}_j(\vec{p}) + k \cdot \begin{pmatrix} S_0(\vec{p}) \\ 1 \end{pmatrix}. \qquad (3)$$

As shown in Eq. (3), the extent of the edit is further scaled by the soft selection mask S_0.

Note that this adjustment is not yet written into the lattice. Therefore, the cost of visualizing each viewfinder frame grows linearly with the viewfinder dimensions, and is independent of the number of nodes in the lattice. Once the user finalizes the edit, we can fold it into the lattice by applying Eq. (3) to every selected patch $\vec{p}$ in the lattice, and reset $\tilde{S}_0(\vec{p})$ to zero. Hence, the cost of finalizing an edit is proportional to the size of the lattice, and independent of the viewfinder dimensions. While this step is slightly more expensive, it is only performed when the user signals the end of a discrete edit.

4.2 Edge-Aware Smoothing

In our online implemention of the visualization, we only process a subsampled set of image patches, saving a significant amount of time, but yielding edit selection masks at a resolution lower than that of the viewfinder. The resulting edit masks undergo edge-aware upsampling using domain transform [Gastal and Oliveira 2011], with respect to the edges of the viewfinder content. Not only does this operation allow us to speed up computation, but it also generates higher-quality masks with better spatial edit propagation. We also experimented with wavelet-based upsampling [Fattal 2009], but found that while faster, it yielded slightly worse results.

We also perform the recursive variant of the domain transform filter across frames to reduce temporal artifacts. In essence, the filter blends the masks for the previous and the current frames together where the pixel difference is small to reduce the temporal noise, but preserves the current frame content where the difference is large (usually because the edges do not align due to camera motion); because of this non-linearity we do not need to register the masks.

This helps to suppress flickering in noisy regions, where the result of the lattice lookup is inconsistent because of spurious texture introduced by the noise.

4.3 Offline Processing

Once the user is satisfied with the edits and presses the shutter, we perform additional computation to generate edit masks with better quality. We process image patches at full resolution, generating a full-size edit mask, which we still smooth spatially with domain transform. The resulting edit masks are thresholded and eroded to create a trimap. The trimap is then passed to manifold-preserving edit propagation [Chen et al. 2012] in order to re-populate low-confidence areas of the edit mask, where confidence is given by the homogeneous coordinates in the lattice. We found that this additional step helps produce cleaner and more homogeneous masks. Figure 4 demonstrates an example.

5 Appearance-Based Camera Control

We have thus far described an interface and underlying algorithms for performing edits directly on the viewfinder and propagating them forward in time. While this benefits the user by allowing him to evaluate and rethink composition based on the edits, it also provides the camera control routines with additional information that can be used to better select capture parameters. We describe two such parameter control algorithms, for exposing and focusing. The capture parameters are optimized for each pixel based on its intended appearance, and the results are aggregated for all the pixels in the viewfinder to produce a set of metering or focusing parameters for the scene.

5.1 HDR Metering for Display

Currently, HDR metering algorithms such as [Gallo et al. 2012; Hasinoff et al. 2010] operate to faithfully acquire scene luminance, attempting to maximize SNR. This philosophy makes sense when the post-processing to be performed on the luminance data is unknown, and there is no additional information on the importance of different scene elements. However, we can leverage the fact that the entire post-processing pipeline, including tonemapping and local edits, is known. The user sees on the viewfinder a tonemapped HDR image, and if some areas are too dark, brightens them, which indicates that longer exposures are needed, or darkens saturated areas to increase contrast, indicating shorter exposures are needed. The viewfinder image reflects the edits, and once the user is happy with the result, he can take the high-resolution HDR image. See Figure 6 for a demonstration of how edits influence HDR metering.

Quantifying Per-Pixel Exposure Requirements. We consider image fidelity at each pixel, and derive the exposure necessary to meet a particular threshold. Let L be the physical scene luminance estimated by the camera, perhaps from multiple exposures; and let I be the k-bit tonemapped result under a global, strictly monotonic tonemapping operator T. The user's edits create an edit map E which, in a spatially varying manner, modulates the luminance estimate L. In our implementation, we set $E(x,y) = 2^{6S_1(\vec{p}_{x,y})}$, corresponding to an adjustment up to ±6 stops. The viewfinder finally clamps the result into k bits:

$$I(x,y) = \min\left(2^k - 1,\ T(L(x,y) \cdot E(x,y))\right). \tag{4}$$

For each of the 2^k display levels, we associate a threshold for acceptable visual distortion, modeled as a Gaussian noise with standard deviation σ_d. In other words, we want the pixel value $I(x,y)$ on display to have at most a standard deviation of σ_d for the final, tonemapped display value. This threshold depends on the viewing condition, display resolution, and the user's visual adaptation, but for a bright display (photopic vision), we assume that σ_d is approximately constant; all the figures in this paper use $\sigma_d = 1$. See [Mantiuk et al. 2008] for work on display-adaptive tonemapping that relaxes this assumption.

