Removing Reflections from RAW Photos

Supplementary Material

Contents

A. Photometric reflection synthesis

As part of our photometric reflection synthesis pipeline, Func. 1, we compute a new exposure and white balance for the simulated mixture image, m, using Func. S1 and Func. S2, respectively. These functions follow ACR color processing, and use methods in the Adobe DNG SDK, Sec. G. The ACR color processing that produces XYZ source images is specified in Func. S3 and discussed in Sec. G.

A.1. White balancing

To compute a new white balance within Func. S2, we use the C5 white balancer of Afifi *et al.* [2]. C5 white balances an input image by using an additional n sample images that were captured from the same camera. We therefore cache n samples for each camera in the dataset of RAW images, and remove all images for which there were not $n = 7$ samples from the camera (7 is the C5 default).

White balancing with C5 requires that simulated mixtures $m = t+r$ in XYZ be transformed into camera color space. In Func. S2 we use the `XYZ_to_CAM` transform associated with the RAW source image of t to simulate a camera from which the mixture was captured, since t typically dominates r in the sum due to attenuation by the glass (see Sec. B). The white balancer produces a new white point `WhiteCAM_awb` in camera color space. We then follow ACR color processing of white points in camera color space by transforming `WhiteCAM_awb` into XY coordinates and computing a new `XYZ_to_CAM` transform using DNG SDK Func. S7. This new transform into XYZ is then composed with Bradford adaptation (Func. S2 line 5) to construct a single, linear white balancing transform that operates on images in XYZ space (line 6).

Over a large scale dataset, white balancer failures in-

Function S1 Compute the exposure of a simulated mixture m.

```
Input: A simulated mixture $m$ and its associated component images $(t, r)$
Output: An exposure value $e$
 1: Compute the WhiteXY of $t$.                              {SDK Func. S9}
 2: Compute transform XYZ_to_sRGB using WhiteXY.             {SDK Func. S11}
 3: Convert $m$ to linear sRGB using XYZ_to_sRGB.
 4: if no pixels in $t$ or $r$ are saturated then
 5:     Compute the mean pixel value $\mu$ of $m$
 6:     Compute the target value $\tau$ = sRGB_to_linear_sRGB(0.4). {SDK Func. S17}
 7:     return $e = \tau/\mu$                {Expose the mean to sRGB 0.4.}
 8: else
 9:     Convert $t$ and $r$ to linear sRGB using XYZ_to_sRGB.
10:     Compute $t_{\max} = \max(t)$ and $r_{\max} = \max(r)$.
11:     return $e = 1/\min(t_{\max}, r_{\max})$
12: end if
```

Function S2 Compute the white balance transform.

```
Input: A re-exposed XYZ mixture, $e \cdot m$, and the transmission $t$
Output: XYZ_to_XYZ_awb
1: Compute the XYZ_to_CAM using the WhiteXY of $t$.    {SDK Func. S7}
2: Transform $e \cdot m$ into camera color space using XYZ_to_CAM.
3: Compute a new white point WhiteCAM_awb in camera color space.
   {This work uses [2]}
4: Compute XYZ_to_CAM_awb and WhiteXY_awb from WhiteCAM_awb.
   {Func. S10+S7 or S8}
5: Compute XYZ_to_XYZ_D50 from WhiteXY_awb.    {SDK Func. S13}
6: return XYZ_to_XYZ_D50 · inv(XYZ_to_CAM_awb) · XYZ_to_CAM.
```

evitably occur. These are handled by culling m if the new white point is extremely different from the as-shot WhiteXY of t. Extreme changes to the true white point are not common because reflections that are of practical interest are either transparent, localized, or both. The new XY white point (WhiteXY_awb) can be further restricted to lie on the Planckian locus, and we found this to be sufficient for our source images, which were captured under typical illuminants. Projection from WhiteCAM_awb to WhiteXY_awb can be done using ACR Func. S10 and S7, or Func. S8 with the Planckian constraint, as noted in Func. S2, line 4.

In summary, the white balance computed in Func. S2 is a linear transform, which we denote XYZ_to_XYZ_awb, that composes three ACR color transforms: 1) it maps images into the camera color space of t, 2) it maps back to XYZ under a new white point, and 3) it applies Bradford adaptation to the D50 illuminant. In the simulation (Func. 1) this transform is applied to m, t, r, and c so they are interpreted with respect to the same white point, lines 8-9.

B. Geometric reflection synthesis

As part of our reflection removal pipeline, a geometric simulation is used to construct transmission and reflection pairs of images (t, r) from a dataset of pairs of scene-referred (RAW) photographs $(i, j) \in \mathcal{D}$ that were not captured with or through glass. These transmission and reflection pairs are then added together to form the training data for our models, with ground truth provided by the constituent images in each pair. This simulation approach overcomes the scaling bottleneck of capturing real reflection images for training, which is difficult because ground truth (without the glass present) is not readily available.

In particular, we synthesize transmission images $t = T(i)$ and reflection images $r = R(j)$ as functions of i and j that appropriately model Fresnel attenuation, perspective projection, double reflection, and defocus blur. We omit from T effects related to global color, dirt, and scratches since existing photo editing tools are well equipped to correct them after reflection removal. Examples are shown in Fig. S1, and an overview of the synthetic images is shown in Fig. S2.

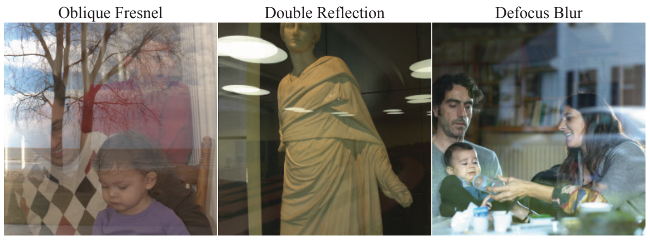

Figure S1. Simulated geometric properties at extreme values. Blurs are typically subtle.

B.1. Fresnel attenuation

Fresnel attenuation is the most essential property to simulate because it reduces the intensity of the reflected image. Specifically, reflections r are attenuated by a spatially varying factor α that depends on the angle of incidence θ_i at which light strikes the glass with respect to the surface normal vector. As derived in [25],

$$\alpha = \frac{1}{2}(\alpha_\perp + \alpha_\parallel) \tag{1}$$

$$\alpha_\perp = \frac{\sin^2(\theta_\mathrm{i} - \theta'_\mathrm{i})}{\sin^2(\theta_\mathrm{i} + \theta'_\mathrm{i})} \tag{2}$$

$$\alpha_\parallel = \frac{\tan^2(\theta_\mathrm{i} - \theta'_\mathrm{i})}{\tan^2(\theta_\mathrm{i} + \theta'_\mathrm{i})} \ , \tag{3}$$

where $\theta'_\mathrm{i} = \arcsin(\frac{1}{\kappa} \sin \theta_\mathrm{i})$, and κ is the refractive index of glass. For $\theta_\mathrm{i} \in [0°, 45°]$, Fresnel attenuation accounts for up to -4 stops (underexposure), and gradually strengthens to -1 stop for rays that glancingly strike the glass at 83°.

To specify θ_i, we define images $r = R(j)$ as originating from a mirror surface, with incident rays reaching the camera by the law of reflection. In the next section we decribe how to simulate a diversity of practical geometric configurations of the glass and camera to construct θ_i and thus compute α. Glass also attenuates the transmission $t = T(i)$ by $1 - \alpha$. This is typically close to 1, but at extreme angles it creates a visible darkening effect (Fig. S2, example 26). Typical attenuation levels are shown in Fig. S2 in the column labeled "reflection."

B.2. Camera projection

We model consumer photography applications in which one sees a subject partially visible behind glass and takes a picture of it. This constrains the relative pose of the camera and glass, and introduces natural priors on the location and appearance of reflections. For example, skies typically reflect near the top of images, and reflections are typically stronger at the edges of photos where the camera rays strike the glass at a relatively higher angle of incidence, θ_i.

Inclination angle ϕ. Most glass is approximately vertical, so if the viewpoint of t looks upward, the viewpoint of r

