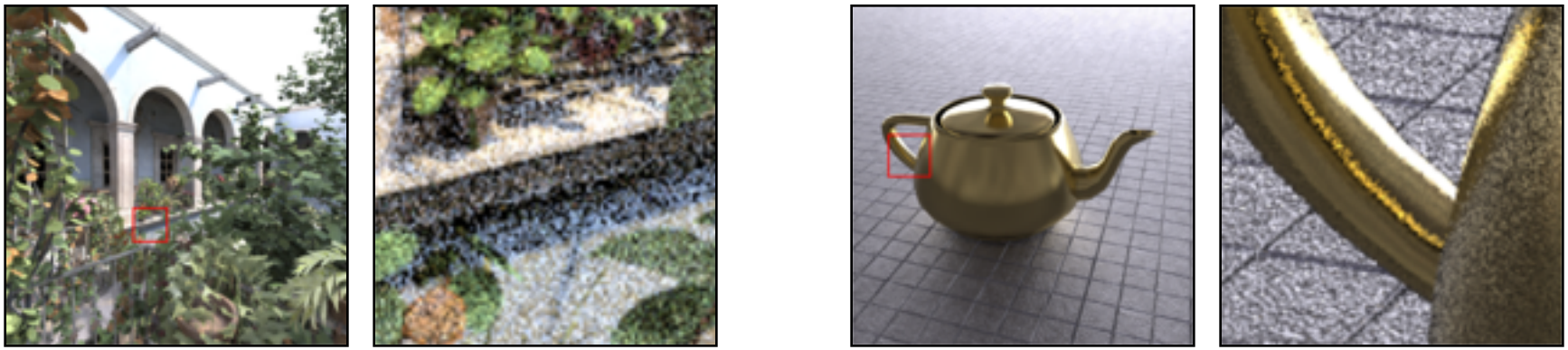
Rendering Algorithms

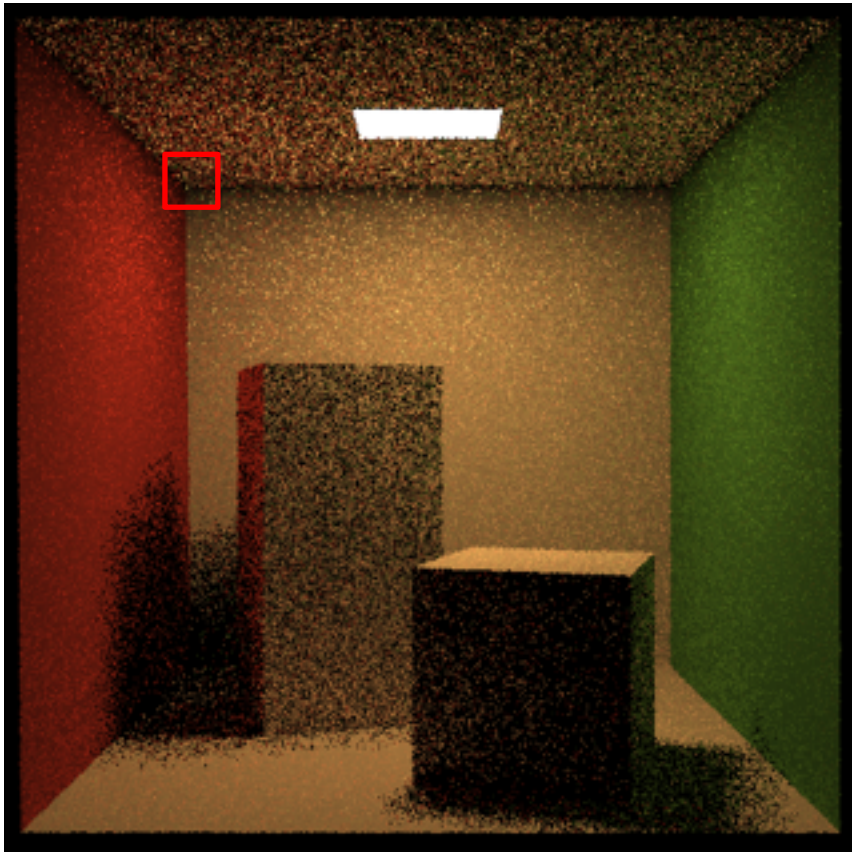
Denoising Monte Carlo Renderings

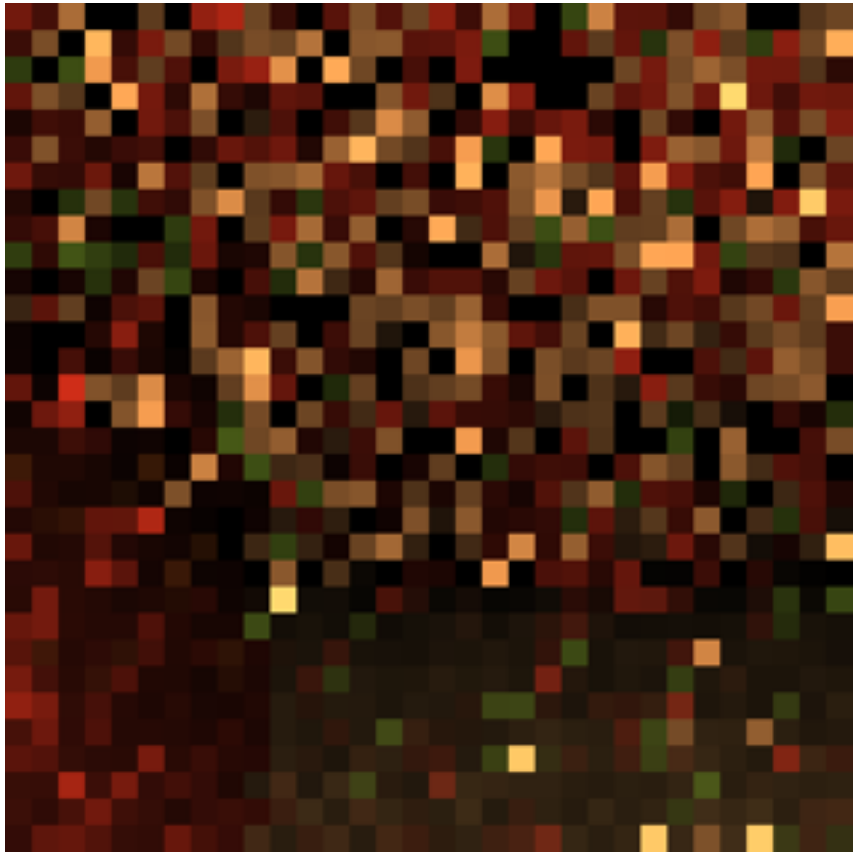


Wojciech Jarosz

(slides courtesy of Dr. Fabrice Rousselle)