Then, assuming non-saturation, the metering algorithm should attempt to record each pixel's physical luminance $L(x,y)$ so that its associated uncertainty σ_w, when carried through the imaging and tonemapping processes, has a standard deviation no larger than σ_d. For sufficiently small uncertainty, we can apply first-order approximation to the tonemapping process to obtain,

$$\begin{aligned} \frac{\sigma_w(x,y)}{\Delta L(x,y)} &\approx \frac{\sigma_d}{\Delta I(x,y)} \\ \implies \sigma_w(x,y) &\approx \frac{\sigma_d}{E(x,y) \cdot T'(L(x,y) \cdot E(x,y))}, \end{aligned} \tag{5}$$

via the chain rule, where $T'(\cdot)$ stands for $\frac{dT}{dL}$.

Finally, we assume a c-bit linear camera that captures the scene and records raw pixel values:

$$p(x,y) = \min\left(2^c - 1,\ L(x,y) \cdot t \cdot K + N(0;\sigma_r)\right), \tag{6}$$

where t is the exposure time; K is a calibration constant; $N(0;\sigma_r)$ is additive (Gaussian) read noise; and we clamp the measurement to model saturation.

The HDR reconstruction algorithm divides each pixel value by $t \cdot K$ to estimate $L(x,y)$, which also lowers the standard deviation of the noise to $\sigma_r/(t \cdot K)$. This noise should be below $\sigma_w(x,y)$ from Eq. (5), providing a lower bound on the exposure time:

$$\frac{\sigma_r}{K} \cdot \frac{E(x,y) \cdot T'(L(x,y) \cdot E(x,y))}{\sigma_d} \leq t. \tag{7}$$

We also enforce an upper bound to avoid sensor saturation:

$$t < \frac{2^c - 1}{K \cdot L(x,y)}. \tag{8}$$

For pixels that saturate on the sensor, the estimate $L(x,y)$ must be such that, when multiplied by $E(x,y)$ and tonemapped, the result should saturate on display:

$$t \leq \frac{(2^c - 1)E(x,y)}{K \cdot T^{-1}(2^k - 1)}. \tag{9}$$

We can easily extend this analysis to handle nonlinear cameras by folding the inverse camera response function into T. Other sources of noise, such as photon noise, can be folded into the read noise by allowing σ_r in Eq. (7) to vary as a function of the pixel value.

Optimizing HDR Stack. Now that we have derived the necessary conditions on each pixel, we can combine them to solve for a set of exposures that best satisfy them. Gallo et al. [2012] note that most scenes are handled by three exposures or fewer, and that most cameras offer only a limited number of possible exposure values.

We implement a greedy approach that seeks to iteratively maximize the aggregate objective function $\sum_{x,y} J(x,y,t)$ with respect to the exposure time t. The objective function should penalize exposure times outside the lower and upper bounds $B_*(x,y)$ and $B^*(x,y)$, derived at each pixel using Eqs. (7–9): there we set $J = 0$. Otherwise, if the output pixel $P(x,y)$ is saturated, we favor shorter exposures. In summary, we use the objective function

$$J(x,y,t) = \begin{cases} 0, & \text{if } t \notin [B_*(x,y), B^*(x,y)], \\ 1 + \alpha(x,y)\log_2 \frac{t}{B_*(x,y)}, & \text{otherwise,} \end{cases} \tag{10}$$

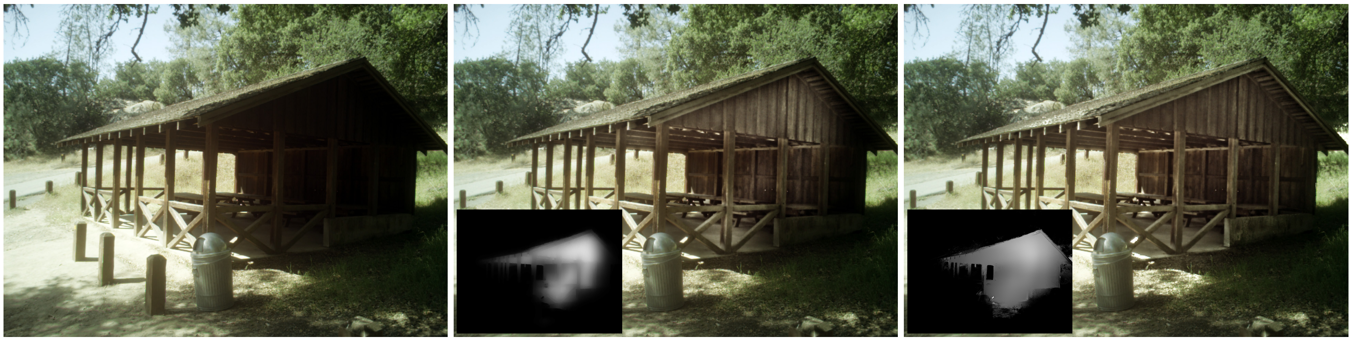
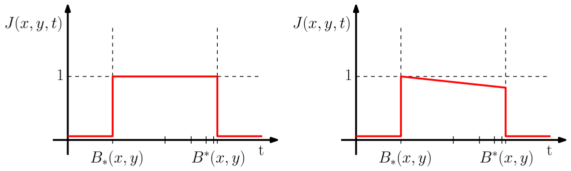


Figure 4: *Improving edit mask quality offline.* **Left:** *An unedited viewfinder frame (from our tablet implementation), which has already undergone HDR blending and global tonemapping.* **Middle:** *The viewfinder frame with edits made by the user, displayed live. In this example, the user has brightened the unlit side of the building.* **Right:** *The final output after additional offline processing. The insets in the middle and right images show the computed mask before and after offline refinement. In the refined version, the selection is more homogeneous and its edges are sharper. While imperfect, the real-time result in the middle is sufficient to give the user a feel for the final output.*

illustrated in Fig. 5 (a) and (b) on a logarithmic time axis, with $\alpha(x, y) = -0.3$ if the pixel is saturated, and 0.0 otherwise.



