A Reduced-Precision Network for Image Reconstruction

MANU MATHEW THOMAS, University of California, Santa Cruz
KARTHIK VAIDYANATHAN, Intel Corporation
GABOR LIKTOR, Intel Corporation
ANGUS G. FORBES, University of California, Santa Cruz

Fig. 1. A comparison of different reconstruction techniques on a frame from the INFILTRATOR scene with PSNR (top row) and SSIM (bottom row) metrics. TAA has the worst reconstruction quality with excessive blurring followed by direct prediction with U-Net. QW-Net produces the best results with quality comparable to brute force sampling with 256 samples per pixel.

Neural networks are often quantized to use reduced-precision arithmetic, as it greatly improves their storage and computational costs. This approach is commonly used in image classification and natural language processing applications. However, using a quantized network for the reconstruction of HDR images can lead to a significant loss in image quality. In this paper, we introduce *QW-Net*, a neural network for image reconstruction, in which close to 95% of the computations can be implemented with 4-bit integers. This is achieved using a combination of two U-shaped networks that are specialized for different tasks, a *feature extraction* network based on the U-Net architecture, coupled to a *filtering* network that reconstructs the output image. The feature extraction network has more computational complexity but is more resilient to quantization errors. The filtering network, on the other hand, has significantly fewer computations but requires higher precision. Our network recurrently warps and accumulates previous frames using motion vectors, producing temporally stable results with significantly better quality than TAA, a widely used technique in current games.

CCS Concepts: • **Computing methodologies** → **Neural networks**; **Rendering**.

Additional Key Words and Phrases: Neural networks, kernel prediction, antialiasing

Authors' addresses: Manu Mathew Thomas, University of California, Santa Cruz, mthomas6@ucsc.edu; Karthik Vaidyanathan, Intel Corporation, karthik.vaidyanathan@intel.com; Gabor Liktor, Intel Corporation, gabor.liktor@intel.com; Angus G. Forbes, University of California, Santa Cruz, angus@ucsc.edu.

ACM Reference Format:
Manu Mathew Thomas, Karthik Vaidyanathan, Gabor Liktor, and Angus G. Forbes. 2020. A Reduced-Precision Network for Image Reconstruction. *ACM Trans. Graph.* 39, 6, Article 231 (December 2020), 12 pages. https://doi.org/10.1145/3414685.3417786

1 INTRODUCTION

Synthesizing computer-generated imagery involves two key functions: sampling the incident radiance on an image plane and applying a reconstruction filter to produce the final image [Cook et al. 1984]. If the incident radiance is undersampled, high-frequency components resulting from visibility discontinuities or specular lighting can appear as aliasing in the reconstructed image. To reduce aliasing, production renderers typically apply supersampling with a large number of samples per pixel. On the other hand, real-time applications rely on a combination of radiance prefiltering and specialized image reconstruction techniques. Temporal antialiasing (*TAA*) [Jimenez et al. 2012; Karis 2014; Yang et al. 2009] is an image reconstruction technique that is widely used in games today. TAA uses renderer-generated motion vectors to gather and accumulate samples from previous frames, effectively increasing the number of samples per pixel. However, TAA is susceptible to ghosting artifacts, loss of detail, and temporal instability [Yang et al. 2020].

Convolutional neural networks offer a promising alternative for image reconstruction, and have been successfully applied to closely related fields such as denoising [Bako et al. 2017; Chaitanya et al. 2017; Vogels et al. 2018]. The U-Net [Ronneberger et al. 2015] architecture is particularly well suited for such tasks as it processes features at multiple scales, achieving a large receptive field and more effective filtering with a relatively low computational cost.

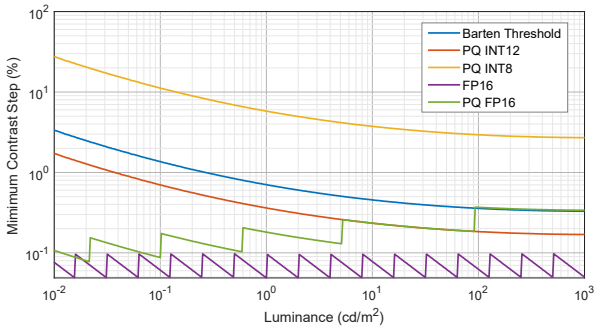

Fig. 2. Minimum contrast step for a 1000 nit display with different numeric formats. The step size with 8-bit integers exceeds the human contrast sensitivity threshold even with perceptual quantization (PQ) to minimize the perceived error [SMPTE 2014].

With the hardware acceleration of 8-bit and 4-bit tensor computations on GPUs [NVIDIA 2018], quantizing the weights and activations of such a network can be key to achieve real-time performance, without compromising its expressive power. However, quantization errors can severely impact the image quality, especially with high-dynamic-range content. We can see from Figure 2 that the minimum contrast step with 8-bit integers, far exceeds the human contrast sensitivity threshold [Barten 2003] on a high-dynamic-range display.

We introduce a novel network called QW-Net that addresses this issue by using a combination of two networks, a feature extraction network and a filtering network. The feature extraction network is based on U-Net and can be quantized to 4-bit integers as feature detection is more resilient to quantization errors. The filtering network is another U-shaped network that uses the extracted features at each scale to predict filters. The filtering operations require a higher precision but involve significantly fewer computations than the feature extraction network. The name QW-Net is motivated by the coupled U-shaped networks resembling a W, as well as the quantized nature of our feature extraction network.

Our network combines the advantages of U-Net and kernel prediction networks [Bako et al. 2017]. It achieves a large receptive field with just a few layers, similar to the U-Net, while avoiding artifacts like color shift and requiring significantly less training time, similar to kernel prediction networks [Vogels et al. 2018]. Moreover, our network temporally refines the reconstructed image by recurrently warping and accumulating previous frames.

Although our motivation for using 4-bit convolutions is improved inference performance, realizing the full potential of a quantized network requires extensive exploration of various implementation aspects such as tensor layouts, tile sizes, kernel fusion and constraints imposed by the hardware, which are beyond the scope of this paper. Our goal is to show the feasibility of a heavily quantized network for image reconstruction using a novel network topology that preserves image quality. To the best of our knowledge, this is the first 4-bit network applied to image processing. We evaluate image quality using simulated quantization and include an analysis of the computational cost and the performance of 4-bit convolutions in section 5.4.

2 BACKGROUND

2.1 Antialiasing and Image Reconstruction

On a fundamental level, the *sampling theorem* defines the minimum sampling frequency of a signal as twice its maximum frequency, also known as the *Nyquist limit* [Proakis and Manolakis 2006]. However, the radiance sampled on a virtual camera's sensor is often not band-limited: while geometric surfaces are continuous, the visibility function of light transport is not. This makes the application of low-pass *reconstruction* filters necessary to remove higher frequencies from the sampled signal in order to suppress *aliasing* [Mitchell and Netravali 1988], which commonly manifests as the "wagon wheel effect" or flickering in the temporal and jagged edges or "fireflies" in the spatial domain. Aliasing can also be reduced by *prefiltering* high-frequency terms in the sampled radiance, for example, specular lighting [Kaplanyan et al. 2016]. The term *antialiasing* broadly encompasses prefiltering, sampling and reconstruction techniques that seek to avoid or remove undersampling artifacts.

A straightforward way to overcome aliasing is *supersampling*, evaluating multiple sub-pixel radiance samples, but this also increases the cost of rendering. A cheaper alternative called Multisampling antialiasing (*MSAA*) [Akeley 1993] was one of the earliest antialiasing methods to be supported by graphics accelerators, where shading is sampled once per pixel, while visibility is sampled at a sub-pixel granularity. However, MSAA is less commonly used in modern games, where deferred shading has become a dominant technique. Deferred shading is evaluated once per pixel and therefore does not benefit from MSAA. Moreover, MSAA can significantly increase the storage cost for intermediate shading data in the *G-buffer*. Instead, modern games typically rely on post-sampling reconstruction techniques to suppress aliasing. Early examples of such a technique include morphological antialiasing (*MLAA*) [Reshetov 2009] and its derivatives [Jimenez et al. 2012] that detect jagged edges and smooth them to improve image quality. However, these approaches can miss fine geometry that is not sufficiently sampled within a single frame often resulting in aliasing in the temporal domain.