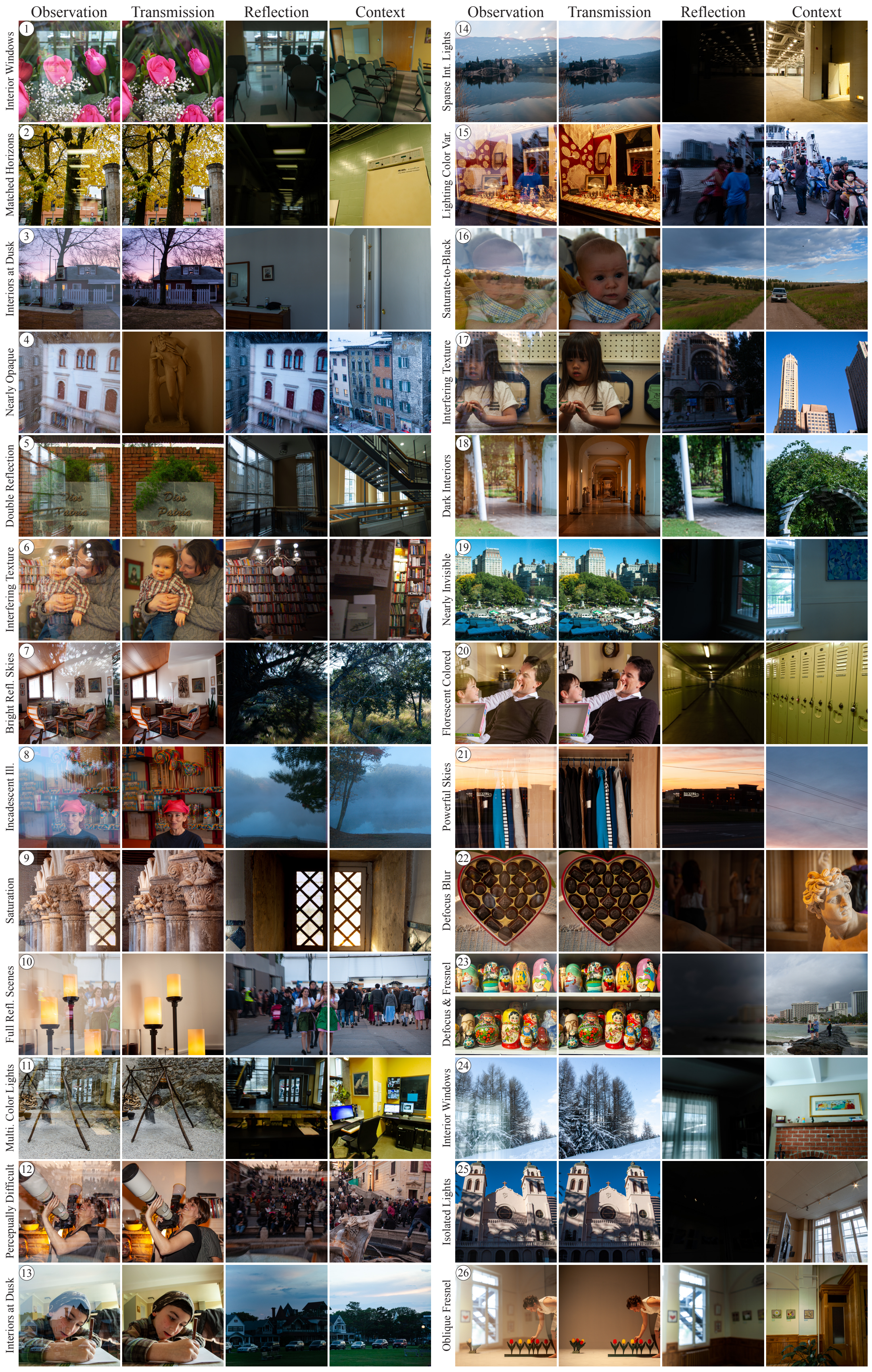

Figure S2. Dataset overview (2048p). **All images are simulations that use RAW sources.** The numbers are referenced in the text.

should as well. We use a pose estimator [18], and augment the search for realistic pairings (t, r), Sec. 3.1, by checking if $\phi_t - \phi_r = \Delta_\phi$ is below a maximum absolute value. In addition to this inclination discrepancy filter, images are culled if their inclination angle ϕ exceeds a threshold. This aligns the horizons of t and r, which introduces spatial priors, as illustrated in Fig. S2 examples 2, 5, 14, 20, and 24.

Roll angle ρ. Images are culled if their estimated roll ρ exceeds a maximum absolute value as these typically indicate that the pose estimator has failed.

Field of view. Images are also culled if the estimated vertical FOV is zero, which indicates general pose estimation failures. Otherwise we randomly sample FOV $\sim \mathcal{U}(\text{FOV}_\text{min}, \text{FOV}_\text{max})$, where $\mathcal{U}$ is the uniform distribution.

Azimuth angle θ. Most glass in consumer photography is roughly planar. We constrain the camera azimuth with respect to the glass so that the camera rays strike this plane (accounting for the FOV). We randomly sample $\theta \sim \mathcal{U}(-\theta_\text{max}, \theta_\text{max})$. The effect of this constraint can be most easily seen in Fig. S2 where highly oblique camera angles create spatially varying Fresnel attenuation across the reflection component (examples 6, 7, 14, 19, 23, and 26).

B.3. Defocus blur

Recently Lei [28] found that performance of state-of-the-art methods degrades significantly for sharp reflections due to an imbalance of blurry images in training and testing data. Physically based methods have been developed to introduce realistic defocus blur using depth maps [24], but this introduces a data collection bottleneck by requiring RGBD cameras that also have physical limitations. We instead model a physically based prior on the amount of defocus blur.

Defocus blurs are determined by the camera focus depth, aperture, and focal length. Points on an object at depth d_o that differs from the focus depth d_f project to a circular region with diameter δ,

$$\delta = \frac{|d_o - d_f|}{d_o} \frac{f^2}{N(d_f - f)} \ , \tag{4}$$

where f is the focal length, and N is the aperture f-number. This *circle of confusion* [15] is magnified with increasing focal length f or decreasing N.

Defocused images are simulated by sampling diameters δ (mm) for the circle of confusion. The focal length f (mm) and aperture N (dimensionless) are sampled according to their physical ranges in mobile cameras. The object and focus depths d_o and d_f (feet) are sampled $d \sim \mathcal{U}(d_\text{min}, d_\text{max})$, in the plausible and finite range of scene depths to which δ is sensitive. The diameter δ (mm) is converted to a percentage of the sensor height, δ_p (the minimum sensor dimension). Reflections $r = R(j)$ are blurred by convolving them with a circular defocus kernel with pixel diameter $h\delta_p$, where h is the minimum dimension of the image j in pixels. We maintain this physical calibration when images are cropped into halves to simulate contextual views (Sec. 3.3 and Section C).

Our physically based sampling procedure simulates reflections with a realistic amount of defocus blur for consumer photography. An example of a strongly blurred reflection is shown in Fig. S1. We however find that reflections are typically sharp, as Lei [28] also notes. Typical defocus blurs are shown in Fig. S2; strong blurs are shown in examples 22 and 23.

B.4. HDR Environment sampling

A dataset of indoor 360° HDR Image-Based-Lights (IBLs) are used as an additional source of scene-referred images [14]. Artificial light sources in HDR images are typically not saturated, which makes it possible to simulate reflections of light sources that are not saturated (or, underexposed RAW images could be used).

When one of the images in a pair $(i, j) \in \mathcal{D}$ is an IBL, a synthetic camera is constructed with a pose that matches the RAW image to which it is paired (see Sec. B.2), excepting that the azimuth θ is sampled independently and uniformly at random in $360°$. Contextual images c are simulated by a second synthetic camera within the IBL with an adjacent, non-overlapping FOV.

The IBLs [14] are captured under a fixed white point, which allows for the color of the illuminant (i.e., its white balance) to be mixed correctly with the RAW data. We calibrate the exposure of these indoor IBLs by setting their median intensity to match the median value of all indoor RAW images (the median contends with saturated pixels). This cropped HDR image can be photometrically combined, and geometrically transformed using functions T or R. The effect of HDRs is shown in Fig. S2: reflected light sources, windows, etc. are produced by HDRs in examples 1, 2, 5, 6, 11, 14, 20, 24, and 26.

B.5. Double reflection

Glass panes introduce multiple reflective surfaces that create a double reflection or "ghosting" effect. Shih et al. [39] ascribe the effect of double reflection to the thickness of a single or double pane, and show shifts up to 4 pixels for thicknesses in 3–10mm under some viewing distances, but double reflections are often much larger. Gaps between panes reach 20mm as reported commercially, and each pane adds up to 7mm. These multiple reflecting surfaces are also not necessarily parallel, uniformly thick, or flat as assumed in [39]. These factors produce significant double reflections even in modern windows, including when the camera is distant. We simulate these complex effects by adopting the geometric model of [39] and allowing a greater range of thicknesses, 8–20mm. We uniformly sample a glass thickness, physical viewing distance, and refractive index. These facilitate a ray tracing procedure, detailed below. Fig. S2 shows double reflections in the dataset; see examples 2, 4, 5, 7, 8, 12, 14, and 15.

The primary reflection that contributes to $r = R(j)$ is determined by the Fresnel attenuation αj as described in Sec. B.1. Specifically, the intensity of light at each homogeneous image coordinate $\mathbf{x}$ is $\alpha(\mathbf{x})j(\mathbf{x})$, because we have defined $j(\mathbf{x})$ as encoding the light along the incident rays $\mathbf{r}$ with $\angle(\mathbf{x}, \mathbf{r}) = 2\theta_{\mathrm{i}}$ where θ_{i} is the angle of incidence. We simulate a second reflection by tracing the camera rays $\mathbf{x}$ through a simulated single pane of uniform thickness to identify the coordinates $\mathbf{x}'$ at which they would emerge from the glass after being internally reflected from the back surface of the pane. Coordinates $\mathbf{x}'$ are shifted according to their transit distance within the glass, which is determined by the law of reflection and Snell's law. We neglect the latter as insignificant in comparison to the former and the various non-modeled physical effects of real glass panes. Under these assumptions, rays that enter the glass at $\mathbf{x}$ emerge at $\mathbf{x}'$ in direction $\mathbf{r}$. An image j' is needed to describe the intensity of incident light in direction $\mathbf{r}$ at $\mathbf{x}'$, but $j'(\mathbf{x}') \neq j(\mathbf{x}')$ because we have defined $j(\mathbf{x}')$ as describing the light reflected from a corresponding direction $\mathbf{r}'$. Since we do not have j', we assume that the light field is sufficiently smooth that $j'(\mathbf{x}') \approx j(\mathbf{x}')$, since $\angle(\mathbf{r}, \mathbf{r}') = \angle(\mathbf{x}, \mathbf{x}')$ is small. We therefore warp j such that $\mathbf{x}' \mapsto \mathbf{x}$, and combine this warped image j_w with j to produce a double reflection image.

The double reflection image is given by $j_d = \alpha j + \beta j_w$, where α is the known Fresnel attenuation due to the primary reflection (see Sec. B.1), and β specifies the attenuation of the rays that travel into the glass before they are internally reflected back to the camera. These latter rays encounter three surfaces, and lose intensity at each one. The first surface is the front face of the glass, where they are mildly attenuated by $1 - \alpha$ as they transmit into the glass. Second is the back face, where they reflect and are attenuated again according to their angle of incidence, which has been altered by Snell's law. This change of incidence angle however has a negligible effect on the the Fresnel attenuation factor within the typical incidence ranges. We therefore use α as the attenuation at the second surface. Lastly, the rays re-encounter the front face of the glass (now from within) where they transmit out of the glass and are attenuated again by approximately $1 - \alpha$. This gives $\beta = (1 - \alpha)\alpha(1 - \alpha)$.

Fig. S1 shows an example of a simulated double reflection, selected to show a case when the primary and secondary reflections are significantly shifted. We note that no doubling effect can occur along the direction of the glass surface normal because the rays that enter the glass re-emerge after internal reflection at the same location they entered. Thus double reflection fields that follow our geometric model (and the model of [39]), in which there are two perfectly parallel planes, must exhibit a radial pattern around the image of the glass surface normal. These patterns are not always apparent in practice, which suggests that the geometrical arrangement of glass surfaces that is described by Shih [39] omits important factors. We nonetheless adopt their model as being sufficient because visible reflections are typically localized to regions of an image, which obscures the presence or absence of a radial center.

