Propagated Image Filtering

CVPR: Jen-Hao Rick Chang, Yu-Chiang Frank Wang

Team : Rebooting

Report by: Aayush Tiwari(201531031)
Nakul Vaidya(201501108)
Vidit Jain(201501021)
Guide : Dr. Avinash Sharma
TA : Aaron Varghese

Introduction:

Image filtering is a process of updating pixel values in an image to achieve particular goals like denoising, smoothing,enhancement, or matting. It typically requires the extraction of particular image characteristics, while undesirable patterns like noise or irrelevant textural regions need to be disregarded. If cross-region mixing occurs during the filtering process, i.e., the characteristics of adjacent image regions are blended, the output image would contain blurry regions which result in degraded visual quality. Bilateral and guided filters are popular edge-preserving image filters. However, these filters require predefined pixel neighborhood regions (via spatial functions or kernels), which are typically difficult to determine beforehand. Geodesic filters which dynamically determines their filtering kernels suffer from cross-region mixing in smoothing and other filtering tasks. Hence we need a filter for solving the above tasks.

Aim of the paper:

To come up with a filter that is able to observe and preserve image characteristics without the need to apply explicit spatial kernel functions. Cross-region mixing is a scenario where the characteristics of adjacent image regions are blended, the output image would contain blurry regions which result in degraded visual quality.So our filter also handles the problem of cross-region mixing.

Previous Works:

Bilateral Filtering

- A bilateral filter is a non-linear, edge-preserving, and noise-reducing smoothing filter for images.
- Bilateral filter replaces the intensity of each pixel with a weighted average of intensity values from nearby pixels.
- This weight can be based on a Gaussian distribution. Crucially, the weights depend not only on Euclidean distance of pixels, but also on the radiometric differences (e.g., range differences, such as color intensity, depth distance, etc.). This preserves sharp edges.

$$I'_s = \frac{1}{Z_s} \sum_{t \in \Omega} g(d_{\mathrm{BF}}(s,t); \sigma_s)\, g(d_{\mathrm{BF}}(I_s, I_t); \sigma_r)\, I_t,$$

Guided Filtering

- The guided filter computes the filtering output by considering the content of a guidance image, which can be the input image itself or another different image.
- The guided filter can be used as an edge-preserving smoothing operator like the popular bilateral filter, but it has better behaviors near edges.
- The output image can be represented as $I'_s = \sum_{t \in \mathcal{N}(s)} w^g_{s,t} I_t$ where,

$$w^g_{s,t} = \frac{1}{|W|^2} \sum_{k \in \{k | s,t \in W_k\}} \left(1 + \frac{\left(I^g_s - \mu_k\right)\left(I^g_t - \mu_k\right)}{\sigma^2_k + \epsilon}\right).$$

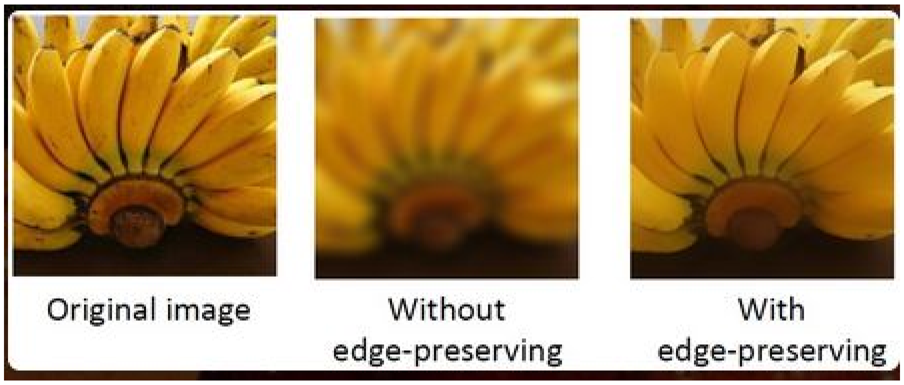
Geodesic Filtering

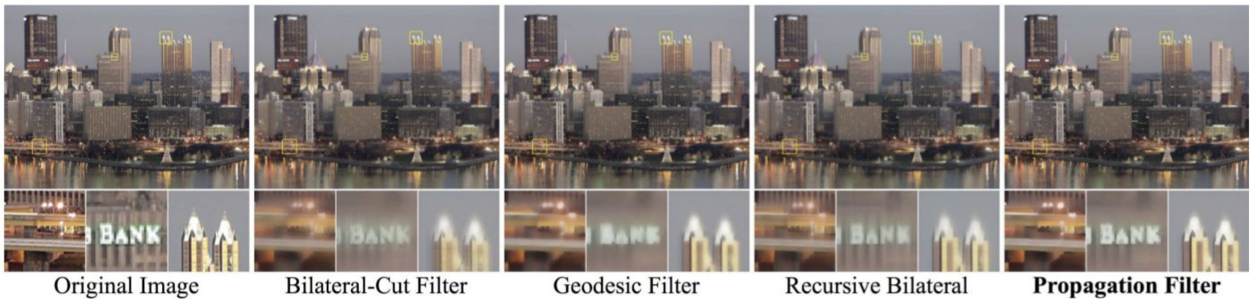
- Geodesic filtering utilizes only photometric distances during the filtering process.
- It calculates the output at pixels by $I'_s = \frac{1}{Z_s} \sum_{t \in \mathcal{N}(s)} g(d_{\mathrm{GF}}(I_s, I_t) ; \sigma_r) \, I_t$
  Where Z_s is the normalization factor, and N(s) contains neighboring pixels of s.
- The photometric distance between pixels s and t is defined as

$$d_{\mathrm{GF}}(I_s, I_t) = \min_{\phi} \sum_{x,\, x+1 \in \phi} \|I_{x+1} - I_x\|$$

Where ∅ is the path connecting pixels s and t.

What to Achieve:





Propagated Filter:

- Propagated filter suppresses undesirable information from adjacent or neighboring pixels during filtering by aiming at taking the context information between image pixels into consideration.

- Without using any explicit spatial functions, propagated filter is formed from a probabilistic point of view.
- Propagation filter essentially cooperates the merits of bilateral and geodesic filtering, while comparable computation costs can be obtained.

One-Dimensional Case:

- Given an input image I , the filtered output I_s' at pixel s produced by our propagation filter is calculated as shown in the formula.

The output pixels s is given as

$$I_s' = \frac{1}{Z_s} \sum_{t \in \mathcal{N}(s)} w_{s,t} \, I_t,$$

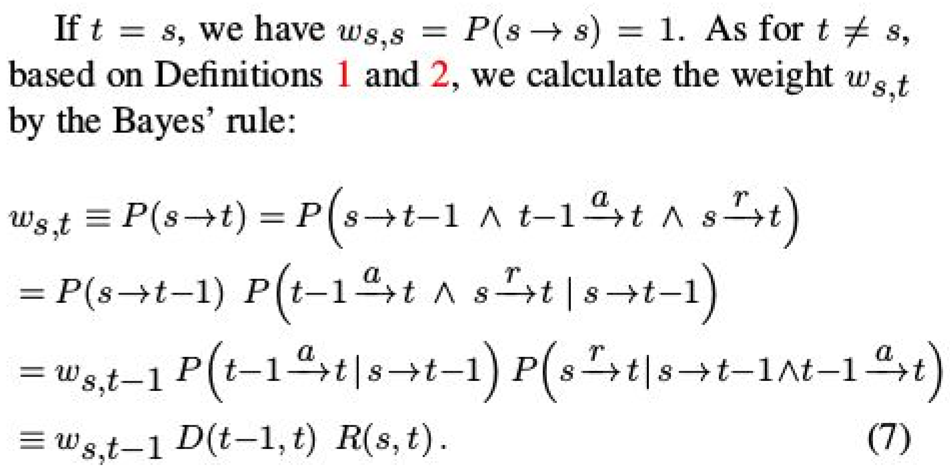
where $Z_s = \sum_{t \in \mathcal{N}(s)} w_{s,t}$ as a normalized factor

Finding $w_{s,t}$:

- Definition 1
  - Suppose there are n singly connected pixels 1, . . . , n , in which s and t are two pixels satisfying $1 \le s < t \le n$ without the loss of generality. Pixel t is related to pixel s , or $s \to t$, if and only if $s \to t-1$, $t-1 \xrightarrow{a} t$, and $s \xrightarrow{r} t$. In addition, each pixel is always self-related, i.e., $s \to s$.