Temporal antialiasing (*TAA*) is a more recent family of techniques that leverage frame-to-frame coherence to amortize supersampling over time [Yang et al. 2009]. Like supersampling these techniques can reduce aliasing caused by high-frequency components of the visibility function as well as shading. TAA uses motion vectors to reproject a sample position in the current frame to its previous location in a temporal accumulation buffer [Nehab et al. 2007], from which a color value can be gathered and blended with the current pixel, effectively adding more samples. However, the gathered color may not match the current pixel, for example, a surface that is currently visible might be occluded in the previous frame or a shaded value may change from one frame to another due to moving shadows and reflections. Most TAA implementations use heuristics like neighborhood clamping [Karis 2014; Salvi 2016] to reject mismatched colors as they can cause severe *ghosting* artifacts. Unfortunately this does not eliminate all ghosting. Moreover, if the rejection heuristic is too aggressive the temporal accumulation becomes ineffective. Yang et al. [2020] survey recent TAA techniques and provide an in-depth analysis of the image quality trade-offs with these heuristics.

2.2 Neural Networks for Reconstruction

In recent years there has been a growing interest in applying neural networks to problems in rendering. One area in particular that has seen rapid advances is *denoising* of Monte Carlo renderings, which aims to remove noise from simulated distributed effects like soft shadows, indirect lighting, etc. While neural networks have been previously explored for denoising natural images [Burger et al. 2012], Kalantari et al. [2015] were the first to apply it to Monte Carlo rendering, using a multi-layer perceptron to drive the parameters of feature-based denoising filters. Later, Bako et al. [2017] used a convolutional neural network to predict the filter kernel itself, achieving significantly better results.

Chaitanya et al. [2017] introduced an alternate approach, using a network based on *U-Net* [Ronneberger et al. 2015] to directly predict the denoised image. By introducing recurrent connections inside a U-Net network they were the first to demonstrate temporally stable results at interactive rates. U-Net also has the advantage of achieving a large receptive field using a multi-scale architecture, which is crucial for denoising sparsely sampled images. On the other hand, achieving a large receptive field with a single predicted kernel [Bako et al. 2017] is impractical, as the complexity of filtering and the last prediction layer grows significantly with the filter size [Gharbi et al. 2019]. This limitation was overcome by Vogels et al. [2018] by predicting filter kernels at different scales using multiple deep residual networks [He et al. 2016]. The filtered images were progressively upsampled and blended using predicted weights to produce a denoised result. They compared their approach against a variant of the direct prediction network of Chaitanya et al. observing over-blurring and color-shift artifacts with direct prediction. They also discussed the possibility of using a U-Net to efficiently predict kernels at different scale. The concurrent work of Hasselgren et al. [2020] uses such a network with a feedback loop to achieve temporally stable results. Along similar lines, we use a U-Net network to predict filter kernels for reconstruction but instead of applying independent filters at each scale, we use a filter network that resembles U-Net, progressively filtering and re-sampling the image. By incorporating predicted filters in between each downsampling and upsampling step, we can better reconstruct feature details at each scale.

Image reconstruction is a more general problem of producing an antialiased image from point samples. In real-time applications, the input to a reconstruction technique like TAA is denoised using filters, specialized for effects like shadows, reflections or ambient occlusion. Neural networks are also promising for image reconstruction as initially demonstrated by Marco Salvi [2017] using a U-Net with a warped feedback loop. Kaplanyan et al. [2019] also used a recurrent U-Net to reconstruct a peripheral image for foveated rendering from a very sparse set of samples. We compare our approach against a U-Net based direct prediction network, but with a simpler form of recurrence [Hasselgren et al. 2020; Sajjadi et al. 2018] using the warped output image instead of a hidden state.

Recently, a combined real-time image reconstruction technique called Deep Learning Super Sampling (*DLSS*) [Liu 2020] was introduced, but the details of the underlying network are unknown. Concurrent to our work, Xiao et al. [2020] introduced a reconstruction technique based on U-Net. Using an optimized inference implementation they reconstruct a 1080p image in 18 to 20 ms on a high-end GPU. In comparison, DLSS reconstructs a 4K image in under 2 ms. Both these approaches can reconstruct images at a higher resolution than the input render. In this paper we reconstruct a supersampled image at the same resolution, focusing on achieving high quality, temporally stable results with aggressive quantization.

2.3 Quantized Networks

A full-precision network can be quantized either post-training or by training the network with simulated quantization. With *post-training quantization*, a network is quantized using the distribution of trained weights and by measuring the distribution of activations on a sample dataset, a process known as *calibration*. For example, TensorRT [Migacz 2017] quantizes weights and activations by minimizing the Kullback-Leibler (KL) divergence between the quantized and un-quantized distributions. TensorFlow [Jacob et al. 2018] on the other hand maps the range of the weight tensor and the average range of the activations computed over several batches, to the range of the quantized format. Unfortunately post training quantization shows as significant degradation in accuracy with 4-bit integers (INT4) and lower precisions [Jacob et al. 2018; Krishnamoorthi 2018].

Jacob et al. [2018] proposed an alternate *quantization-aware training* approach that simulates quantization errors during training, achieving significantly better accuracy. Simulated quantization introduces quantization errors in the forward pass by quantizing the weights and activations and dequantizing them back to the original range. However, in the backward pass, weights and biases are updated with floating point precision without any loss in precision.

The choice of threshold values for quantizing weights and activations greatly impacts the accuracy of the network. While Jacob et al. [2018] derived the quantization thresholds based on a measurement of the tensor range, Jain et al.[2019] and Esser et al. [2019] showed that the quantization thresholds could be trained to further improve accuracy. They derived the gradient of the quantization function applying a *straight through estimator* [Bengio 2013] for the gradient of round/ceil operations but without approximating these operations with an identity function. This allowed the thresholds to grow (favoring larger dynamic range) or shrink (favoring higher precision) based on the gradients.

A majority of existing work on reduced precision networks use a uniform quantization scheme where the quantized values are evenly spaced [Lin et al. 2016; Nayak et al. 2019; Zhou et al. 2017, 2016]. A uniform quantizer can be further classified into *asymmetric* and *symmetric* quantizers. An asymmetric quantizer applies an affine transformation with a scale and a zero-point to map values in the floating-point range to the integer range. In a symmetric quantizer, the zero point is set to 0 reducing the affine mapping to a linear mapping. The symmetric quantizer avoids the overhead of handling zero-points making it computationally efficient. A large body of work explores binary and ternary neural networks, where the weights and/or activations are quantized to binary or ternary values [Courbariaux et al. 2015, 2016; Rastegari et al. 2016; Zhu et al. 2016]. These networks push quantization to its limits but also lose a significant amount of accuracy.