C. The contextual photo

One arrangement of a primary and a contextual camera is shown in Fig. S3 (see caption for explanation). This specific arrangement of cameras is neither required nor typical in practice, but it reveals general geometric differences between the views of primary and contextual cameras. The view of the so-called reflection camera is translated by $2d$, twice the distance d to the glass. Furthermore, if the contextual camera is rotated 180° from the primary, the latter view will be in an opposite direction. At extreme rotations, the views will have little or no overlap.

Because the translation and rotation of the contextual camera view can differ significantly from the primary camera, it is difficult to simulate a contextual view using a dataset of image pairs $t = T(i)$ and $r = R(j)$ that are used to create mixture images m from the perspective of a primary camera. In particular, content from j should not be copied into a simulated contextual image c, as trained models could learn to cheat by searching for patches of r that have the same perspective projection in c. Such patches will not be present in practice.

We create a scalably large dataset of contextual images c by noting that c will often contain no common content with r unless the photographer is asked to point the camera at what they see in the reflection. We minimize such burden on the photographer, and define the contextual image c as any image of the reflection scene that does not view the same parts of the scene as r. This definition allows contextual images to capture lighting information (sunlight, incandescent, etc.) and scene semantics (outdoor, indoor, city, nature, etc.) to aid reflection removal.

To construct c, we crop reflection source images, j, into non-overlapping left/right or top/bottom squares (j_0, j_1), and similarly for transmission source images i. This yields four pairs for simulation $(i_a, j_b) : (a, b) \in \{0, 1\}^2$ for all $(i, j) \in \mathcal{D}$. The mixture and context images are (m_{ab}, c_{1-b}), where $m_{ab} = \mathcal{C}(T(i_a) + R(j_b))$, and $c_{1-b} = \mathcal{C}(j_{1-b})$. We fix the capture function $\mathcal{C}$ for both m and c to have the same white balance so c can describe the color of the reflection scene in addition to its semantics. Fig. S2 shows example contextual photos.

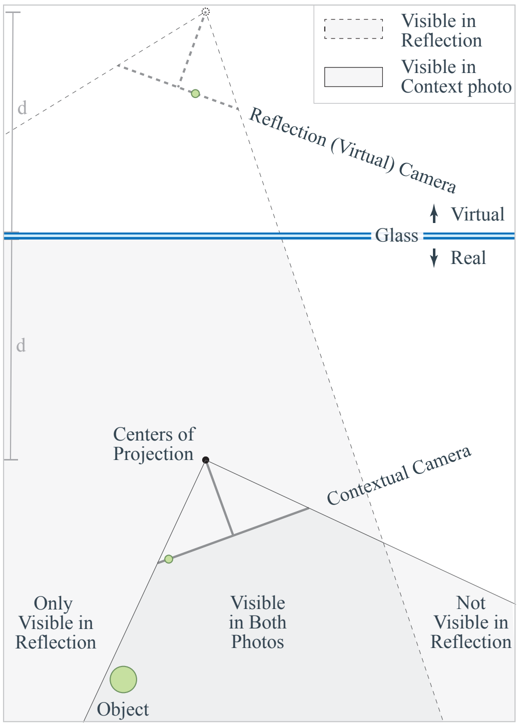

Figure S3. The geometry of a primary and contextual camera view. In this figure the two views are co-located (black dot), and the latter is rotated 180° with respect to the former. Neither condition is required or typical; they are shown to illustrate one possible geometric arrangement. The contextual camera frustum is shown at the bottom ($\perp$ symbol); the primary camera frustum is not drawn. The reflection contains the scenery that would be captured by a virtual camera behind the glass (open circle, dashed $\perp$), equidistant to the glass wrt the primary camera, and swung azimuthally in the opposite direction. Note, the object (green circle) appears at the right edge of the contextual camera's image (small green circle above it), but slightly left of center in the virtual camera view (small green circle near top of figure), and hence it is slightly right of center in the captured primary photo because the virtual camera is flipped left/right.

D. Data collection

Below are the data search and collection methods summarized in Sec. 3.1.

D.1. Mixture search

Well-exposed mixtures m are identified by checking if the mean pixel value is within a normal distribution over the pixel values in the dataset of RAW images.

Well mixed m are identified by computing the SSIM between m and t as a block-wise image, and checking if the mean of this SSIM image is within a useful range: if the SSIM is too high, the reflection is imperceptible; if it is too low, the mixture is not visually interpretable, even by a human. We compute this single channel SSIM image as a weighted average of the corresponding per-channel SSIM images. The weights are the average value in each color channel, which better accounts for strongly colored images. Lastly, the standard deviation of this single channel SSIM image is checked to remove reflections that are imperceptible, but nonetheless produce a low mean SSIM by spreading their power broadly (they have low spatial variance). The effect of this search can be seen in Fig. S2: simulated reflections can be faint or nearly invisible (examples 19, 22, 23, and 25); or so strong that the transmission is nearly invisible or slightly difficult interpret (examples 4, 12, 16, and 21).

D.2. Source images

We collect all images at their native RAW camera resolution to facilitate training upsampling methods. We label all images, including IBLs (Sec. B.4), as outdoor $\mathcal{O}$ and indoor $\mathcal{I}$ since glass typically separates indoor and outdoor spaces. This information is available in existing datasets [8, 10, 14], and can be collected at large scales via crowd-sourcing. We define our dataset $\mathcal{D}$ of pairs (i, j) for simulation as $\mathcal{D} = (\mathcal{O} \times \mathcal{I}) \cup (\mathcal{I} \times \mathcal{O}) \cup (\mathcal{I} \times \mathcal{I}) - \mathcal{P}$, where $\mathcal{P}$ is all pairs $i = j$. The set $\mathcal{O} \times \mathcal{O}$ is uncommon, and should be included sparingly following empirical priors. We omit them.

These pairings of source images interact with the mixture search process (Sec. D.1) to introduce photometric and semantic priors, as seen in Fig. S2. Outdoor transmission scenes typically have reflections of indoor light sources or windows (examples 1, 2, 5, 9, 11, 14, 19, 24) unless it is dusk (example 3). Indoor transmission scenes typically have strong reflections of skies or full outdoor scenes due to the brightness of natural light (examples 4, 7, 8, 10, 12, 15, 16, 17, 18, 21) unless it is dusk (example 13). Lastly, indoor-indoor pairings are often complex because t and r are typically similar in brightness (examples 6, 20, 26).

D.3. Simulation settings

Two *capture scenarios* are generated for each pair $(i, j) \in \mathcal{D}$. A virtual camera is posed randomly with maximum azimuth $\theta_{\max} = 50°$ toward the glass and FOV $\sim \mathcal{U}(50°, 80°)$, where $\mathcal{U}$ denotes the uniform distribution. Pairs (i, j) are culled if $|\Delta_\phi| > 15°$, or either image has absolute inclination value $|\phi| > 45°$ or roll $|\rho| > 10°$ (see Sec. B.2). Lastly, capture scenarios are also culled if the camera rays from more than 4 pixels do not strike the glass (they are parallel or divergent). This final check ensures that the glass fills the FOV.

We compute spatially varying Fresnel attenuation with index of refraction $\kappa \sim \mathcal{U}(1.47, 1.53)$ [45]; see Sec. B.1. Double reflections are simulated with glass thickness (mm) in $\mathcal{U}(8, 20)$ at distances (mm) in $\mathcal{U}(500, 2000)$, with 50% probability of being a double pane; see Sec. B.5. Defocus blur is simulated with object and focus distances (ft) in $\mathcal{U}(1, 100)$ with aperture and focal length of iPhone main

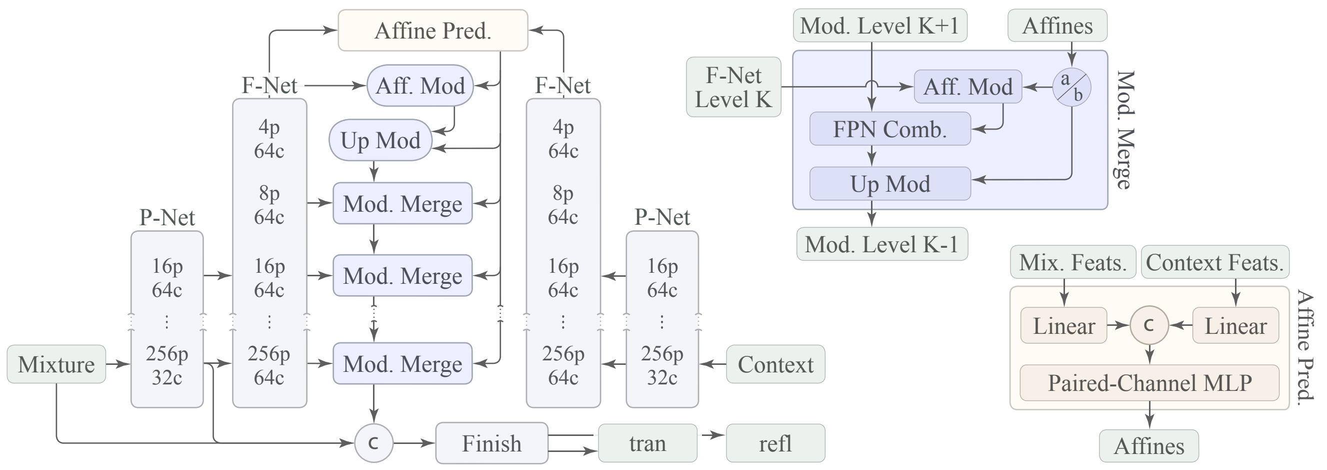

Figure S4. The base model. Mixture and context images are projected into a high dimensional space using a shared backbone [43] (labeled *P-Net*; weights are shared), and a feature fusion network [44] (labeled *F-Net*; weights are shared). The context features are used to predict affine transforms for each feature channel at each resolution. Channel-wise modulation is used because contextual photos do not always include content that can be matched. Modulation can help identify the reflection in the feature space. We use two conv-mod-deconv operations of [23] within the modulated merge blocks. The FPN combine op is a fast normalized fusion module from the BiFPN architecture [44].

cameras, $N \approx 1.6$ and $f \approx 26$mm (35mm equivalent units); see Sec. B.3. Simulated mixtures are culled if the mean SSIM between m and t falls outside of $[0.4, 0.94]$, or if the standard deviation of this SSIM is below 0.05; see Sec. D.1.

E. Reflection removal

Here we provide details of the base model architecture, as summarized in Sec. 4.1, and the upsampler, Sec. 4.2.

E.1. Base model

The base model is designed to leverage local and global features, Fig. S4, and produce 256^2 pixel outputs in about 1 second on a mobile device to meet req. 4 (see Sec. 1).