Case a: $P(x, y)$ *is not saturated.* **Case b:** $P(x, y)$ *is saturated.*

Figure 5: *Appearance-based metering via per-pixel analysis. For each pixel on the screen, we compute the minimal and maximal permissible exposure values, accounting for the local and global transforms raw sensor values undergo.* **(a, b):** *For metering, each pixel yields an objective function* $J(x, y, t)$ *based on the minimum and maximum per-pixel exposure values* $B_*(x, y)$ *and* $B^*(x, y)$*.*

When t is mapped to logarithmic domain, the objective in Eq. (10) becomes a sum of piecewise-linear functions, which we maximize in linear time using dynamic programming, by pre-computing and caching the first- and second-order derivatives of the objective. We greedily find exposures that maximize the objective, removing pixels from consideration whenever their requirements are met. We terminate the procedure upon reaching the maximum stack size or satisfying a certain percentage of per-pixel requirements.

5.2 Stack Focusing for Display

Another popular target for manipulation is the depth of field. A focal stack, a set of images focused at different depths, can be combined to simulate extended depth of field [Hasinoff and Kutulakos 2008]; reduced depth of field can be obtained likewise [Jacobs et al. 2012]. If the user can interactively specify the desired manipulation prior to capture and verify it via visualization in the viewfinder, the autofocus routine can deduce the minimal focal stack needed.

Quantifying Per-Pixel Focus Requirement. Using our interactive viewfinder, the user paints a mask $F : \{(x, y)\} \to [-1, 1]$ specifying which regions should be sharper or blurrier in a reference photograph focused at depth $z_0 \in [z_{min}, z_{max}]$. We measure the depths in diopters; under a simple thin-lens model, the blur size changes linearly with the offset in diopters. Meanwhile, the viewfinder stream cycles through a number of focus settings to continuously acquire the scene at various depths and builds a rough depthmap based on a local contrast measure. Then we build an indexing map $\hat{z}$, with which we will create the focal stack composite, as follows: at $F = 0$ we use the reference depth z_0; at $F = 1$, the maximally sharp depth z_*; at -1, the maximally blurred depth (either z_{min} or z_{max}); at other values of F we linearly interpolate.

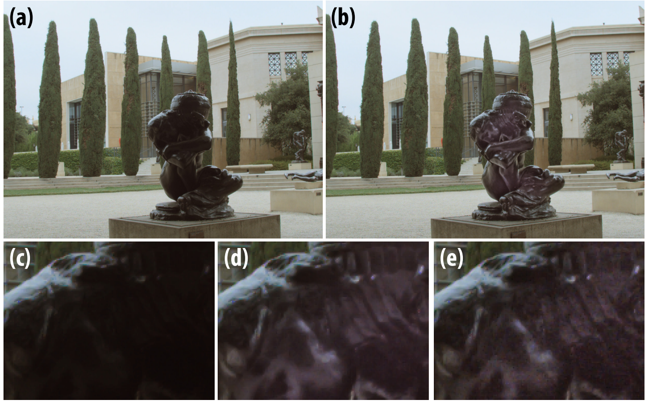


Figure 6: *The real-time effect of local tonal edits on appearance-based metering.* **(a):** *This tonemapped image was produced from a 2-image stack at (1.607 ms, 14.874 ms), as determined by our appearance-based metering.* **(b):** *The user brightened the dark statue on the viewfinder and retook the photo. Our algorithm automatically adjusted to the new appearance on the viewfinder and appended an extra shot (79.613 ms) to the stack.* **(c, d):** *We show insets from the center of the two images.* **(e):** *Just applying the local edit to the statue without taking it into consideration during metering yields much more noticeable noise in the output, compared to (d) where we accounted for the local edit during metering. The insets are best viewed on a screen while zoomed in.*

After $\hat{z}$ is regularized with a cross-bilateral filter using the scene image, we synthesize the output by indexing into the appropriate slice of the focal stack at each pixel. When the appropriate depth is not available, we interpolate linearly from the two nearest slices. The viewfinder is updated with this synthetic image continuously.

Optimizing Focal Stack. The map $\hat{z}$ obtained above covers a continuous range of depths, which is impractical to capture. To discretize $\hat{z}$ into a few representative values, we reuse the framework from Section 5.1 for optimizing the sum of piecewise-linear functions. The per-pixel objective is 1 at the desired depth $\hat{z}(x, y)$, linearly reducing to zero at depth error ϵ (we use $\epsilon = 1.0$ for a lens with depth range of $0.0 - 10.0$ diopters):

$$J(x, y, z) = \max\left(0,\ \frac{\epsilon - \|z - \hat{z}(x, y)\|}{\epsilon}\right). \tag{11}$$

We aggregate this per-pixel objective over all pixels. Because Eq. (11) is piecewise linear, the aggregate objective can be optimized quickly as in Section 5.1. Once again, we greedily select focus distances that maximize the objective, stopping when 10 slices are ordered or if for most of the pixels $\hat{z}$ is close to one of the focus distances in the set.