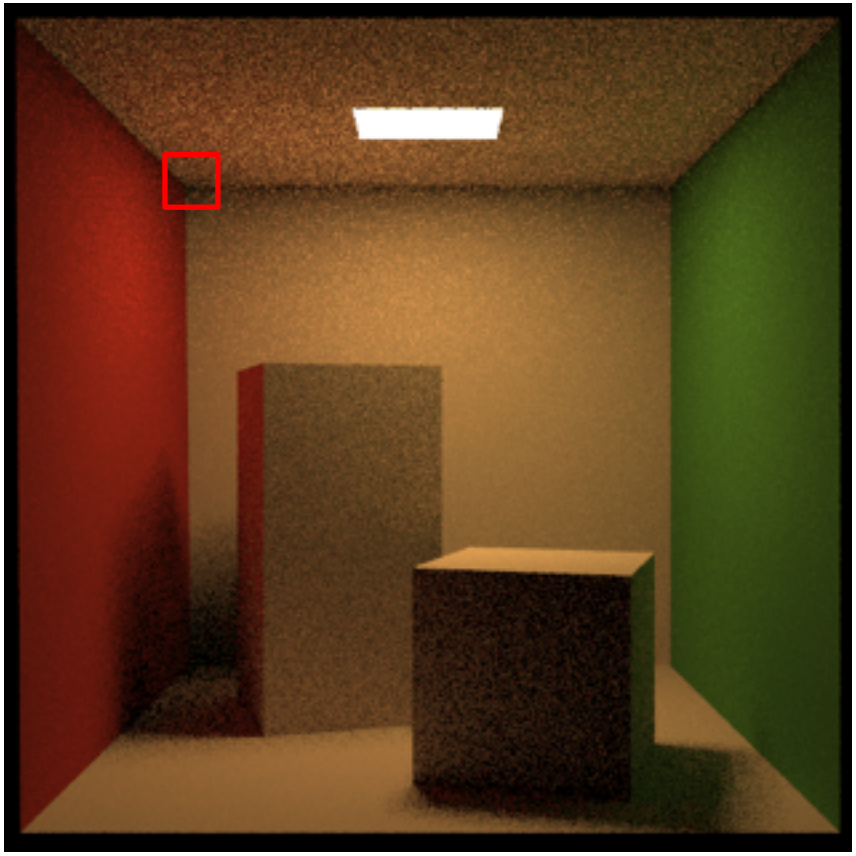
Monte Carlo Rendering

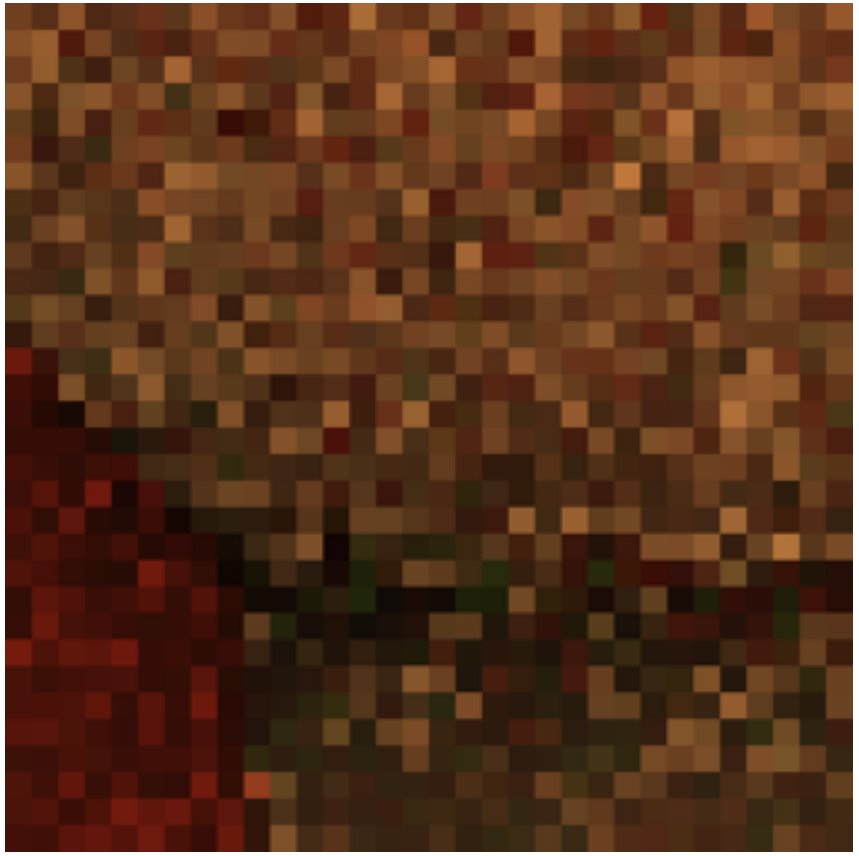




1 sample per pixel, 0.3s

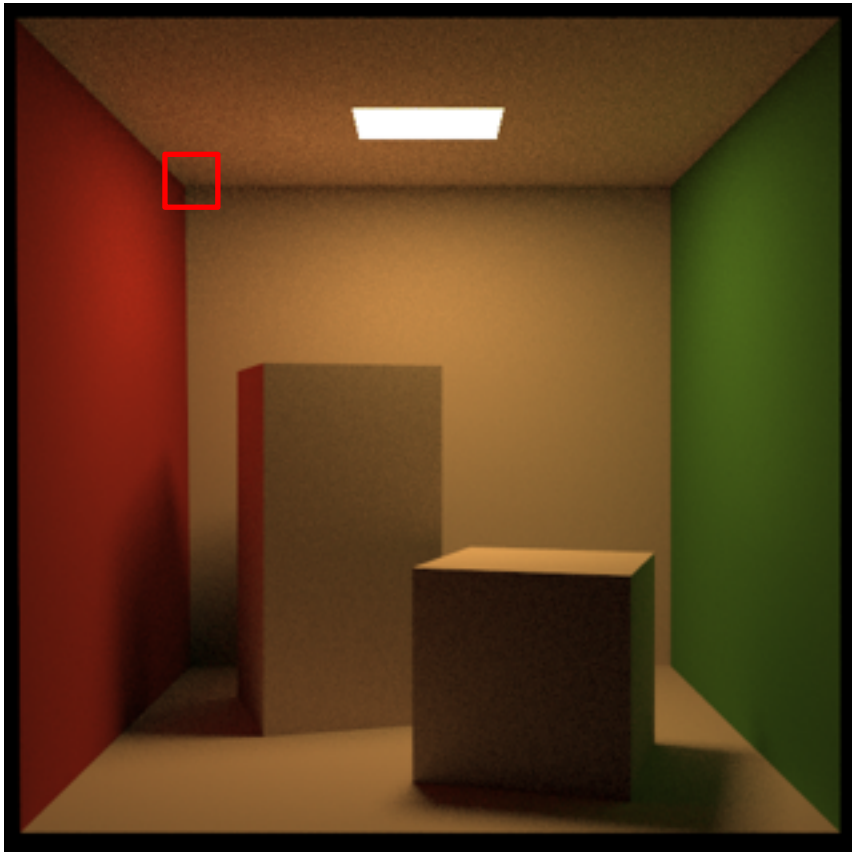
Monte Carlo Rendering





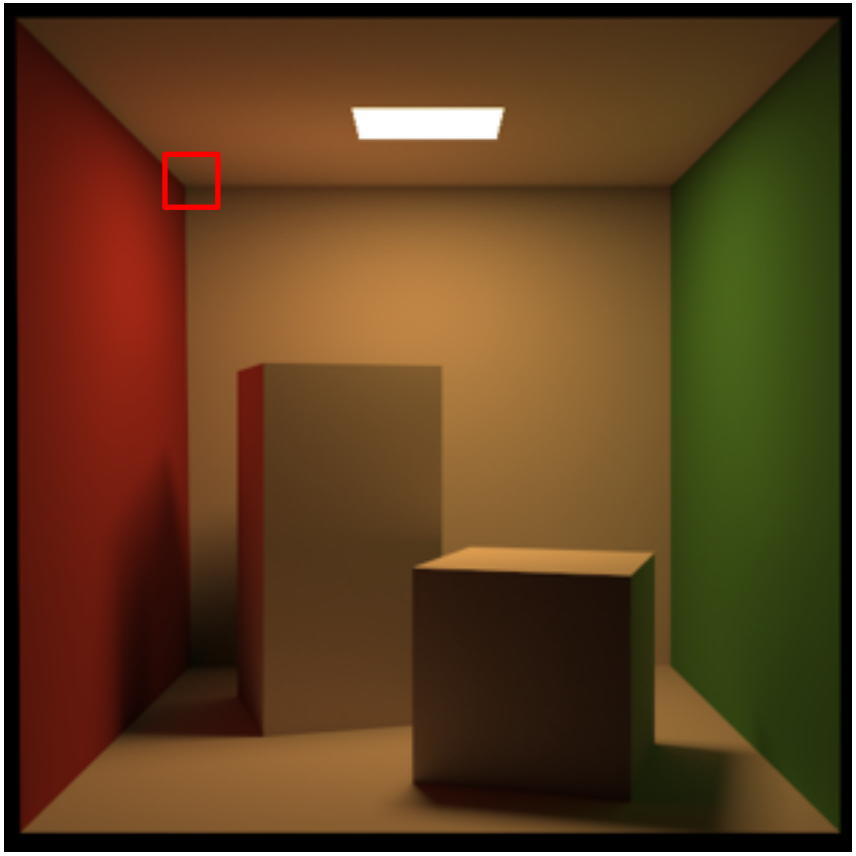
10 samples per pixel, 2.6s

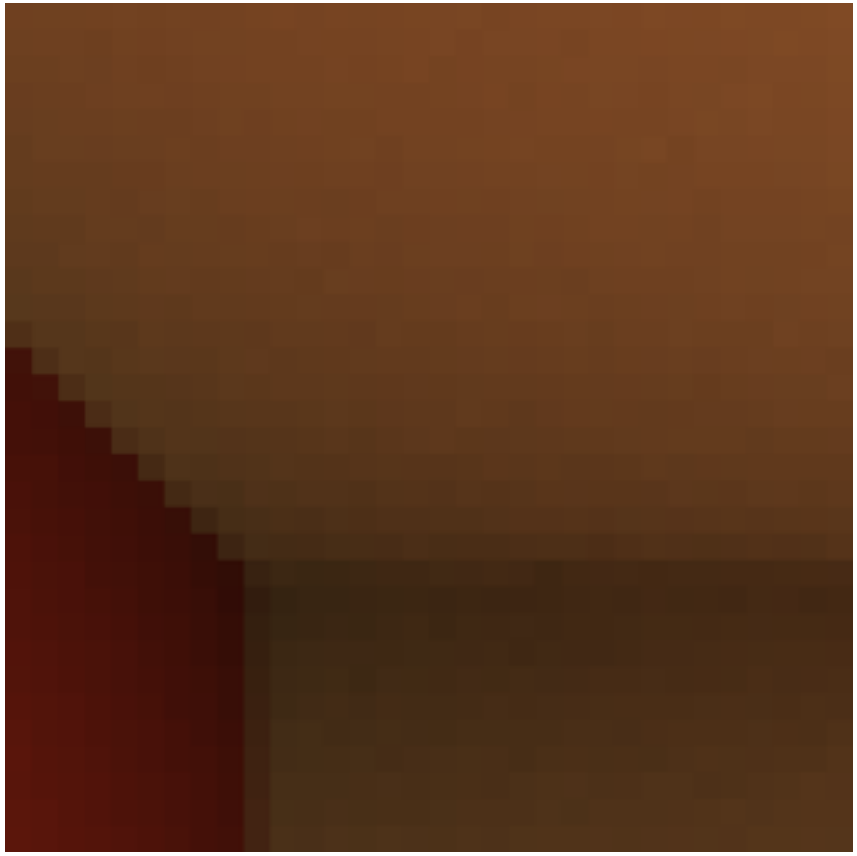
Monte Carlo Rendering



100 samples per pixel, 26s

Monte Carlo Rendering

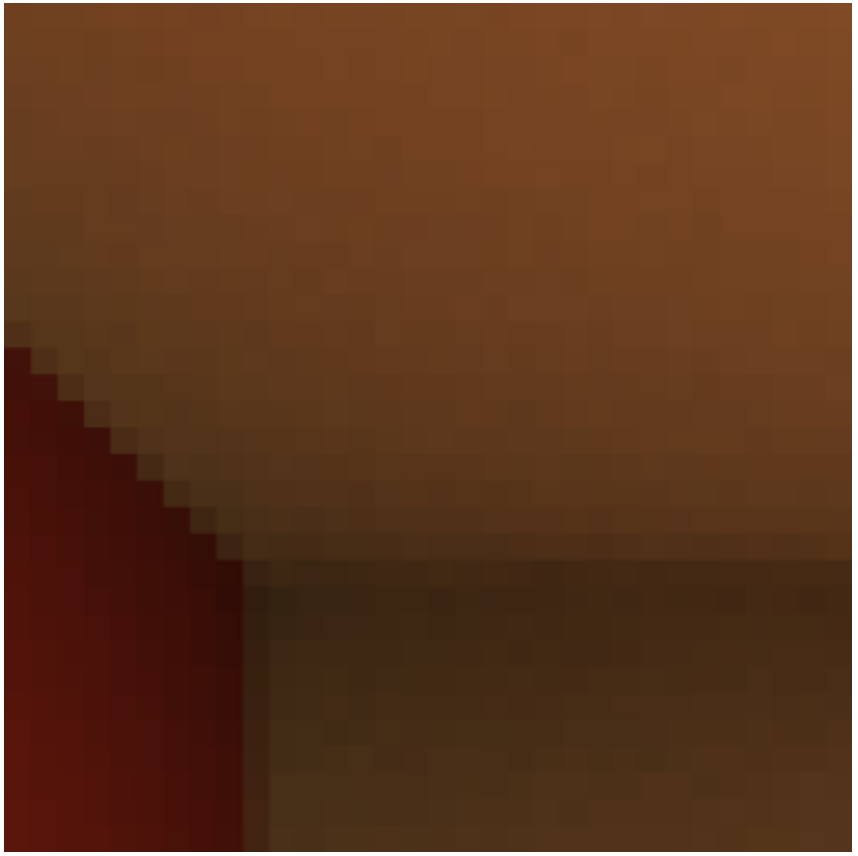




10 000 samples per pixel, 51mn

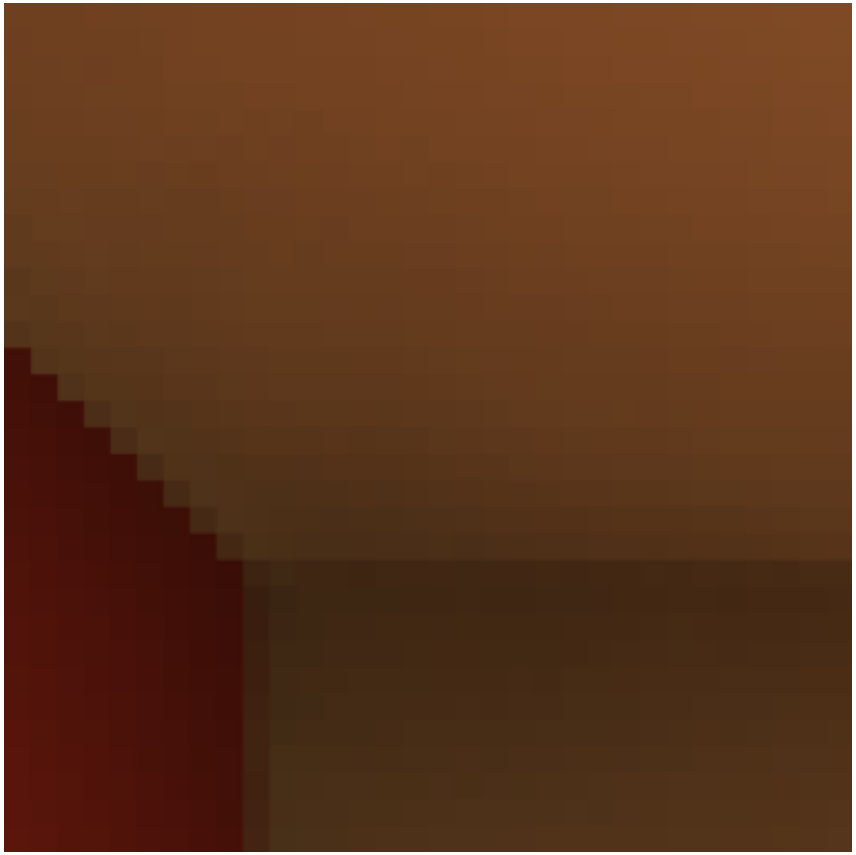
Monte Carlo Rendering

100 samples per pixel, 26s

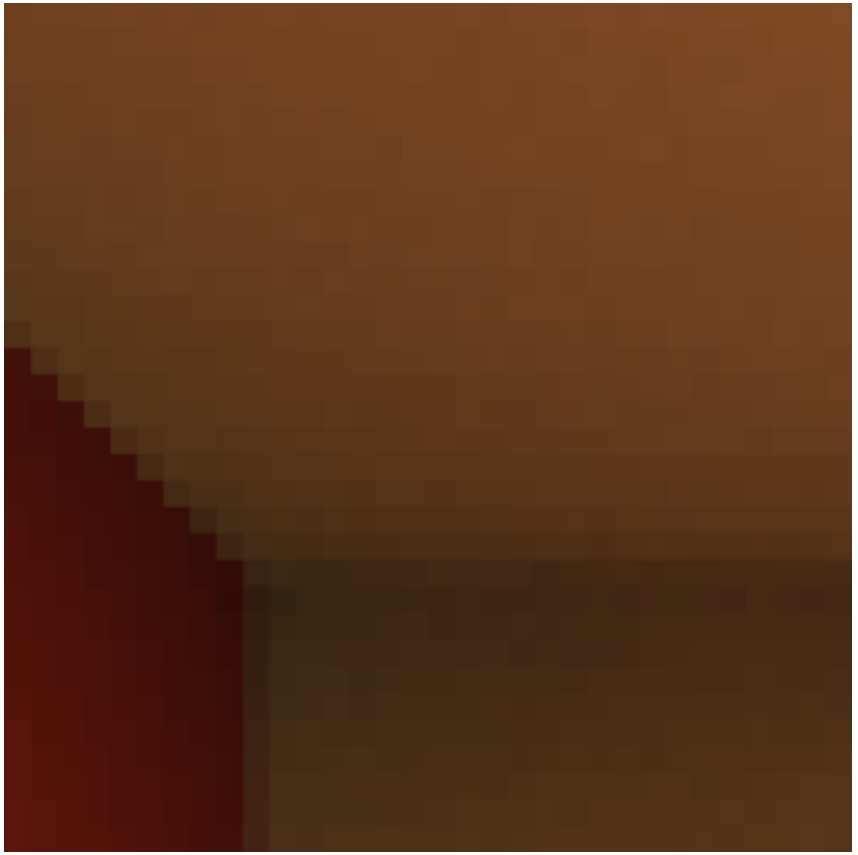


Ground truth (10 000 spp)

Monte Carlo Rendering

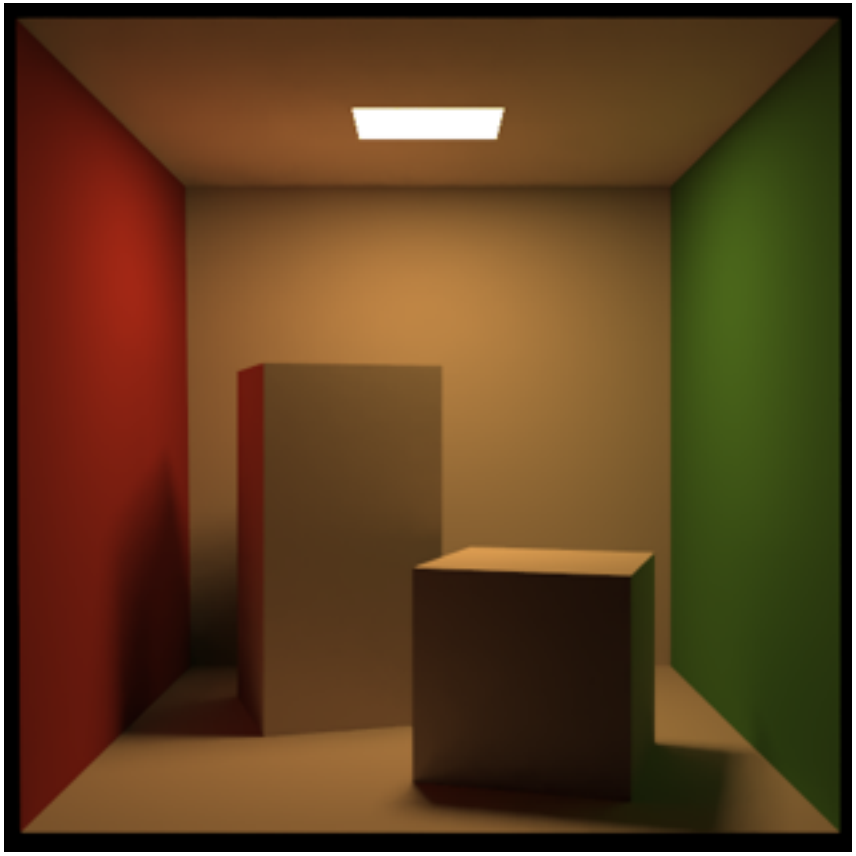


100 samples per pixel, 26s
+ denoising, 15s (matlab)

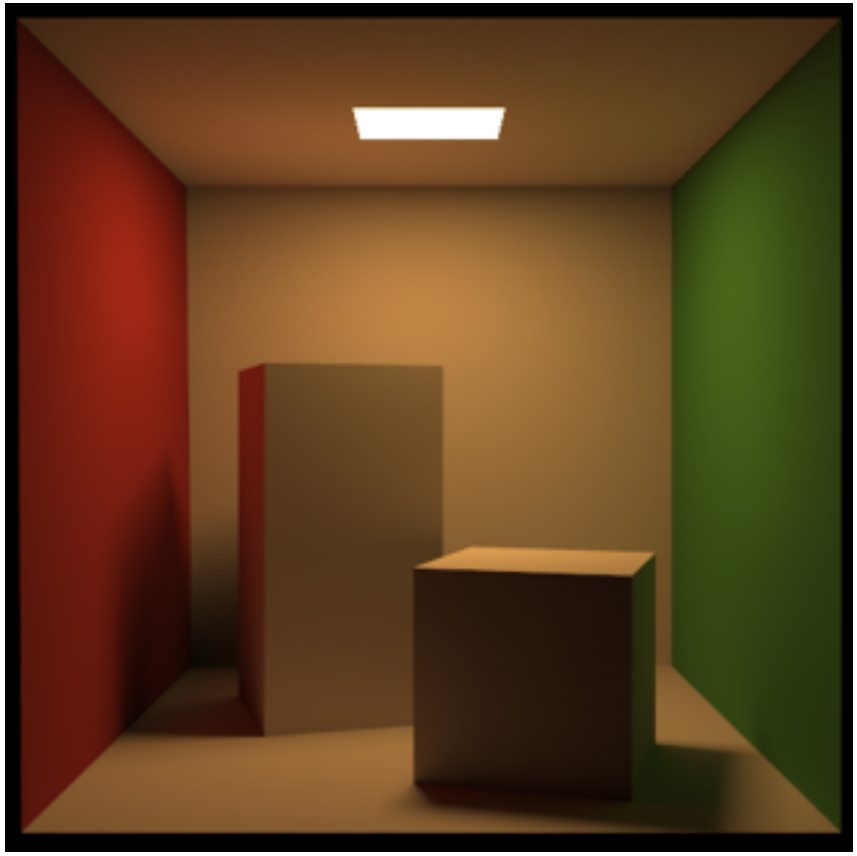


Ground truth (10 000 spp)

Monte Carlo Rendering



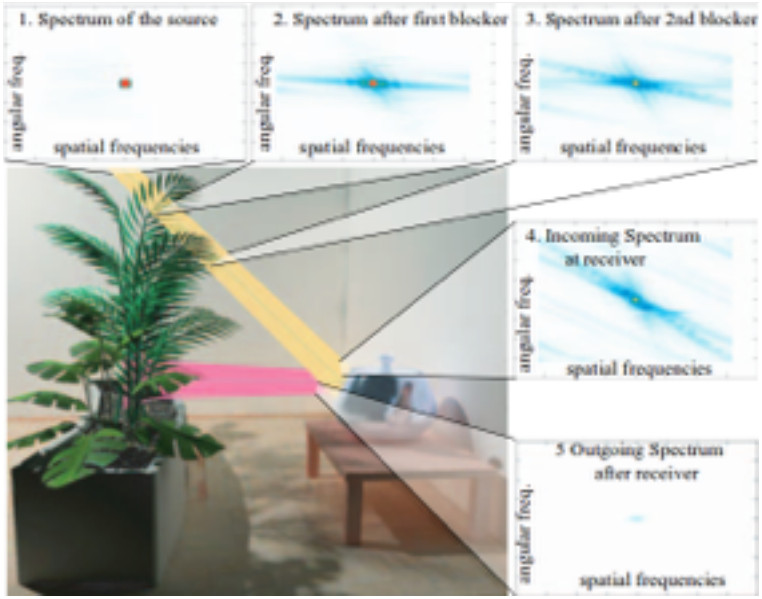
100 spp, denoised



Ground truth (10 000 spp)

Denoising – Two Approaches

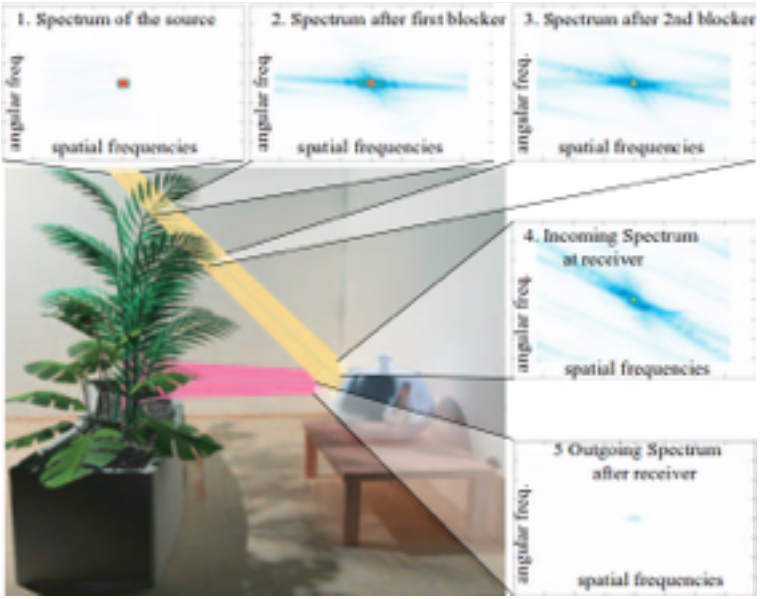
- *A priori* methods
  Analytical



Durand et al. 2005

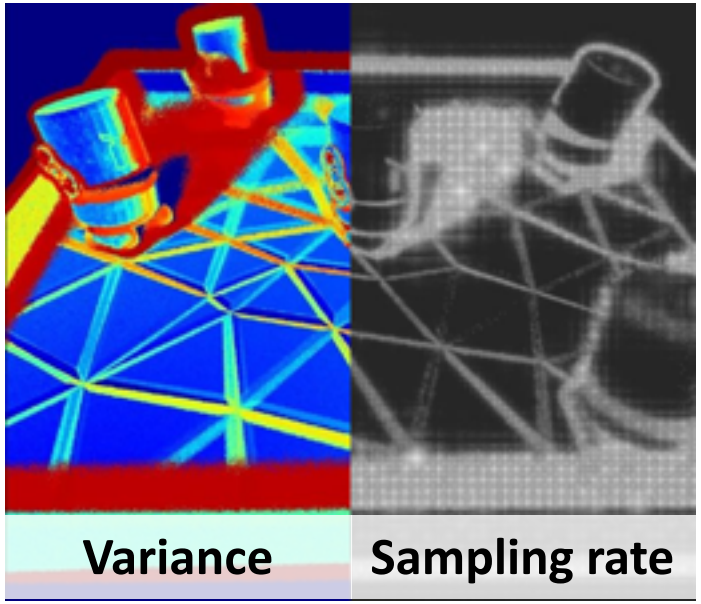
MC Noise – Two Approaches

- *A priori* methods
  Analytical



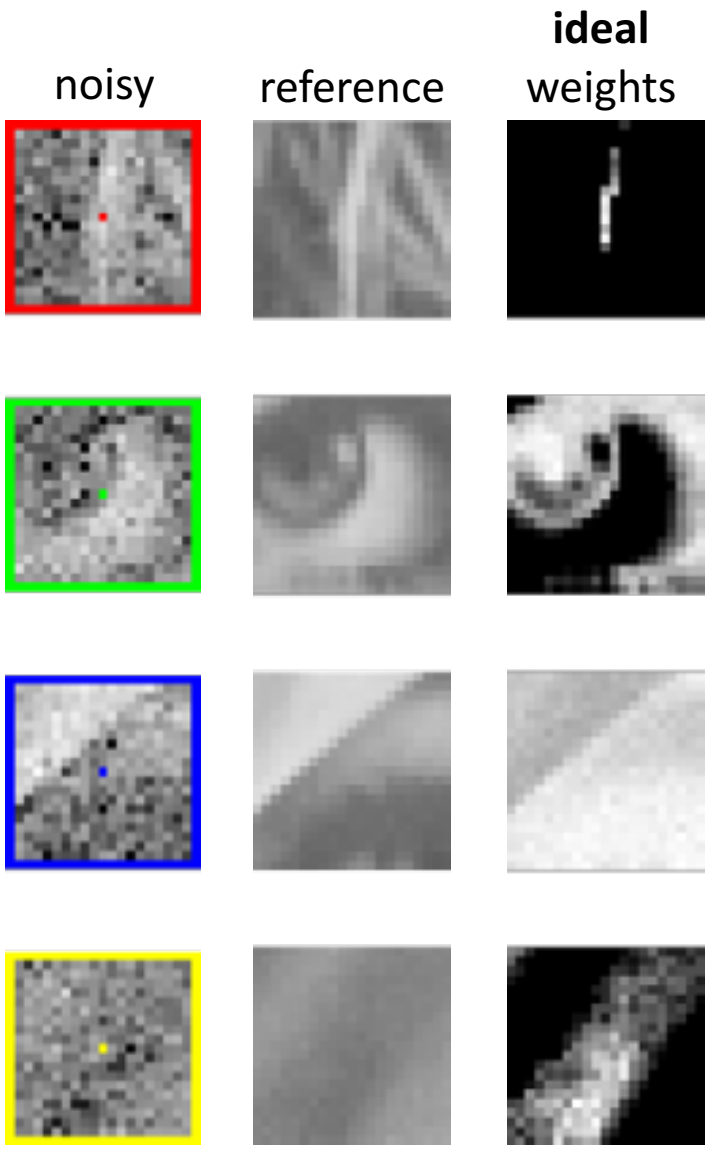
A frequency analysis of light transport
Durand et al., SIGGRAPH 2005