A feature backbone [43] is used to project m into a linear, high dimensional space (32-D) and compute semantic features (labeled *P-Net*). Features are at a variety of spatial and channel resolutions: $(256, 32)$, $(256, 16)$, $(128, 24)$, $(64, 40)$, $(32, 112)$. These features include the outputs of the initial convolution layer of the `EfficientNet-B1` variant of [43] (as implemented by [54]), which we modify to use an initial stride of 1 rather than 2 so no initial down-sampling is performed on the input 256×256 pixel images.

The multi-resolution feature tensors from the backbone are next fused into 64 channels at the input resolution using the `D0` variant of the EfficientDet feature pyramid architecture [44, 54] (labeled *F-Net*). This architecture first augments the input features with three additional levels: $(16, 64)$, $(8, 64)$, $(4, 64)$, where each results from a 2×2 maxpool with stride 2, and the first is preceded by a 1×1 conv, batch norm, and no activation. The augmented input features are then input to a series of so-called BiFPN layers [44] (see Fig. 3), which fuse features from low resolution to high, and then back to low resolution, in a zigzag operation that is repeated three times. To obtain high resolution fused features at only the input resolution of 256p, we add a fourth repetition in which we omit the final high-to-low pass. We furthermore modify the low-to-high pass to incorporate the contextual image, as described next.

The contextual image, c, is passed through the same F-Net and P-Net using the same weights as m. The features of c and m at the lowest resolution are input to an affine prediction module, Fig. S4 (lower right). This module first vectorizes its two $64 \times 4 \times 4$ inputs, and passes them through a fully connected layer to transform them into two 64-D vectors. These vectors embody 64 pairs of channels, which we concatenate and input to an MLP (labeled *paired-channel MLP*) that predicts affine transforms that modify the features of the FPN during the final low-to-high pass.

The paired-channel MLP is a series of grouped convolutions that implement 64 independent MLPs followed by a fully connected layer. These 64 MLPs each have 2 inputs, 2 hidden, and 1 output dimension, with leaky ReLUs after each layer. The inputs to these MLPs are corresponding pairs of channels from m and c. The outputs compose a single 64-D vector that is input to a fully connected layer to predict $64 \times K \times 2$ affine transforms, two for each of the $64 \times K$ channels and levels of the FPN.[7] Conceptually, this paired-channel MLP has the capacity to compare c and m to identify channels that match the reflection scene, and to determine

[7]Note that we include two levels at 256^2 pixels, in correspondence with the resolutions of the features that we extract from the backbone.

how to transform those channels to remove reflections.

The predicted affines from the paired-channel MLP are used in the final low-to-high pass of the FPN to perform a series of modulated merge operations, Fig. S4 (upper right). These merge operations use two affine transforms per feature channel, labeled a and b in Fig. S4. Transforms a are used to perform a conv-mod-deconv operation ala StyleGan [23] on the features of m from FPN level K. These features are subsequently combined with the features from level $K+1$ by resampling the latter features $2\times$ and using fast normalized fusion [44] (labeled *FPN Combine*). These combined features are modified with a second conv-mod-deconv operation using the second group of affines, b.

The final modulated merge produces 64-D features at 256^2 pixels. These features are concatenated with m and the features from the first layer, for a total of $3+32+64=99$ channels. A convolutional finishing module is then applied. This module has the capacity to further identify and finally render t and r. To simplify comparison to prior work, our finishing module is the head in [59]. The first layer is 1×1, and projects the 99 input features to 64-D, which is maintained in the remaining operations. Those operations are 3×3 convolutions dilated by $(1,2,4,8,16,32,64,1)$; each is followed by a batch norm and leaky ReLU. A final 1×1 convolution generates 6 channels: t and r.

The model is trained using the perceptual, adversarial, and gradient losses of Zhang *et al*. [59] with a ResNet-based discriminator [23], and optimized 5-tap derivatives [13] in the exclusion loss to suppress grid artifacts. We also adopt the l_1 reflection loss of [59] to minimize arbitrary differences to prior work. Perceptual losses are computed in non-linear sRGB by applying gamma compression and using VGG19 features, weighted to contribute equally. Crucially, we train end-to-end from randomly initialized weights.

E.2. Upsampler

The upsampler is shown in Fig. S5 and introduced in Sec. 4.2. It transforms low resolution outputs t, r from the base model to a flexible output resolution. We use a Gaussian pyramid and apply the same upsampler at each level. The upsampler projects the low-res, 3-channel inputs (m,r,t) into a high dimensional space ϕ using convolutions in an expand-and-contract pattern. We use 3×3 kernels for feature expansion, and 1×1 kernels for contraction. There are 3 layers with hidden dimensions $(32,16)$, $(64,32)$, $(128,64)$. We use leaky ReLU between the hidden layers, and no skip connections. Batch norms are omitted to facilitate a feature matching process, described next.

The components t and r are separated by identifying low resolution features ϕ_t and ϕ_r in the low resolution mixture ϕ_m. We predict 2, per-pixel, per-channel low resolution masks using a mask prediction module, Fig. S5 (bottom), which uses a paired-channel MLP (defined in Sec. E.1) to predict its affine transforms (see also Sec. 4.2). The joint mask predictor also uses a paired-channel MLP, but it directly outputs the final masks rather than affine transforms. The input, hidden, and output dimensions of both paired-channel MLPs are $(2,2,2,1,2)$. The final layer is fully connected. As noted in Sec. E.1, these MLPs can be implemented efficiently as a series of grouped 1×1 convolutions.

The finishing network is a series of 3×3 convolutions that are distinguished by the number of channels and dilation rates, 128:$(1,2)$, 96:$(1,2,4,8)$, 64:$(1,1)$. A final 1×1 convolution produces the 6 channels of output for T and R.

The upsampler is trained using a cycle-consistency loss on the predicted transmission and reflection, in addition to the losses of Pawan et al. [35]. Perceptual features are computed by converting to non-linear sRGB. For each predicted high resolution image $x'_{\mathrm{H}} \in \{t'_{\mathrm{H}}, r'_{\mathrm{H}}\}$ the loss is a weighted sum of the following terms: $\mathbb{E}[|x'_{\mathrm{H}} - x_{\mathrm{H}}|]$, $\mathbb{E}[(x'_{\mathrm{H}} - x_{\mathrm{H}})^2]$, $\mathbb{E}[|\nabla x'_{\mathrm{H}} - \nabla x_{\mathrm{H}}|]$, $\mathrm{LPIPS}(x'_{\mathrm{H}}, x_{\mathrm{H}})$ and $\mathbb{E}[|D(x'_{\mathrm{H}}) - x_{\mathrm{L}}|]$, where D downsamples x'_{H} to compute the cycle consistency loss, and $\mathbb{E}$ denotes expectation over spatial dimensions and (where applicable) the output of the gradient operator. We use both the l_1 and l_2 norms to avoid introducing arbitrary differences to prior work [35]. The norms are weighted 0.2, gradient terms 0.4, LPIPS 0.8, and cycle consistency 10. These losses are accumulated over three upsampling levels from 128^2 to 1024^2 pixels (smaller than the 256^2 pixel output size of the base model to contend with memory constraints during training). The upsampler is trained first on the ground truth low resolution inputs, and fine tuned on the output of the base de-reflection model. When fine tuning, the 256^2 pixel outputs of the base model are downsampled to 128^2 pixels. At test time, no downsampling is applied. The upsampler takes 256^2 pixel images as input, and produces output at 2048^2 pixels and higher.

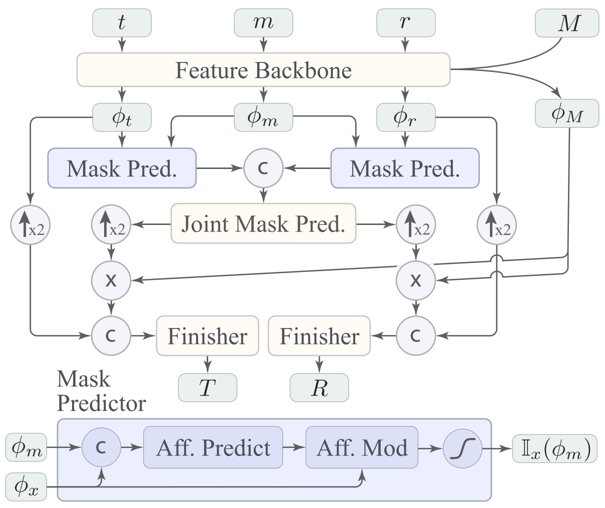

Figure S5. Upsampler at one pyramid level.

F. Results

Here we provide results and discussion in addition to Sec. 5.

F.1. Evaluation methods

Ground truth capture. Ground truth capture was used for Fig. 5, Fig. 7, and Fig. S6. Mixture images m were captured with ground truth r by placing a black material behind the glass and taking a second photo. Images t were computed $t = m - r$ in linear sRGB after normalizing the exposures and using the white point of m (see ACR Step 6, Sec. 3). Nonetheless, we found it necessary to capture m and r with fixed exposures and white points to avoid imprecisions in the values that are stored in RAW metadata.

SSIM computation. We report SSIM values between pairs of RAW images (a, b) by first transforming them into linear sRGB (ACR Step 6, Sec. 3) using the white point of a. By using a consistent white balance, across the ground truth m, t, and r, we penalize errors in the white balance of the estimated t and r. For SSIM computation, images are then converted to non-linear sRGB by applying standard gamma compression (ACR Step 8):

$$x_{\text{sRGB}} = \begin{cases} 12.92x & x \leq 0.0031308 \\ 1.055x^{1/2.4} & x > 0.0031308 \end{cases}$$

where x are pixel values in a linear sRGB image. We omit tone mapping operations (ACR Step 7) to remove subjectivity from the SSIM values. Lastly, SSIMs are reported as averages over the low-resolution images (denoted t, r) and high-resolution images (denoted T, R).

Ground truth photographs. Ground truth images were captured in three common scenarios: 1) looking out of a home window, Fig. 7; 2) photographing artwork, Fig. S6 (left); and 3) looking into a display case, Fig. S6 (right). In scenario one, the illuminant in the reflection scene is approximately 3,000K, and the white point of the mixture image is 7,300K. In scenario two, these values are 6,000K and 6,850K, and in scenario three they are 6,000K and 3,627K.