- For pixel “t” being related to pixels, the intermediate pixels betweens and “t” not only need to be photometrically related to “s”, they are also required to be adjacent photometrically related to their predecessors.

- Definition 2
    - Suppose there are n singly connected pixels, 1, . . . , n , and pixels s and t satisfying 1 ≤ s ≤ t ≤ n . The weight ws,t for filtering pixel s with pixel t is the probability value of t being related to s , i.e., P (s → t) .



If $t = s$, we have $w_{s,s} = P(s \to s) = 1$. As for $t \neq s$, based on Definitions 1 and 2, we calculate the weight $w_{s,t}$ by the Bayes’ rule:

$$\begin{aligned}
w_{s,t} &\equiv P(s \to t) = P\left(s \to t-1 \ \wedge \ t-1 \xrightarrow{a} t \ \wedge \ s \xrightarrow{r} t\right) \\
&= P(s \to t-1) \ P\left(t-1 \xrightarrow{a} t \ \wedge \ s \xrightarrow{r} t \mid s \to t-1\right) \\
&= w_{s,t-1} \ P\left(t-1 \xrightarrow{a} t \mid s \to t-1\right) P\left(s \xrightarrow{r} t \mid s \to t-1 \wedge t-1 \xrightarrow{a} t\right) \\
&\equiv w_{s,t-1} \ D(t-1,t) \ R(s,t) . \qquad (7)
\end{aligned}$$

If t=s, we have ws, s=P(s→s) = 1.

As for “t!=s”, based on Definitions 1 and 2, we calculate the weight “ws”, “t” by the Bayes’ rule:

$$\begin{aligned}
w_{s,t} &\equiv P(s \to t) = P\left(s \to t-1 \ \wedge \ t-1 \xrightarrow{a} t \ \wedge \ s \xrightarrow{r} t\right) \\
&= P(s \to t-1) \ P\left(t-1 \xrightarrow{a} t \ \wedge \ s \xrightarrow{r} t \mid s \to t-1\right) \\
&= w_{s,t-1} \ P\left(t-1 \xrightarrow{a} t \mid s \to t-1\right) P\left(s \xrightarrow{r} t \mid s \to t-1 \wedge t-1 \xrightarrow{a} t\right)
\end{aligned}$$

$$\equiv w_{s,t-1} \ D(t-1,t) \ R(s,t) .$$

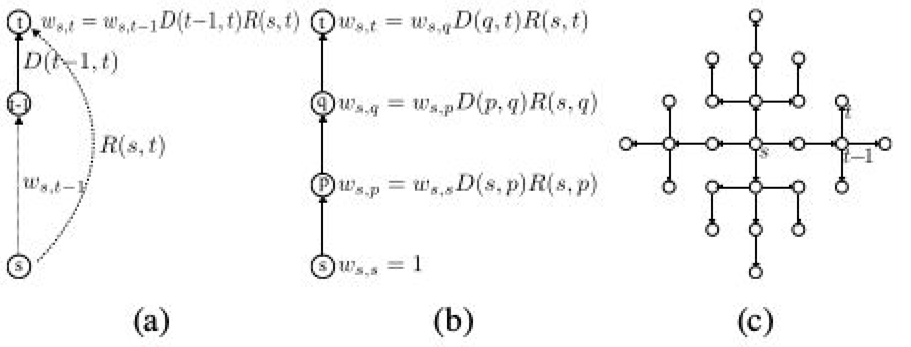

$w_{s,t} = w_{s,t-1}D(t-1,t)R(s,t)$
$D(t-1,t)$
$R(s,t)$
$w_{s,t-1}$
$w_{s,t} = w_{s,q}D(q,t)R(s,t)$
$w_{s,q} = w_{s,p}D(p,q)R(s,q)$
$w_{s,p} = w_{s,s}D(s,p)R(s,p)$
$w_{s,s} = 1$
(a)
(b)
(c)

Figure 2: Illustration of propagation filtering. (a) the definition of filtering weight $w_{s,t}$, (b) the calculation of $w_{s,t}$, and (c) the pattern for performing 2D filtering with $d = 3$ pixels.

R(s, t) is measured as the adjacent-photometric relationship between s and t . As a result, we define

$$R(x,y) = g(\|I_x - I_y\|; \sigma_r) = \exp\left(\frac{-\|I_x - I_y\|^2}{2\sigma_r^2}\right)$$

For simplicity, we choose $\sigma_a = \sigma_r$, and thus D(·) = R(·) .

The probability value of two adjacent pixels being photometric related is proportional to the value of a Gaussian function of their pixel value difference, we define

$$D(x,y) = g(\|I_x - I_y\|; \sigma_a) = \exp\left(\frac{-\|I_x - I_y\|^2}{2\sigma_a^2}\right)$$

where I_x is the value of pixel x , and $\| I_x - I_y \|$ measures the Euclidean distance between the corresponding pixel value difference.

Comparison to Bilateral and Geodesic Filtering:

In propagation filters the filter weights are derived from a probability point of view. the filtering weights $w_{s,t}$ regarding the pixel distances as:

$$w_{s,t} = g\left(d^{a}_{\mathrm{PF}}(Is, I_t);\ \sigma a\right)\ g\left(d^{r}_{\mathrm{PF}}(Is, I_t);\ \sigma r\right)$$

$$d^{a}_{\mathrm{PF}}(Is, I_t) = \sqrt{\sum_{x,x+1\in\phi} \|I_{x+1} - Ix\|^2}$$

$$d^{r}_{\mathrm{PF}}(Is, I_t) = \sqrt{\sum_{x\in\phi} \|Ix - Is\|^2},$$

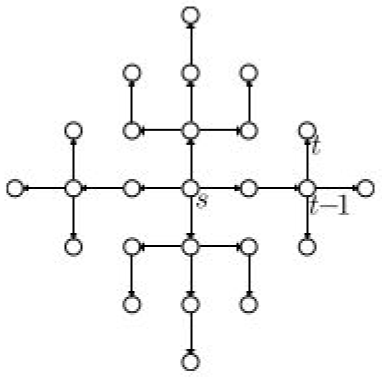
Note that ϕ is the path connecting pixels s and t . The propagation filter inherits merits from both bilateral and geodesic filters while compensating their deficiencies.

- Instead of using explicit spatial kernels, we have $d^{a}_{PF}(I_s, I_t)$ utilize intermediate adjacent pixel information between s and t , which is similar to the photometric distances applied in geodesic filtering and recursive bilateral filtering.
- The deficiency of geodesic filters is complemented by $d^{r}_{PF}(I_s, I_t)$, which measures the photometric distance between pixels along the path of interest.
- Geodesic filters cannot suppress effects from nearby image regions well, especially when only negligible noise is observed in those regions, Instead, our propagation filter is able to observe large $d^{a}_{PF}(I_s, I_t)$ and $d^{r}_{PF}(I_s, I_t)$ and have $w_{s,q}<w_{s,p}$.
- This would alleviate cross-region mixing problems, and this is the reason why propagation filters better discriminate between image regions with different context information.

The Two-Dimensional Case:

For image filtering, we now extend the above process to two-dimensional scenarios

- Given pixels s and t in an image, we need to first determine a path connecting s and t for deriving the weight $w_{s,t}$ accordingly.
- We consider a particular 2D pattern for filtering all pixels in an image.



- It can be seen that, the path would be a straight line connecting pixels s and t, if these two pixels are horizontally or vertically aligned. If pixels s and t are not simply horizontally or vertically connected, we determine the path based on their Manhattan distance.
- If the Manhattan distance between s and t is an odd number, we choose the path for traversing from that pixel to its predecessor in the vertical direction; otherwise, the horizontal path will be chosen.
- By using this 2D pattern for filtering, the computation complexity of our propagation filter will be reduced to O(NW).