- *A posteriori* methods
  Empirical



Adaptive Wavelet Rendering
Overbeck et al., SIGGRAPH Asia 2009

Image-Based Denoising



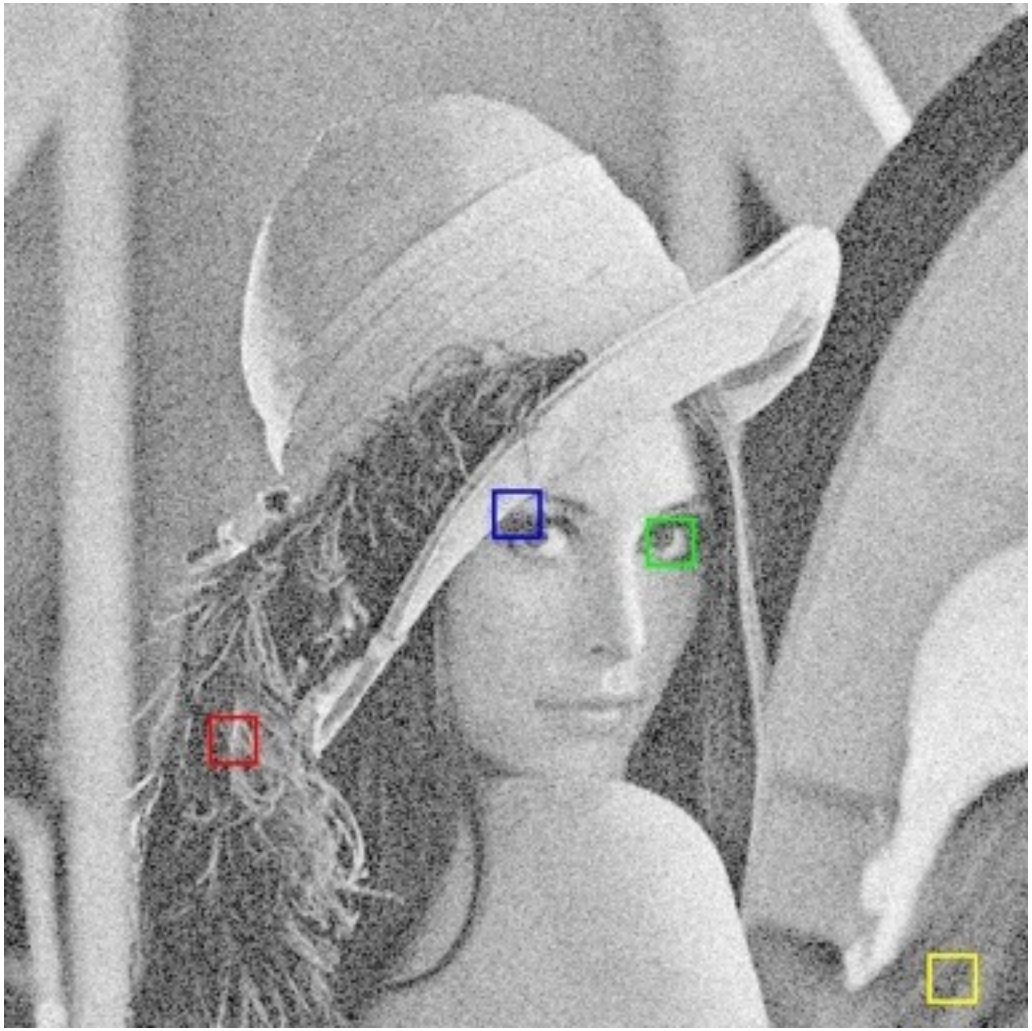
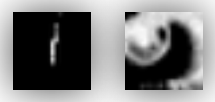


Image-Based Denoising

- Use a **flexible** and **robust** filter
  - Flexible: arbitrary filter support
  - Robust: few denoising artifacts



Noisy data

Bilateral Filter

Non-Local Means Filter

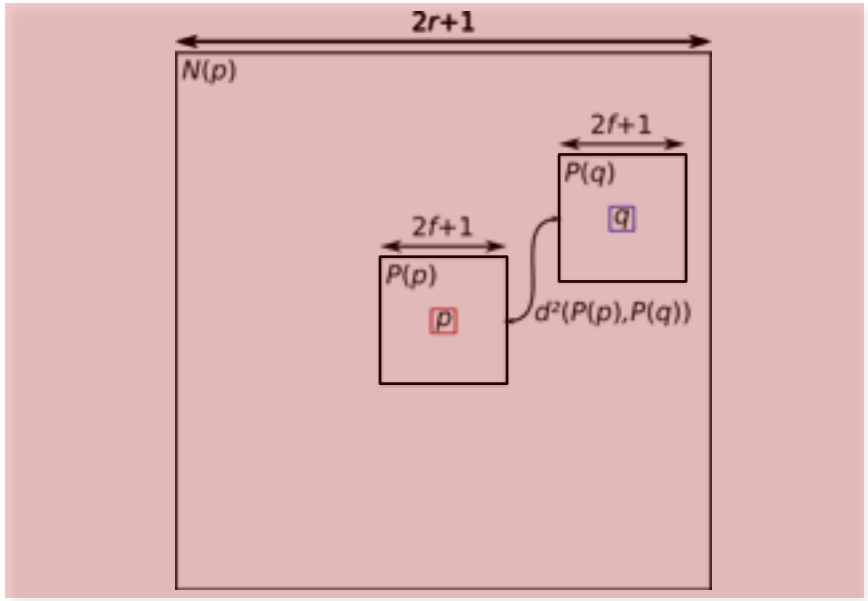
Buades et al. 2006

The Non-Local Means Filter

- Per pixel weight $w(p,q)$
- Generalized distance d
  - spatial (in pixels, exact)
  - range (difference in value u, noisy)

$$d^2(p,q) = \frac{(u(p) - u(q))^2 - 2\sigma^2}{\epsilon + k^2 2\sigma^2}$$

- Weights
  - decrease exponentially with the distance

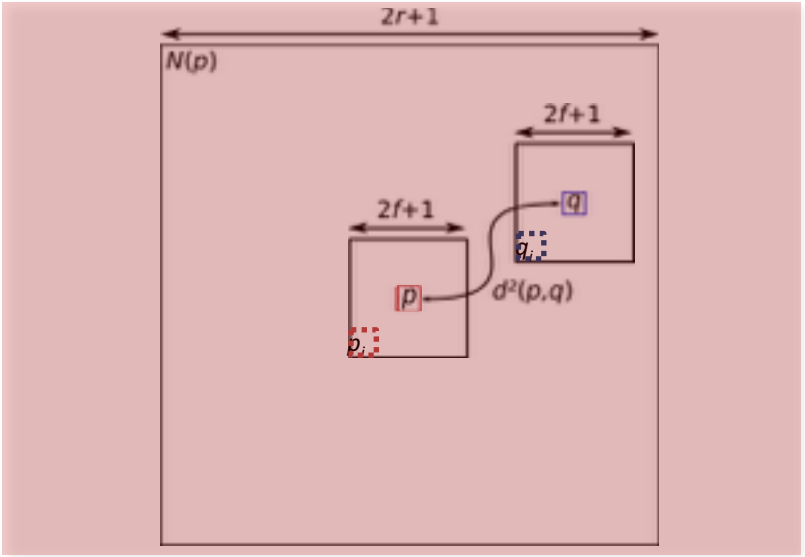


Variance σ^2

$ff=30$ bilateral filter

NL-Means – Naïve Implementation

```
INPUT: dat, datvar
INPUT: r(10), f(3), k(0.45)
flt = wgtsum = 0                          // 1. Initialize output
for each pixel p                          // 2. Loop over pixels
   for each pixel q in N(p)               // 3. Loop over neighbors (r²)
      d2patch = 0                         // 4. Loop over pairs
      for each offset i in P()            //    (P(p), P(q)) to get
         d2patch += d2(p_i,q_i,k)         //    mean distance over
      d2patch /= area(P())                //    patch
```



Complexity: $O(r^2 \times f^2)$

NL-Means – Naïve Implementation

```
INPUT: dat, datvar
INPUT: r(10), f(3), k(0.45)
flt = wgtsum = 0                          // 1. Initialize output
for each pixel p                          // 2. Loop over pixels
   for each pixel q in N(p)               // 3. Loop over neighbors
      d2patch = 0                         // 4. Loop over pairs
      for each offset i in P()            //    (P(p), P(q)) to get
         d2patch += d2(p_i,q_i,k)         //    mean distance over
      d2patch /= area(P())                //    patch
      wgt = exp(-max(0,d2patch))          //    Weight of P(q)
      for each offset i in P()            // 5. Store weighted
         wgtsum(p_i) += wgt               //    contribution of P(q)
         flt(p_i) += wgt * dat(q_i)       //
flt /= wgtsum                             // 6. Normalize output
                                          //
```

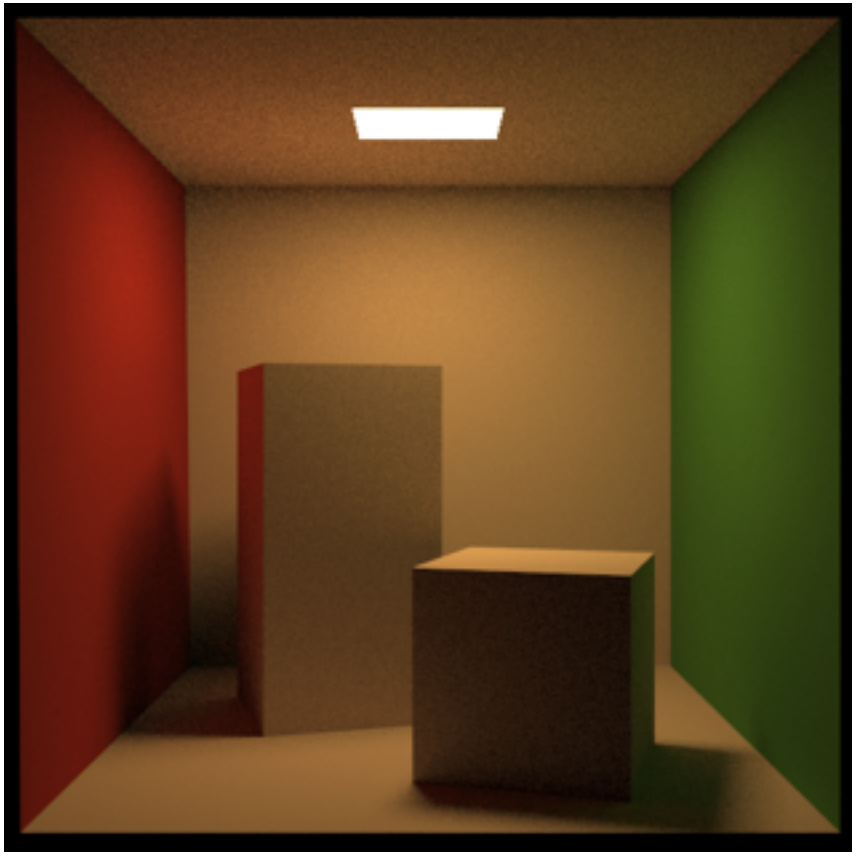
Complexity: $O(r^2 \times f^2)$

NL-Means – Fast Implementation

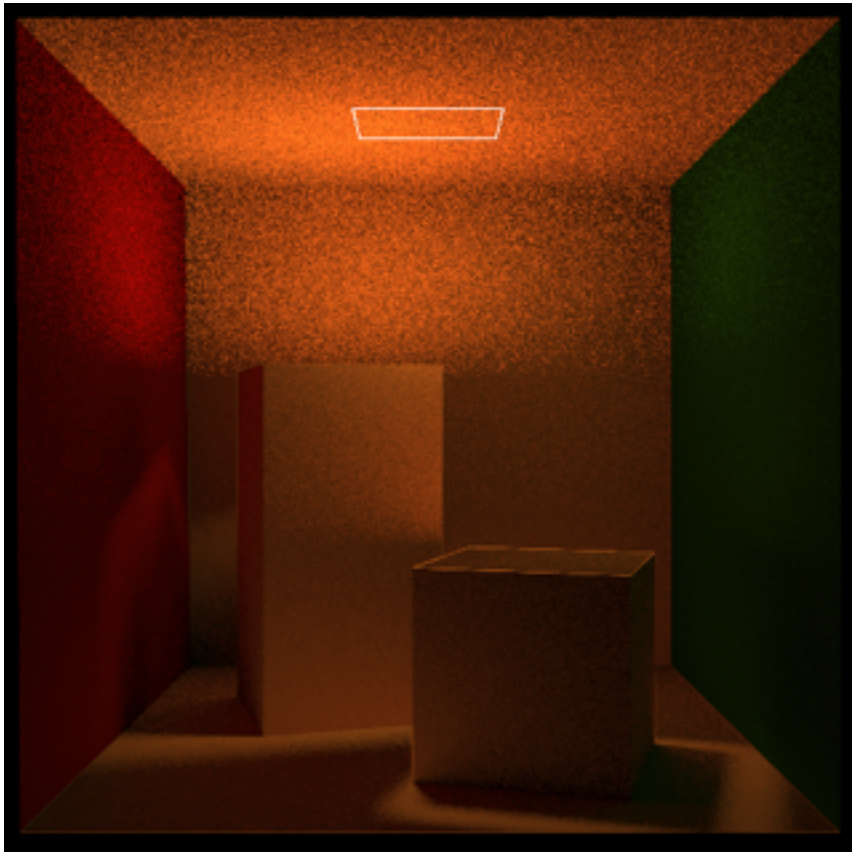
```
INPUT: dat, datvar
INPUT: r(10), f(3), k(0.45)
flt = wgtsum = 0                          // 1. Initialize output
for each offset (dx,dy) in N()            // 2. Loop over neighbors
   ngb = shift(dat,dx,dy)                 // 3. Compute distance to
   d2pixel = d2(ngb,dat,k)                //    neighbor (dx,dy)
   d2patch = conv(d2pixel,box_f)          // 4. Apply box filter for
   wgt = exp(-max(0,d2patch)              //    patch distance+weight
   wgt = conv(wgt,box_f-1)                // 5. Box filter weights for
   flt += wgt * ngb                       //    patch contribution
   wgtsum += wgt                          //
flt /= wgtsum                             // 6. Normalize
```

Complexity: $O(r^2 \times f)$

NL-Means – Application



100 samples per pixel



Variance of pixel mean (x1000)

NL-Means – Non-Uniform Variance

- Uniform variance

$$d^2(p,q) = \frac{(u(p) - u(q))^2 - 2\sigma^2}{\epsilon + k^2 2\sigma^2}$$

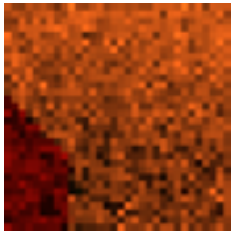
- Non-uniform variance

$$d^2(p,q) = \frac{(u(p) - u(q))^2 - (\mathrm{Var}\,[p] + \min(\mathrm{Var}\,[q]\,, \mathrm{Var}\,[p]))}{\epsilon + k^2(\mathrm{Var}\,[p] + \mathrm{Var}\,[q])}$$

Variance Estimation

- For independent samples, use sample mean variance

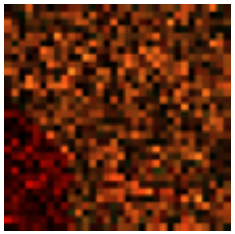
$$\mathrm{Var}[p] = \left(\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2 \right) / n$$



$(\times 1000)$

- For correlated samples, use buffer difference
  - Stratified sampling, low-discrepancy sampling, MLT, etc.
  - Generate two images with independent seeds
  - Compute the variance of the mean of the two images

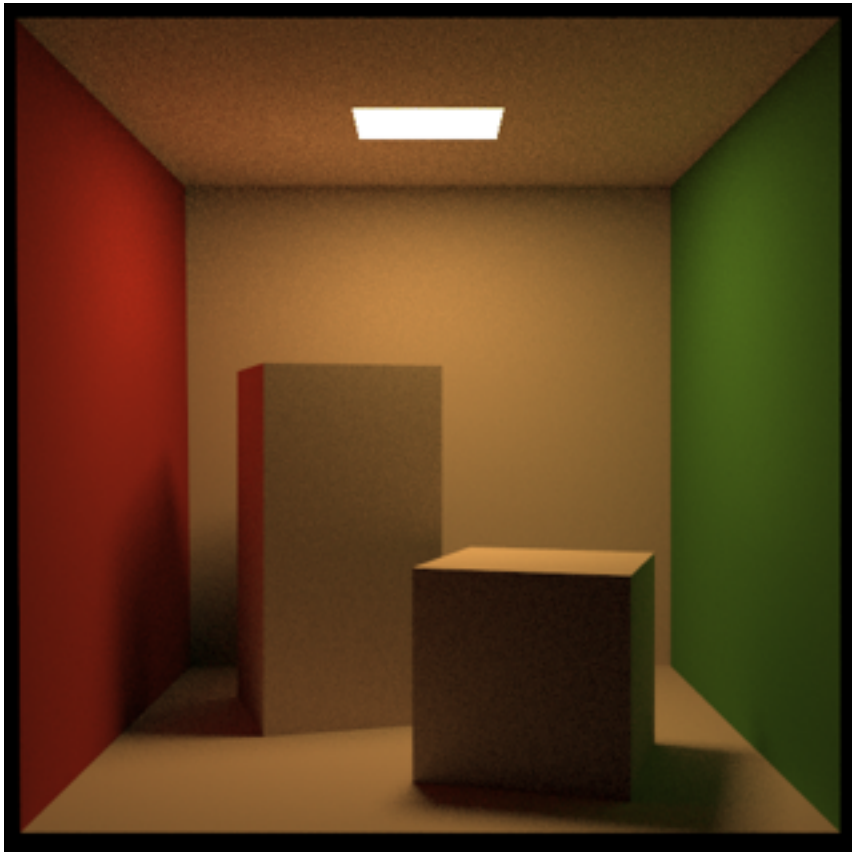
$$\left(\text{  } - \text{  } \right)^2 /4 =$$