F.2. Base model comparisons

Reflections with ground truth. In Fig. S6 we show two images with ground truth reflections: photographing artwork, and looking into a display case. These complement Fig. 7, in which a photo was taken when looking out of a home window. The home window view includes an interior reflection that is strongly yellow due to the indoor illuminant color. In contrast, the artwork in Fig. S6 is illuminated by the same light source as the reflection scene, which produces a correctly white balanced reflection that consequently has more diverse colors. In contrast, the display case in Fig. S6 (right) reflects an outdoor scene with a different white balance, which produces a strong, blue reflection. This outdoor reflection is visible over broad regions because the illuminant is powerful enough to reflect off of the diffuse ground and sidewalk surfaces with sufficient intensity to be visible over the contents of the display case. These exterior reflections are also qualitatively different from those in photo of the artwork, where the reflection is sparse. The SSIM of the artwork is therefore high on average (0.994), but the reflections are locally strong, whereas the SSIM of the display case is low (0.833) because the reflection affects broad regions.

Our base models improve the SSIM of t and r in all of these ground truth examples (labeled in lower case t, r), and this extends to the upsampled results (labeled in uppercase T, R) whereas prior works do not perform as well (Fig. S6 and Fig. 7). Our contextual model produces a more correct transmission and reflection image on the artwork. On the display case, the contextual model improves the reflection, whereas the cars are not fully removed. We believe this variation results from saturated regions in the sky of the contextual photo, where we use the as-shot illuminant color in the EXIF to recover the color of the saturated pixels. Lastly, the method of Zhang [59] associates blue colored content with the reflection in both images, but this is incorrect for the painting; the transmission image is therefore distorted.

We found that our model removes blue colored reflections consistently well, and we believe that this results from their commonness (outdoor illuminants are powerful and therefore frequently create reflections that appear blue when mixed with interior scenes). The yellow color of interior reflections on outdoor scenes also seems to help, as our model can separate textured objects like the painting on the wall in Fig. 7 from the tree texture. We also tried illuminating the indoor scene in Fig. 7 with a special studio light that is much more powerful than a typical interior light source. This created a warm indoor reflection that was analogous to the display case in Fig. S6 (right), where the reflection covers large parts of the transmission scene. Our model results are less consistent in this artificial situation. We believe this is because it is rare for interiors to be flooded with such strong lights, and so their reflections are uncommon in our training data. We find that interior reflections tend to appear in small regions because most artificial light sources are weak—only objects near the light will be bright enough to reflect over outdoor scenes. At nighttime, however, consumer illuminants easily reflect over dark cityscapes. We found that our model results are less consistent on such images. This can be improved by augmenting the dataset with source images $t = T(i)$ that were taken at nighttime.

Reflections in the wild. In Fig. S7, S8, and Fig. 8 we show results on reflections that were captured in-the-wild, where it was not possible to capture ground truth: (left) shopping in the morning, (middle) looking into a building at midday in a city, and (right) photographing artwork in an outdoor mall (Fig. S7); while traveling and photographing art-

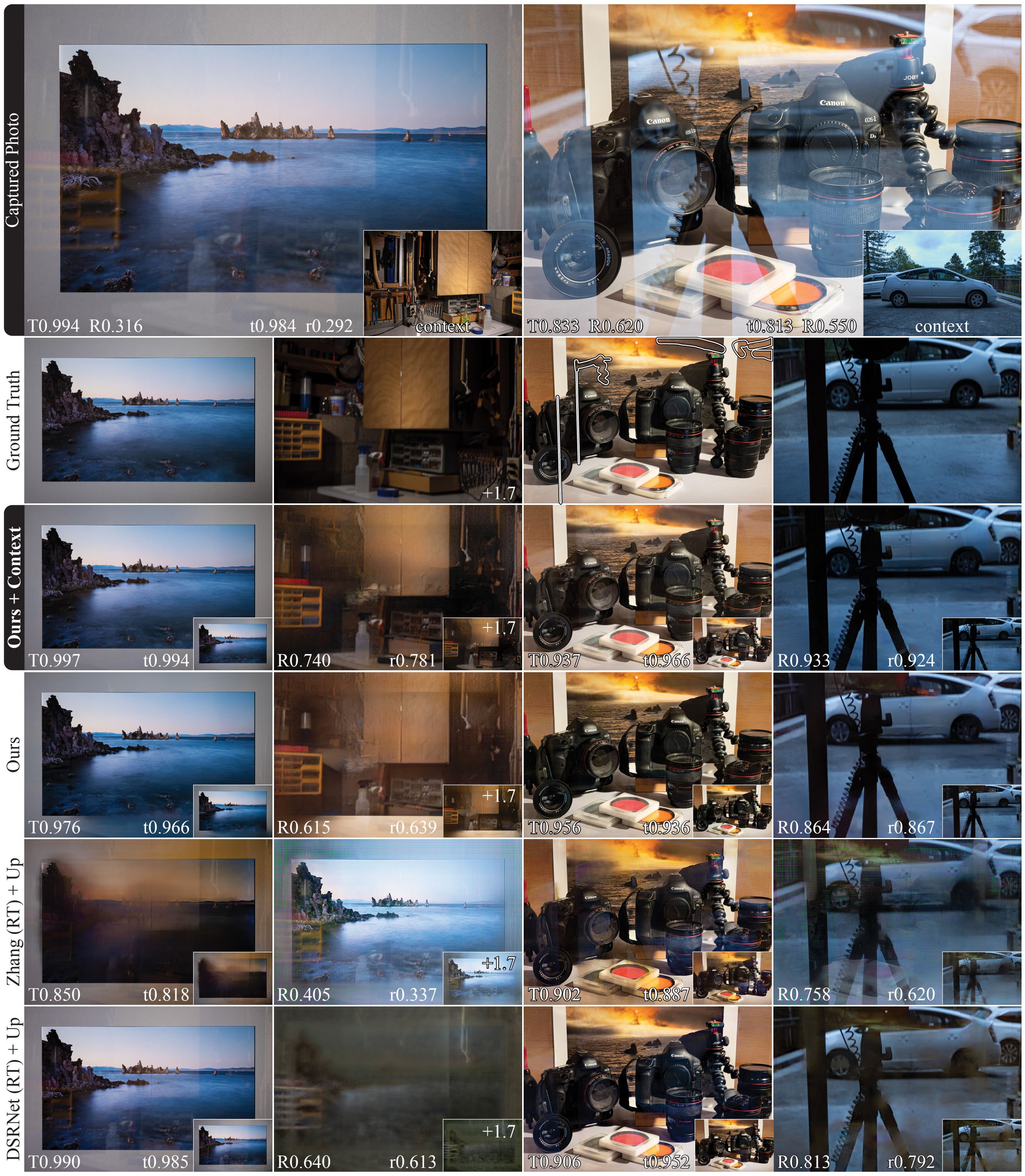

Figure S6. Comparisons with ground truth at 256 × 384 and 2048 × 3072. The re-trained, low-resolution methods Zhang [59] and DSRNet [20] are upsampled using our upsampler, and both low- and high-resolution results are shown. The SSIM of the predicted t and r is reported at low resolution (labeled t, r) and high (labeled T, R). Errors in ground truth are outlined and omitted from the SSIM.

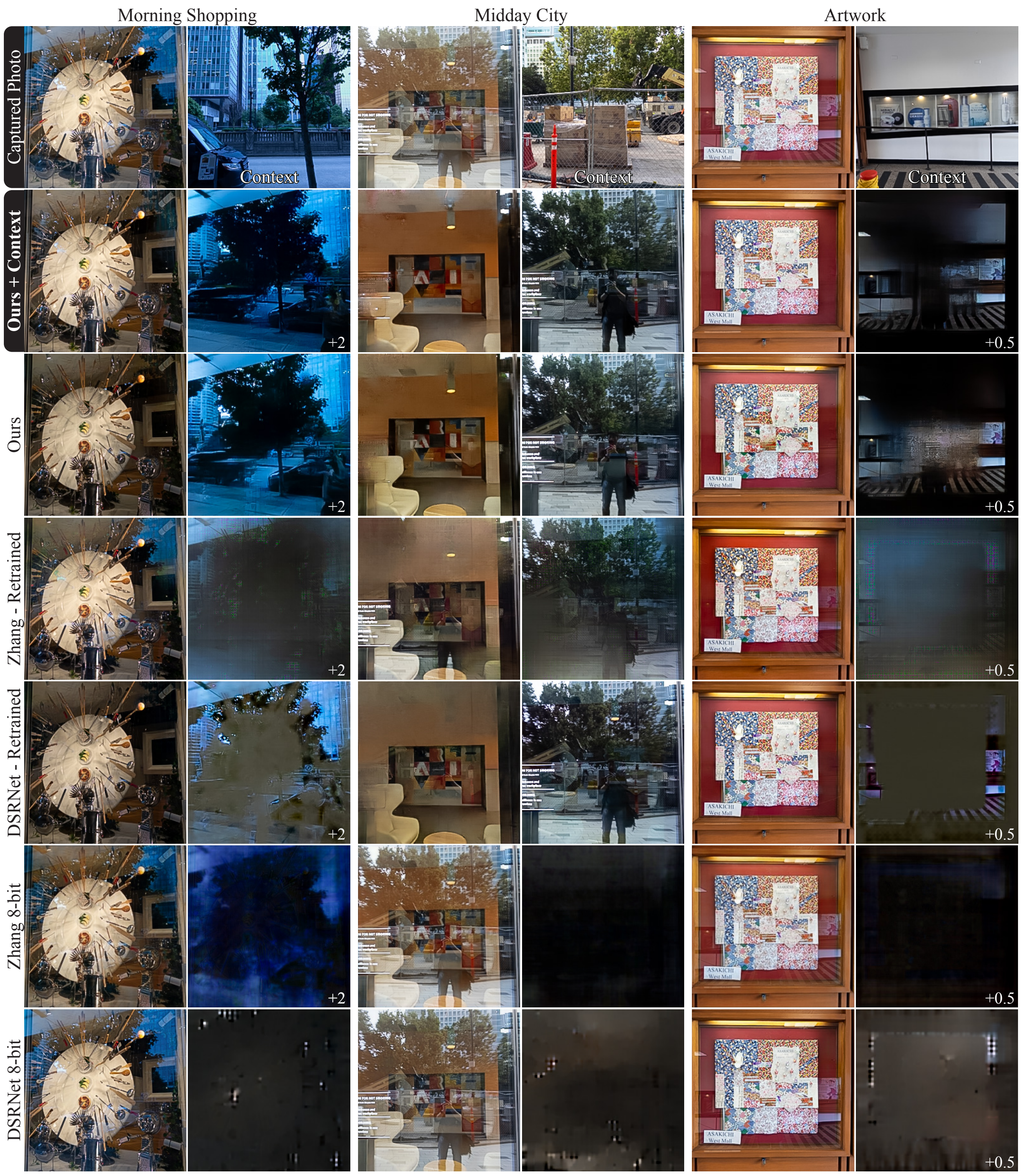

Figure S7. Results for base models at 256^2 pixels. Reflections marked "+X" are brightened by X stops compared to the transmission.