6 Applications and Results

To demonstrate the use of our viewfinder-editing framework (Section 4) and the appearance-based camera control routines (Section 5), we implemented a camera application incorporating these modules on two platforms: an x86-architecture laptop[1] with a USB camera (PointGrey Flea3), and an ARM-based tablet[2] with its rear camera controlled by the FCam API [Adams et al. 2010b]. Below we discuss our results and implementation details.

6.1 Appearance-based HDR Acquisition

In this application, we allow the user to make tonal and color edits on the viewfinder in order to drive HDR acquisition. We request the image sensor to continuously cycle through a number of exposures. These exposures are preliminary and are metered for the scene based on the scene luminance histogram. The image processing is performed in its own thread that runs asynchronously from the image capture. In each iteration, the most recent frames are registered by warping them via homographies, recovered from either sparse image features or gyroscope trace, and blended into an HDR scene luminance map [Reinhard et al. 2005] using the weights proposed by Robertson et al. [1999]. The thread then converts the resulting HDR image to LogLUV format, generates the descriptors for the image patches, looks up the edit values for those descriptors in the permutohedral lattice, and performs edge-aware upsampling of the masks. The remaining tasks of applying the edits and tonemapping the result (we use Reinhard et al.'s [2002] global tonemapping operator) are executed in the GPU with fragment shaders. Table 1 summarizes the execution times.

Task	Platform	
	x86	ARM
HDR blending	12.18 ms	48.46 ms
Patch descriptor computation	1.49 ms	18.37 ms
Lattice lookup	3.34 ms	12.92 ms
Domain transform	8.77 ms	24.64 ms
Appearance-based metering	1.79 ms	10.35 ms
GPU Processing	1.70 ms	44.38 ms
Total	29.27 ms	159.12 ms
RAW capture rate	20 FPS	10 FPS

Table 1: *Timing of the viewfinder editing framework on a laptop PC and a tablet at VGA resolution. The GPU processing consists of LogLUV decoding, tonemapping, applying edits, and UI rendering.*

During the editing session, the user may alter the tone, hue, or saturation locally, or tweak the global tonemapping. Once the shutter is actuated, we order the image sensor to capture high-resolution images at the exposure(s) chosen by our appearance-based metering.

Evaluation. Using publicly available HDR datasets, we simulated HDR capture with our appearance-based metering algorithm, and compare the resulting tonemapped output with [Hasinoff et al. 2010], which optimizes for the minimum SNR of the raw luminance data. As Figure 7 shows, the knowledge of the processing applied to the HDR data after acquisition is crucial in optimizing overall quality, since it allows our metering algorithm to safely ignore regions that will be saturated or too dark on the display. In contrast, Hasinoff's method prescribes a considerably longer exposure in order to match the SNR of the shadows with that of other regions. For this comparison, we assumed a 12-bit sensor and simulated read noise reported by [Granados et al. 2010] for a point-and-shoot camera.

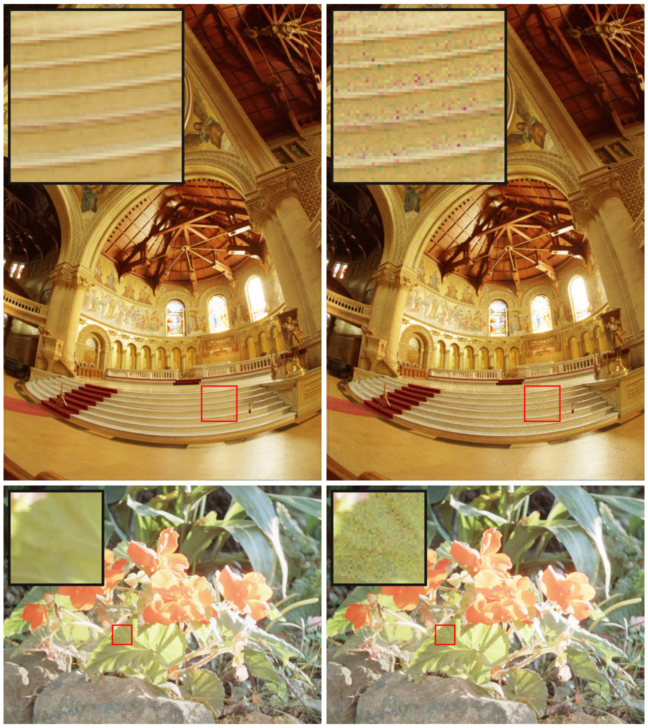


Figure 7: *Metering with a fixed time budget.* **Left:** *We simulated an HDR stack capture and tonemapping with our appearance-based metering, assuming the tonemapping shown. Our algorithm chose to capture at (0.50, 9.96, 92.18 ms) for the* `memorial` *dataset on the top, and (5.56, 55.31, 74.06 ms) for the* `wardflower` *dataset on the bottom.* **Right:** *We solved for the optimal exposures using Hasinoff's method for the same time budget and up to the same number of shots. The optimization yielded (0.027, 1.70, 101.30 ms) and (0.70, 3.01, 131.87 ms) for the two datasets. The tonemapped results generated from these exposures are shown here. These results exhibit more noise in regions that matter perceptually after tonemapping, compared to our results on the left. While our metering emphasizes faithfully acquiring regions that will maintain high contrast after tonemapping, Hasinoff's method maximizes the worst per-pixel SNR of the raw radiance, which causes much of the budget to be spent on the darkest parts of the scene, which are eventually tonemapped to black. This issue is aggravated by the very large dynamic range of the scene.*