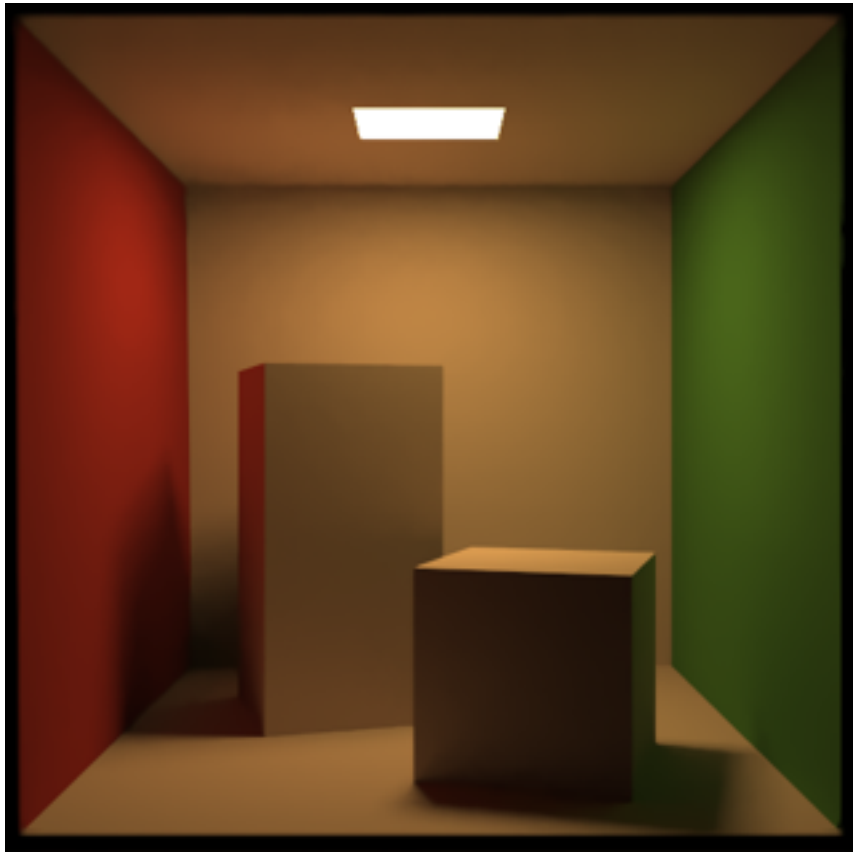
$(\times 1000)$

NL-Means – Application

Independent samples



100 samples per pixel

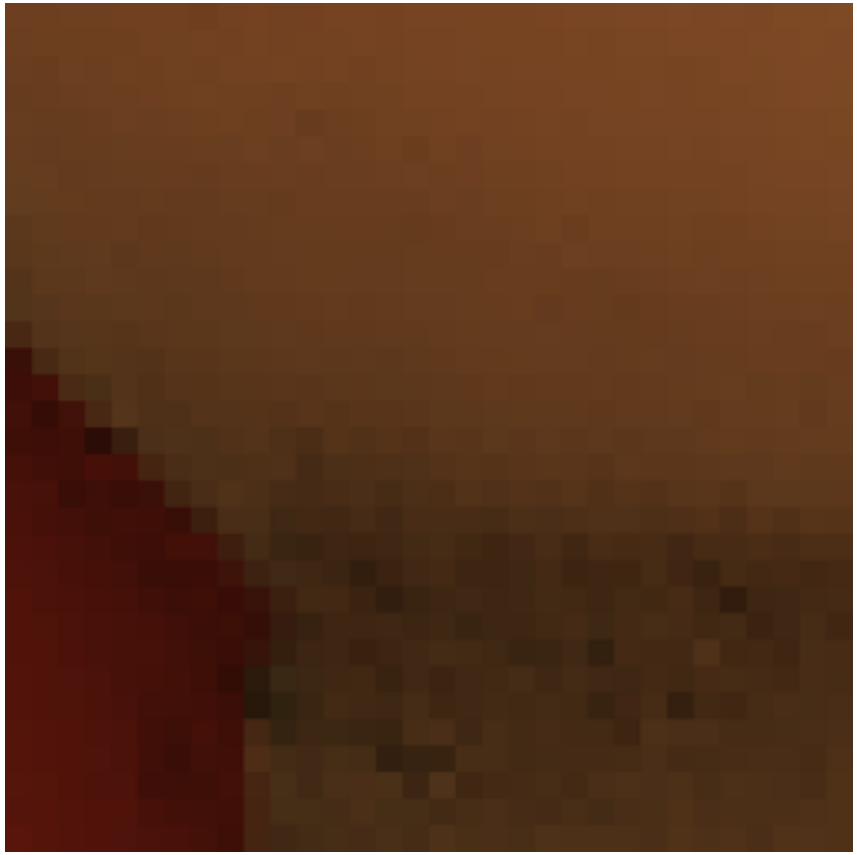


Denoised

NL-Means – Application

Independent samples

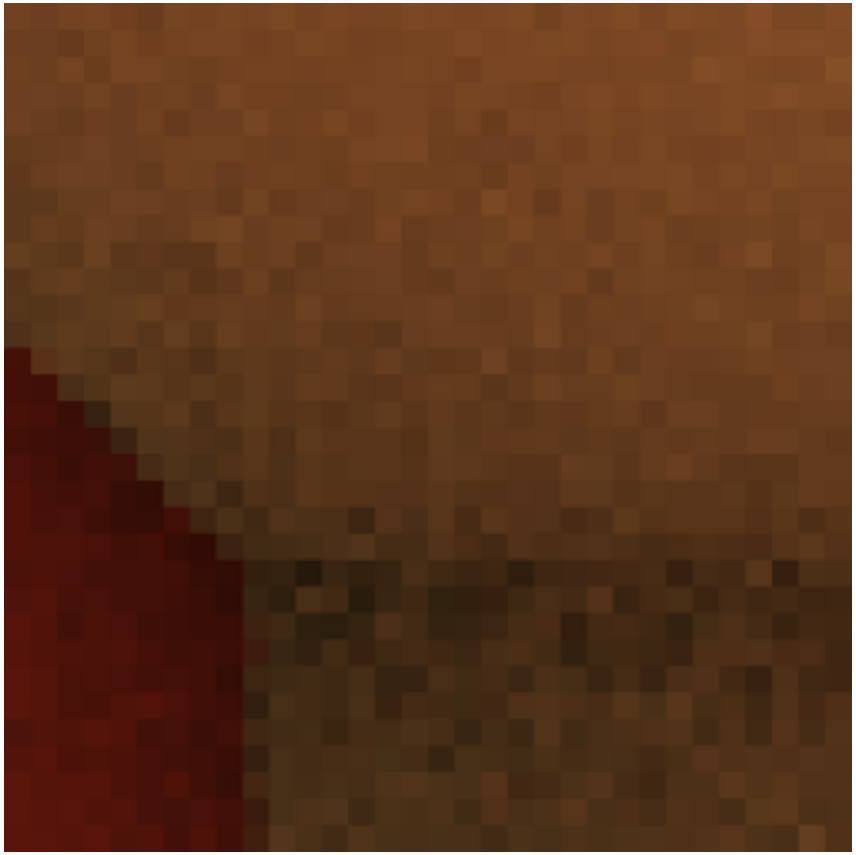
100 samples per pixel
relative MSE: 1.573E-3



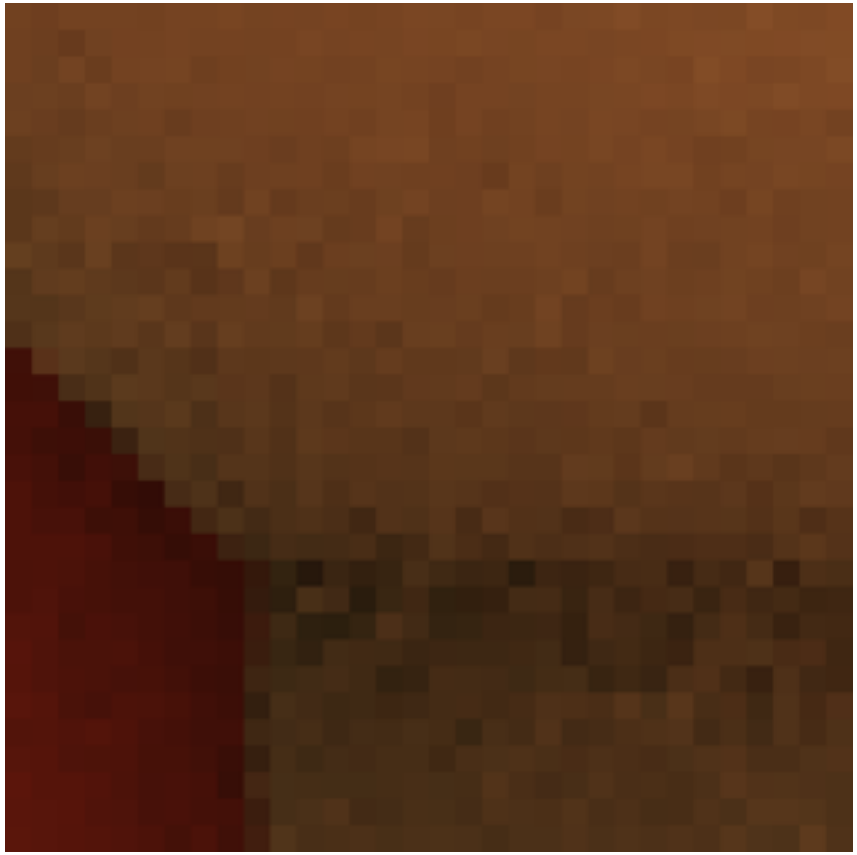
Denoised
relative MSE: 0.198E-3

NL-Means – Application

Low-discrepancy samples



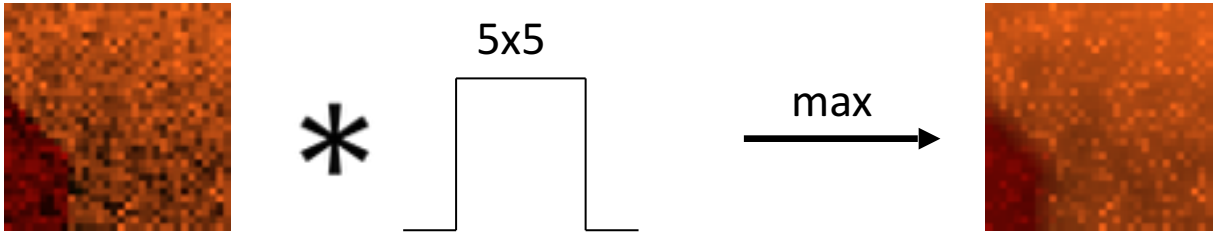
2 x 50 LD samples per pixel
relative MSE: 0.405E-3



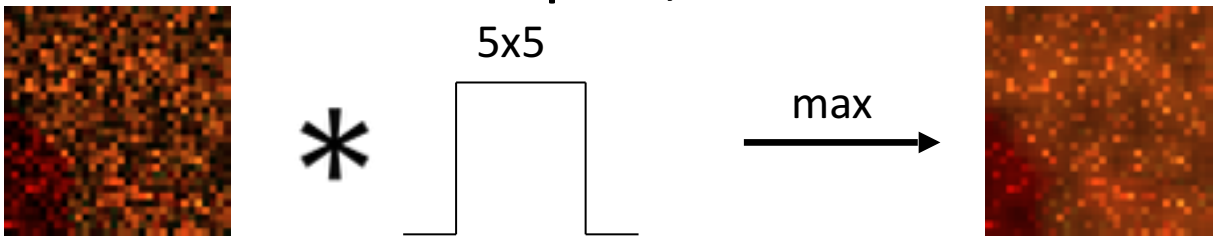
Denoised
relative MSE: 0.257E-3

Variance Estimation

- For independent samples, use sample mean variance



- For correlated samples, use buffer difference

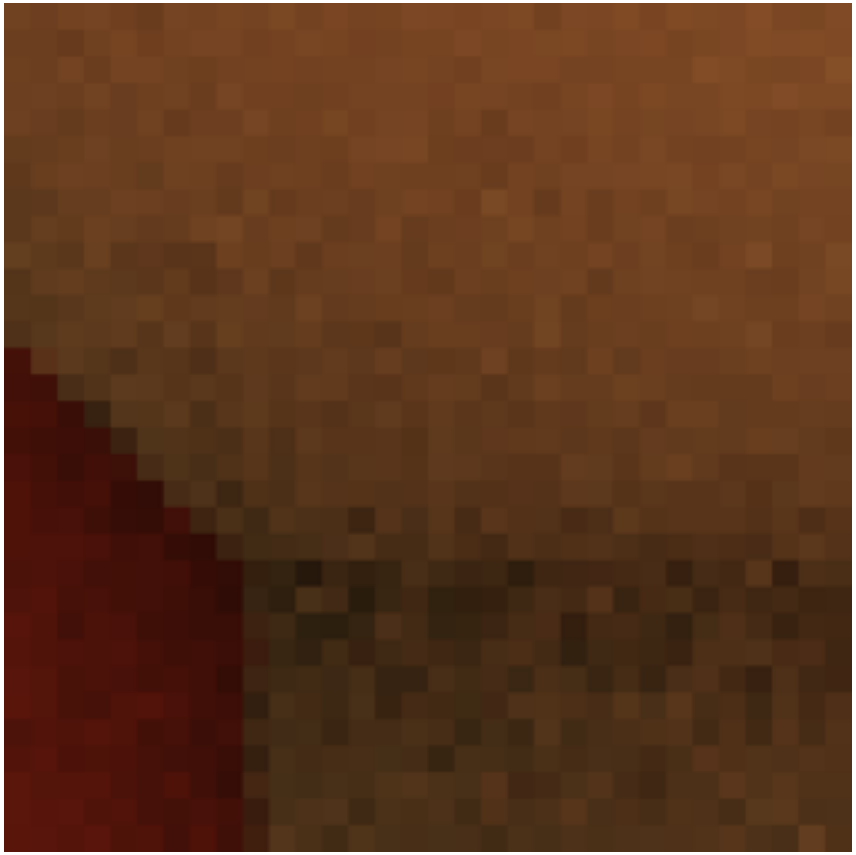


NL-Means – Fast Implementation

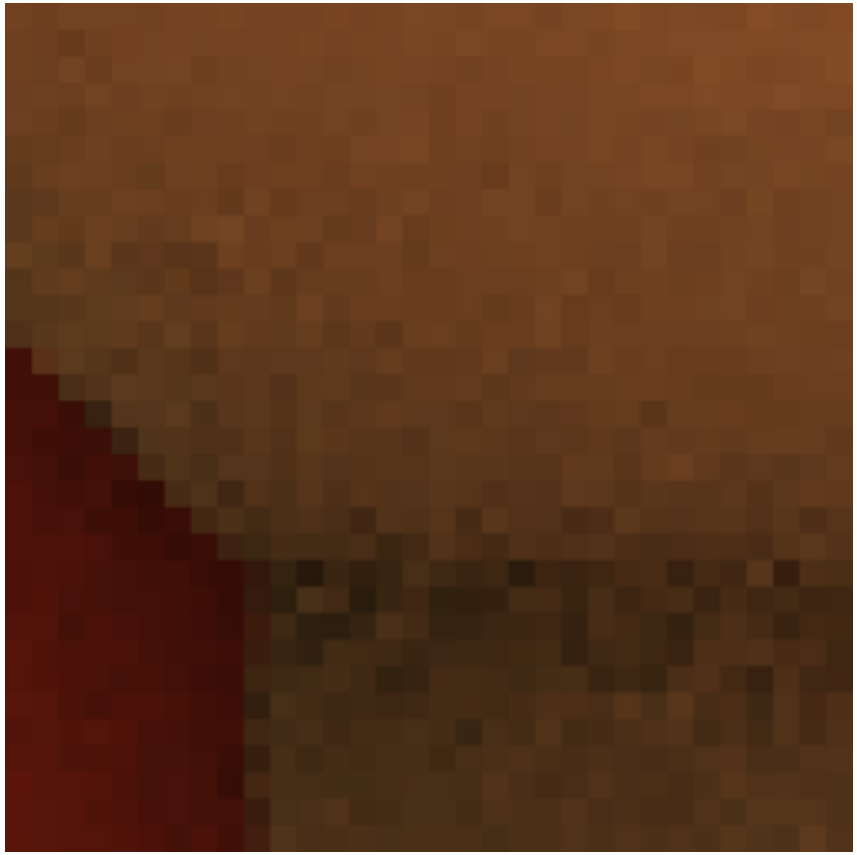
```
INPUT: dat, datvar
datvar = max(datvar, conv(datvar,box_2)
flt = wgtsum = 0
for each offset (dx,dy) in N()
   ngb = shift(dat,dx,dy)
   d2pixel = d2(ngb,dat)
   d2patch = conv(d2pixel,box_f)
   wgt = exp(-max(0,d2patch)
   wgt = conv(wgt,box_f-1)
   flt += wgt * ngb
   wgtsum += wgt
flt /= wgtsum
```

NL-Means – Application

Low-discrepancy samples



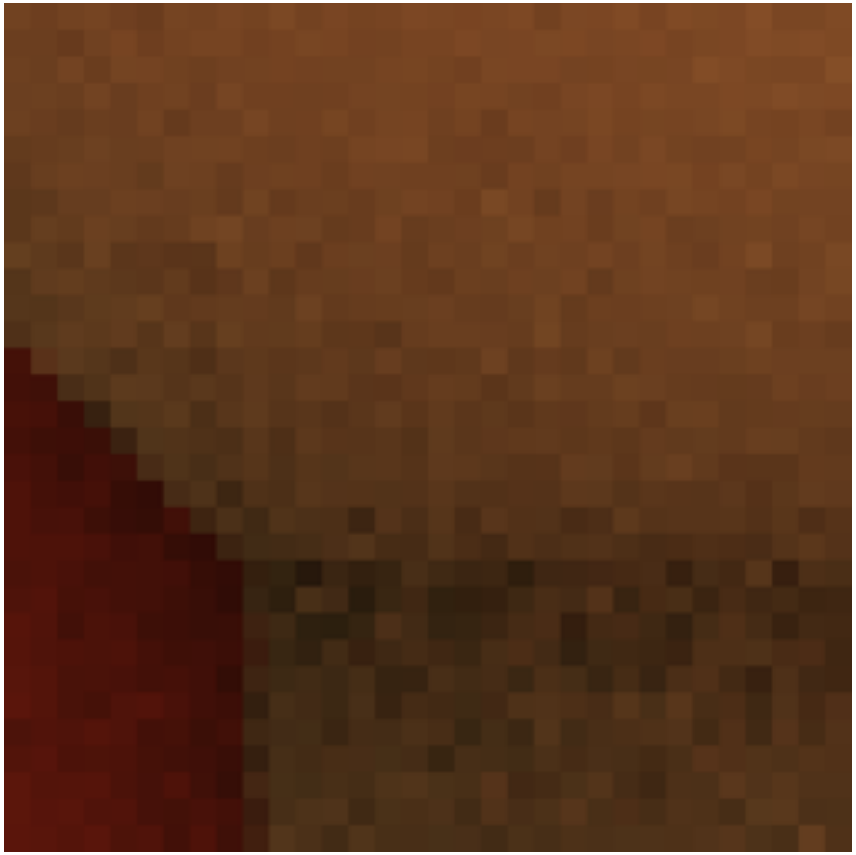
2 x 50 LD samples per pixel
relative MSE: 0.405E-3



Denoised
relative MSE: 0.257E-3

NL-Means – Application

Low-discrepancy samples



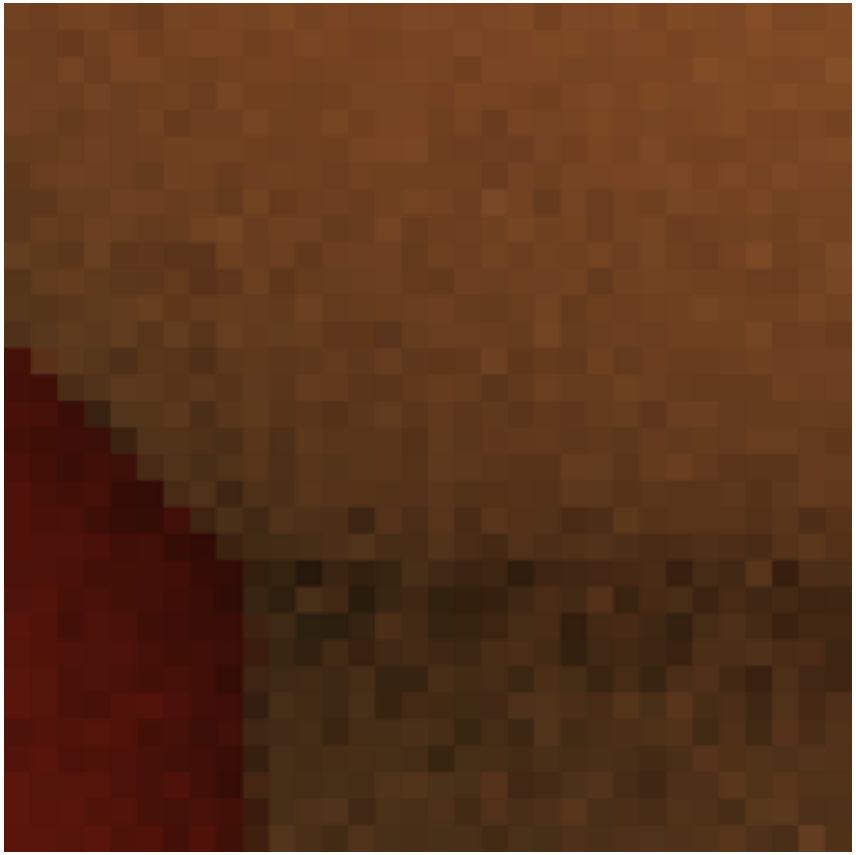
2 x 50 LD samples per pixel
relative MSE: 0.405E-3