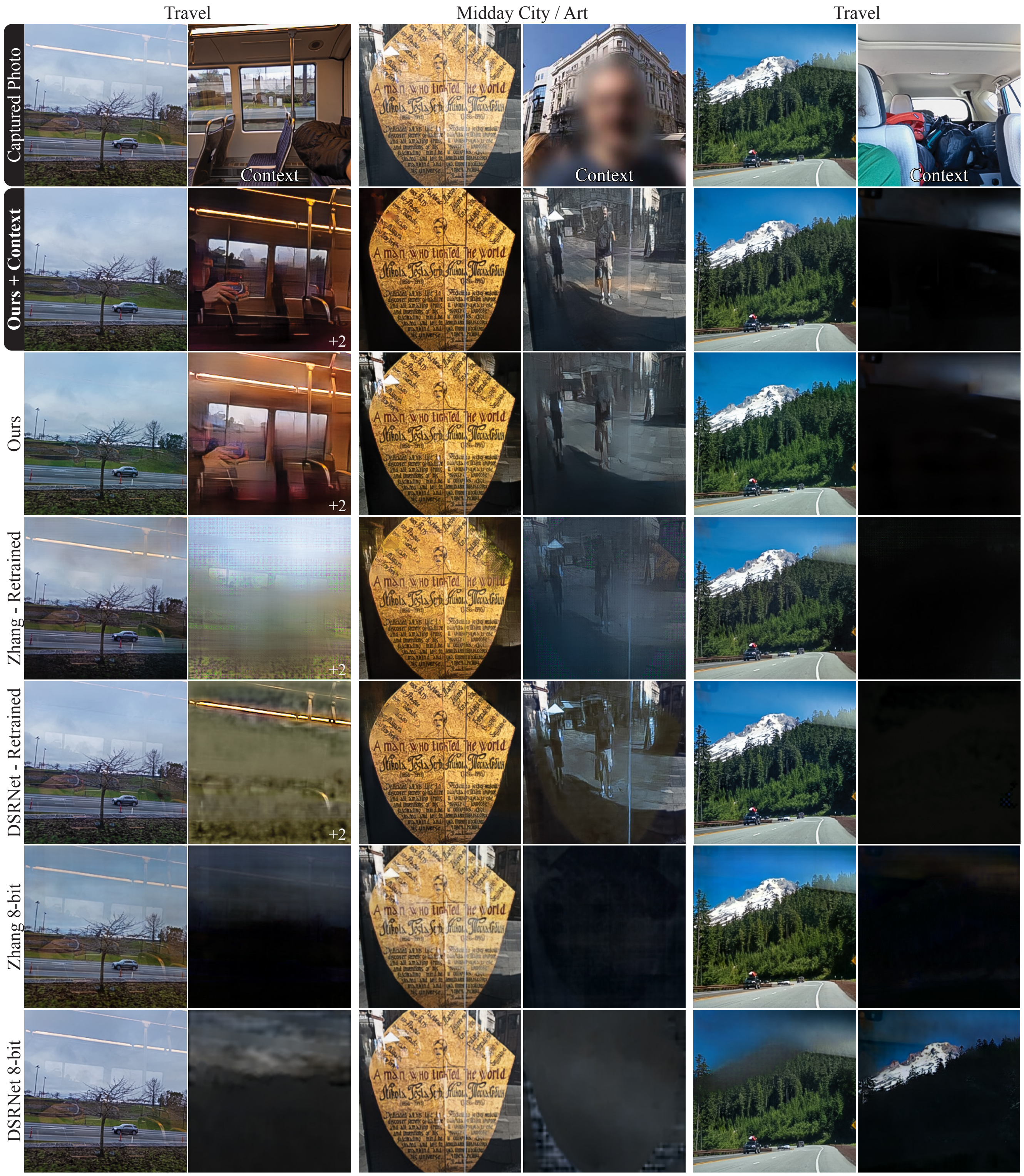

Figure S8. Results for base models at 256^2 pixels. Reflections marked "+X" are brightened by X stops compared to the transmission.

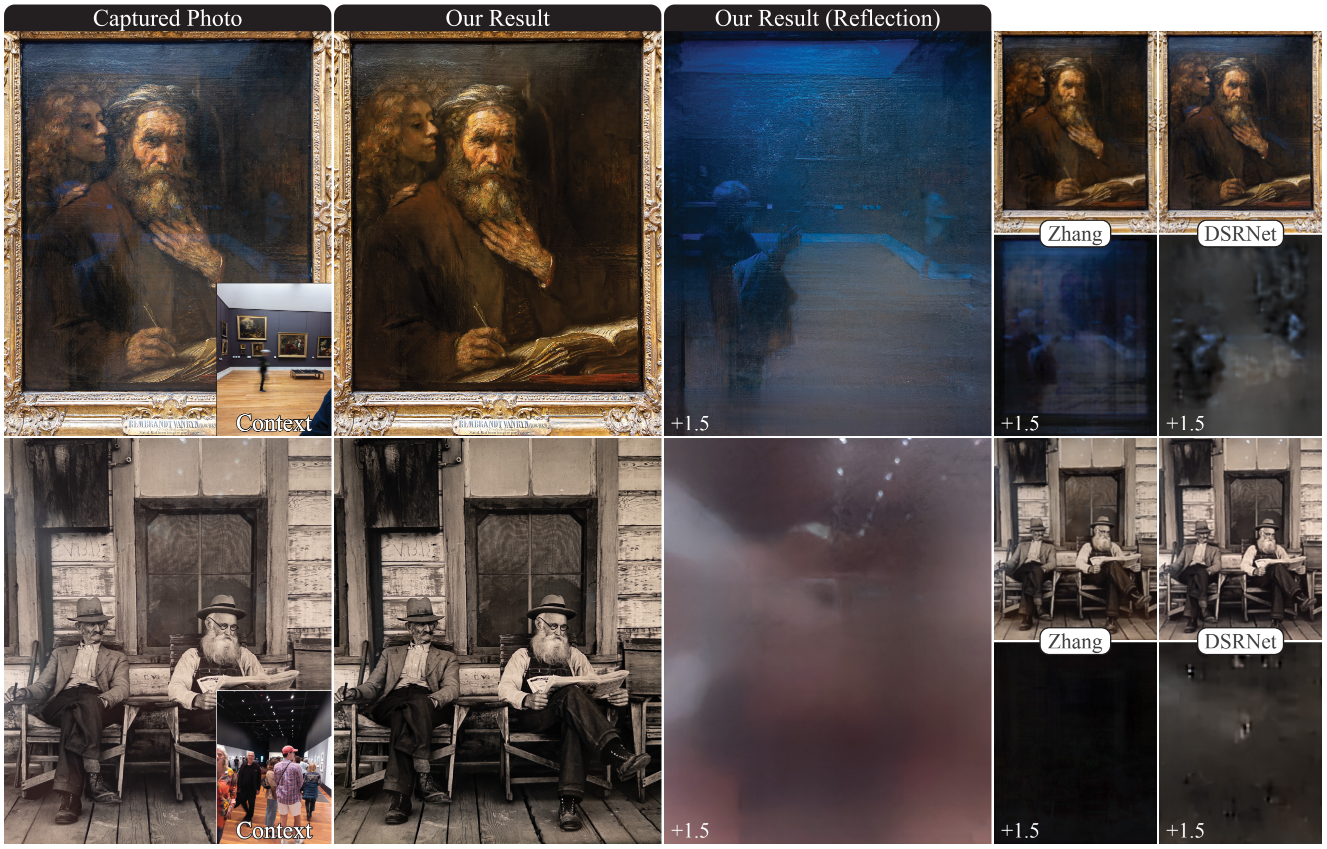

Figure S9. Reflection recovery. Our model separates reflections that can be difficult to spot with the naked eye. See Sec. F.2.

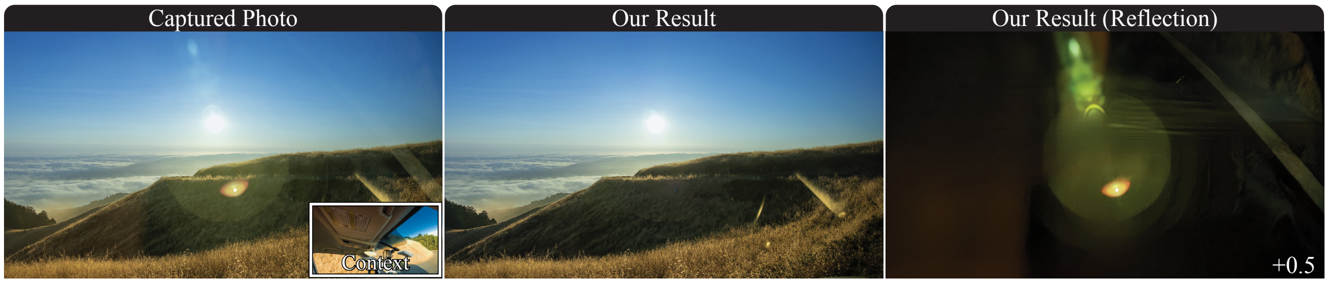

Figure S10. Removing lens flares. Although our training data do not include images of lens flares, our model can sometimes remove them.