We also collected a high-frame-rate image sequence from which we can simulate stack acquisition by integrating the frames appropriately, with the intention of demonstrating the effect of scene and camera motion on HDR photography. See Figure 8, in which we controlled for the image quality. Our appearance-based metering successfully captures both the shadows and the edited flowers to

[1] Intel Core i7-2760QM with NVIDIA Quadro 1000M GPU

[2] NVIDIA Tegra 3: quad-core Cortex-A9 CPU and ULP GeForce GPU

the extent needed, avoiding excessive motion blur in the process, whereas Hasinoff's metering parameters introduce more noticeable motion blur from the scene and leads to artifacts.

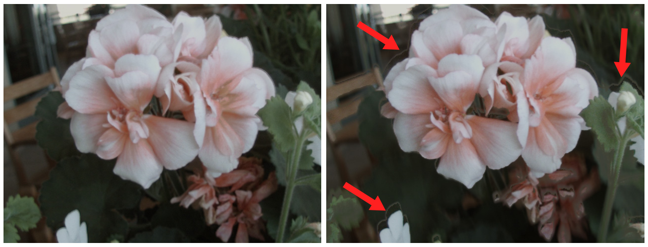


Figure 8: *Metering for a dynamic HDR scene. The scene contains flowers on a balcony swaying in the wind.* **Left:** *We simulated an HDR stack capture and tonemapping with our appearance-based metering, assuming the tonemapping shown. Our algorithm chose to capture at (5.00 ms, 22.00 ms).* **Right:** *We solved for the lowest total budget (85 ms) that would yield the same tonemapped output quality for Hasinoff's method, which is considerably longer than our total (27.00 ms). The tonemapped output created from these metering parameters is shown here. Note the halo and ghosting artifacts stemming from motion blur in the scene is more apparent on the right. We remark that the two simulated stack acquisitions are based on a high-speed multiframe dataset, and correspond to the same physical point in time.*

All in all, existing methods that seek to optimize for the raw signal would be more suitable for creating a very high-fidelity radiance map for a static HDR scene. When the time budget is limited or a lower-quality sensor is being used, knowing the post-processing pipeline helps our metering algorithm deal with these limitations. Empirical comparisons are given in Figure 1, in which we compare against a histogram-based HDR metering method.

6.2 Appearance-based Focal Stack Acquisition

The lens module on many mobile devices is equipped with a voice coil motor that can change the lens position very quickly. Exploiting this feature, we stream images to our pipeline while repeatedly cycling through multiple focus positions, effectively acquiring a "live" focal stack. Because changing the focus position also slightly alters the field of view, we scale each frame appropriately with a constant for which we calibrate beforehand. Then, the processing thread coalesces the most recent stack of frames and builds a depthmap and an all-focus image. In this step, the frames are registered to the most recent frame by analyzing the gyroscope trace. The all-focus image is analogous to the HDR scene radiance map in the HDR application, acting as the source for image patches for selections, as described in Section 4.1. The rendering thread continuously recomputes and displays the viewfinder output, obtained by merging the registered focal stack slices as described in Section 5.2.

The result of such compositions are shown in Figure 9. We found that for most practical scenes, local edits were crucial for producing a pronounced reduction in depth of field. The chief reason is that multiple objects at a meter away or farther $(0.0 - 1.0$ diopters) will have approximately equal-sized blur, since the circle of confusion grows linearly with the depth disparity in diopters. Hence, global edits cannot cause them to have distinctive blurs.

Lastly, producing a faithful all-focus image on a jerky camera is difficult because of imperfect registration will duplicate strong edges. This can be quite jarring, so we disable image blending when the camera is moving very fast, since the user cannot distinguish between the ghosting artifacts and the motion blur of the device itself.

6.3 Live Video Editing

Because the edit-propagation algorithm we have described is real-time, it can be used to edit live HDR videos as they are filmed. This obviates the need to revisit the videos afterwards and decode them for processing. See Figure 10 for a demonstration. We also note that while many existing edit propagation work on videos require keyframes with user strokes every 20 frames or so, our method does not; we trade off selection quality for robustness against scene and camera motion.

Finally, Figure 11 shows additional results created with viewfinder editing, demonstrating scenarios in which our framework can be useful. Supplementary video provides more demonstrations.

6.4 Limitations

Because the edit-propagation algorithm is based on appearance, our framework cannot distinguish similar but distinct objects, often forcing the user to select all or none of them. Using depth cues or reconstructing the scene geometry may help with discriminating objects with similar texture. User experience would also be improved by tolerance to inaccurate strokes [Subr et al. 2013]. We also found that specular objects can be difficult to track temporally, as their appearance can change drastically at different orientations.