With filtered variance

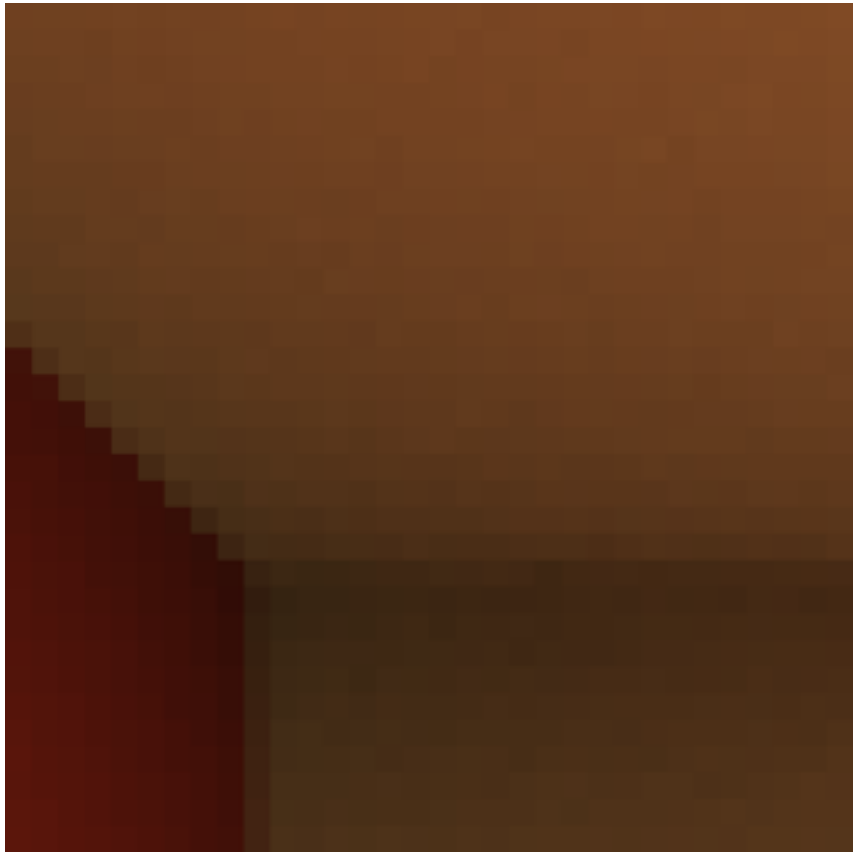
Denoised
relative MSE: 0.038E-3

NL-Means – Application

Low-discrepancy samples



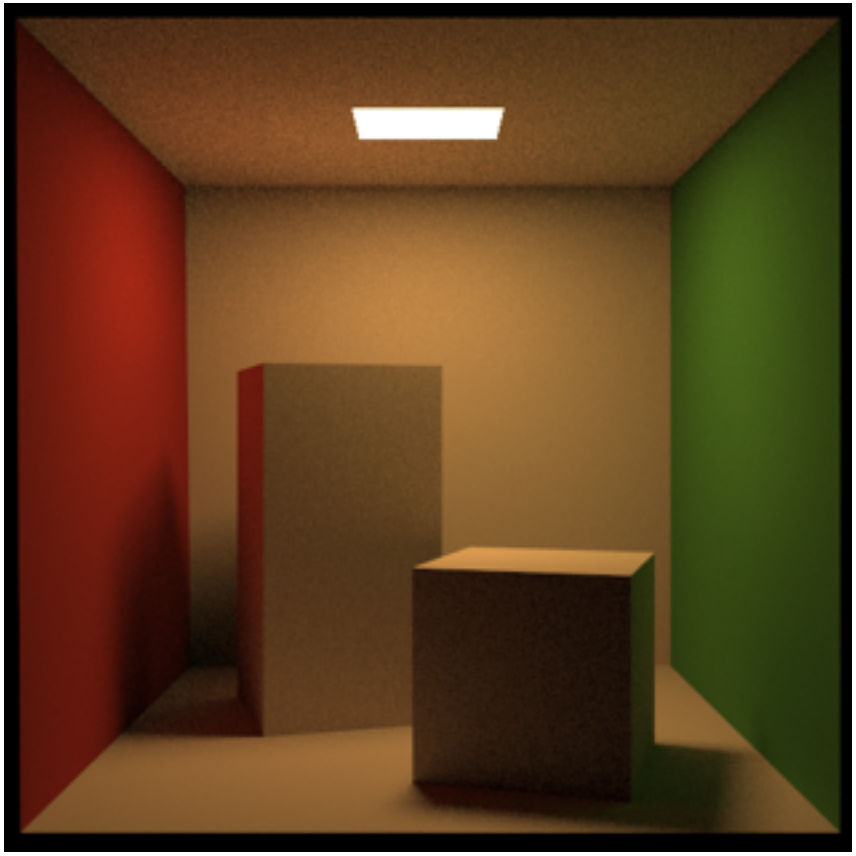
2 x 50 LD samples per pixel
relative MSE: 0.405E-3



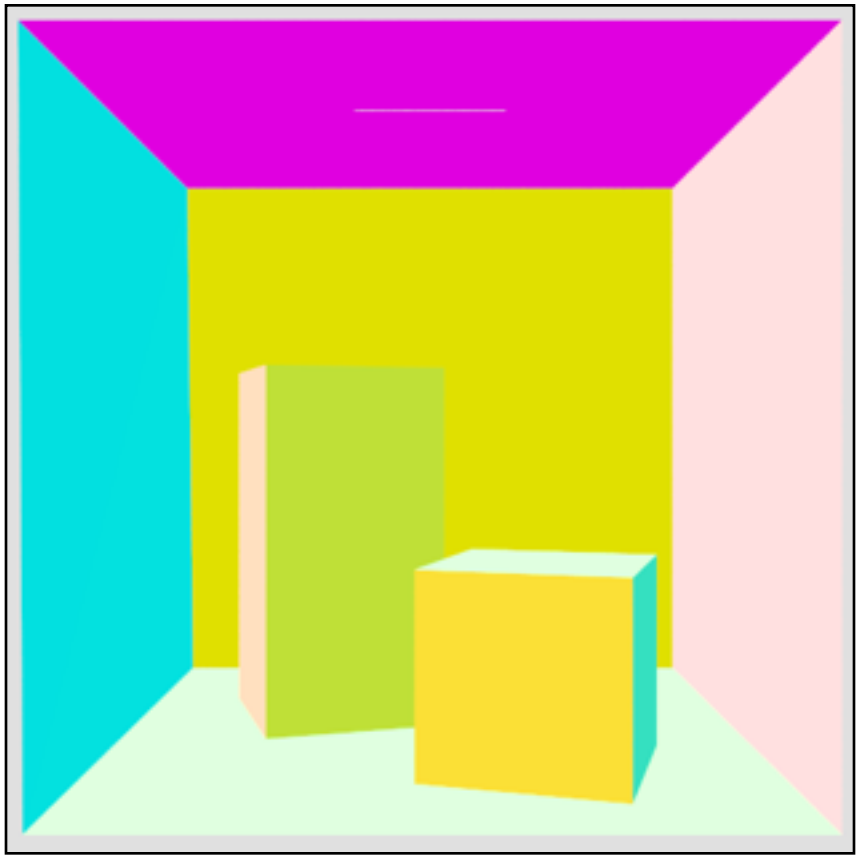
Ground truth (10 000 spp)

Leveraging Scene Information

Joint NL-Means Filter

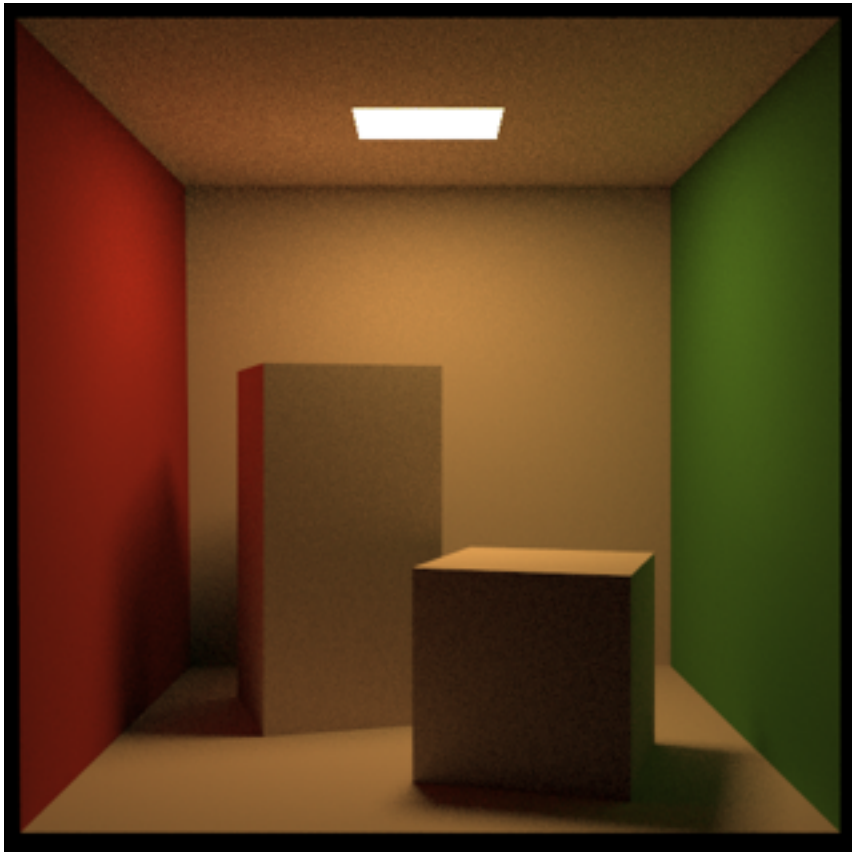


100 samples per pixel, 26s

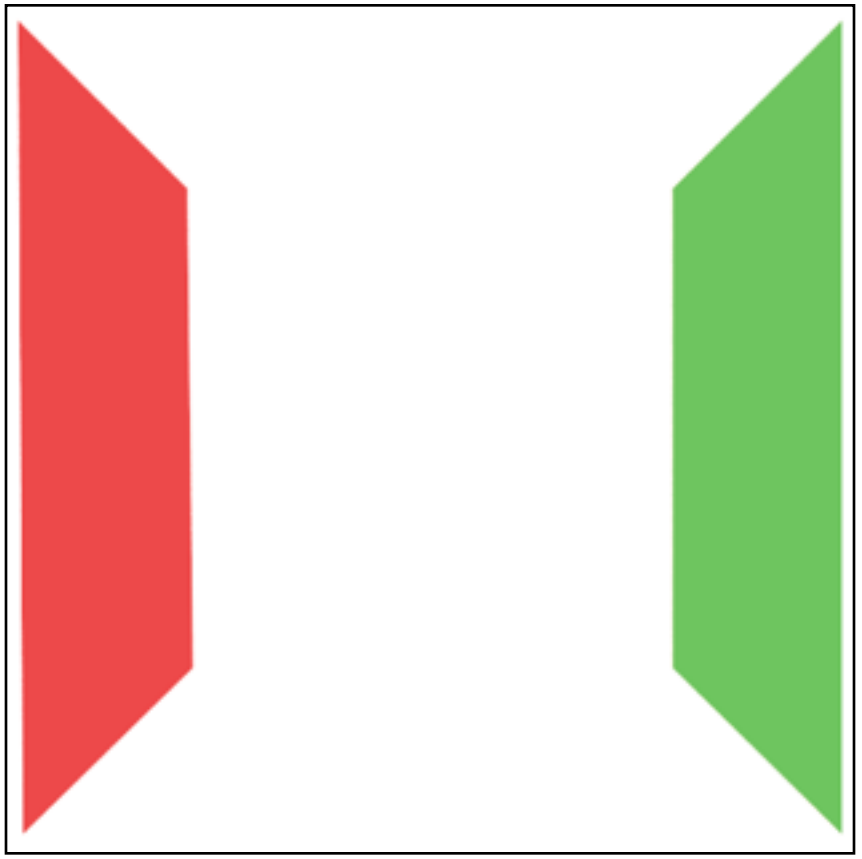


Normal, remapped to [0,1]

Joint NL-Means Filter

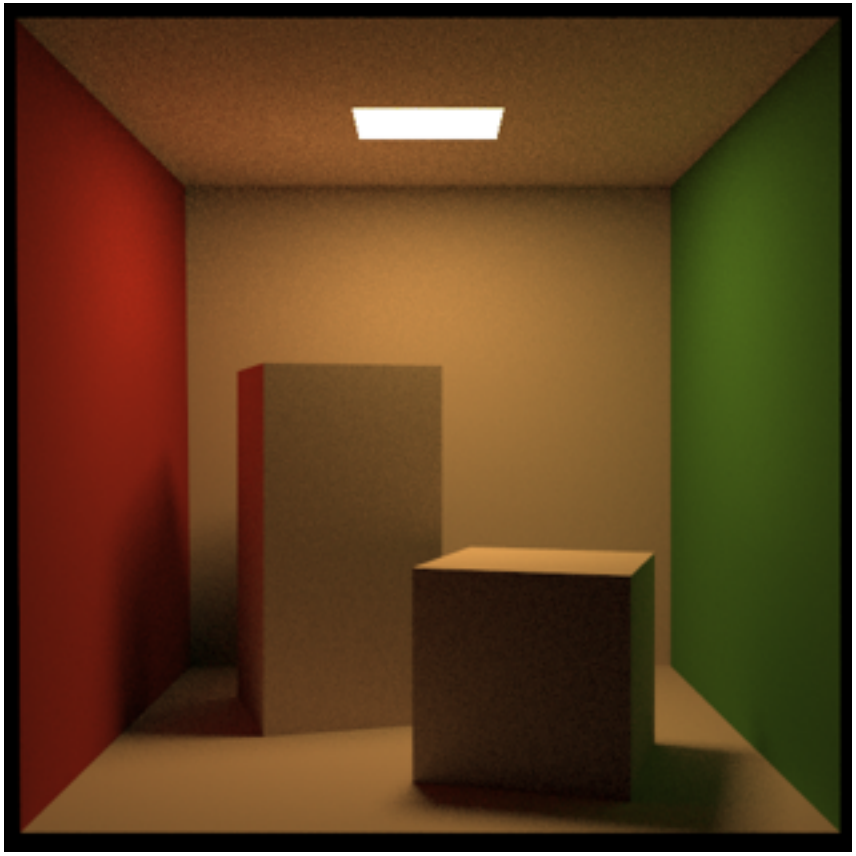


100 samples per pixel, 26s

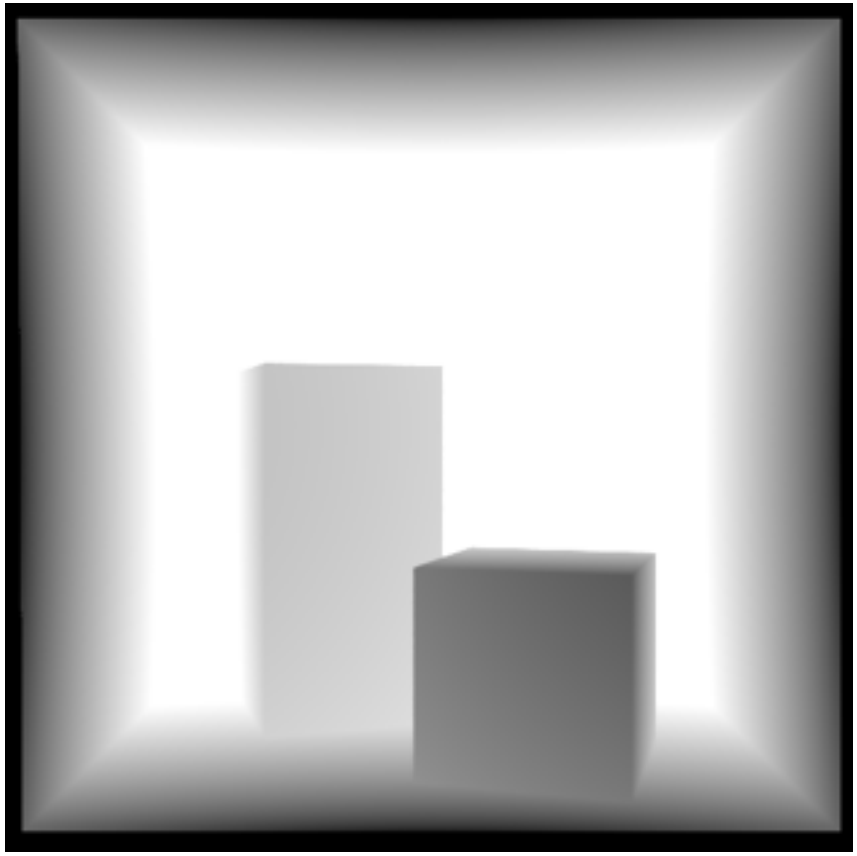


Albedo

Joint NL-Means Filter



100 samples per pixel, 26s



Depth, remapped to [0,1]

Joint NL-Means – Feature Weights

- Feature Distance
  - Assume noise-free features

$$\Phi_j^2(p,q) = \frac{(f_j(p) - f_j(q))^2}{k_f^2 \max(\tau, \|\mathrm{Grad}_j[p]\|^2)}$$

$$k_f = 0.60$$

$$\tau = 1e-3$$

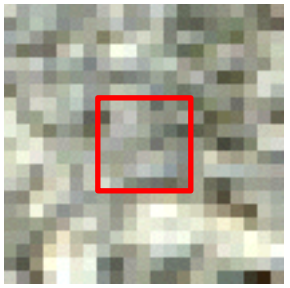
$$d_f^2(p,q) = \underset{j \in [1 \ldots M]}{\mathrm{argmax}}\, \Phi_j^2(p,q)$$

- Pixel-based distance – ***not*** patch-based

$$w_f(p,q) = \exp^{-d_f^2(p,q)}$$

- Joint weight

$$w(p,q) = \min(w_c(p,q), w_f(p,q))$$

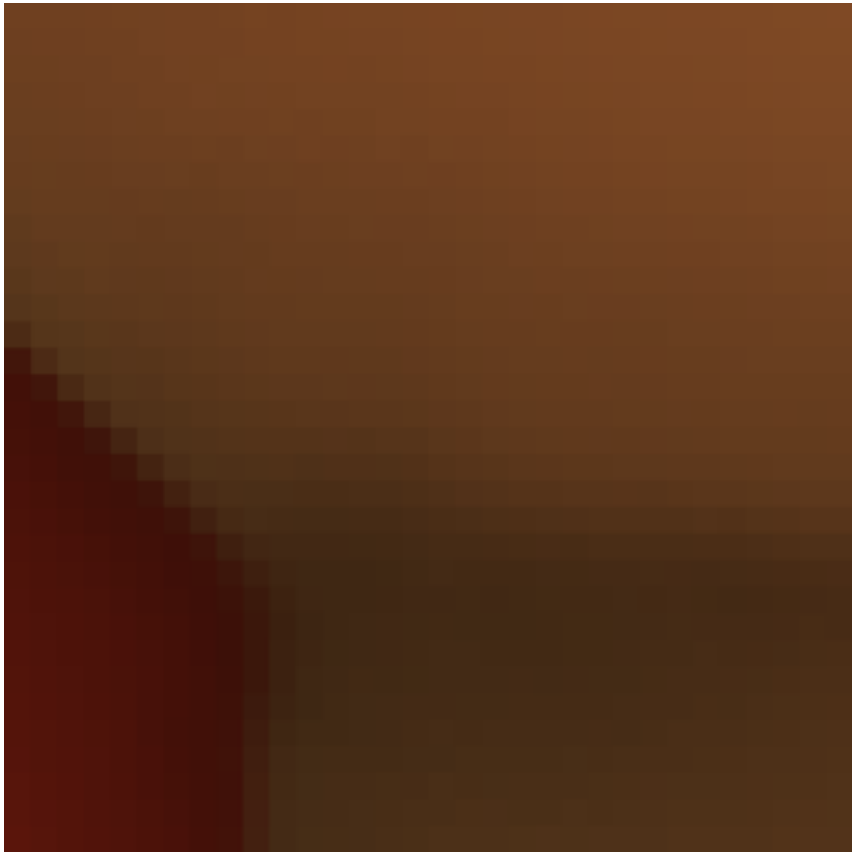


Features Squared Gradient

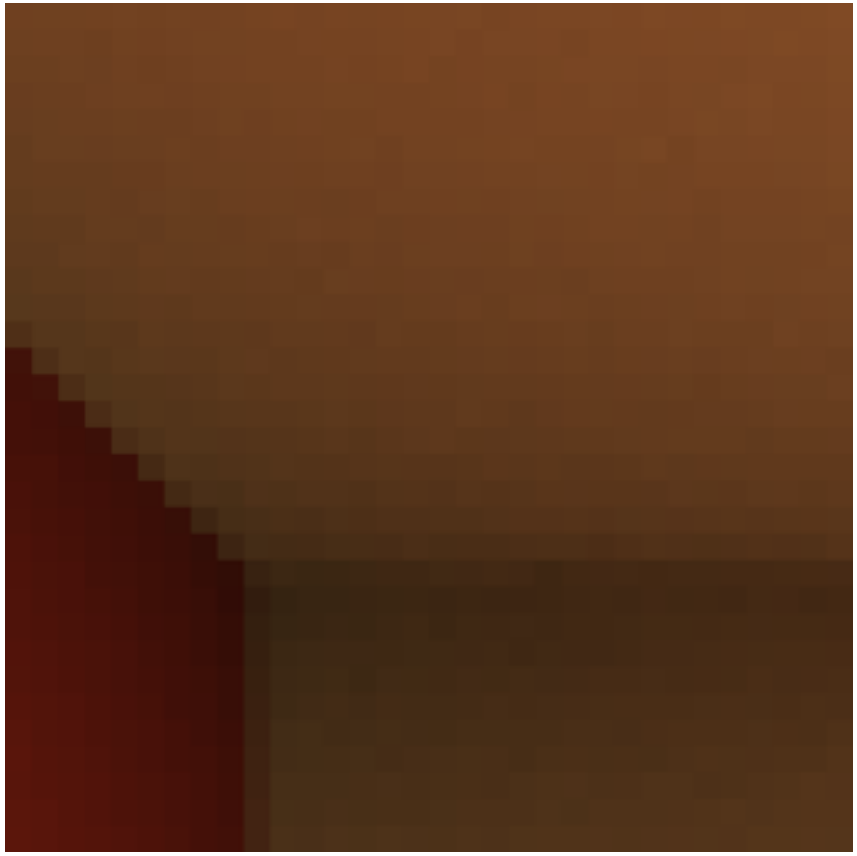
```
INPUT: features (n-dimensional array)
gL = (features - shift(features, [0 -1]))/2
gR = (features - shift(features, [0 +1]))/2
gU = (features - shift(features, [+1 0]))/2
gD = (features - shift(features, [-1 0]))/2
sqrgrad = min(gL^2,gR^2) + min(gU^2,gD^2)
```