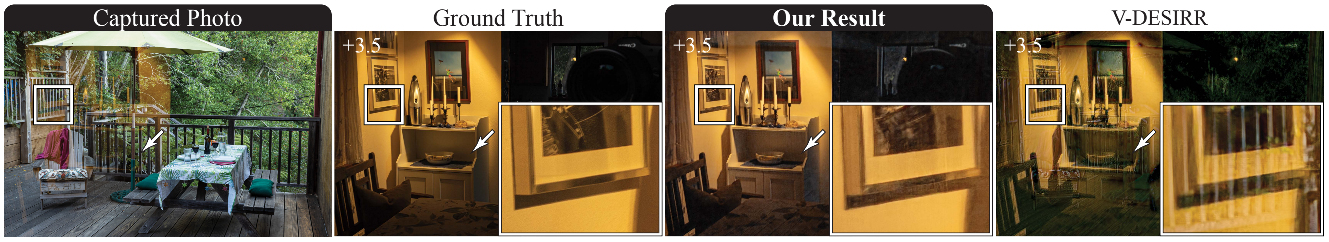

Figure S11. Upsampler performance comparison. We upsample a ground truth reflection image from 256 × 384 to 2048 × 3072 using our method and V-DESIRR [35]. The latter creates strong artifacts due to resize operations that directly synthesize output pixels. Our model eliminates these artifacts while using 29% fewer parameters.

work from a city street (S8). We compare to Zhang [59] and DSRNet [20] using their published 8-bit models (bottom two rows). The 8-bit models do not consistently remove in-the-wild reflections, with the exception that the method of Zhang et al. seems to have learned to remove blue colored content (left); see also Fig. S9 (top). We retrained these prior works on our photometrically accurate training data (marked "retrained"), and they improve significantly. This suggests that the muted response of the 8-bit models on in-the-wild reflections results from differences between their training data and real world reflections, and furthermore that the training process of prior works is insufficient: pre-training on photometrically inaccurate images, and fine-tuning on small datasets of ground truth reflections does not produce models that generalize as well. Looking closely, however, note that the retrained models do still introduce blur and colored artifacts.

Our methods perform well across these diverse in-the-wild use cases. In Fig. S7, both our contextual and non-contextual models separate the strongly blue colored reflection, the complex cityscape reflection, and the artwork reflection (in the entrance to an indoor shopping mall). The contextual model improves each of these cases, excepting the transmission image for the shopping photo. We believe this also results from the saturated regions in the upper right of the contextual photo, where we have used the as-shot illuminant color in the EXIF to recover the pixels.

The midday city photo represents a difficult reflection, and both of our models improve this image. The result of the contextual model is more correct: the interior mural is more accurately reconstructed, and the white sign that is adhered to the glass is left more intact. Saturated regions near the ceiling have color artifacts in both cases.

Lastly, the artwork example (right) has two illuminants, a small warm colored artificial light, and a dominant outdoor illuminant. The white balance of the image is determined by the outdoor illuminant, which also illuminates the reflection scene, and this leads to a diversely colored reflection that both of our models are able to remove. The contextual model better preserves the texture of the artwork, and does not associate that texture with the reflection, whereas the non-contextual model does.

Subtle reflections. In Fig. S9 we show results on images with subtle reflections, which are common in art museums where the glass and lighting are optimized to reduce reflections. Our models are able to remove these reflections, and recover r even when it is invisible to the naked eye. In the painting (Rembrandt's "St. Matthew and the Angel"), the recovered reflection has a strong blue color due to the special glass that is used in museums, and it correctly depicts the photographer and gallery. The colors in the recovered painting are also more correct. The specular reflection from the surface of the painting (top) is not removed, and we believe that this is the desired result. We have found that our models will sometimes remove sharp specular regions that are similar in color to the reflection from the glass, and this is visible in the upper quarter of the reflection image, where some specular texture from the surface of the painting is visible, and has been removed from the transmission.

In Fig. S9 (bottom) we remove reflections from Ansel Adams' "Residents of Hornitos." Ceiling light sources are visible at the top of the photo, and on the left there is a general loss of contrast. The recovered transmission image has accurately uniform contrast, and the reflection reveals the hidden image of the photographer holding their cell phone. Our model is able to recover hidden reflections like this in part due to the bit depth of RAW images. The reflection is also blurry and differs in color.

Our ability to uncover subtle reflections is in part due to the extended bit depth of RAW images. The images in Fig. 1, Fig. 5-8, Fig. S6-S15 are $\geq$15-bit. ACR Step 3 is uint16. Together, the source datasets for the simulation (MIT5K [8] and RAISE [10]), are 12- and 14-bit (43%, 57%).

Lens flares. Our model can also remove lens flares, which are reflections from the optical elements within the camera lens. An example is shown in Fig. S10. Most of the lens flare is removed, excepting one saturated region. The flare itself is also recovered in the reflection image even though the simulated reflections r in the training dataset do not include lens flares. Since our method has some ability to generalize to flare removal, and it is difficult to create training data for flare removal, it could be helpful to pre-train flare removal methods first to remove photometrically accurate simulated reflections.

Failures. Failures typically occur when an image is too difficult for a person to visually interpret, as shown in Fig. S12. Light sources might also be missed, or a halo might be left behind, but in-painting methods can fill these holes. Reflections are usually removed best when glass covers the field-of-view, blocking the photographer from their subject, as this matches how the simulation is designed. As a result, local reflections on objects are typically not modified, which is desirable (*e.g.* shiny cars, distant windows).

Results on 8-bit photos. In this work, all results are presented on RAW photos because our models are trained for RAW. Nonetheless, our RAW-trained models have some ability to remove reflections in 8-bit photos if gamma is pseudo-inverted, as shown in Fig. S13, but removal may be incomplete. This difference between RAW and 8-bit photos appears because the latter are produced by diverse camera pipelines and artistic adjustments. This diversity of finishing operations must be considered when training and testing models that have been trained on RAW-based simulation images. It is therefore not meaningful to test our models on datasets of 8-bit images.

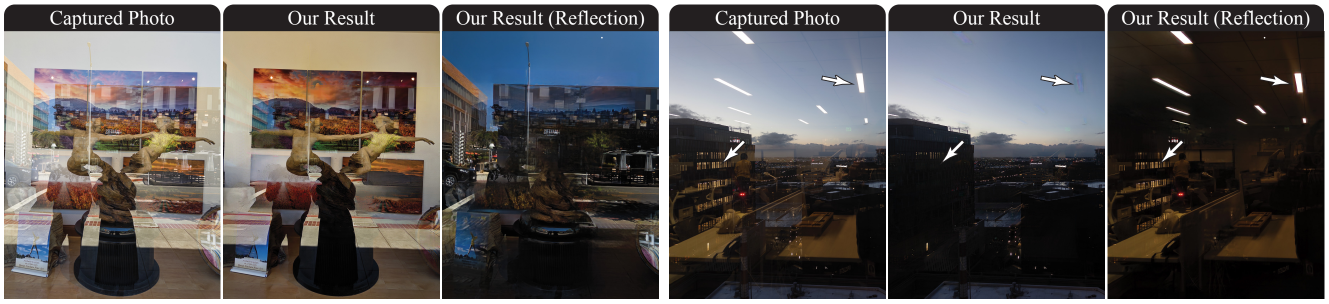

Figure S12. Failure cases. Hard-to-interpret photos can cause reflection removal to be imperfect.

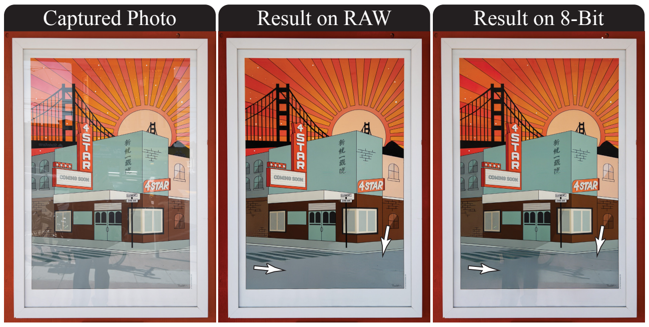

Figure S13. Results when running a RAW-trained model on 8-bit inputs. Results on 8-bit photos can be obtained by pseudo-inverting gamma, but they are often imperfect because such photos have undergone a variety of non-linear finishing effects that are not present in RAW images (see ACR Steps 7-8).

F.3. Upsampler comparisons

In Fig. S11 we provide additional comparison to V-DESIRR [35] in which we upsample a ground truth reflection image (a transmission image is upsampled in Fig. 5). A magnified region is inset, and shows that our method preserves details of a framed photo through the upsampling process, whereas V-DESIRR re-introduces the reflection content (the bars of a fence; see captured photo, white box). In the rest of the image, V-DESIRR introduces strong color artifacts around sharp edges. This is a consistent issue that appears to result from the propagation and amplification of small errors that are made at low resolutions. Our model reduces error propagation by masking t and r out of the high-resolution mixture features at each level. We are able to predict an effective, high resolution mask with a lightweight and fast model by matching features, whereas V-DESIRR must use its model capacity to infer from the high-resolution mixture image what details to add into the low resolution clean images. This latter problem is more difficult to solve with limited model capacity, and it is difficult to avoid propagating errors.

In Fig. S14 we show a typical failure mode of our model. The input image (top and center left) has a reflection of a tex-

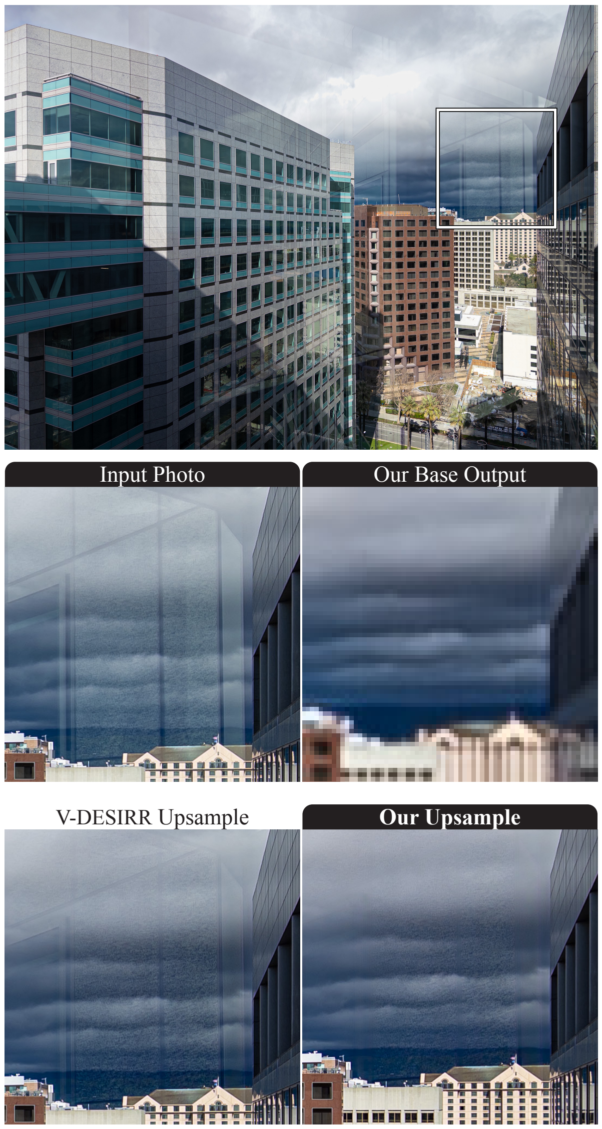

Figure S14. Upsampling errors. V-DESIRR [35] re-introduces low-frequency reflections. Our method avoids this error, but copies high-frequency textures that were not visible at 256^2.