Propagation Filtering as a Robust Estimator:

Our proposed propagation filter can be viewed as a one-step estimator, which aims at minimizing the error between the filtered and desirable outputs we first transform our filtering algorithm of into the following formulation:

$$I'_s = \frac{1}{Z_s} \sum_{t \in \mathcal{N}(s)} w_{s,t}\, I_t = I_s - \frac{1}{Z_s} \sum_{t \in \mathcal{N}(s)} w_{s,t}\,(I_s - I_t).$$

The above equation is effectively a gradient descent solver optimizing the objective function 'f', whose gradient at pixel 's' is derived as:

$$\nabla f = \sum_{t \in \mathcal{N}(s)} w_{s,t}\; (I_s - I_t)\, .$$

With this gradient solver, the optimization problem can be recovered as:

$$\min_I \sum_{s \in \Omega} \sum_{t \in \mathcal{N}(s)} \int w_{s,t}\; (I_s - I_t)\;\, \mathrm{d}(I_s - I_t),$$

Where Ω denotes the pixel set of the input image.
our propagation filter assumes that related pixels exhibit similar pixel values.Therefore, we apply $w_{s,t}$ as the belief $P(I_s - I_t)$ and have $\int w_{s,t}(I_s - I_t)\, d(I_s - I_t) = \int P(s \to t)\, (I_s - I_t)\, d(I_s - I_t) \equiv \int P(I_s - I_t)\, (I_s - I_t)\, d(I_s - I_t)$, which indicates the expectation of the difference between pixels 's' and 't' with probability $P(s \to t)$. Now,

$$\min_I \sum_{s \in \Omega} \sum_{t \in \mathcal{N}(s)} \mathrm{E}\,[I_s - I_t]\, .$$

Propagation Filtering as Belief Propagation:

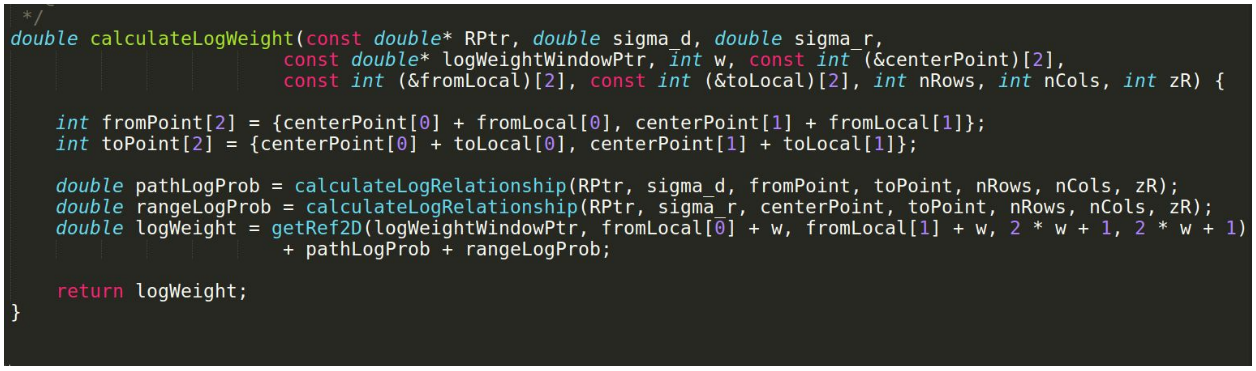
Propagation filtering can be viewed as belief propagation with a Bayesian network. Let X_t as the random variable representing that pixel t is related to pixels. A Bayesian network can be constructed using our proposed 2D pattern, which reflects the dependency between its nodes. By Definition, we have s and $t-1$ as the parent nodes of t, and $t+1$ as its child node. Since there only exists a single directed path from t to $t+1$, the probability $P(X_t)$ can be expressed as $P(X_t)=P(X_s)P(X_{t-1}|X_s)P(X_t|X_s \wedge X_{t-1})$, where $P(X_s)$ and $P(X_{t-1}|X_s)$ indicate the beliefs of X_s and X_{t-1}. We now show that $P(X_t)$ is effectively the weight $w_{s,t}$ as determined in our propagation filter. According to Definition 2, since pixel s is to be filtered, we have $P(X_s) = w_{s,s} = 1$ and $P(X_{t-1}|X_s) = P(X_{t-1}) = w_{s,t-1}$. Based on the same definition, we calculate $P(X_t|X_s, X_{t-1})$ by $P(t-1 \xrightarrow{a} t \wedge s \xrightarrow{r} t | s \rightarrow t-1)$, and thus $P(X_t) = w_{s,t-1} P(t-1 \xrightarrow{a} t \wedge s \xrightarrow{r} t | s \rightarrow t-1)$. From the derivations, we have $P(X_t) = w_{s,t}$, which verifies that our propagation filtering can be regarded as the belief propagation polytree algorithm.

Speedup

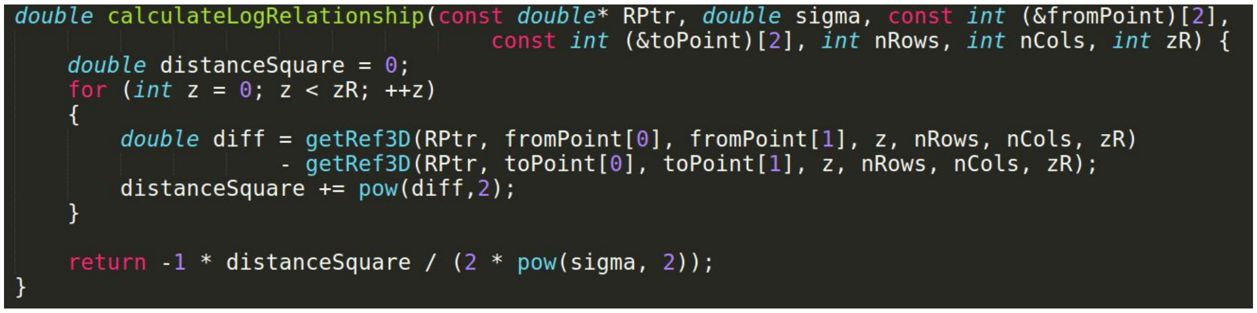
In addition to parallelization processing by multi-core techniques, further acceleration for propagation filtering can be achieved by early cutoff in the derivation of the filter weights.

Implementation Details

For finding the D(x, y) we instead find log(D(x, y)) to avoid the overflow problem and later we will take the exponential of the sum of log Weights.