The quality of selection and edit masks during interaction is far from perfect, albeit by a design choice. The goal was to provide an approximately WYSIWYG experience during the viewfindering phase to aid in composition and parameter selection, as well as to engage the user in a novel photography or videography experience. While the edit mask quality is improved via offline processing, increased capacity and availability of computational power in mobile cameras should close this gap in the future.

Next, when we blend multiple exposures to create a composite in real time, scene and camera motion can lead to artifacts along object boundaries. While camera motion can be combated with gyroscopes or vision-based algorithms as we do, scene motion requires non-rigid registration, which is prohibitively expensive. However, this limitation renders time-efficient stack acquisition even more important. Note that we avoid registration artifacts in the viewfinder by using edge-avoiding filtering across time (see Section 4.2).

We also found that bright outdoor conditions can produce glare on the display and lower the perceivable contrast range, altering the appearance of the edited image. However, we believe that this problem can be addressed to a large extent by recent advances in display technology, such as OLED displays.

Lastly, neither platform (the PointGrey SDK nor the FCam implementation on the Tegra tablet) used in the paper fully supported per-frame alteration of capture parameters, and as a result, the rate at which our application fetched image frames was cut in a third. The ability to alter capture parameters per-frame without loss of frame rate is critical in stack photography applications.

7 Conclusion and Future Work

We have presented the first system that implements image editing, stack composition, and tonemapping directly on a viewfinder, using a fast edit propagation framework, and demonstrated how to exploit the resulting WYSIWYG property to drive capture parameters in a way that leads to less noise and/or less capture time. Our algorithms are designed to handle streaming data, require no preprocessing, and are fast enough to provide useful feedback on a mobile device, for both the user and the camera control algorithms.

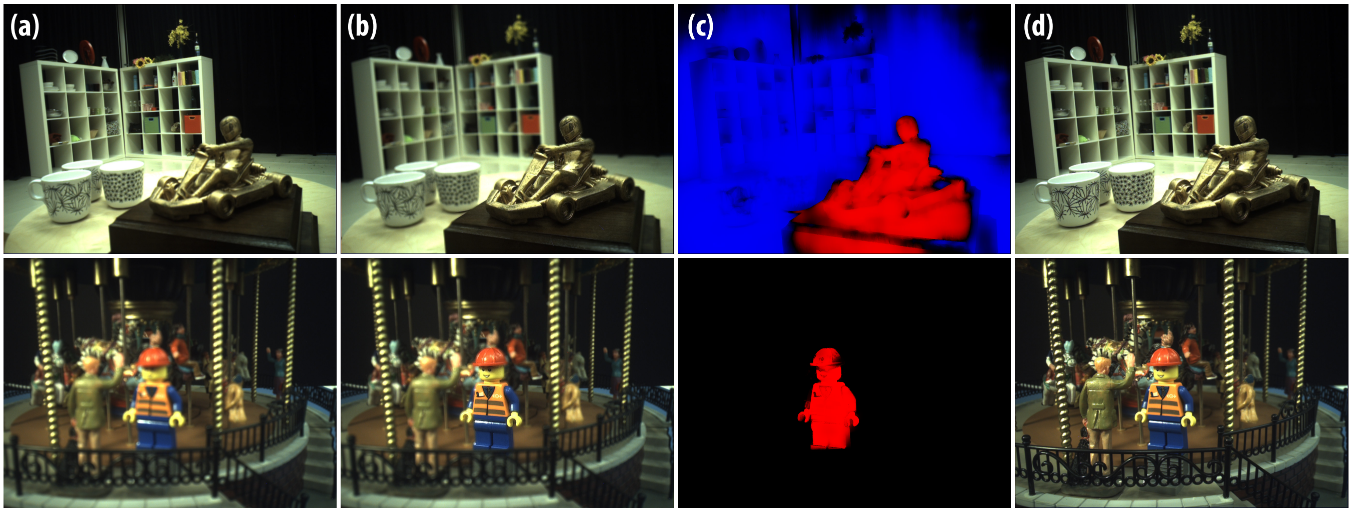


Figure 9: *Focus adjustment via viewfinder editing. In our focus editing application, we continuously sample the scene at several depths to build a depthmap, from which the viewfinder is synthesized.* **(a):** *Before any user interaction, the viewfinder shows a regular image.* **(b):** *The result of viewfinder editing, produced by acquiring a suitable focal stack and compositing the slices. On the top image, the user sharpened the figurine and blurred the background; on the bottom image, the user sharpened the Lego figure. See the supplementary video for the user interactions.* **(c):** *The edit mask produced for these compositions. Red corresponds to sharpening, and blue corresponds to blurring.* **(d):** *For reference, here we show photographs taken with the same tablet, each focused on the respective subject. In the top row, note that the defocus in the background is limited, whereas the user was able to attain the appearance of a much larger aperture with our system; in the bottom row, a conventional camera is unable to replicate our nonphysical depth of field, in which objects at the same depth are defocused differently.*