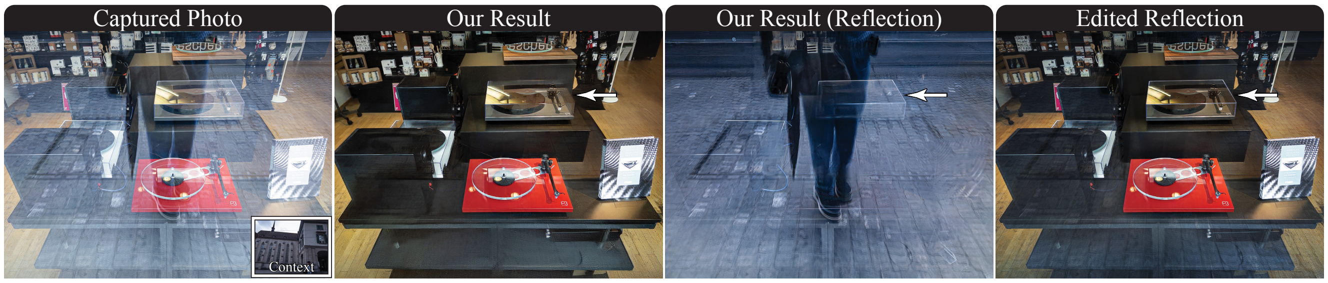

Figure S15. Predicting the reflection enables aesthetic editing and error correction.

tured exterior wall. Our base model correctly removes this reflection at 256^2 pixels (center right). Notice that, at low resolution, it is possible to represent the vertical edges of the exterior wall, but not the texture of the wall. We upsampled this result with both V-DESIRR and our model.

V-DESIRR re-introduces the low frequency edges that were removed by the base model, and it copies the high frequency wall texture into the transmission as well. Our model does not re-introduce the low frequency edges, but the high frequency texture of the wall that is not present at low resolution is copied into the transmission image. This produces a texture artifact in the final result. Future work should reduce these kinds of errors while keeping the architecture lightweight. This might be done by reusing feature information from the base model, and across levels of the Gaussian pyramid as the upsampler is iteratively applied.

F.4. Editing reflections

In Fig. 6 and Fig. S15 we show examples of reflection editing for aesthetic control and error correction (see also Sec. 5.4). For Fig. S15, a photographer was asked to finish the photo using the outputs of the reflection removal system. They chose to re-introduce the reflection for aesthetic purposes, and to correct errors. The reflections from the edges of the top record player cover were removed by our system (white arrows); the photographer pulled them back into their edited image. Editing was performed in Photoshop using the tone-mapped transmission and reflection images, and the "Linear Dodge (Add)" layer blend mode.

G. Adobe Camera RAW, DNG SDK

Here we detail the code within the DNG SDK [1] version 1.7.1 that is used in our simulation functions Func. 1, Func. S1 and Func. S2. In this work, the necessary SDK functionality was transliterated into Python to interoperate with the geometric simulation (Sec. B) and mixture search (Sec. D). To simplify the exposition, we however describe the SDK code here in a functional manner, whereas the SDK is a class system. Consequently, some of the SDK functions use class member variables, and not all functions in this exposition of the code map one-to-one with functions of the same name in the SDK.

Function S3 Adobe DNG SDK, convert RAW images to XYZ.

```
Input: A RAW image
Output: A linear image in XYZ color
1: Extract the ACR Stage-3 image I with dng_validate option -3.
                                                  {ACR Step 2}
2: Subtract the Stage-3 black level from I.       {SDK Func. S5}
3: Divide I by the maximum pixel value.
4: Compute WhiteXY.                               {SDK Func. S9}
5: Compute the transform XYZ_to_CAM from WhiteXY. {SDK Func. S7}
6: Recover saturated highlights in I.             {SDK Func. S6}
7: Transform I to XYZ using inv(XYZ_to_CAM).
                                         {see also SDK Func. S4}
8: return the linear XYZ image I.
```

As discussed in Sec. 3 and Sec. A, reflections are simulated in XYZ color space by using the color processing of the DNG SDK, which supports two paths to a white balanced, linear RGB image: the `ForwardMatrix` or the `ColorMatrix`. Only the latter path facilitates conversion to a device-independent color space (XYZ) before white balancing, as required for reflection simulation. We therefore implement in Func. S3 the ACR color process in which the `ColorMatrix` is used (cf. Func. S4). Note that both paths account for the as-shot illuminant because the `ColorMatrix` is interpolated according to the as-shot illuminant (see Func. S9 and Func. S7).

All supporting DNG SDK functions are listed below.

Function S4 Adobe DNG SDK, `CAM_to_RGB`

Input: `WhiteXY`
Output: Transform to linear RGB

This function is included here as reference to the entire computation of the color transform to linear RGB. Note, the DNG SDK implements the DNG Spec. [1]. It uses the `ForwardMatrix` when it is available, and otherwise uses the `CameraMatrix`. In this work we use only the `CameraMatrix`, since this supports white balancing after conversion to XYZ.

```
1: See dng_color_spec.cpp:573-609.
```

Function S5 Adobe DNG SDK, get `Stage3BlackLevel`

Input: A DNG file
Output: The black level

The Stage-3 black level is not stored in the DNG EXIF header. It is a global scalar offset that remains after spatially varying black levels have been removed by parsing and applying the black level tags. By default, the DNG SDK Stage-3 image has a black level of zero, and negative noise values have been clipped to zero. In this work, we adopt that convention and clip the negative noise for simplicity. Clipping can however be disabled, and the non-zero Stage-3 black level can be recovered with a minor modification to the `dng_validate` binary, described below. The black level can then be read from the printed output of `dng_validate` when the stage-3 image is extracted with the `-3` option.

```
1: return true for SupportsPreservedBlackLevels
                                        {dng_mosaic_info.cpp:2014}
2: return true for SupportsPreservedBlackLevels
                                        {dng_negative.cpp:5814}
3: printf((uint16) negative->Stage3BlackLevel())
                                        {dng_validate.cpp:293}
```

Function S6 Adobe DNG SDK, recover saturated highlights

Input: An image in camera color space
Output: Image with clipped higlights repaired

```
1: Compute CameraWhite
                                        {dng_color_spec.cpp:548-568}
2: return $\forall c, \min(c,$ CameraWhite$)$
                    {dng_render.cpp:1785 → dng_reference.cpp:1389}
```

Function S7 Adobe DNG SDK, `FindXYZtoCamera`

Input: White point XY
Output: Matrix `XYZ_to_CAM`

```
1: See dng_color_spec.cpp:541
                 {In practice, calls FindXYZtoCamera_SingleOrDual.}
```

Function S8 Adobe DNG SDK, `NeutralToXY` (projected, cf. SDK Func. S10)

Input: An AWB white point in camera color space.
Output: `XYZ_to_CAM_awb` and `WhiteXY_awb`

```
 1: for all kelvins $K$ do {plausible kelvin values}
 2:     Compute the XY coordinate of $K$.            {SDK Func. S15}
 3:     Compute the transform XYZ_to_CAM from XY.    {SDK Func. S7}
 4:     Compute the XYZ coordinate of XY.            {SDK Func. S16}
 5:     Project the XYZ coordinate into camera color using XYZ_to_CAM.
 6:     if the projected XYZ is closer to the AWB point than previous values then
 7:         Save XY
 8:     end if
 9: end for
10: return the saved XY value and its associated XYZ_to_CAM matrix.
```

Function S9 Adobe DNG SDK, computing `WhiteXY` and `CameraWhite`

Input: DNG `AsShotXY` XOR `AsShotNeutral` {All DNGs have one xor the other.}
Output: White point in XY

```
1: Compute WhiteXY.                     {dng_render.cpp:892,899}
```

Function S10 Adobe DNG SDK, `NeutralToXY`

Input: DNG `AsShotNeutral` value
Output: An XY coordinate

```
1: See dng_color_spec.cpp:659
```

Function S11 Adobe DNG SDK, compute `XYZ_to_sRGB`

Input: An XY white point, `XYPoint`.
Output: Matrix `XYZ_to_sRGB`.

```
1: Get the transform XYZ_D50_to_sRGB.           {SDK Func. S12}
2: Get the D50 XY coordinate XY_D50.            {SDK Func. S14}
3: Compute matrix XYZ_to_XYZ_D50 using XY_D50 and XYPoint.
                                                {SDK Func. S13}
4: return XYZ_D50_to_sRGB · XYZ_to_XYZ_D50
```

Function S12 Adobe DNG SDK, get `XYZ_D50_to_sRGB`

Input: None
Output: The transform from XYZ D50 to linear sRGB.

```
1: See dng_color_space.cpp:254, which specifies the inverse matrix.
```

Function S13 Adobe DNG SDK, `MapWhiteMatrix`

Input: Two white points, `w1` to `w2`.
Output: Bradford adaptation matrix.

```
1: See dng_color_spec.cpp:22.
```

Function S14 Adobe DNG SDK, `D50_xy_coord`

Input: None
Output: XY coordinate of the D50 illuminant

```
1: See dng_xy_coord.h:145.
```

Function S15 Adobe DNG SDK, `Get_xy_coord`.

Input: A scalar temperature value, K.
Output: An XY coordinate.

```
1: See dng_temperature.cpp:173.
```

Function S16 Adobe DNG SDK, `XYtoXYZ`.

Input: An XY vaue.
Output: An XYZ value

```
1: See dng_xy_coord.cpp:47.
```

Function S17 Adobe DNG SDK, `sRGB_to_linear_sRGB`.

Input: A gamma compressed sRGB value
Output: A linear sRGB value

```
1: See dng_color_space.cpp:34.
```