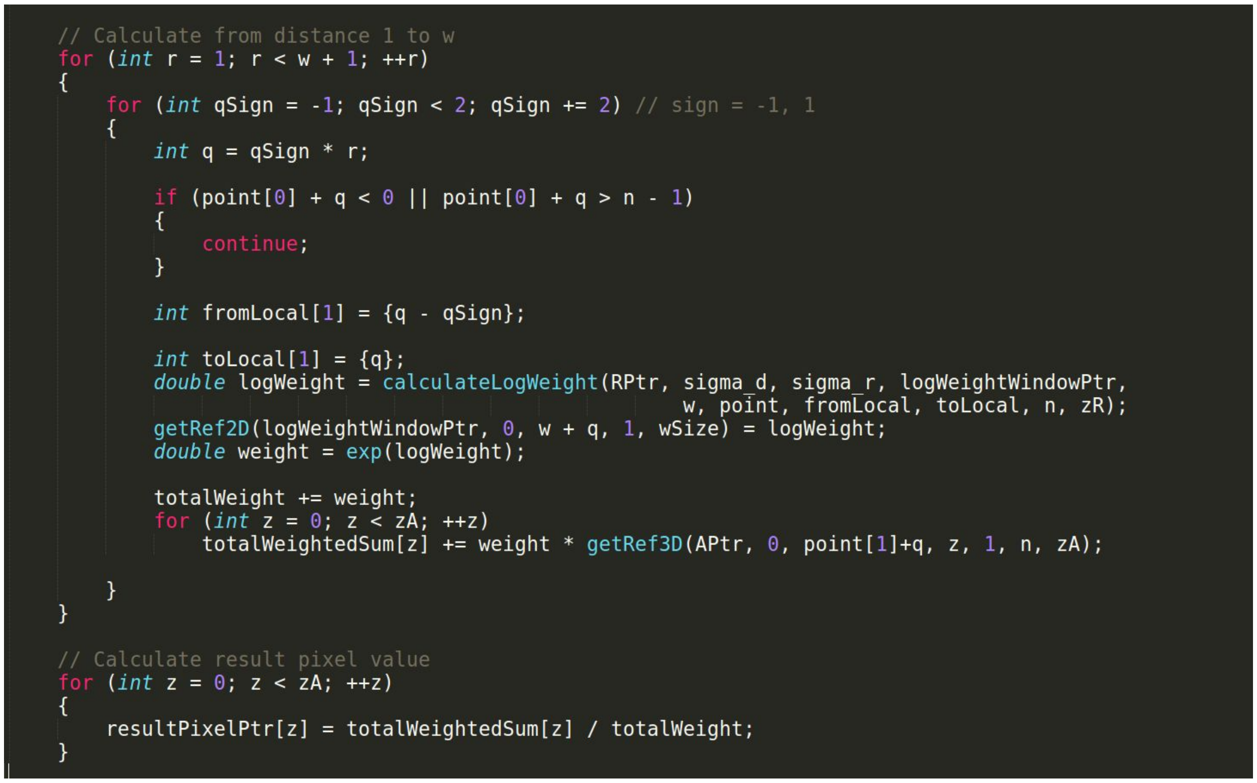


```
 */
double calculateLogWeight(const double* RPtr, double sigma_d, double sigma_r,
                        const double* logWeightWindowPtr, int w, const int (&centerPoint)[2],
                        const int (&fromLocal)[2], const int (&toLocal)[2], int nRows, int nCols, int zR) {

    int fromPoint[2] = {centerPoint[0] + fromLocal[0], centerPoint[1] + fromLocal[1]};
    int toPoint[2] = {centerPoint[0] + toLocal[0], centerPoint[1] + toLocal[1]};

    double pathLogProb = calculateLogRelationship(RPtr, sigma_d, fromPoint, toPoint, nRows, nCols, zR);
    double rangeLogProb = calculateLogRelationship(RPtr, sigma_r, centerPoint, toPoint, nRows, nCols, zR);
    double logWeight = getRef2D(logWeightWindowPtr, fromLocal[0] + w, fromLocal[1] + w, 2 * w + 1, 2 * w + 1)
                        + pathLogProb + rangeLogProb;

    return logWeight;
}
```



```
double calculateLogRelationship(const double* RPtr, double sigma, const int (&fromPoint)[2],
                                const int (&toPoint)[2], int nRows, int nCols, int zR) {
    double distanceSquare = 0;
    for (int z = 0; z < zR; ++z)
    {
        double diff = getRef3D(RPtr, fromPoint[0], fromPoint[1], z, nRows, nCols, zR)
                    - getRef3D(RPtr, toPoint[0], toPoint[1], z, nRows, nCols, zR);
        distanceSquare += pow(diff,2);
    }

    return -1 * distanceSquare / (2 * pow(sigma, 2));
}
```

Finding new pixel value in 1D case



```
    // Calculate from distance 1 to w
    for (int r = 1; r < w + 1; ++r)
    {
        for (int qSign = -1; qSign < 2; qSign += 2) // sign = -1, 1
        {
            int q = qSign * r;

            if (point[0] + q < 0 || point[0] + q > n - 1)
            {
                continue;
            }

            int fromLocal[1] = {q - qSign};

            int toLocal[1] = {q};
            double logWeight = calculateLogWeight(RPtr, sigma_d, sigma_r, logWeightWindowPtr,
                                                  w, point, fromLocal, toLocal, n, zR);
            getRef2D(logWeightWindowPtr, 0, w + q, 1, wSize) = logWeight;
            double weight = exp(logWeight);

            totalWeight += weight;
            for (int z = 0; z < zA; ++z)
                totalWeightedSum[z] += weight * getRef3D(APtr, 0, point[1]+q, z, 1, n, zA);

        }
    }

    // Calculate result pixel value
    for (int z = 0; z < zA; ++z)
    {
        resultPixelPtr[z] = totalWeightedSum[z] / totalWeight;
    }
```

Applications and Experimental Results:

Denoising:

Original Images

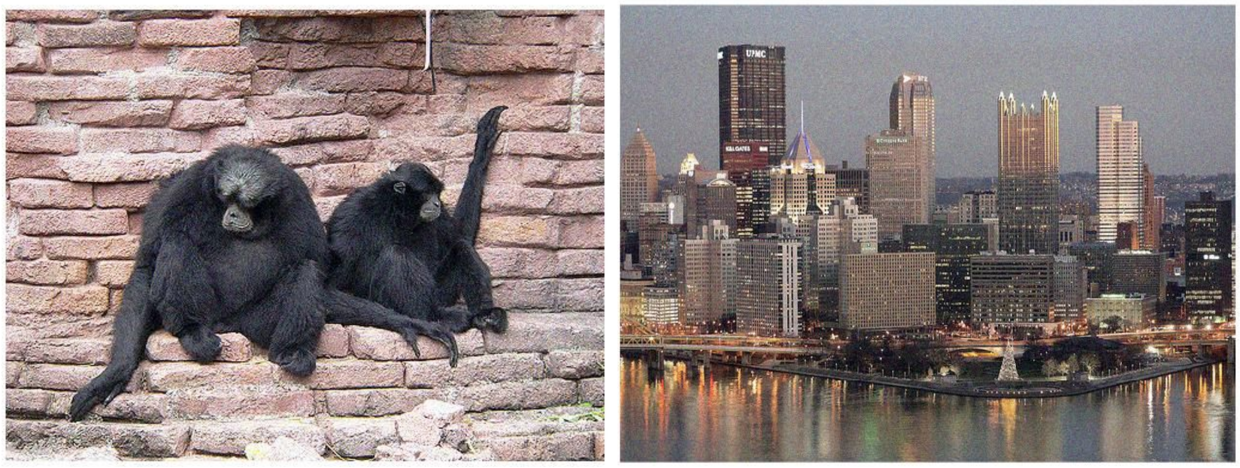


Image after adding gaussian white noise with standard deviation 0.05

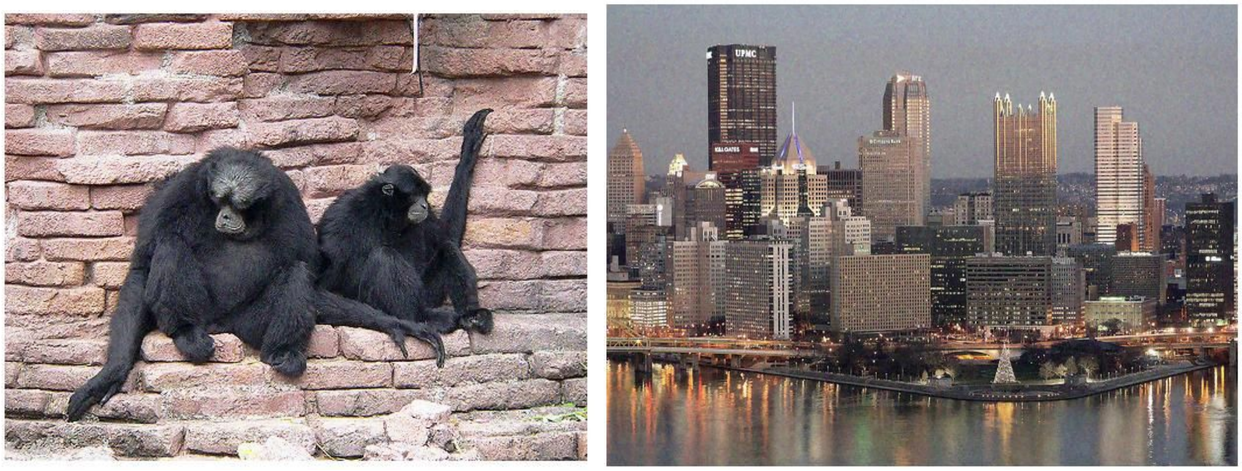


Image after Denoising through bilateral filter ($\sigma_s = 5, \sigma_r = 40$)