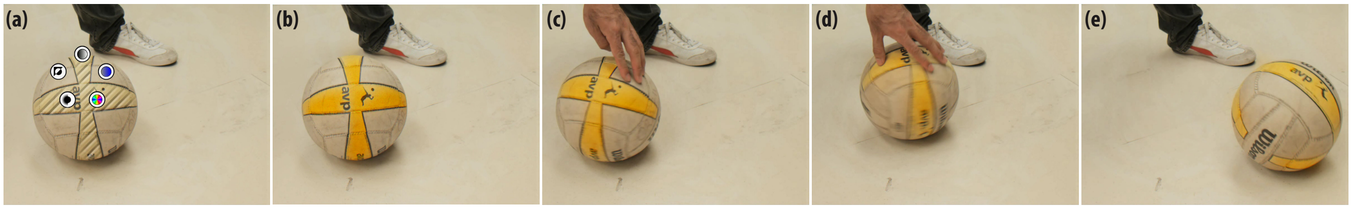


Figure 10: *Edit propagation on a live video.* **(a):** *In this example featuring a faded volleyball, the user marks a region as selected, corresponding to the ball's stripes as shown.* **(b):** *The user then applies an edit to accentuate the color, restoring the unfaded look.* **(c-e):** *This edit can be seen persisting through frames #150, #300, and #420 of a 30-fps video sequence, despite considerable motion of the ball between frames and the resulting motion blur. See the supplementary video for the entire sequence.*

For future work, we intend to re-examine aspects of the edit propagation that were sacrificed for performance, such as explicit per-frame tracking. A global coordinate system would provide more discriminativeness, for example to select only one of several similarly textured objects. We would like to mitigate issues inherent to a handheld viewfinder, such as noise at high gain, motion blur, and inaccurate user gestures. The descriptor could be improved to be robust against lighting changes or specular materials.

Acknowledgements We thank the anonymous reviewers and the NVIDIA Mobile Visual Computing team for their advice. Jongmin Baek acknowledges Lucent-Alcatel Stanford Graduate Fellowship. David Jacobs, Sung Hee Park, Sean Kim aided data collection.

References

Adams, A. B., Baek, J., and Davis, A. 2010. Fast high-dimensional filtering using the permutohedral lattice. *Computer Graphics Forum 29*, 2.

Adams, A. B., Talvala, E., Park, S. H., Jacobs, D. E., et al. 2010. The Frankencamera: An experimental platform for computational photography. *ACM Trans. Graph. 29*, 4.

Agarwala, A., Dontcheva, M., Agrawala, M., Drucker, S., Colburn, A., Curless, B., Salesin, D., and Cohen, M. 2004. Interactive digital photomontage. *ACM Trans. Graph. 23*, 3.

An, X., and Pellacini, F. 2008. Appprop: all-pairs appearance-space edit propagation. *ACM Trans. Graph. 27*, 3.

Bie, X., Huang, H., and Wang, W. 2011. Real time edit propagation by efficient sampling. *Comp. Graphics Forum 30*, 7.

Chen, X., Zou, D., Zhao, Q., and Tan, P. 2012. Manifold preserving edit propagation. *ACM Trans. Graph. 31*, 6.

Farbman, Z., Fattal, R., Lischinski, D., and Szeliski, R. 2008. Edge-preserving decompositions for multi-scale tone and detail manipulation. *ACM Trans. Graph. 27*, 3.

Farbman, Z., Fattal, R., and Lischinski, D. 2010. Diffusion maps for edge-aware image editing. *ACM Trans. Graph. 29*, 6.

Fattal, R. 2009. Edge-avoiding wavelets and their applications. *ACM Trans. Graph. 28*, 3.