NL-Means Filter

NL-Means Denoising



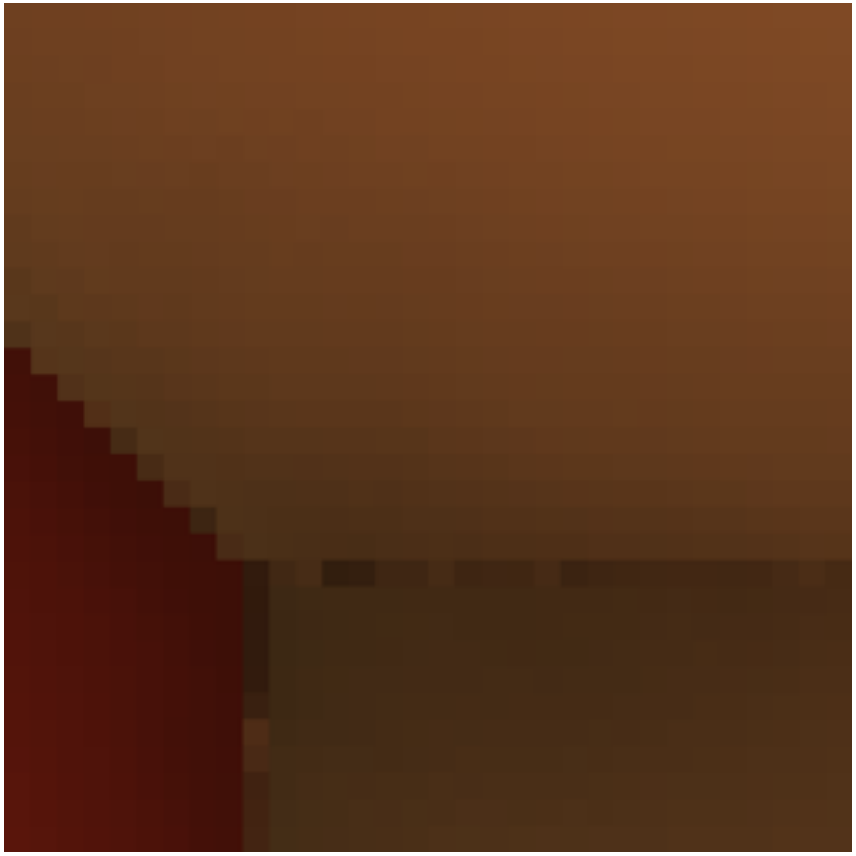
100 samples per pixel
relative MSE: 0.089E-3



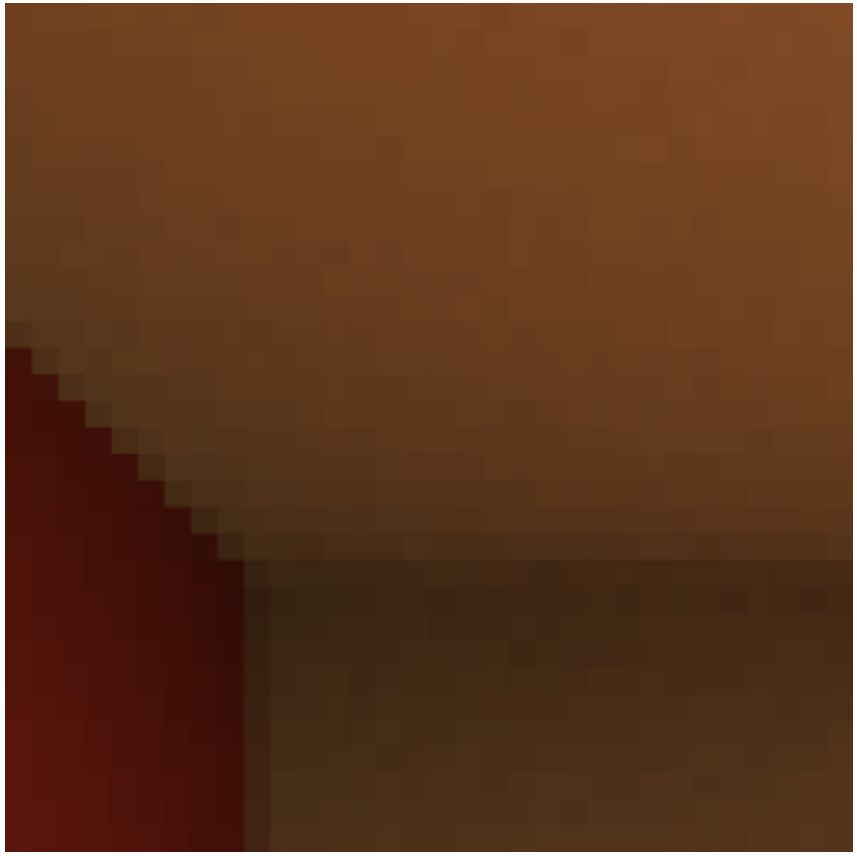
Ground truth (10 000 spp)

Joint NL-Means Filter

Joint NL-Means denoising
Without feature gradients



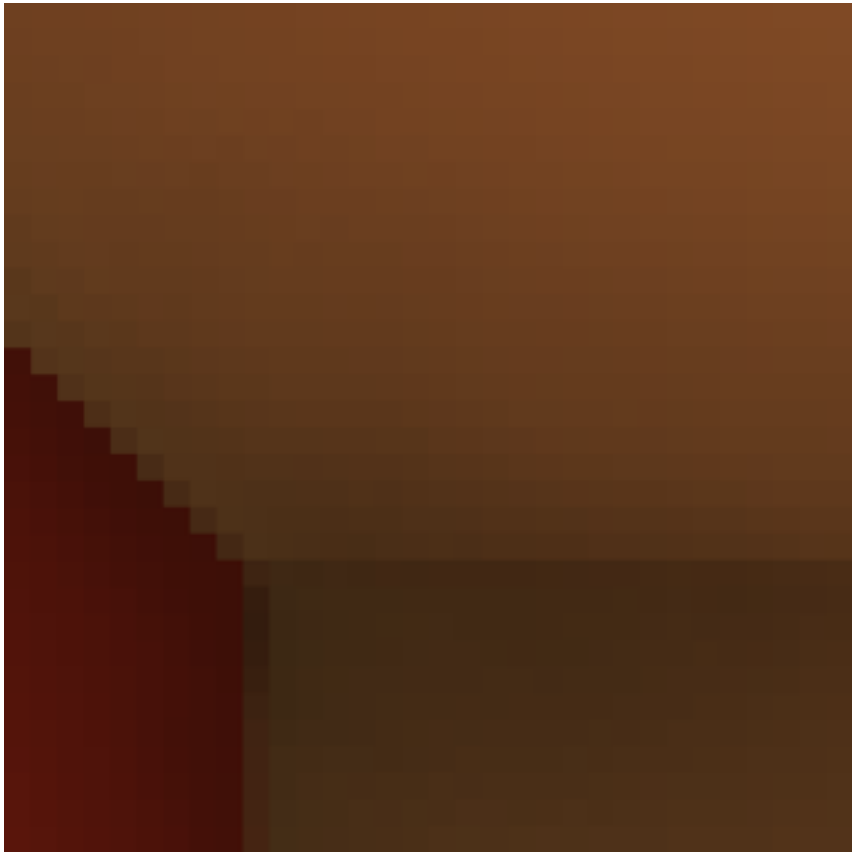
100 samples per pixel
relative MSE: 0.081E-3



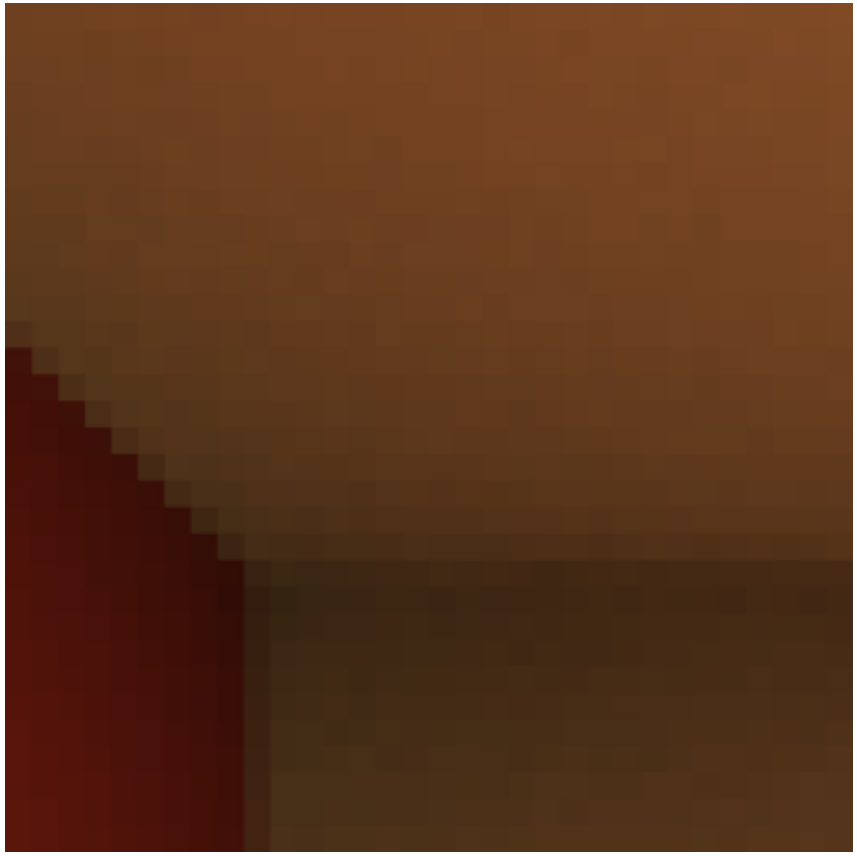
Ground truth (10 000 spp)

Joint NL-Means Filter

Joint NL-Means denoising
With feature gradients

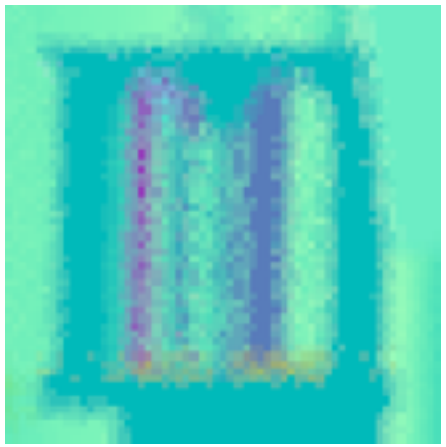


100 samples per pixel
relative MSE: 0.071E-3

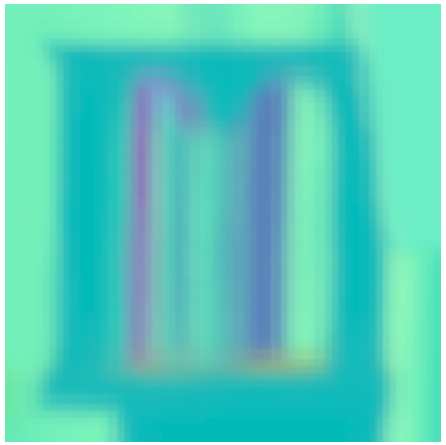


Ground truth (10 000 spp)