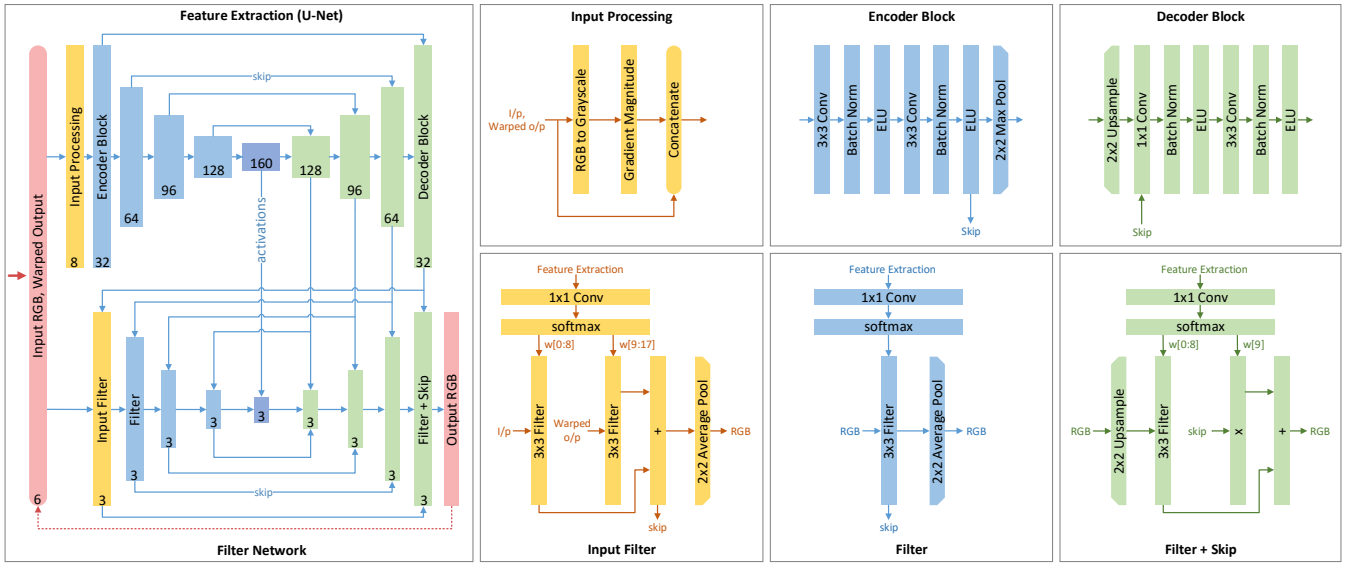

Fig. 3. The QW-Net architecture including the feature extraction network (top left) and the filtering network (bottom-left). The number of output channels is shown at the bottom of each stage of the network. The details of the encoder and decoder blocks as well as the filter stages are shown on the right.

3 QW-NET

Figure 3 shows the details of the QW-Net architecture, comprising the feature extraction network and the filtering network. The input to our network is a sequence of images and per-pixel motion vectors generated by the renderer. The network processes images in tone-mapped space, similar to the approach of Bako et al. [2017]. However, instead of the log transform, we use the inverse of the perceptual Electro-Optical Transfer Function (EOTF) [SMPTE 2014] which is a better match for the human contrast sensitivity model. We map the EOTF to a luminance range of up to a 1000 nits. Besides achieving better results with high-dynamic-range content, tone mapping also reduces the overall perceptual error with 8-bit and 4-bit integer formats.

Similar to TAA, the input images are rendered with a sub-pixel jitter sequence producing a spatial distribution of samples over multiple frames. In order to leverage this distribution and temporally accumulate samples, we apply the frame-recurrent approach of Sajjadi et al. [2018], where the previously reconstructed frame is warped and concatenated with the input frame, forming the current input to the network. Assuming $\mathrm{U_e}$ is the feature extraction network and $\mathrm{U_f}$ is the filter network, the reconstructed output $\boldsymbol{I}_{\boldsymbol{o}}^{k}$ at frame k is given by

$$\boldsymbol{I}_{\boldsymbol{o}}^{k} = \mathrm{U_f}\left(\mathrm{U_e}\left(\boldsymbol{I}_{\boldsymbol{a}}^{k}, \boldsymbol{I}_{\boldsymbol{w}}^{k}\right), \boldsymbol{I}_{\boldsymbol{a}}^{k}, \boldsymbol{I}_{\boldsymbol{w}}^{k}\right)$$
$$\boldsymbol{I}_{\boldsymbol{w}}^{k} = \mathrm{W}\left(\boldsymbol{I}_{\boldsymbol{o}}^{k-1}, \boldsymbol{I}_{\boldsymbol{v}}^{k}\right),$$

where $\boldsymbol{I}_{\boldsymbol{a}}^{k}$ is the aliased input image, $\boldsymbol{I}_{\boldsymbol{w}}^{k}$ is the warped previous output, $\boldsymbol{I}_{\boldsymbol{v}}^{k}$ is a 2D grid of motion vectors and W is a bilinear warp function.

3.1 Feature Extraction

The feature extraction network is based on the U-Net architecture which includes a series of encoder blocks that downsample the image followed by decoder blocks that reverse this process. The first stages in the network convert the input images $\boldsymbol{I_a}$ and $\boldsymbol{I_w}$ to grayscale and compute their gradient magnitudes. The two gradient magnitude images are concatenated with $\boldsymbol{I_a}$ and $\boldsymbol{I_w}$ forming the input for the first convolution layer. The gradients highlight aliased regions in the image which aids training [Bako et al. 2017].

Each encoder block has two convolution layers with a 3×3 spatial footprint, each followed by batch normalization [Ioffe and Szegedy 2015] and Exponential Linear Unit (ELU) activation [Clevert et al. 2015]. The last stage in the encoder block is downsampling with 2×2 max pooling. We increase the number of channels (tensor depth) by 32 at each successive block starting with 32 in the first encoder block and reaching 160 at the bottleneck. Encoder blocks have skip connections to the corresponding decoder blocks relaying high-frequency details to the decoder. The bottleneck is similar to an encoder block but excludes max pooling and skip connections.

The first stage in the decoder block is a 2×2 nearest-neighbor upsampling operation. The upsampled activations are concatenated with the skip connection and projected to the same size as the encoder output using a 1×1 convolution layer [Szegedy et al. 2015]. The decoder block includes a single 3×3 convolution layer resulting in three such layers at each scale, excluding the bottleneck, which has two convolution layers.

Batch Normalization: While batch normalization has been successfully applied to U-Net [Çiçek et al. 2016], recent recurrent variants tend to avoid it or replace it with a layer norm. With our frame recurrent network we observe that batch norm achieves significantly better training convergence, even with direct prediction.

3.2 Filter Network

The filter network has a similar topology to the feature extraction network with a series of downsampling filters followed by upsampling filters with skip connections between them. The pair of downsampling and upsampling filters at each scale are coupled to the output of the corresponding decoder block in the feature extraction network.

Each filter uses the activations from the decoder block to predict a 3×3 kernel that is applied to the input image. Similar to Bako et al. [2017] we use a 1×1 convolution layer with softmax activation to predict the kernel resulting in normalized weights. The input filter predicts 18 normalized filter weights corresponding to two 3×3 filters with 9 weights each. These filters are applied to $\boldsymbol{I_a}$ and $\boldsymbol{I_w}$ respectively and the results are summed to produce a single image. The subsequent downsampling filters apply a 3×3 kernel to a single image. The last stage in each downsampling filter is a 2×2 average pooling operation. The bottleneck filter excludes this pooling operation.

The first stage in each upsampling filter is bilinear upsampling, following which the image is filtered and combined with the skip connection. The upsampling filters use 10 filter weights, 9 weights for the 3×3 filter kernel and one for scaling the skip connection. We use average pooling and bilinear upsampling in the filtering network as it results in better image quality. On the other hand we use max pooling and nearest neighbor upsampling in the feature extraction network as they are computationally cheaper and do not significantly impact feature extraction.

4 TRAINING

We train our network on blocks of $N_t \times N_x \times N_y$ pixels, where $N_x = N_y = 256$ are the spatial dimensions of the block and $N_t = 8$ is the number of frames (time steps). Each block is extracted from a sequence of rendered frames that belong to the same camera shot. Our training dataset includes a collection of sequences from different scenes as discussed in Section 6.

The network weights are initialized following He et al. [2015] with a uniform distribution. At each training iteration, we evaluate the network on a mini-batch of 64 blocks for each time step and then back propagate the loss through all time steps. At the beginning of each iteration, we initialize the warped previous frame $\mathbf{I_w}$ to the same value as the input frame $\mathbf{I_a}$. Initializing $\mathbf{I_w}$ to zero produces comparable results except for the first few warm-up frames. We use the recent Ranger optimizer [Wright 2019] that combines Rectified Adam [Liu et al. 2019] and Lookahead [Zhang et al. 2019] with default parameters and a learning rate of 0.0005.