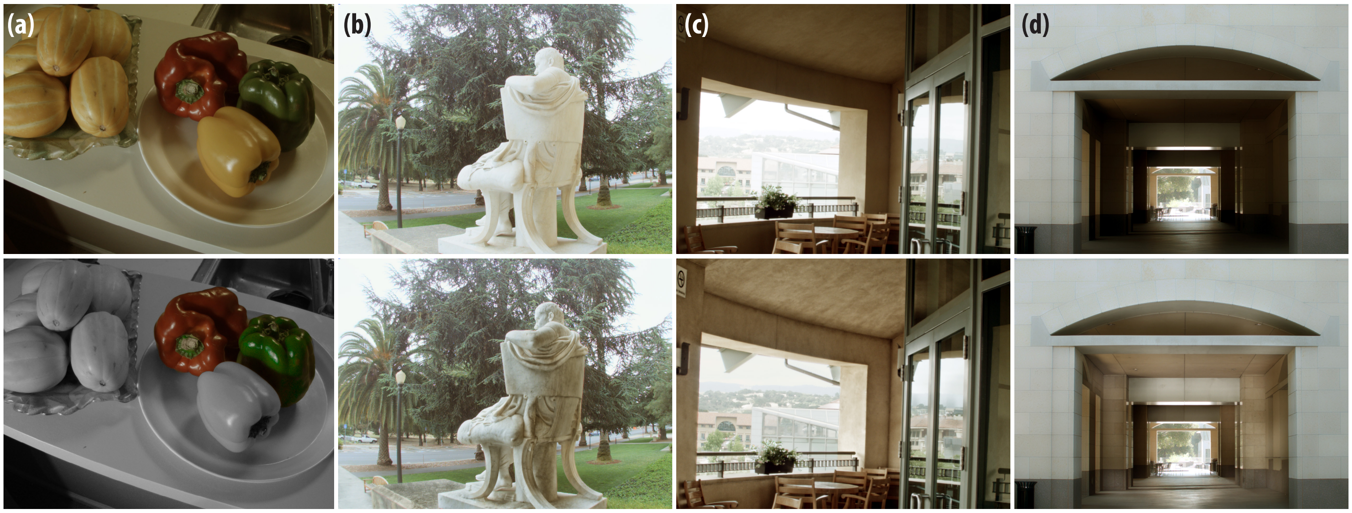


Figure 11: *Additional results of viewfinder editing. The HDR composites without local edits are shown on the top for reference. Our results are on the bottom.* **(a):** *The user desaturated all but two peppers on a countertop, and deepened the color of the green pepper.* **(b):** *The user darkened the white statue to enhance the local contrast. Its geometry and texture are now more conspicuous.* **(c):** *The user added local edits to brighten the indoor regions and darken the outdoors.* **(d):** *The user increased the brightness of the otherwise dimly lit corridor.*

GALLO, O., TICO, M., MANDUCHI, R., GELFAND, N., AND PULLI, K. 2012. Metering for exposure stacks. *Computer Graphics Forum 31*, 2.

GASTAL, E. S. L., AND OLIVEIRA, M. M. 2011. Domain transform for edge-aware image and video processing. *ACM Trans. Graph. 30*, 4.

GASTAL, E. S. L., AND OLIVEIRA, M. M. 2012. Adaptive manifolds for real-time high-dimensional filtering. *ACM Trans. Graph. 31*, 4.

GRANADOS, M., AJDIN, B., WAND, M., THEOBALT, C., SEIDEL, H.-P., AND LENSCH, H. P. A. 2010. Optimal HDR reconstruction with linear digital cameras. *IEEE Computer Vision and Pattern Recognition*, 215–222.

HASINOFF, S. W., AND KUTULAKOS, K. N. 2008. Light-efficient photography. In *European Conf. on Computer Vision*, Springer.

HASINOFF, S. W., DURAND, F., AND FREEMAN, W. T. 2010. Noise-optimal capture for high dynamic range photography. In *Computer Vision and Pattern Recognition*, IEEE.

JACOBS, D. E., BAEK, J., AND LEVOY, M. 2012. Focal stack compositing for depth of field control. Tech. Rep. 1, Stanford Computer Graphics Laboratory, 10.

LEVIN, A., LISCHINSKI, D., AND WEISS, Y. 2004. Colorization using optimization. *ACM Trans. Graph. 23*, 3.

LI, Y., ADELSON, E., AND AGARWALA, A. 2008. ScribbleBoost: Adding classification to edge-aware interpolation of local image and video adjustments. *Computer Graphics Forum 27*, 4.

LI, Y., JU, T., AND HU, S.-M. 2010. Instant propagation of sparse edits on images and videos. *Computer Graphics Forum*, 7.

LIANG, C. K., CHEN, W. C., AND GELFAND, N. 2010. Touchtone: Interactive local image adjustment using point-and-swipe. *Computer Graphics Forum 29*, 2.

LISCHINSKI, D., FARBMAN, Z., UYTTENDAELE, M., AND SZELISKI, R. 2006. Interactive local adjustment of tonal values. *ACM Trans. Graph. 25*, 3.

LIU, L., CHEN, R., WOLF, L., AND COHEN-OR, D. 2010. Optimizing photo composition. *Computer Graphic Forum 29*, 2.

MANTIUK, R., DALY, S., AND KEROFSKY, L. 2008. Display adaptive tone mapping. *ACM Trans. Graph. 27*, 3.

NIK SOFTWARE, 2012. Snapseed | snap it. tweak it. love it. share it. `http://www.snapseed.com`. Accessed: 12/11/2012.

RAGAN-KELLEY, J., ADAMS, A., PARIS, S., LEVOY, M., AMARASINGHE, S., AND DURAND, F. 2012. Decoupling algorithms from schedules for easy optimization of image processing pipelines. *ACM Trans. Graph. 31*, 4.

REINHARD, E., STARK, M., SHIRLEY, P., AND FERWERDA, J. 2002. Photographic tone reproduction for digital images. *ACM Trans. Graph. 21*, 3.

REINHARD, E., WARD, G., PATTANAIK, S., AND DEBEVEC, P. 2005. *High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting*. Morgan Kaufmann.

ROBERTSON, M. A., BORMAN, S., AND STEVENSON, R. L. 1999. Dynamic range improvement through multiple exposures. In *Int. Conf. on Image Processing*, IEEE.

RUBINSTEIN, M., SHAMIR, A., AND AVIDAN, S. 2008. Improved seam carving for video retargeting. *ACM Trans. Graph. 27*, 3.

SUBR, K., PARIS, S., SOLER, C., AND KAUTZ, J. 2013. Accurate binary image selection from inaccurate user input. *Computer Graphics Forum 32*, 2.

VAQUERO, D., GELFAND, N., TICO, M., PULLI, K., AND TURK, M. 2011. Generalized autofocus. In *Workshop on Applications of Computer Vision*, IEEE.

WINNEMÖLLER, H., OLSEN, S. C., AND GOOCH, B. 2006. Realtime video abstraction. *ACM Trans. Graph. 25*, 3.

XU, K., LI, Y., JU, T., HU, S.-M., AND LIU, T.-Q. 2009. Efficient affinity-based edit propagation using k-d tree. *ACM Trans. Graph. 28*, 5.