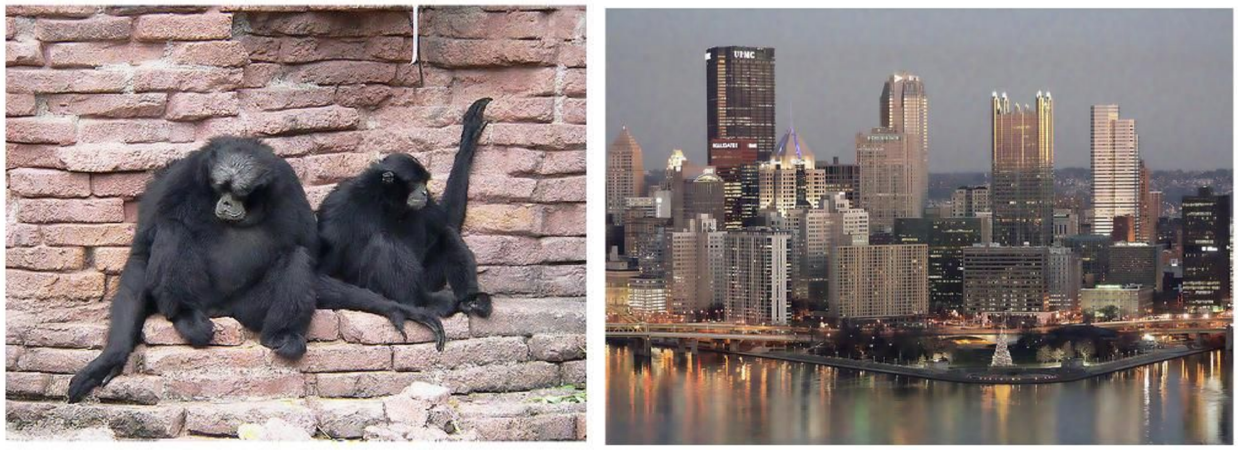


Image after Denoising through guided filter with window size 5x5

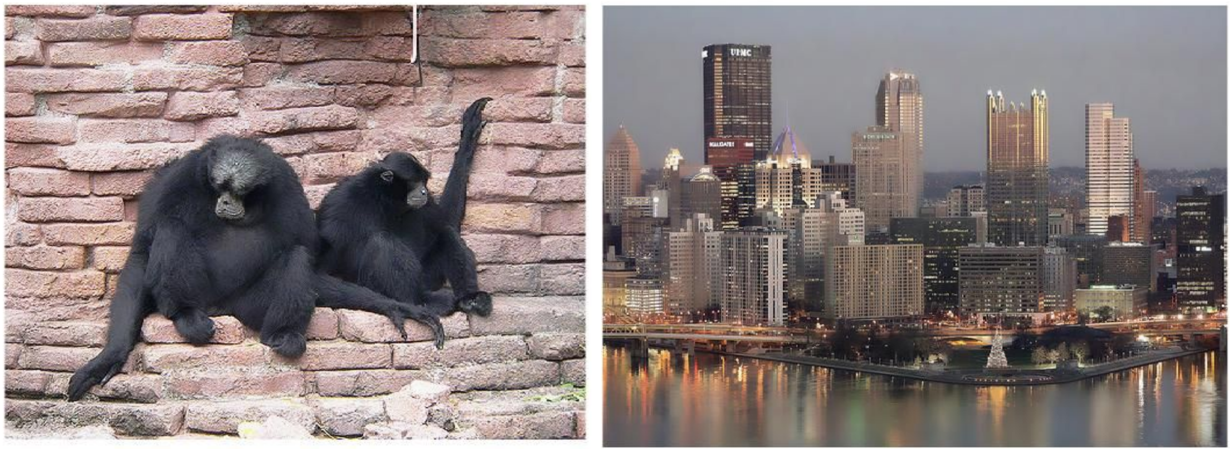


Image after Denoising through guided filter with window size 10x10

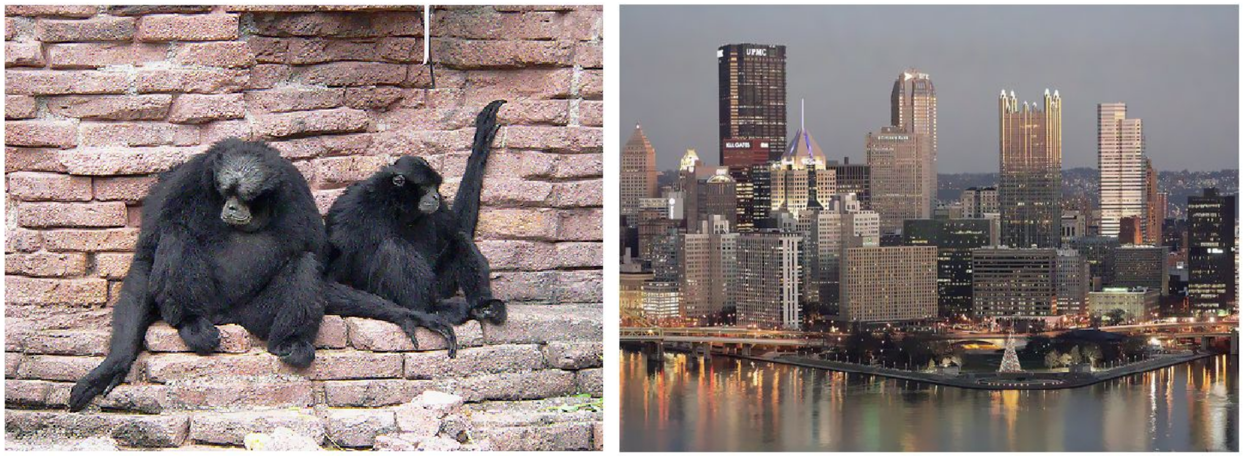


Image after Denoising through Propagated Image Filter with window size 5x5

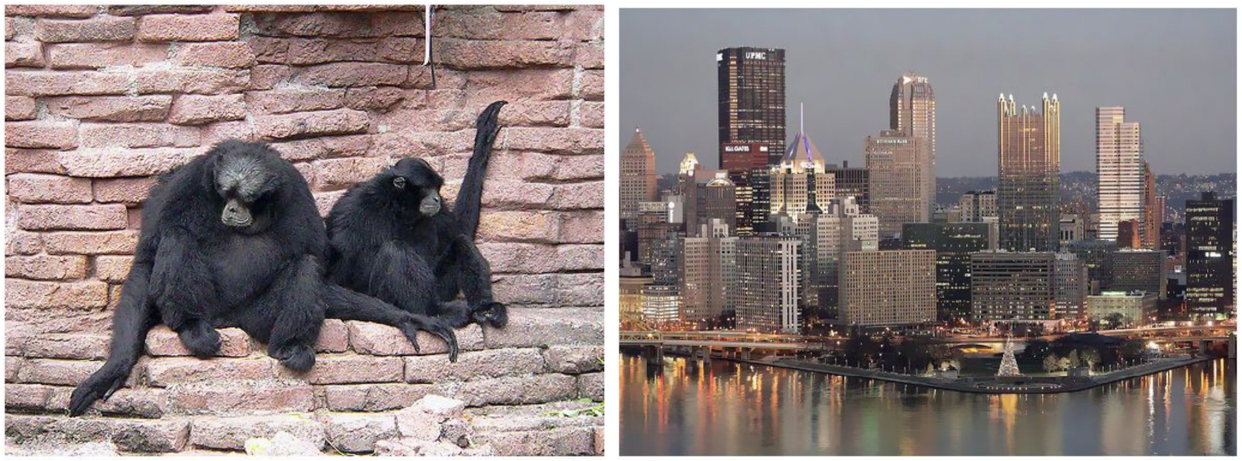
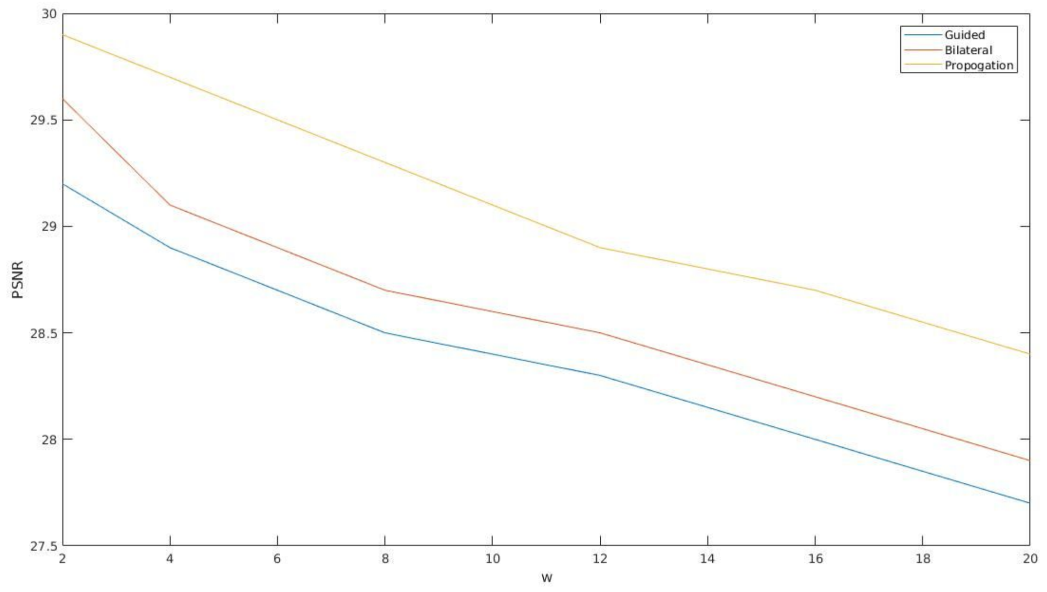