4.1 Loss

We use a loss function L that combines a spatial loss L_s and a temporal loss L_t. The spatial loss is the L_1 loss commonly used in denoising and super-resolution networks, computed over the $N_s = N_x N_y$ spatial pixels and N_t time steps in a block:

$$L_s = \frac{1}{N_t N_s} \sum_{k=1}^{N_t} \sum_{i=1}^{N_s} \left| \boldsymbol{o}_i^k - \boldsymbol{r}_i^k \right|, \tag{1}$$

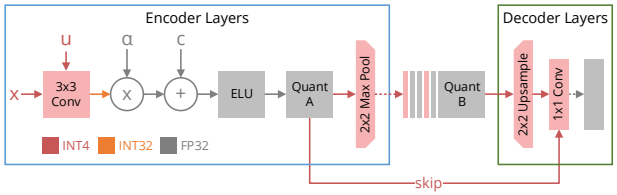

Fig. 4. Convolution layers with 4-bit activations and weights. When the outputs of quantizers are concatenated together (e.g. quantizers A and B) we ensure that they use the same quantization range.

where $\boldsymbol{o}_i^k$ is the output color at pixel i and time step k and $\boldsymbol{r}$ is the corresponding reference color rendered with 256 samples. The temporal loss is the mean absolute error in the temporal gradient and aims to achieve temporal stability:

$$L_t = \frac{1}{(N_t - 1)N_s} \sum_{k=2}^{N_t} \sum_{i=1}^{N_s} \left| \left(\boldsymbol{o}_i^k - \boldsymbol{\phi}_i^k \right) - \left(\boldsymbol{r}_i^k - \boldsymbol{\psi}_i^k \right) \right|, \tag{2}$$

where $\boldsymbol{\phi}$ and $\boldsymbol{\psi}$ are the reconstructed and reference color values from the warped previous frame. Our temporal loss is similar to the one used by Chaitanya et al. [2017] with the difference that we compute the temporal gradient w.r.t the warped previous frame, highlighting regions that are temporally mismatched. This is possible since we assume the availability of motion vectors from the renderer.

The combined loss is then given by $L = 0.3L_s + 0.7L_t$. We derived this ratio of L_s and L_t experimentally starting with an equal blend and then decreasing the spatial loss contribution until the temporal quality was acceptable.

5 QUANTIZATION

Our quantization approach targets GPU architectures that support accelerated tensor computations with 8-bit and 4-bit integers, such as Nvidia Turing [2018]. The throughput of tensor operations on Turing scales inversely with the bit width, where 8-bit computations have 2× the throughput and 4-bit computations have 4× the throughput of half precision. This gives us the flexibility to select a different trade-off between precision and throughput at different stages in our network.

We quantize all layers of the feature extraction network to use 4-bit weights and activations, except the first convolution layer, which uses 8-bit weights and 4-bit activations. As previously observed with other networks [Choi et al. 2018; Zhang et al. 2018; Zhu et al. 2016], having the input layer at a higher precision leads to a significant improvement in the loss. We use per-channel symmetric quantization for the weights and affine per-layer quantization for the activations [Krishnamoorthi 2018]. This approach achieves good results with 4-bit quantization with a relatively small overhead discussed later in this section. When the outputs of two layers are concatenated together, we use the same quantization range for both the activations as shown in Figure 4, ensuring a uniform quantization range for the input to the next convolution layer.

The mapping of quantized integer weights and activations u, v to their real values w, x is given by

$$w = s_w u, \tag{3}$$

$$x = s_x(v - z), \tag{4}$$

where s_w is a per-channel scale (step size) for the quantized weights, s_x is a per-layer scale for the quantized activations and z is the zero point. We can then represent the convolution for a single channel as:

$$y = \sum w_i x_i + b_i,$$

where b_i is the bias. Substituting with Equations 3 and 4 we get:

$$\begin{aligned} y &= s_w s_x \left(\sum u_i v_i - \sum u_i z \right) + b_i, \\ &= \alpha \sum u_i v_i + c_i, \end{aligned}$$

where $\alpha = s_w s_x$ and $c_i = b_i - \alpha \sum u_i z$, represents a single-precision floating point bias that can be precomputed.

Figure 4 shows the convolution and activation layers for a single channel along with their numeric formats. The convolutions constitute the majority of the computations but can be mapped to the tensor cores on Turing that can efficiently evaluate 4-bit multiplications and accumulate the result in a 32-bit integer. An additional floating-point MAC scales the convolution output by α and introduces the bias c, following which an ELU activation is computed with single-precision floating point. A final MAC operation then maps the floating-point activation back to a 4-bit integer.

The activations are quantized using a function $\mathrm{Q}(x, l, u)$ that maps an activation x in the range $[l, u]$ to a quantized integer x_q in the range $[M_l, M_u]$. For a b-bit integer, $M_l = -2^{b-1}$, $M_u = 2^{b-1} - 1$ and $M = M_l + M_u$ is the number of quantization levels. Following Jacob et al. [2018], we adjust the quantization range such that a zero value gets quantized without error, preserving zero-padded data. The adjusted range $[l_a, u_a]$ is derived as follows:

$$l_a = s \left\lfloor \frac{l}{s} \right\rceil, \tag{5}$$

$$u_a = l_a + sM, \tag{6}$$

where $s = \frac{u-l}{M}$ is the quantization step size and a common term for both s_w and s_x. The quantization function Q is then given by

$$x_q = \begin{cases} \left\lfloor \frac{x}{s} \right\rceil + z & \text{if } l_a \le x \le u_a \\ M_l & \text{if } x < l_a \\ M_u & \text{if } x > u_a, \end{cases} \tag{7}$$

where $z = M_l - \frac{l_a}{s}$ is the zero point introduced earlier in Equation 4.

5.1 Trained Quantization

We initially train our network with full precision and then quantize the weights and activations by fine tuning the network with simulated quantization [Jacob et al. 2018]. Training with simulated quantization achieves significant improvements over post-training quantization, especially with 4-bit precision [Krishnamoorthi 2018]. Simulated quantization applies a quantization followed by a dequantization in the forward pass, introducing quantization errors while maintaining and updating the weights as un-quantized variables.

The simulated quantization function $\mathrm{Q}'(x, l, u)$ can be derived from Equations 4 and 7:

$$\mathrm{Q}'(x, l, u) = s\,(\mathrm{Q}(x, l, u) - z) \tag{8}$$

While Jacob et al. [2018] use the moving averages of the minimum and maximum activations in a batch to derive the quantization thresholds l and u, we adopt a more recent approach [Esser et al. 2019; Jain et al. 2019] where the quantization thresholds are derived from trained variables.

Following Jain et al. [2019], we train the thresholds in log space using variables t_l and t_u, such that $l = -e^{t_l}$ and $u = e^{t_u}$ to improve stability with quantized training. We derive the gradient of Q' for backpropagation, using a straight through estimator [Bengio 2013] for rounding operations i.e. $\frac{\mathrm{d}}{\mathrm{d}x} \lfloor x \rceil = 1$. For reference, we provide the equations for the gradients in Appendix A. We initialize $t_l = \ln(1)$ and $t_u = \ln(3)$, where $l = -1$ corresponds to the lower limit of the ELU activation and $u = 3$ is a value that we have chosen based on experimenting with $u = 6$ [Krizhevsky 2010] and $u = 3$.

We use the same quantization functions Q and Q' for the weights but enforce a symmetric range by setting $l = -u$. Unlike the activations we did not derive the quantization threshold from a trained variable. Instead we set u to the maximum absolute value of the weights corresponding to a channel.

5.2 Batch Normalization

It is common practice to eliminate the batch normalization layer in the inference network by folding the layer parameters into the weights and bias of the preceding convolution layer. The folded weights and bias are given by:

$$w_{\text{inference}} = \frac{\gamma}{\sigma} w,$$

$$b_{\text{inference}} = \beta - \frac{\gamma}{\sigma} \mu,$$

where μ and σ are the moving average of the batch mean and standard deviation and γ and β are the trained scale and bias respectively. While scaling the weights by $\lambda = \frac{\gamma}{\sigma}$ we also scale the quantization threshold by the same amount. This has no impact on quantization error as $\mathrm{Q}(\lambda w, \lambda l, \lambda l) = \mathrm{Q}(w, l, l)$. During trained quantization, we freeze the values of γ and β and use the moving average of the batch mean and variance computed during the initial training.