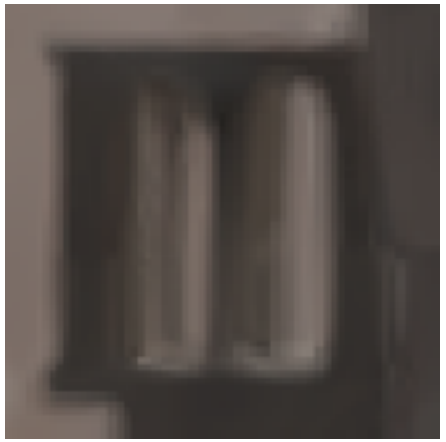
Joint NL-Means – Noisy Features



Normal

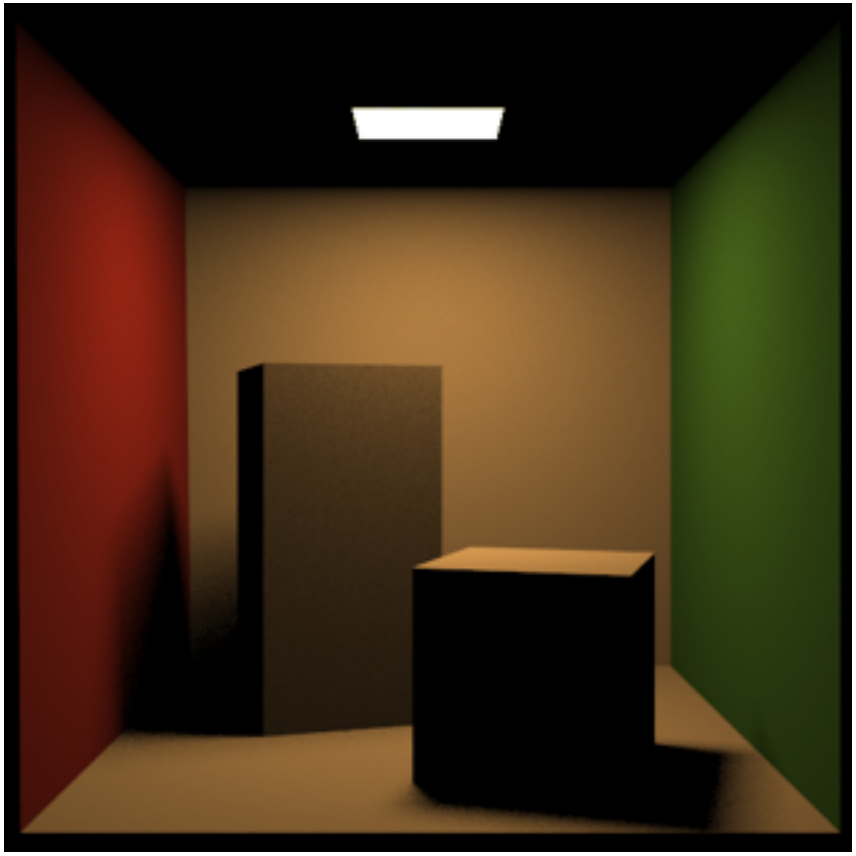


Filtered normal

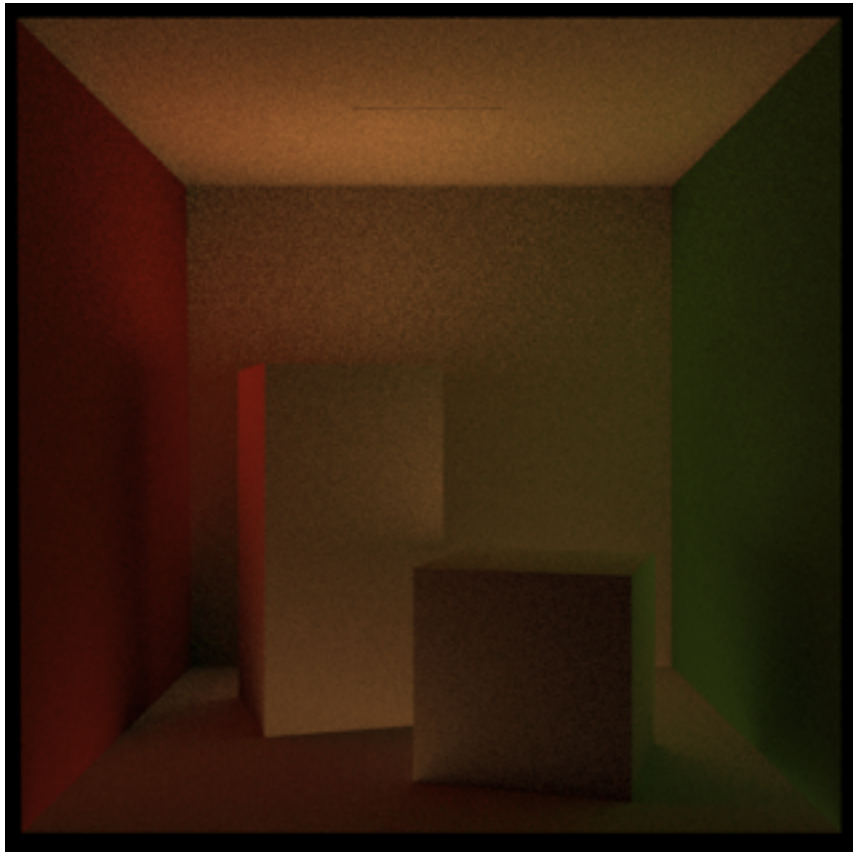


Separate Direct/Indirect Illumination

Direct Illumination



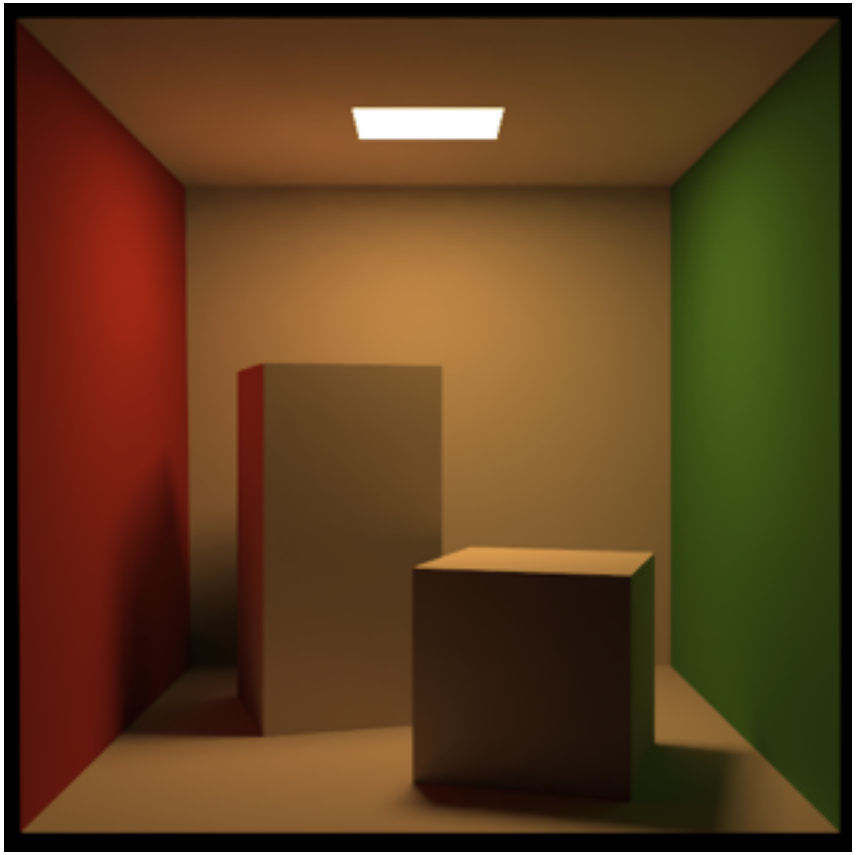
Indirect Illumination



100 samples per pixel
relative MSE: 1.573E-3

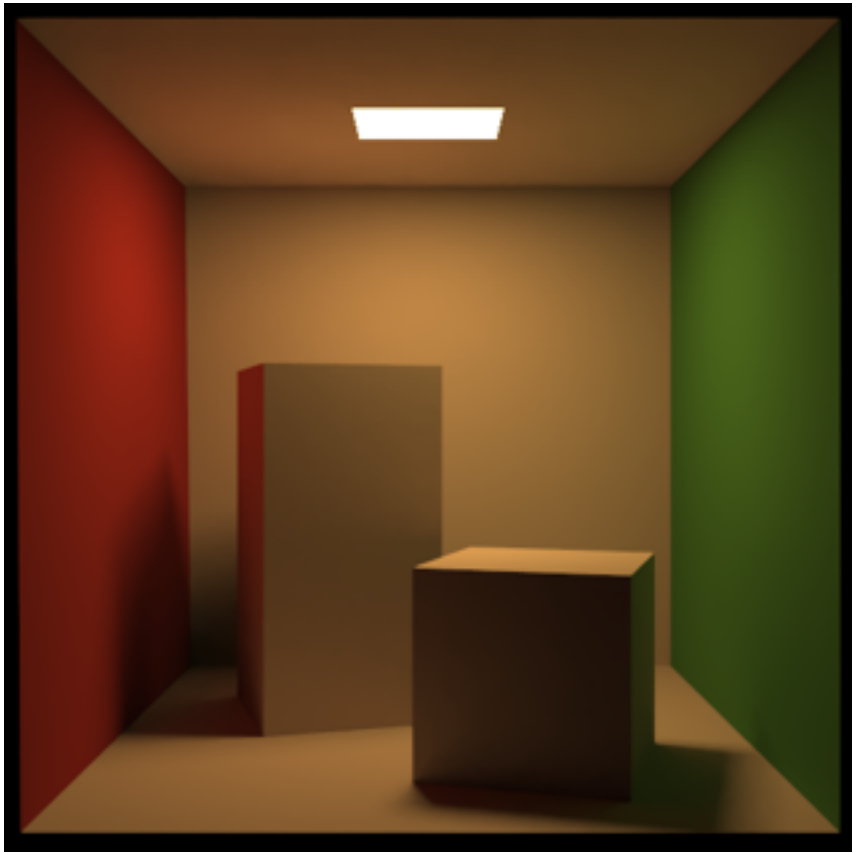
Separate Direct/Indirect Illumination

Final color denoising



100 samples per pixel
relative MSE: 0.071E-3

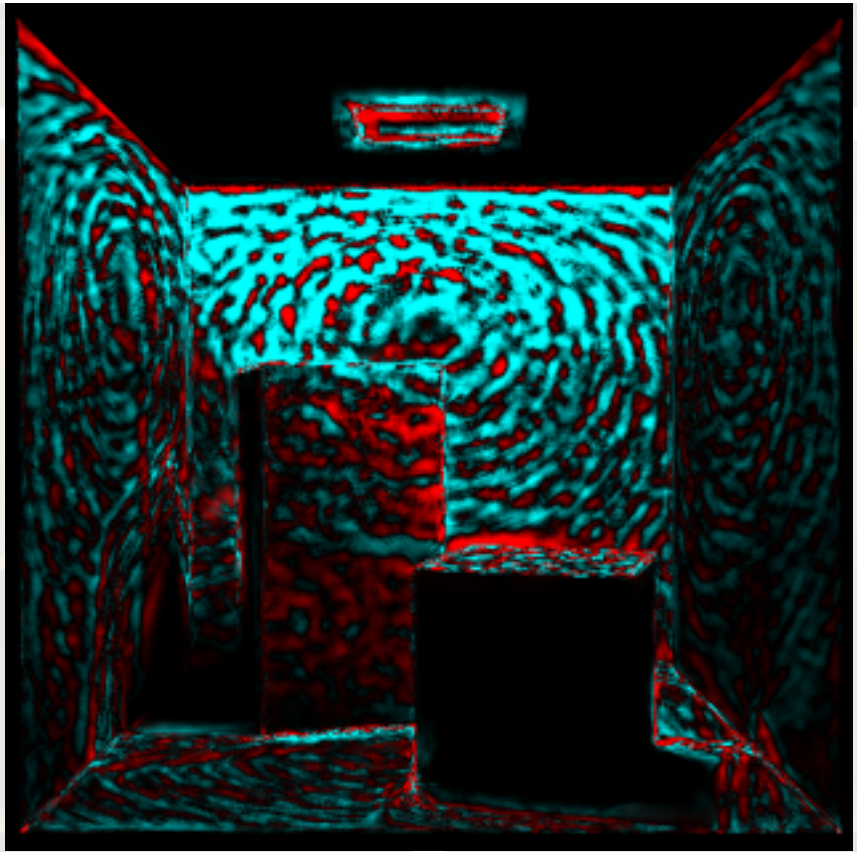
Separate
direct/indirect denoising



100 samples per pixel
relative MSE: 0.059E-3

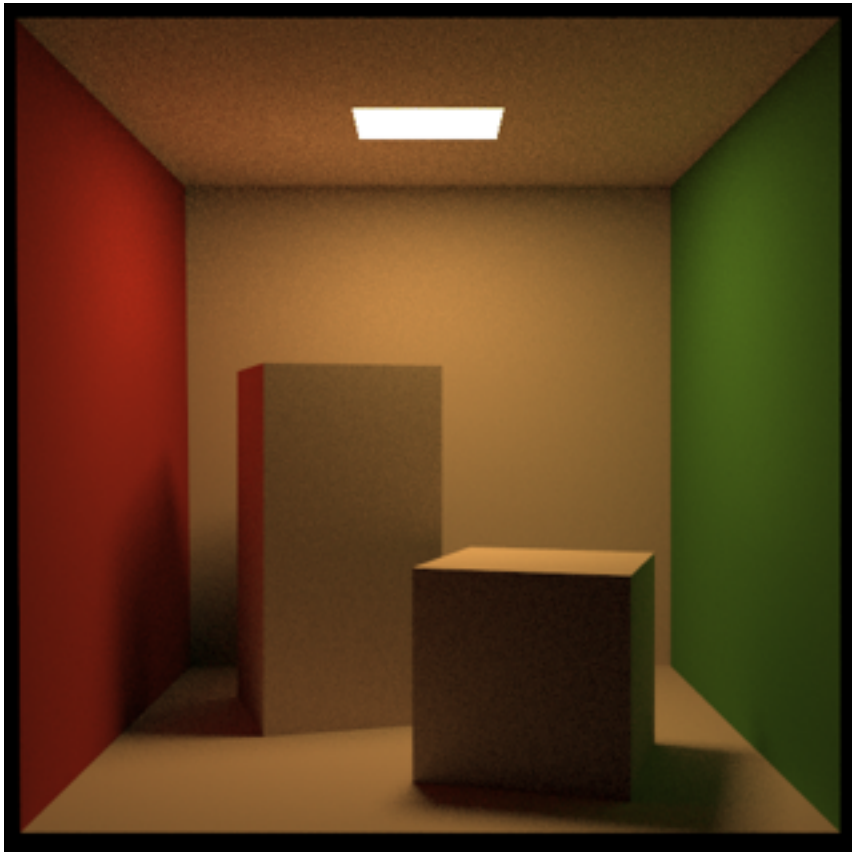
Separate Direct/Indirect Illumination

Difference x1000



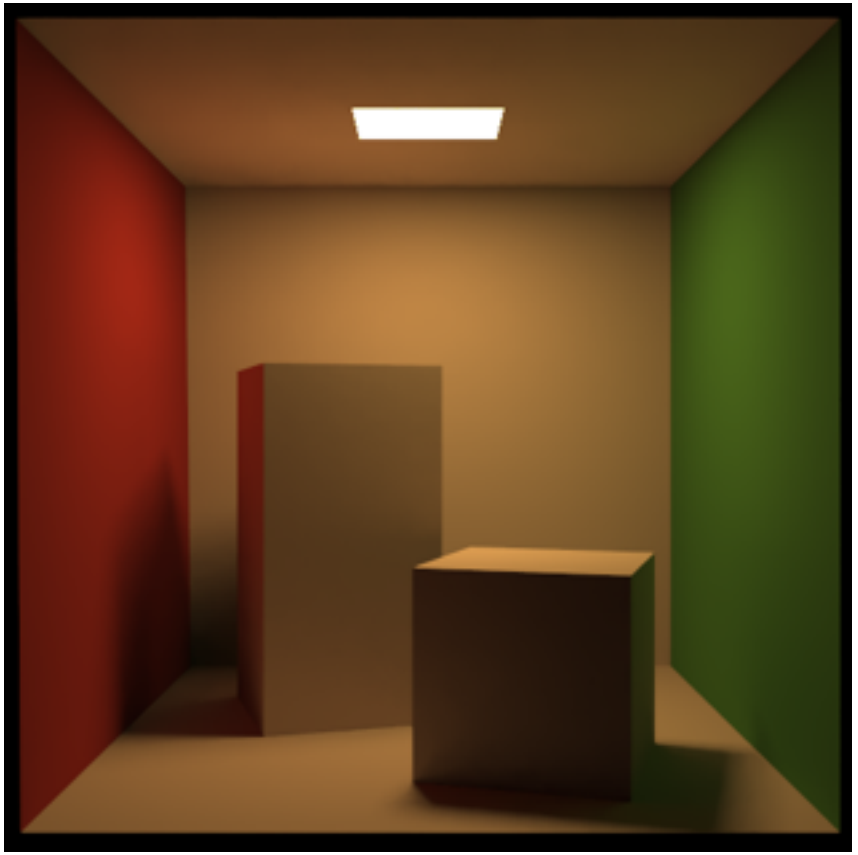
Final Result: 26x reduction of rMSE

Input



100 samples per pixel
relative MSE: 1.573E-3

Denoised



100 samples per pixel
relative MSE: 0.059E-3

Denoising Results

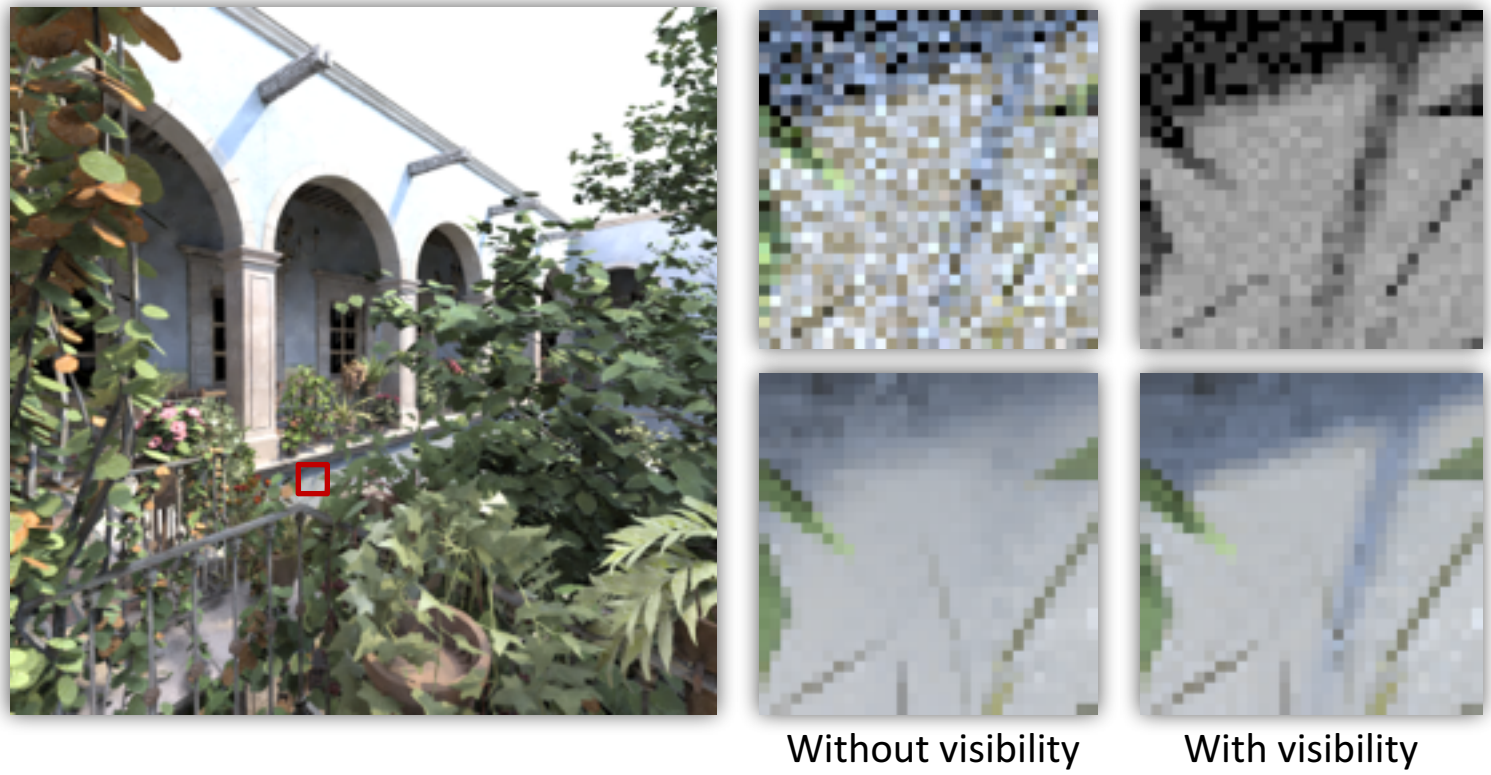
Environment lighting + indirect illumination