Image after Denoising through Propagated Image Filter with window size 10x10

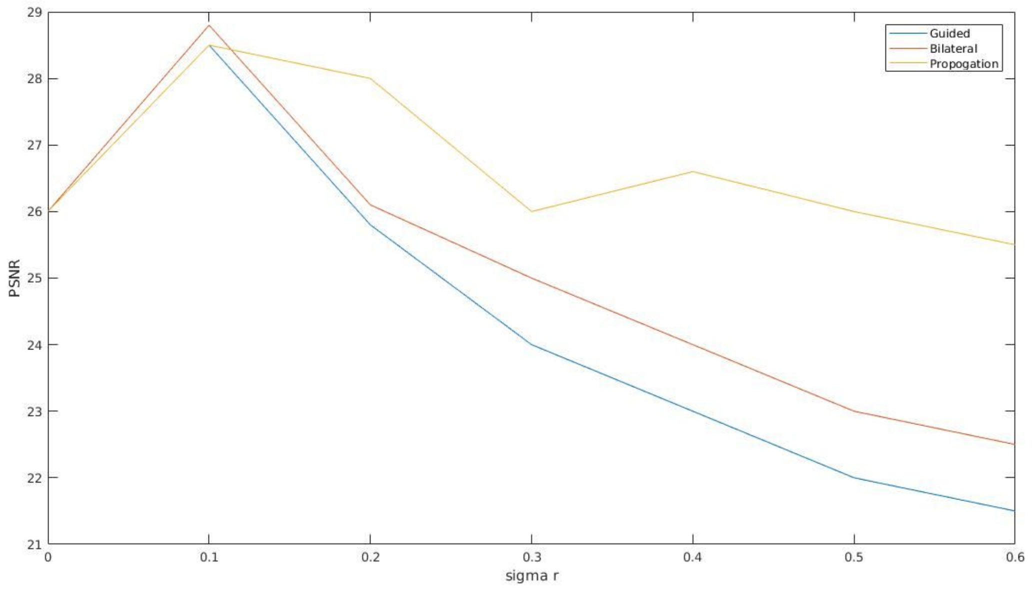
Graphs:

For Monkey Image

PSNR vs w



PSNR vs sigma r



SSIM vs w

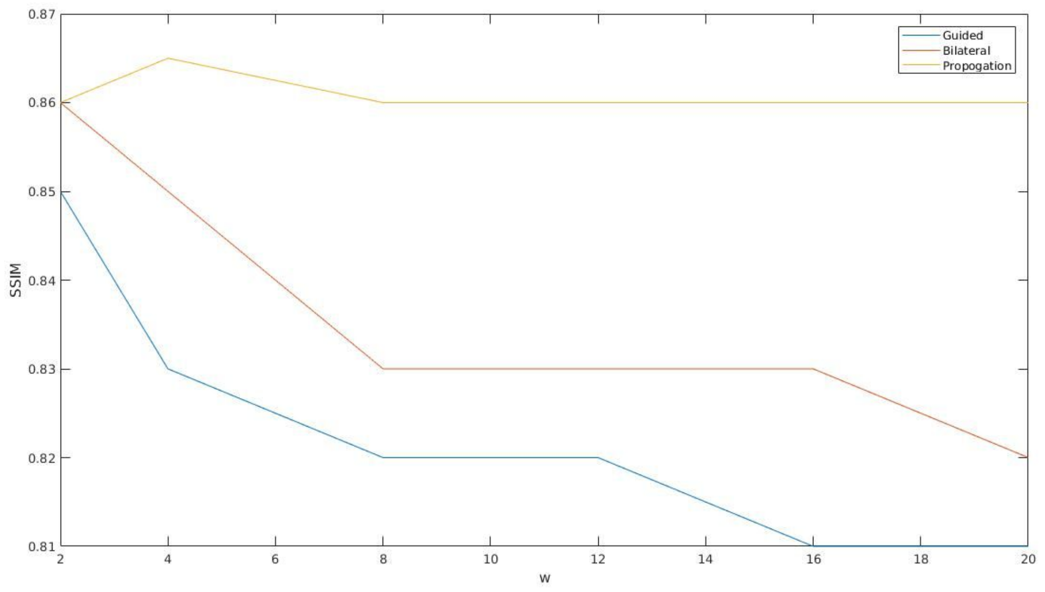


Image Smoothing:

Original Images

Image smoothing through bilateral filter (σ_s = 0.25)

Image smoothing using guided filter with window size 5x5 and σ_s = 0.25

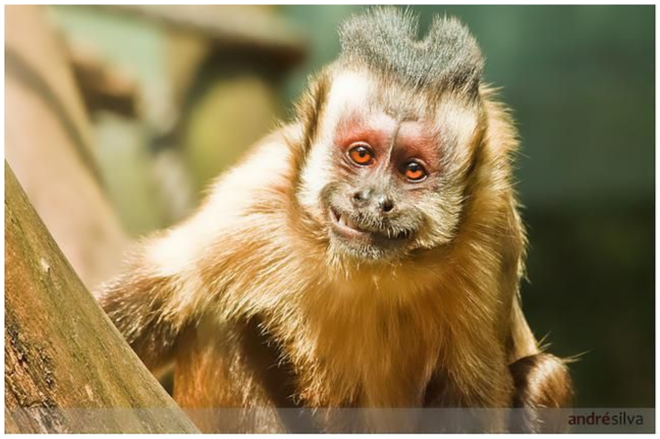


Image smoothing using guided filter with window size 10x10 and $\sigma_s = 0.25$

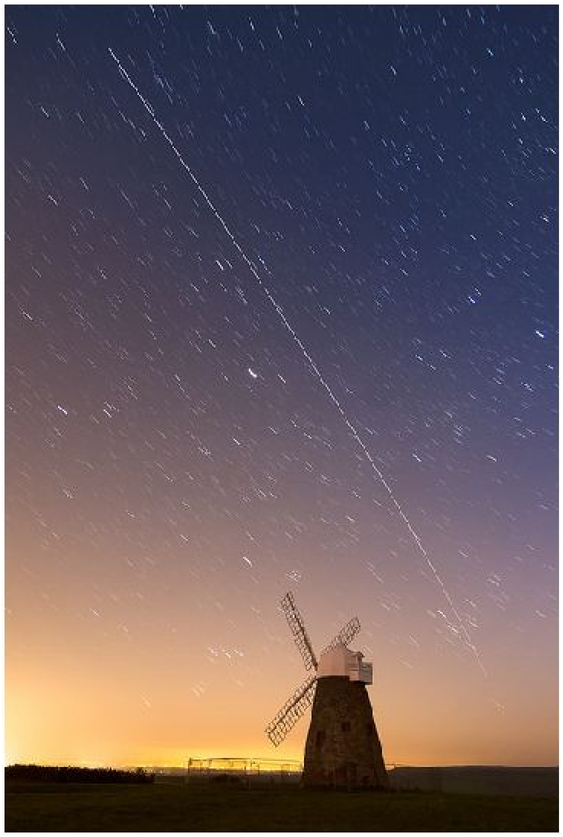
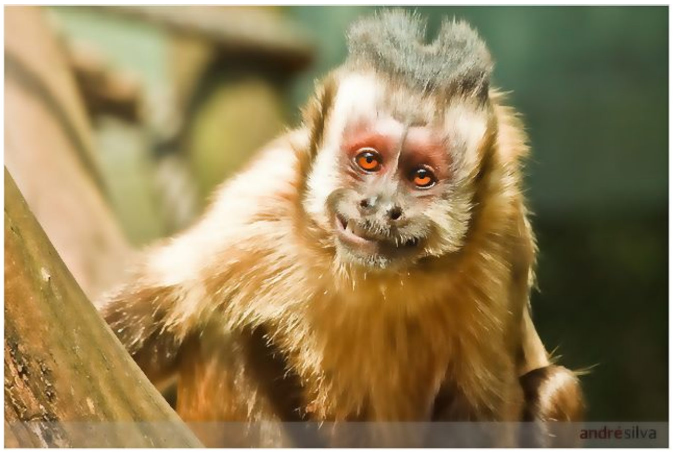