5.3 Filtering Network

As discussed in Section 3.2, the filter network includes a 1×1 convolution layer to predict the filter kernels. We quantize the weights of this layer to 8-bit using trained quantization but we do not quantize the activations (filter kernel) as all filtering operations are computed with single-precision floating-point.

5.4 Computation Costs

Although the focus of our paper is high-quality reconstruction with quantization, we provide some data to support the feasibility of high performance 4-bit inference. Table 1 lists the number of MAC operations per pixel for the QW-Net network and the corresponding numeric formats.

Table 1. Multiply-Accumulate (MAC) operations per pixel.

Network	INT4	INT8	FP32	% Total
U-Net				
Input Conv			2304	3.13
Encoder			37344	50.73
Decoder			33472	45.47
Pool / Upsample			404	0.55
Output Layer			96	0.13
Total			70912	
QW-Net				
Input Conv		2304		3.07
Encoder	37344			49.80
Decoder	33472			44.63
Quantization			387	0.52
Kernel Prediction		1358		1.81
Pool / Upsample			28	0.04
Filter			99	0.13
Total	70816	3662	514	

We also list the additional overhead for quantization which is less than 1% of the total computations. This includes one MAC per channel for scaling and biasing the convolution output and another MAC for quantizing the activations. The overhead of kernel prediction and filtering is close to 2%, while 4-bit convolutions account for 95% of the computations. Table 1 also lists the number of operations for direct prediction with a comparable U-Net network. We assume single-precision floats for the filtering network as well as the U-Net network since we reconstruct the image in tone-mapped space, which leaves very little precision headroom with 16-bit floats as shown in Figure 2. The computational cost of the ELU and softmax activation functions are not included in this table even though it can be significant depending on the performance of transcendental operations on the GPU. This cost could be avoided through a different choice of activation such as LRelu [Maas et al. 2013] for the feature extraction network and linear activation for the kernel prediction layers [Mildenhall et al. 2018].

To further study the feasibility of a 4-bit network, we implemented a convolution layer in CUDA with 32 input and output channels. This represents the second and last layer in the feature extraction network which are the most expensive. We also benchmarked a similar convolution layer in NVIDIA TensorRT which is a general-purpose inference runtime. Both these implementations were evaluated on a TITAN RTX GPU and utilized tensor cores. Table 2 shows the execution time with different numeric formats at a 1080p resolution. We see a 2× performance gain going from 8-bit integers to 4-bit integers using our custom kernel, which is in-line with the expected performance on a Turing GPU. A similar trend is observed with the 8-bit and half precision convolutions in TensorRT. Unfortunately, TensorRT does not support 4-bit convolutions.

Table 2. Execution time for a single convolution operation.

Precision Level	TensorRT(ms)	Custom(ms)
FP16	1.03	
INT8	0.61	0.29
INT4		0.14

6 RESULTS

We train and evaluate our network (QW-Net) and a comparable direct prediction network (U-Net) using images rendered with Unreal Engine 4 (UE4) [2019]. The direct prediction U-Net is the same as the feature-extraction network but with an additional 1×1 convolution layer at the output to directly produce the reconstructed color. In Section 6.1 we describe our modifications to the game engine for our data-acquisition pipeline. Section 6.2 provides an overview of the datasets and the methodology used for training. In Section 6.3 we evaluate the training convergence of QW-Net and U-Net and present some ablation studies. Finally, in Section 6.4 we compare the image quality of our network against TAA and U-Net.

6.1 Data Acquisition

We generated our training and test datasets using a modified version of UE4. Obtaining large-scale datasets that are representative of modern game workloads is a challenging task because most game engines cannot produce reference images with the sampling density that we require. Therefore, we render an image sequence with one sample per pixel and repeat this multiple times with different sub-pixel viewport offsets. The frames from each render are then weighted by a 2D Blackman-Harris window and accumulated to produce a supersampled reference sequence. This accumulation is performed in a perceptually tone-mapped space [SMPTE 2014] in order to suppress "fireflies" caused by high-energy outliers in the reference image. The accumulated result is inverse tone mapped to bring it back into linear space. This is analogous to what most TAA implementations do when accumulating samples.

In order to produce matched results with each render we used cinematic UE4 demos that are fully keyframed. However, a significant issue we encountered is that *randomness* is deeply rooted in game engines. Although this can be a desirable property for games, it results in some assets behaving differently between sequence replays, for example, particle systems, procedural lights or materials and even some skinned characters. To ensure repeatable playback, we hard-coded the seeds of random number generators inside the engine and also turned off some particle systems, where we were unable to achieve repeatable results. Moreover, particles in UE4 that do not include motion vectors are separately handled by the TAA shader based on a particle mask. In the future, we plan to incorporate this particle mask in our network.

For extracting image data and motion vectors from UE4 we made a few modifications to the renderer. Since our reconstruction technique is likely to be implemented at the same stage in the rendering pipeline as TAA, we disabled the stages that execute after TAA (e.g. motion blur, post-processing, fog, etc.) and modified the TAA shader to a "pass-through" function that skips temporal accumulation. To generate the motion vectors we rendered the image sequence with another modified TAA shader that encodes and writes the motion vectors to the output image. We extracted the output images using a linear 16-bit EXR [Kainz et al. 2004] format to preserve the dynamic range. We also biased the texture MIP-level by -1 to preserve high-frequency texture details.

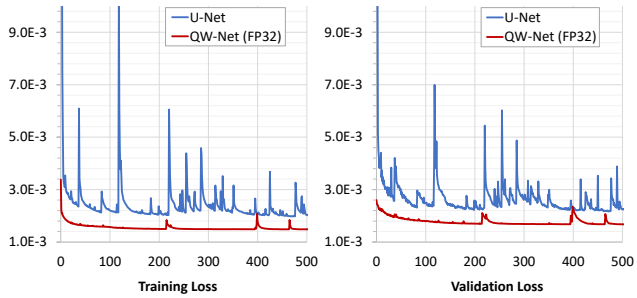

Fig. 5. Loss versus epochs during full-precision training.

6.2 Datasets

We prepared our datasets from four cinematic scenes publicly available for UE4. ZENGARDEN has large, smooth surfaces with well defined edges in an outdoor setting and a slow panning camera, making it the least challenging for reconstruction. INFILTRATOR features dark indoor scenes with several light sources, and highly specular materials, as well as an outdoor cityscape. KITE is a brightly-lit landscape sequence with a large amount of alpha-tested foliage. Finally, SHOWDOWN is another city scene with several reflective materials, which was not included in the training data. Both the training and test datasets consist of several image sequences, each having 120 continuous frames. The training set has 6 sequences from INFILTRATOR, 4 from KITE and 2 from ZENGARDEN. The test set contains 2 sequences of each of these three scenes, and an additional sequence from the SHOWDOWN demo which was excluded from training. We also set aside a sequence from the KITE demo for validation.

As discussed in Section 4, we use image blocks comprising 8 time steps (frames) of 256×256 pixel tiles to train the network. We extract a set of overlapping blocks by iterating over the pixels and frames with a stride of 192 in the spatial domain and 4 in time. The overlap approximately matches the receptive field of the network to better weight the edges of the block which are affected by zero padding. After extracting the blocks we cull approximately 50% of them based on the spatial loss (Equation 1), where blocks with a lower spatial loss are culled with a higher probability. After culling, we end up with 13712 blocks which are used for training.

6.3 Network Analysis

Figure 5 shows the loss profile for QW-Net and U-Net with full-precision training. This aligns with earlier findings [Bako et al. 2017; Vogels et al. 2018] showing faster convergence with kernel prediction compared to direct prediction. Vogels et al. also provide theoretical insights into this behavior in a simplified convex setting.