Complex geometry

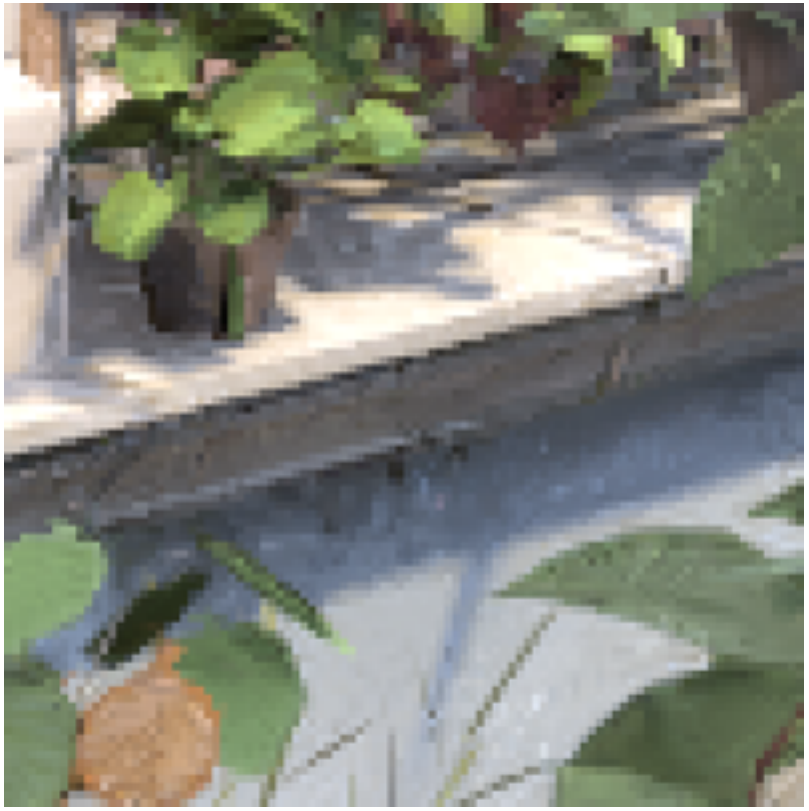
Robustness – new features

Robust Denoising using Features and Color Information

Rousselle et al., Pacific Graphics 2013



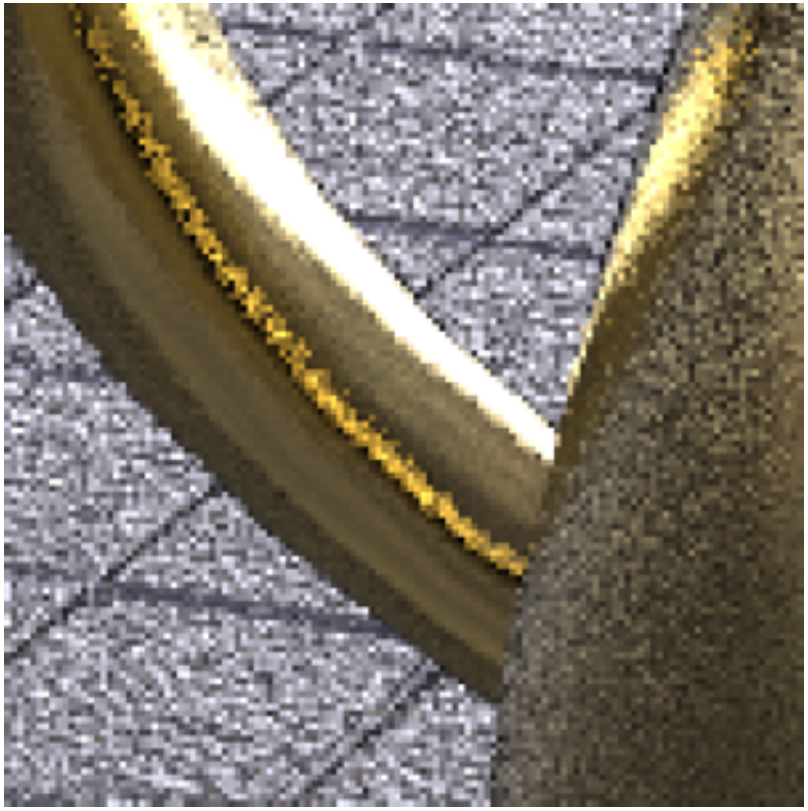
Denoising Results

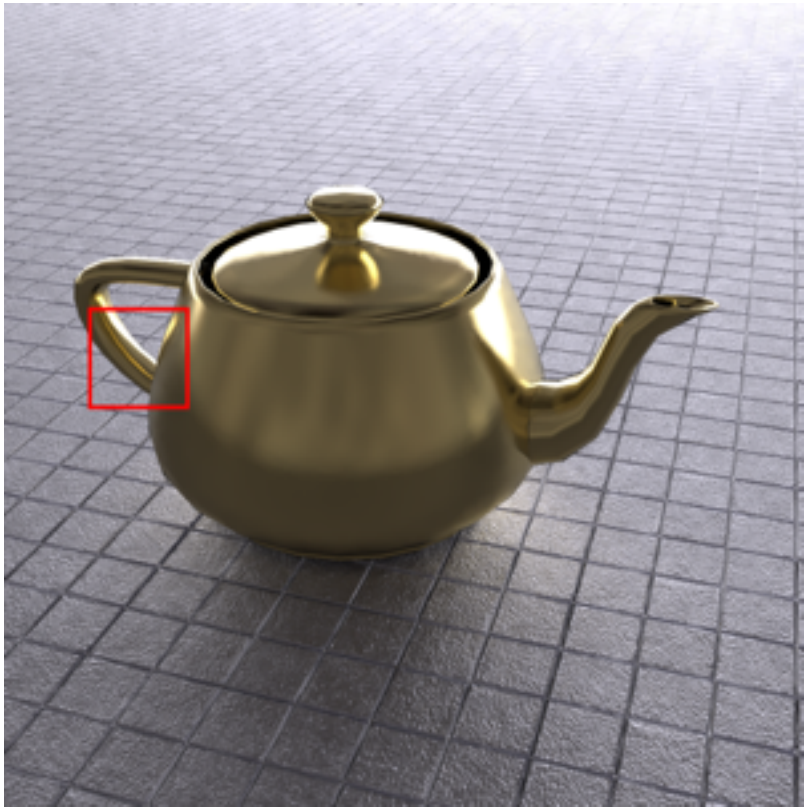


Environment lighting + indirect illumination

Complex geometry

Denoising Results

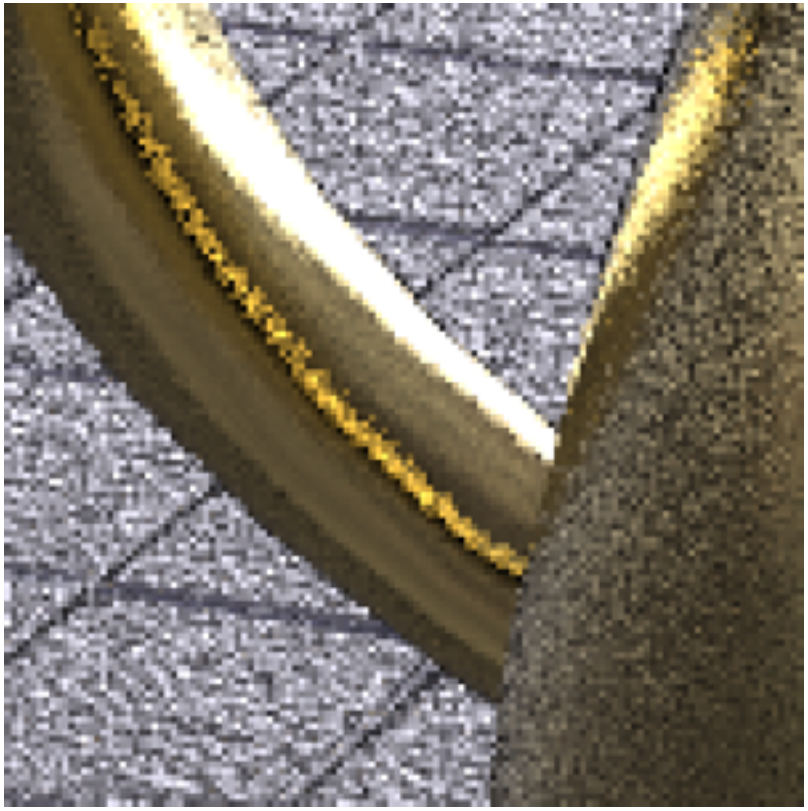


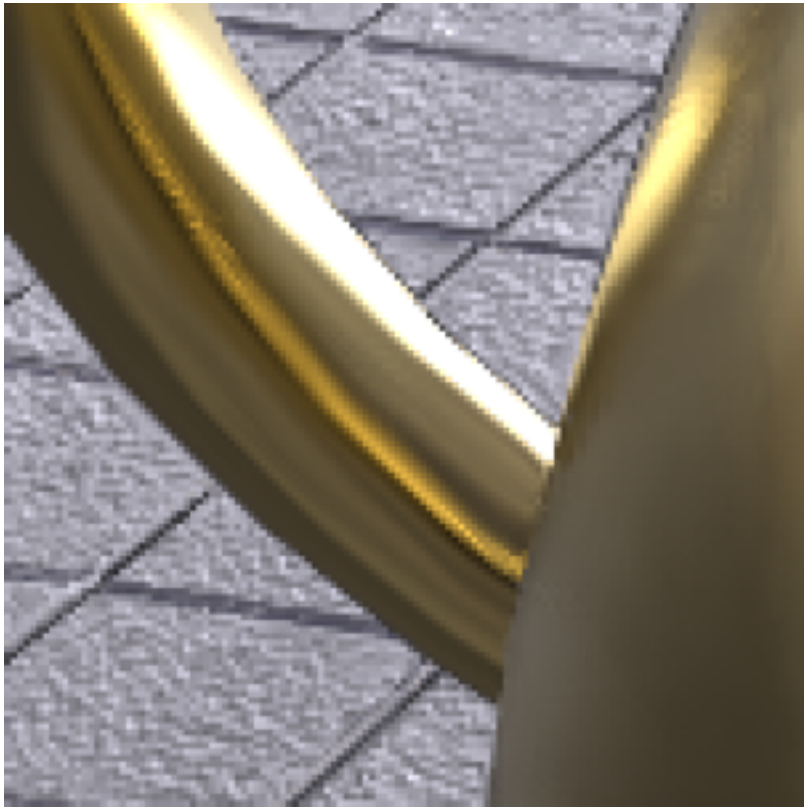


Environment lighting + indirect illumination

Bump map + glossy reflections

Denoising Results

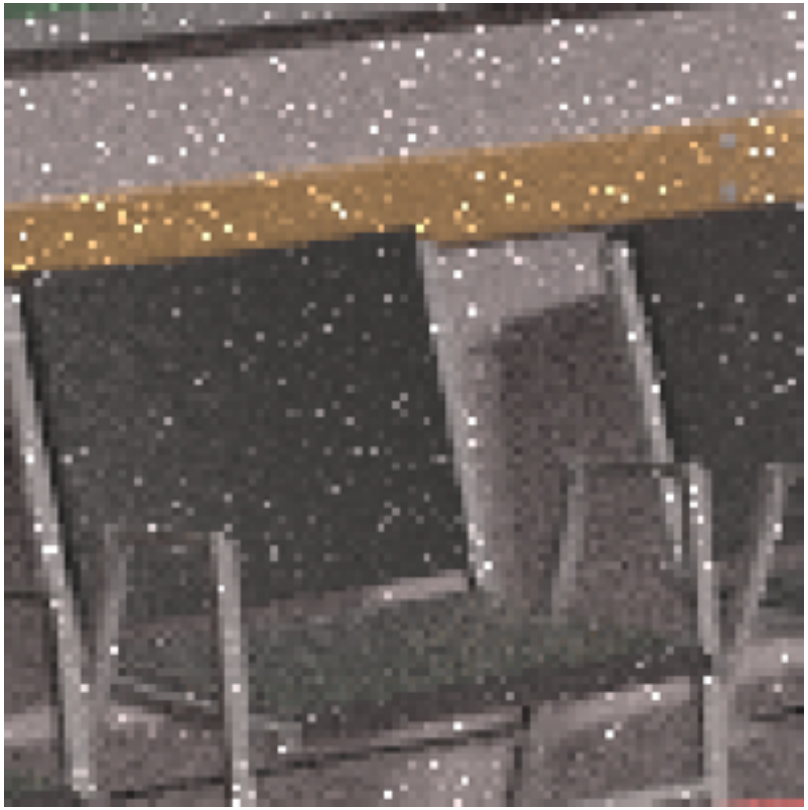


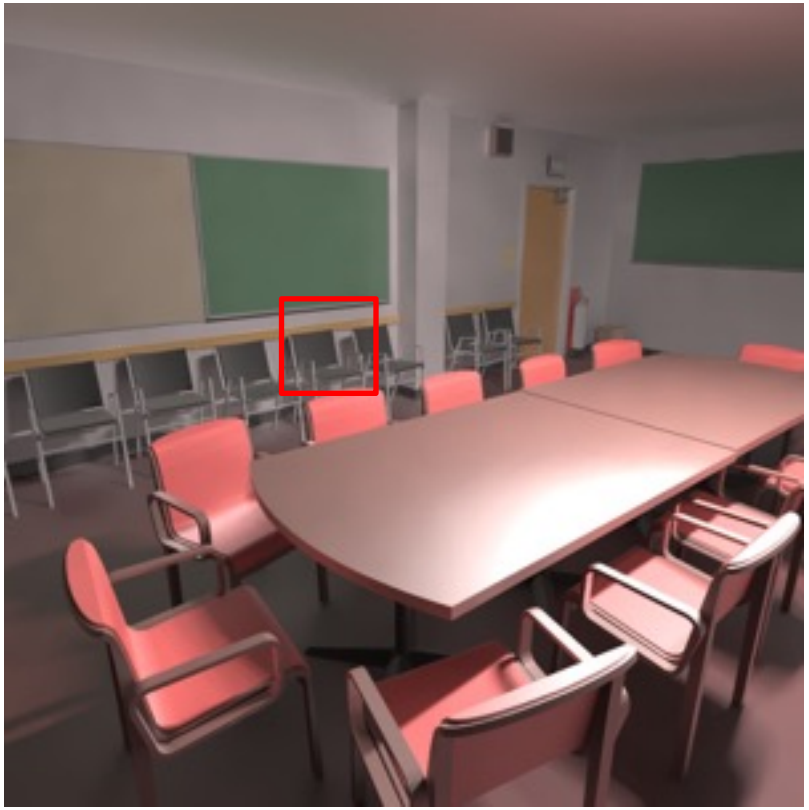


Environment lighting + indirect illumination

Bump map + glossy reflections

Denoising Results

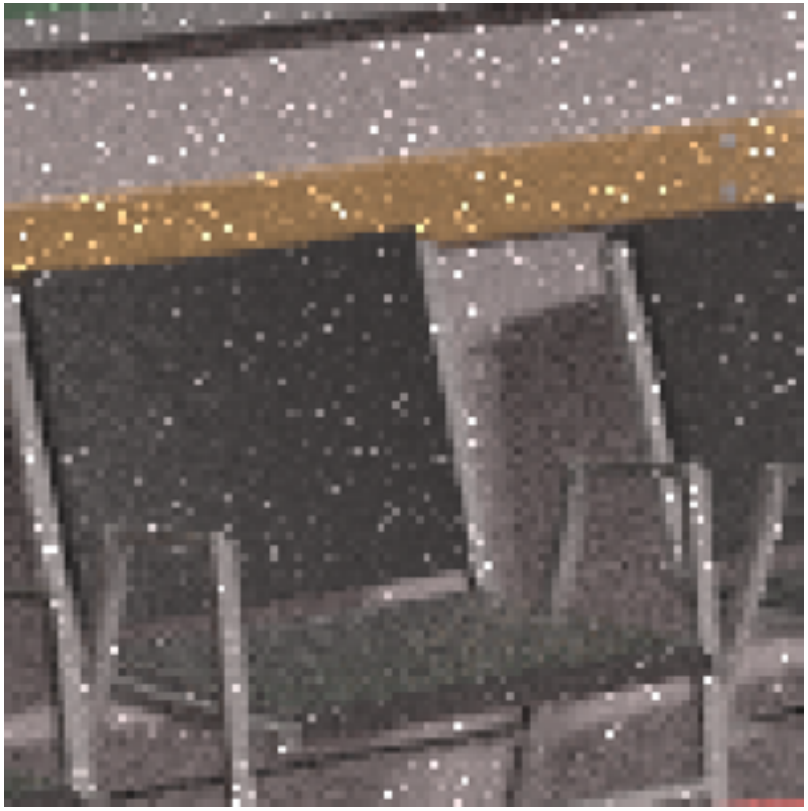




Area lighting + indirect illumination

Spike noise

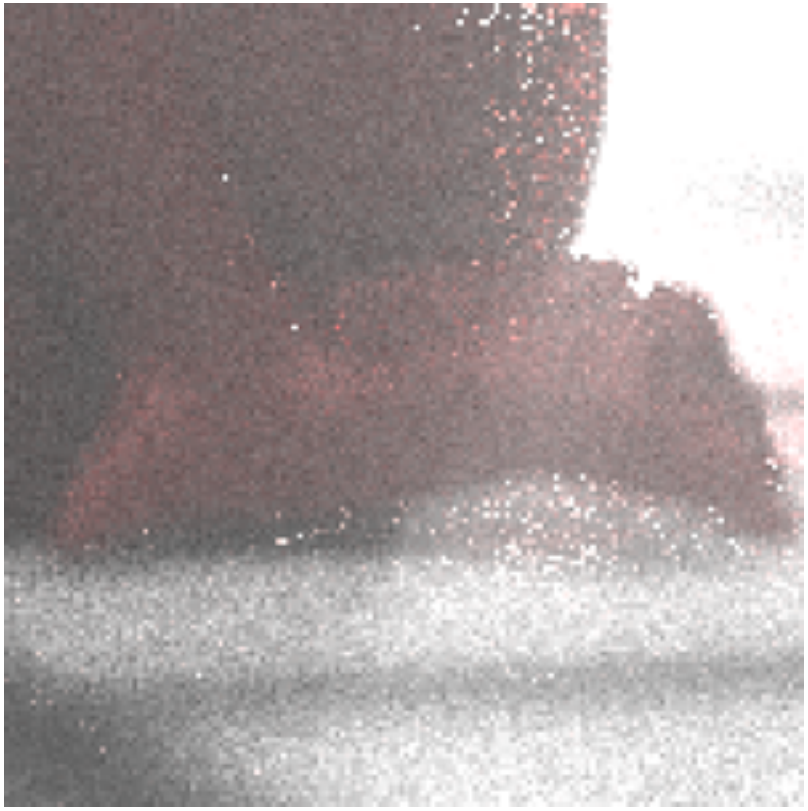
Denoising Results



Area lighting + indirect illumination

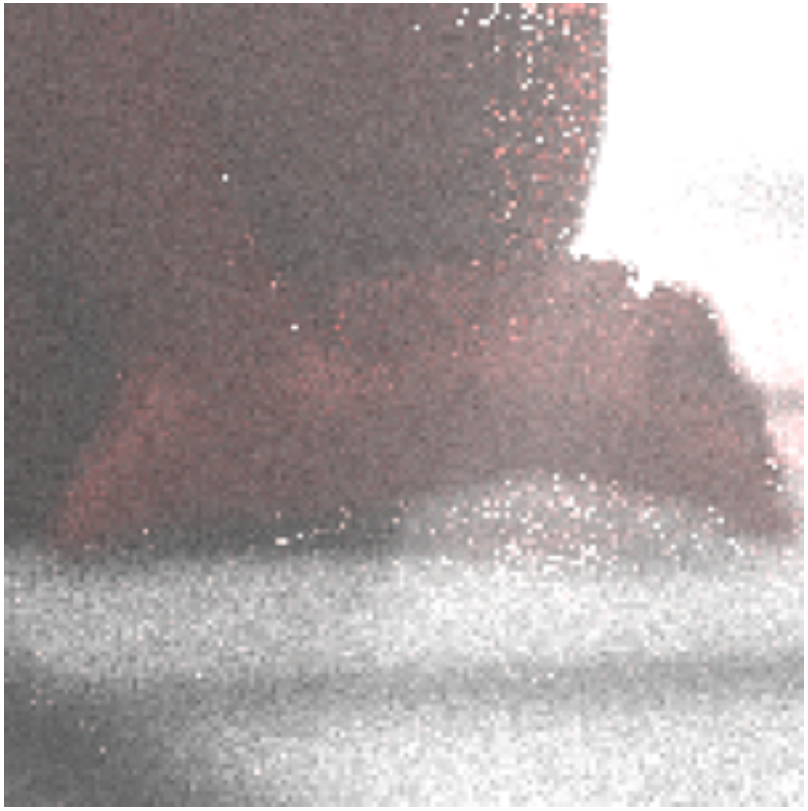
Spike noise

Denoising Results



Area lighting + indirect illumination + depth-of-field + participating media

Denoising Results



Area lighting + indirect illumination + depth-of-field + participating media

Denoising Results – Animation

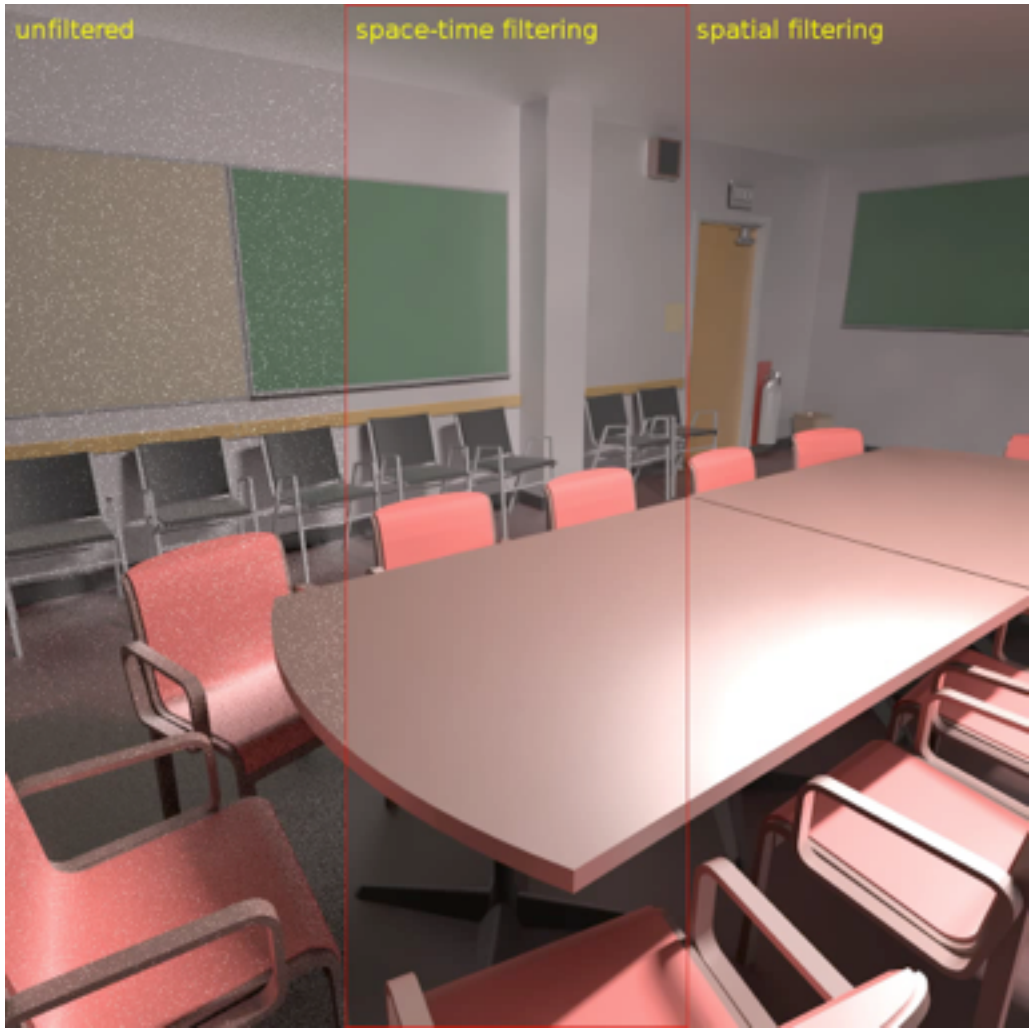


Image-based *a posteriori* methods

- Very effective
  - Preserve the generality of MC rendering
  - Low computational overhead
  - Moderately intrusive for the renderer
- Limitations
  - Residual low-frequency noise (use spatio-temporal denoising)
  - Not robust at low sampling rates
  - Cannot handle conflicting constraints (motion blur in front of a static object)

NL-Means filter – production use