We derived the quantized weights by first training our network for 500 epochs and then re-training the network with simulated quantization. Figure 6 shows that re-training converges quickly to a loss that is close to the full precision loss. With 8-bit quantization the loss converges to a value that is slightly below the full precision loss while with 4-bit quantization it remains slightly above. This small difference in loss values has little impact in terms of image quality as we show in the next section. Figure 6 also shows the

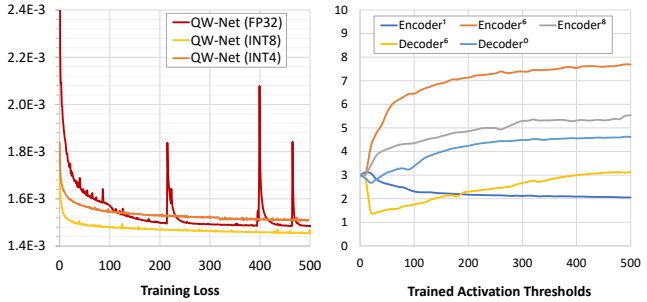

Fig. 6. On the left, we show that training loss for both 4-bit and 8-bit quantization converges quickly to a value that is close to the full precision loss. On the right, we show how the upper quantization thresholds u change during reduced precision re-training. The encoder and decoder layers are numbered from the input to the output. Layers closer to the inputs and outputs (Encoder 1, Decoder 6) have a lower threshold compared to inner layers (Encoder 6, Decoder 0).

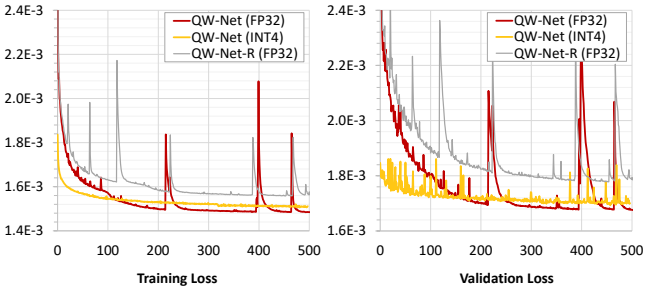

Fig. 7. QW-Net-R trained with single-precision float versus QW-Net re-trained with 4-bit quantization.

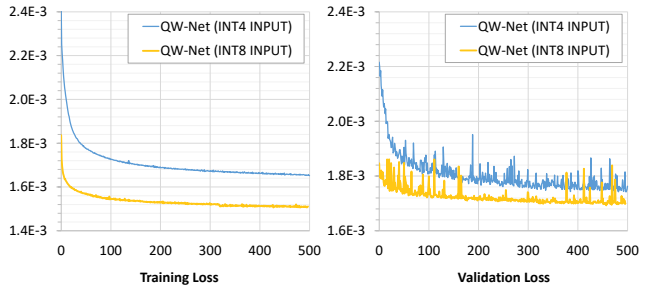

Fig. 8. QW-Net re-trained with 4-bit and 8-bit quantization at the input of the feature extraction network.

quantization threshold u for the activations from a few selected layers in the network. The layers closest to the input and the output seem to adapt to a lower threshold compared to the inner layers.

To verify that our network is not over-parameterized, we evaluated a network called QW-Net-R, where we set the tensor depth of the first stage to 24 (instead of 32) and increased it by 24 at each following stage. As a result, the computational complexity of QW-Net-R is approximately half of QW-Net. Figure 7 shows that QW-Net quantized to 4-bits still converges to a significantly lower loss than QW-Net-R without any quantization.

We also studied the impact of reduced precision at the input layer of the feature extraction network, by quantizing this layer to 4 bits

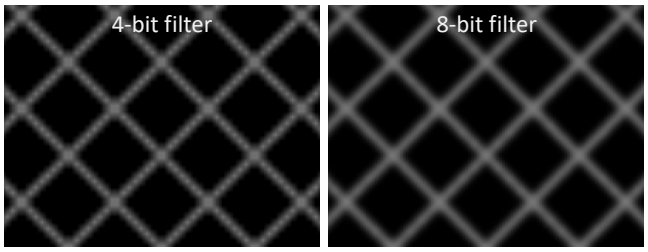

Fig. 9. A comparison of filtering quality with a Gaussian kernel quantized to 4-bits and 8-bits.

instead of 8 bits, while the remaining layers were quantized to 4 bits as usual. This resulted in a significant increase in loss as shown in in Figure 8, highlighting the importance of higher input precision.

Our choice of higher precision 8-bit filter kernels is motivated by the experiment shown in Figure 9, where we filter a high-frequency image with 4-bit and 8-bit quantized Gaussian filter kernels. The 4-bit filter shows clear artifacts which would be unacceptable for high quality reconstruction.

6.4 Image Quality

We include a qualitative and quantitative comparison of the reconstructed images using the default TAA implementation in UE4, direct prediction (U-Net) and our network (QW-Net). Unless otherwise noted, all QW-Net images were produced with 4-bit quantization. QW-Net was trained for 500 epochs at full precision followed by trained quantization for 1000 epochs while U-Net was trained for 1200 epochs at full precision. For each network we use the weights from the epoch with the minimum training loss. We use highest-quality settings when rendering images with TAA.

For the quantitative analysis we use two widely-used metrics: the Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity (SSIM) using a reference image rendered with 256 samples per pixel. Unfortunately, neither of these metrics sufficiently penalizes local phenomena such as jaggies or ghosting. However, we still find them to be useful as they clearly show an order of quality across different techniques. The problem of quantitatively analyzing aliasing artifacts in synthetic images was studied by [Patney and Lefohn 2018], where they proposed a neural network to generate a metric for aliasing.

Figures 1 and 10 show a few representative frames from our dataset. With the exception of Figure 1, all results shown in this paper are from the test dataset and were not used during training. In Figure 1, TAA blurs the text on the sign significantly and loses details on the railings as a result of color clamping. In Figure 10, blurring is present in almost all TAA examples, with ZENGARDEN also displaying severe ghosting artifacts.

The differences between QW-Net and U-Net on the other hand are more subtle. However, looking at the quality metrics, we can see that QW-Net consistently outperforms U-Net, and these differences can also be perceived with some visual scrutiny. While U-Net can remove aliasing in many cases, it fails on the railings in Figure 1. The results of QW-Net are also noticeably sharper, especially in the KITE scene. The bottom row of the INFILTRATOR scene in Figure 10 also shows a color shift around the collar. QW-Net on the other hand cannot produce a color shift, since it predicts a single set of kernels that are applied to all color channels. The top row of the SHOWDOWN scene shows a region where some ghosting is observed with all reconstruction techniques but to a lesser degree with QW-Net.

While these figures show selected frames, we observe superior quality metrics for all sequences with QW-Net. Figure 11 plots the per-frame metrics for an example sequence from KITE. It shows a notable drop in quality around frame 100, which is the result of a large disoccluded region where temporal supersampling has to discard pixel history. Even in this scenario, QW-Net stays above U-Net in quality and both networks recover quickly after a few frames. Figure 12 shows a region in the frame that becomes blurry at the disocclusion event and regains sharpness in a few frames.

Inspired by the evaluation of video super-resolution methods [Caballero et al. 2017; Sajjadi et al. 2018], we also show *temporal profiles* in Figure 13. These images highlight temporal discontinuities by tracing a column of pixels over a sequence of frames. While the temporal profiles with QW-Net appear smooth and relatively close to the reference, U-Net shows visible aliasing on the railing in INFILTRATOR and temporal noise in the KITE scene.

7 CONCLUSIONS AND FUTURE WORK

Quantization can be a key to overcome the computational barriers of deep neural networks for their wider application in real-time rendering. It has particularly good potential on modern GPU architectures that accelerate reduced-precision tensor operations.

In this paper, we demonstrated a network where most of the convolutions can be mapped to 4-bit operations without a significant loss in dynamic range or quality. While preserving image quality with 4-bit quantization has been the main focus of our research, an optimized implementation for real-time inference remains future work. In addition, we plan to explore leveraging auxiliary features like depth or normals, optimizing the activation functions to avoid transcendental functions and reducing memory traffic by fusing layers. A potential direction for future research is applying this network for other problems in the field of image processing.

ACKNOWLEDGMENTS

We thank Epic Games, Inc. for providing Unreal Engine with demo scenes for training and testing, and members of the Intel Advanced Research and Technology group for discussions and insightful feedback. We also thank David Blythe and Charles Lingle for supporting this research.

REFERENCES

Kurt Akeley. 1993. RealityEngine Graphics. In *Proc. ACM SIGGRAPH*. 109–116.

Steve Bako, Thijs Vogels, Brian Mcwilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony Derose, and Fabrice Rousselle. 2017. Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings. *ACM Transactions on Graphics* 36, 4, Article 97 (July 2017), 14 pages.

Peter G. J. Barten. 2003. Formula for the contrast sensitivity of the human eye. In *Image Quality and System Performance*, Yoichi Miyake and D. Rene Rasmussen (Eds.), Vol. 5294. International Society for Optics and Photonics, SPIE, 231–238.

Fig. 10. Representative frames from our test datasets. We compare the visual quality of TAA, U-Net, and QW-Net (INT4), compared to the 256× supersampled reference. For each frame we magnify two areas of interests for easier comparison. We also show the PSNR metrics in the upper comparison rows and the SSIM values in the lower rows. TAA produces significantly lower metrics for all test frames, losing sharpness, and sometimes producing noticeable ghosting (e.g. ZENGARDEN, top row). U-Net closely approximates the reference, but is also consistently below QW-Net in quality metrics, and sometimes produces color shifts (e.g. INFILTRATOR, bottom row).

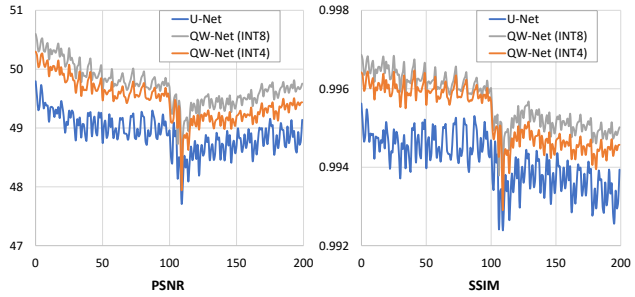

Fig. 11. Per-frame quality metrics on a continuous frame sequence from the KITE dataset. QW-Net consistently outperforms U-Net, with a slight degradation in the case of INT4.

Fig. 12. The loss of history samples due to disoccluded regions results in a short-term reduction in image quality. Before occlusion (left), at disocclusion (middle), 10th frame after disocclusion.

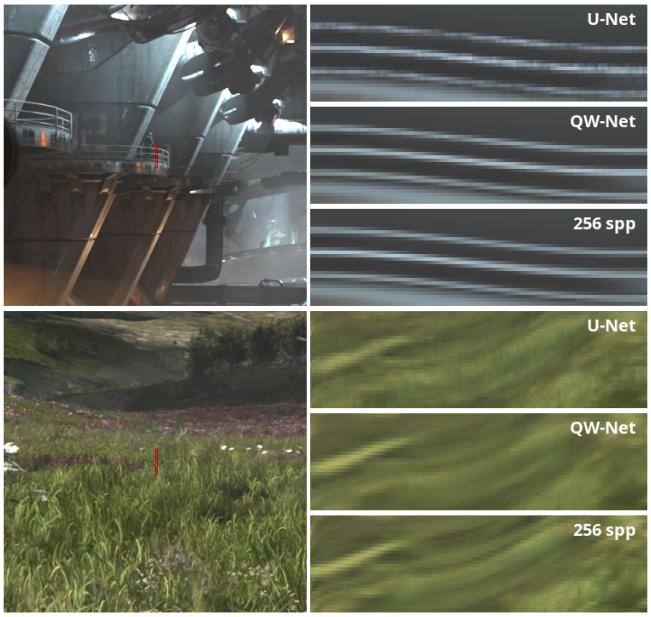

Fig. 13. Temporal Profiles from the INFILTRATOR (top) and KITE (bottom). QW-Net has a significantly smoother profile than U-Net, indicating better temporal coherence.

Yoshua Bengio. 2013. Estimating or Propagating Gradients Through Stochastic Neurons. *arXiv preprint arXiv:1305.2982* (2013).

H. C. Burger, C. J. Schuler, and S. Harmeling. 2012. Image denoising: Can plain neural networks compete with BM3D?. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2392–2399.

Jose Caballero, Christian Ledig, Andrew Aitken, Alejandro Acosta, Johannes Totz, Zehan Wang, and Wenzhe Shi. 2017. Real-Time Video Super-Resolution with Spatio-Temporal Networks and Motion Compensation. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 4778–4787.

Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. 2017. Interactive Reconstruction of Monte Carlo Image Sequences Using a Recurrent Denoising Autoencoder. *ACM Transactions on Graphics* 36, 4, Article 98 (July 2017), 12 pages.

Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. 2018. PACT: Parameterized Clipping Activation for Quantized Neural Networks. *arXiv preprint arXiv:1805.06085* (2018).

Ö. Çiçek, A. Abdulkadir, S.S. Lienkamp, T. Brox, and O. Ronneberger. 2016. 3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, Vol. 9901. Springer, 424–432.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and accurate deep network learning by exponential linear units (ELUs). *arXiv preprint arXiv:1511.07289* (2015).

Robert L. Cook, Thomas Porter, and Loren Carpenter. 1984. Distributed Ray Tracing. In *Computer Graphics (Proc. ACM SIGGRAPH)*, Vol. 18. 137–145.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. BinaryConnect: Training deep neural networks with binary weights during propagations. *arXiv preprint arXiv:1511.00363* (2015).

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. *arXiv preprint arXiv:1602.02830* (2016).

Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. 2019. Learned step size quantization. *arXiv preprint arXiv:1902.08153* (2019).

Epic Games. 2019. Unreal Engine 4.24 on GitHub. https://github.com/EpicGames/UnrealEngine/tree/4.24

Michaël Gharbi, Tzu-Mao Li, Miika Aittala, Jaakko Lehtinen, and Frédo Durand. 2019. Sample-Based Monte Carlo Denoising Using a Kernel-Splatting Network. *ACM Transactions on Graphics* 38, 4, Article 125 (July 2019), 12 pages.

J. Hasselgren, J. Munkberg, M. Salvi, A. Patney, and A. Lefohn. 2020. Neural Temporal Adaptive Sampling and Denoising. *Computer Graphics Forum* 39, 2 (2020), 147–155.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *Proc. IEEE International Conference on Computer Vision (ICCV)* (Dec 2015), 1026–1034.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Jun 2016), 770–778.

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2704–2713.

Sambhav R Jain, Albert Gural, Michael Wu, and Chris H Dick. 2019. Trained quantization thresholds for accurate and efficient fixed-point inference of deep neural networks. *arXiv preprint arXiv:1903.08066* (2019).

Jorge Jimenez, Jose I. Echevarria, Tiago Sousa, and Diego Gutierrez. 2012. SMAA: Enhanced Morphological Antialiasing. *Computer Graphics Forum* 31, 2 (2012), 355–364.

Florian Kainz, Rod Bogart, and Drew Hess. 2004. The OpenEXR image file format. In *GPU Gems*, Randima Fernando (Ed.). Addison-Wesley Reading, Chapter 26.

Nima Khademi Kalantari, Steve Bako, and Pradeep Sen. 2015. A Machine Learning Approach for Filtering Monte Carlo Noise. *ACM Transactions on Graphics*, 34, 4, Article 122 (2015), 12 pages.

A. S. Kaplanyan, S. Hill, A. Patney, and A. Lefohn. 2016. Filtering Distributions of Normals for Shading Antialiasing. In *Proc. High-Performance Graphics (HPG)*. 151–162.

Anton S. Kaplanyan, Anton Sochenov, Thomas Leimkühler, Mikhail Okunev, Todd Goodall, and Gizem Rufo. 2019. DeepFovea: Neural Reconstruction for Foveated Rendering and Video Compression Using Learned Statistics of Natural Videos. *ACM Transactions on Graphics* 38, 6, Article 212 (Nov. 2019), 13 pages.

Brian Karis. 2014. High Quality Temporal Supersampling. SIGGRAPH 2014 Advances in Real-Time Rendering in Games course. http://advances.realtimerendering.com/s2014/

Raghuraman Krishnamoorthi. 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342* (2018).

Alex Krizhevsky. 2010. Convolutional deep belief networks on CIFAR-10. https://www.cs.toronto.edu/~kriz/conv-cifar10-aug2010.pdf

Darryl D. Lin, Sachin S. Talathi, and V. Sreekanth Annapureddy. 2016. Fixed Point Quantization of Deep Convolutional Networks. In *Proc. International Conference on Machine Learning (ICML)*. 2849–2858.

Edward Liu. 2020. DLSS 2.0 - Image Reconstruction for Real-time Rendering with Deep Learning. In *GPU Technology Conference (GTC)*. https://developer.nvidia.com/gtc/2020/video/s22698-vid

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. 2019. On the Variance of the Adaptive Learning Rate and Beyond. *arXiv preprint arXiv:1908.03265* (2019).

Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. 2013. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML Workshop on Deep Learning for Audio, Speech and Language Processing*.

Szymon Migacz. 2017. 8-bit Inference with TensorRT. In *GPU Technology Conference (GTC)*. https://on-demand.gputechconf.com/gtc/2017/presentation/s7310-8-bit-inference-with-tensorrt.pdf

Ben Mildenhall, Jonathan T Barron, Jiawen Chen, Dillon Sharlet, Ren Ng, and Robert Carroll. 2018. Burst Denoising with Kernel Prediction Networks. In *Proc. IEEE Computer Vision and Pattern Recognition (CVPR)*. 2502–2510.

Don P. Mitchell and Arun N. Netravali. 1988. Reconstruction Filters in Computer Graphics. *SIGGRAPH Computer Graphics* 22, 4 (1988), 221–228.

Prateeth Nayak, Degan Zhang, and Sek Chai. 2019. Bit Efficient Quantization for Deep Neural Networks. *arXiv preprint arXiv:1910.04877* (2019).

Diego Nehab, Pedro V. Sander, Jason Lawrence, Natalya Tatarchuk, and John R. Isidoro. 2007. Accelerating Real-time Shading with Reverse Reprojection Caching. In *Graphics Hardware*. 25–35.

NVIDIA. 2018. NVIDIA Turing GPU Architecture Whitepaper. https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf

Anjul Patney and Aaron Lefohn. 2018. Detecting Aliasing Artifacts in Image Sequences Using Deep Neural Networks. In *Proc. High-Performance Graphics (HPG)*. Article 4, 4 pages.

John G. Proakis and Dimitris K Manolakis. 2006. *Digital Signal Processing (4th Edition)*. Prentice Hall.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-NET: ImageNet classification using binary convolutional neural networks. In *Proc. European Conference on Computer Vision (ECCV)*. 525–542.

Alexander Reshetov. 2009. Morphological Antialiasing. In *Proc. High-Performance Graphics (HPG)*. 109–116.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. *arXiv preprint arXiv 1505.04597* (2015).

Mehdi SM Sajjadi, Raviteja Vemulapalli, and Matthew Brown. 2018. Frame-recurrent video super-resolution. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 6626–6634.

Marco Salvi. 2016. An excursion in temporal supersampling. In *Game Developer's Conference (GDC)*. https://developer.download.nvidia.com/gameworks/events/GDC2016/msalvi_temporal_supersampling.pdf

Marco Salvi. 2017. Deep Learning: The Future of Real-Time Rendering?. In *ACM SIGGRAPH Courses: Open Problems in Real-Time Rendering*. https://openproblems.realtimerendering.com/s2017/index.html

SMPTE. 2014. ST 2084:2014 - SMPTE Standard - High Dynamic Range Electro-Optical Transfer Function of Mastering Reference Displays. *ST 2084:2014* (2014), 1–14.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proc. IEEE Computer Vision and Pattern Recognition (CVPR)*. 1–9.

Thijs Vogels, Fabrice Rousselle, Brian Mcwilliams, Gerhard Röthlin, Alex Harvill, David Adler, Mark Meyer, and Jan Novák. 2018. Denoising with Kernel Prediction and Asymmetric Loss Functions. *ACM Transactions on Graphics* 37, 4, Article 124 (July 2018), 15 pages.

Less Wright. 2019. New Deep Learning Optimizer, Ranger: Synergistic combination of RAdam + LookAhead for the best of both. https://medium.com/@lessw/new-deep-learning-optimizer-ranger-synergistic-combination-of-radam-lookahead-for-the-best-of-2dc83f79a48d

Lei Xiao, Salah Nouri, Matt Chapman, Alexander Fix, Douglas Lanman, and Anton Kaplanyan. 2020. Neural supersampling for real-time rendering. *ACM Transactions on Graphics* 39, 4, Article 142 (2020), 12 pages.

Lei Yang, Shiqiu Liu, and Marco Salvi. 2020. A Survey of Temporal Antialiasing Techniques. *Computer Graphics Forum* 39 (2020), 607–621.

Lei Yang, Diego Nehab, Pedro V. Sander, Pitchaya Sitthi-amorn, Jason Lawrence, and Hugues Hoppe. 2009. Amortized Supersampling. *ACM Transactions on Graphics* 28, 5, Article 135 (Dec. 2009), 12 pages.

Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. 2018. LQ-Nets: Learned Quantization for Highly Accurate and Compact Deep Neural Networks. In *Proc. European Conference on Computer Vision (ECCV)*. 365–382.

Michael R. Zhang, James Lucas, Geoffrey Hinton, and Jimmy Ba. 2019. Lookahead Optimizer: k steps forward, 1 step back. *arXiv preprint arXiv:1907.08610* (2019).

Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. 2017. Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights. In *International Conference on Learning Representations (ICLR)*.

Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. 2016. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *arXiv preprint arXiv:1606.06160* (2016).

Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. 2016. Trained Ternary Quantization. *arXiv preprint arXiv:1612.01064* (2016).

A APPENDIX A

In this section, we derive the gradients for our simulated quantization function. Expanding Equation 8 we get the simulated quantization function as:

$$x'_q = \begin{cases} s\left\lfloor \frac{x}{s} \right\rceil & \text{if } l_a \le x \le u_a \\ l_a & \text{if } x < l_a \\ u_a & \text{if } x > u_a, \end{cases} \tag{9}$$

where s is the quantization step, l_a and u_a are the adjusted range as defined in Equation 5 and 6. The local gradient with respect to trainable lower bound t_l is given by:

$$\frac{\partial x'_q}{\partial t_l} = \begin{cases} \frac{l}{M}\left(\frac{x}{s} - \left\lfloor \frac{x}{s} \right\rceil\right) & \text{if } l_a \le x \le u_a \\ \frac{l}{M}\left(\frac{l}{s} - \left\lfloor \frac{l}{s} \right\rceil + M\right) & \text{if } x < l_a \\ \frac{l}{M}\left(\frac{l}{s} - \left\lfloor \frac{l}{s} \right\rceil\right) & \text{if } x > u_a \end{cases} \tag{10}$$

Similarly, the local gradient with respect to trainable upper bound t_u is:

$$\frac{\partial x'_q}{\partial t_u} = \begin{cases} \frac{u}{M}\left(\left\lfloor \frac{x}{s} \right\rceil - \frac{x}{s}\right) & \text{if } l_a \le x \le u_a \\ \frac{u}{M}\left(\left\lfloor \frac{l}{s} \right\rceil - \frac{l}{s}\right) & \text{if } x < l_a \\ \frac{u}{M}\left(\left\lfloor \frac{l}{s} \right\rceil - \frac{l}{s} + M\right) & \text{if } x > u_a \end{cases} \tag{11}$$