Image smoothing using Propagated Image Filter with window size 5x5, σ_s = 0.25

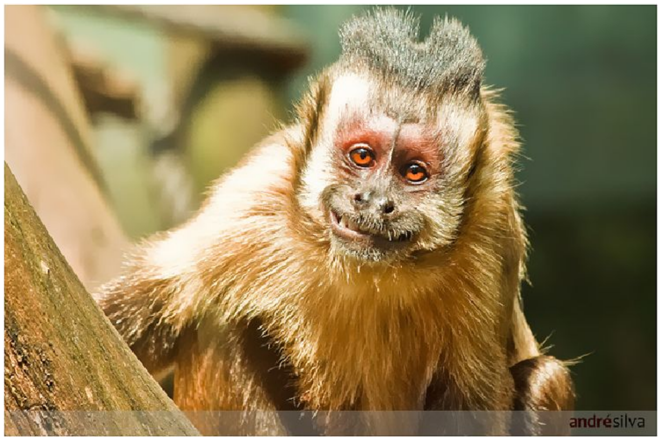
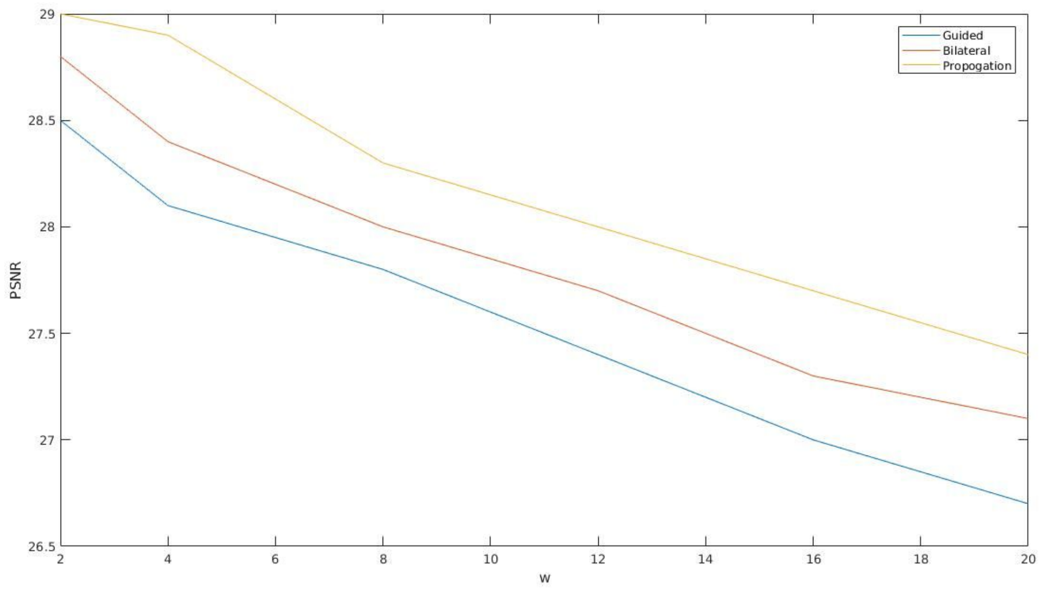


Image smoothing using Propagated Image Filter with window size 10x10, σ_s = 0.25

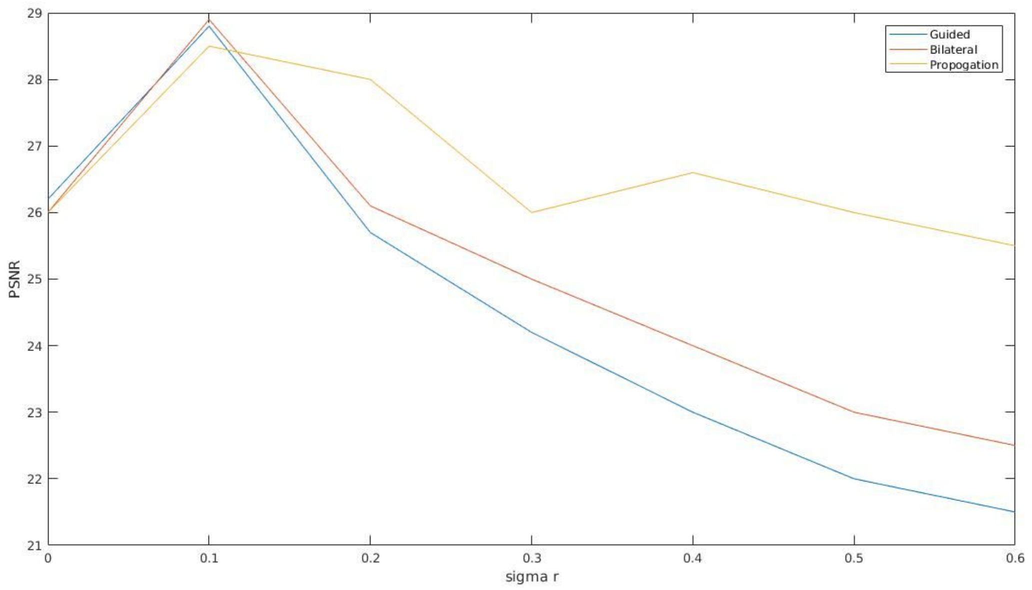
Graphs:

For Monkey Image

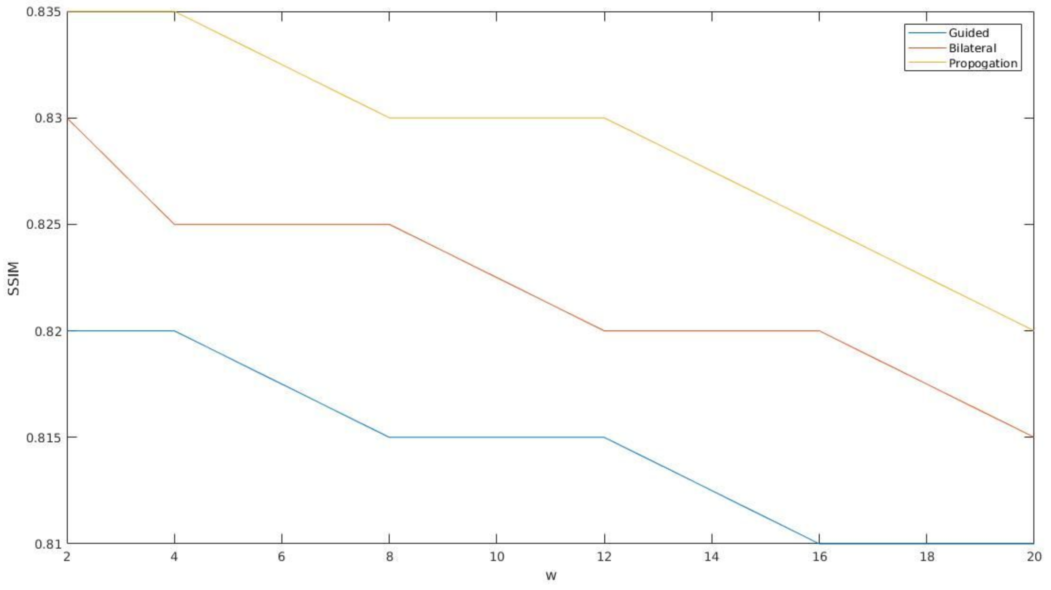
PSNR vs w



PSNR vs sigma r



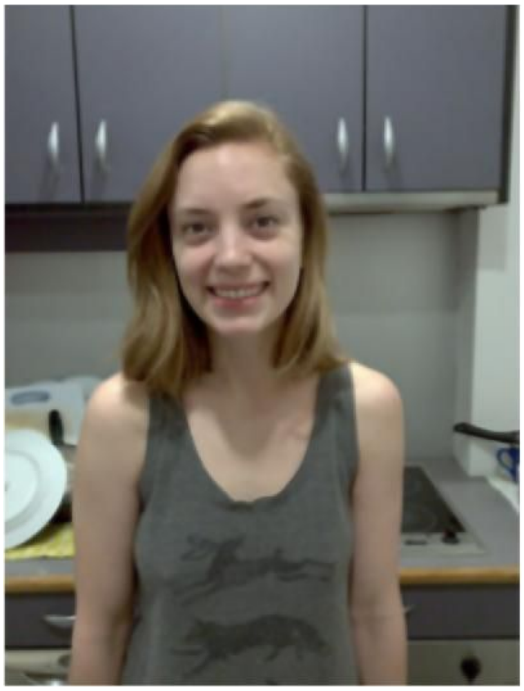
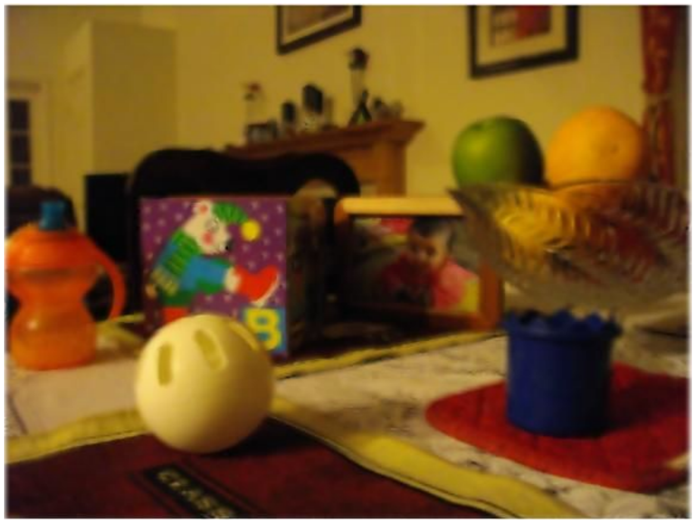
SSIM vs w



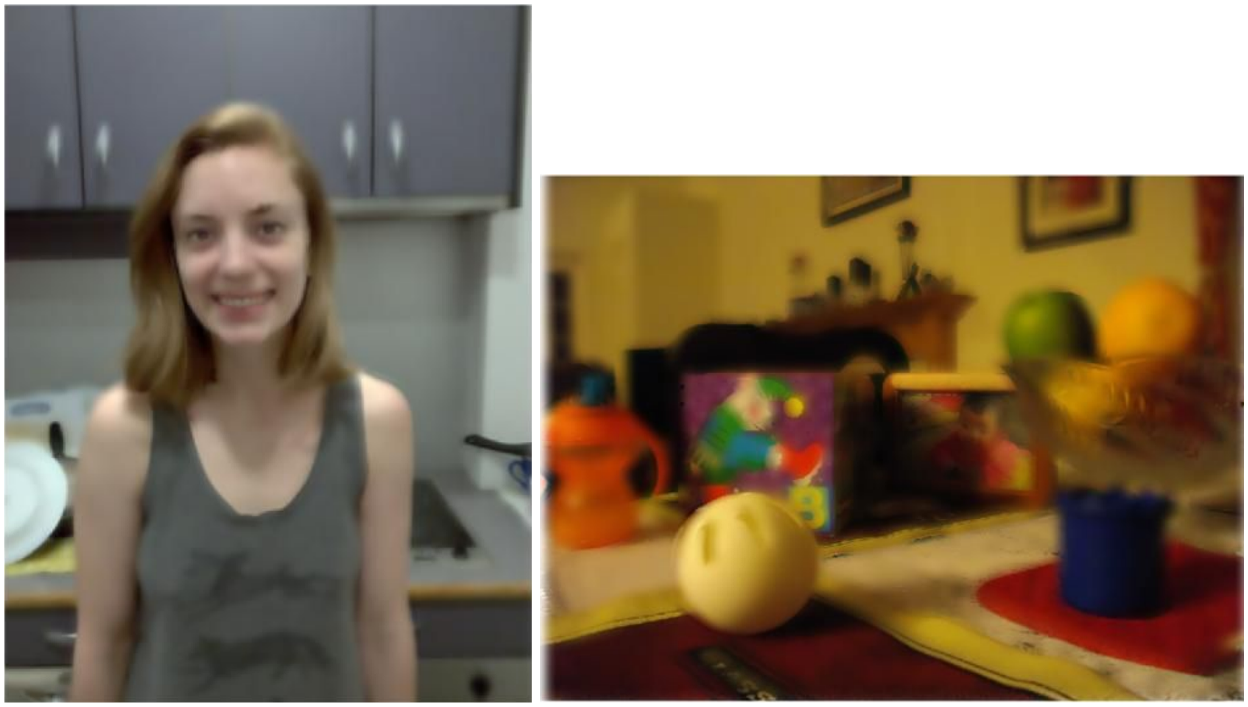
Flash/No-Flash Denoising:

Flash Images

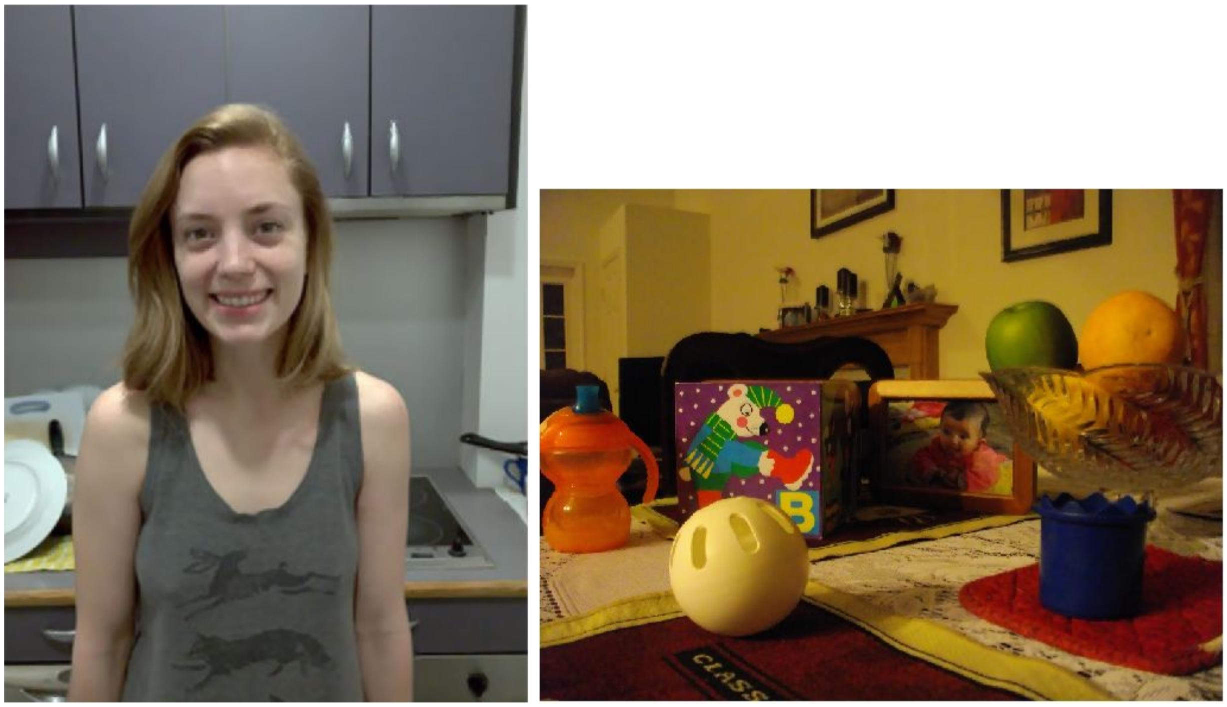
Images Without Flash

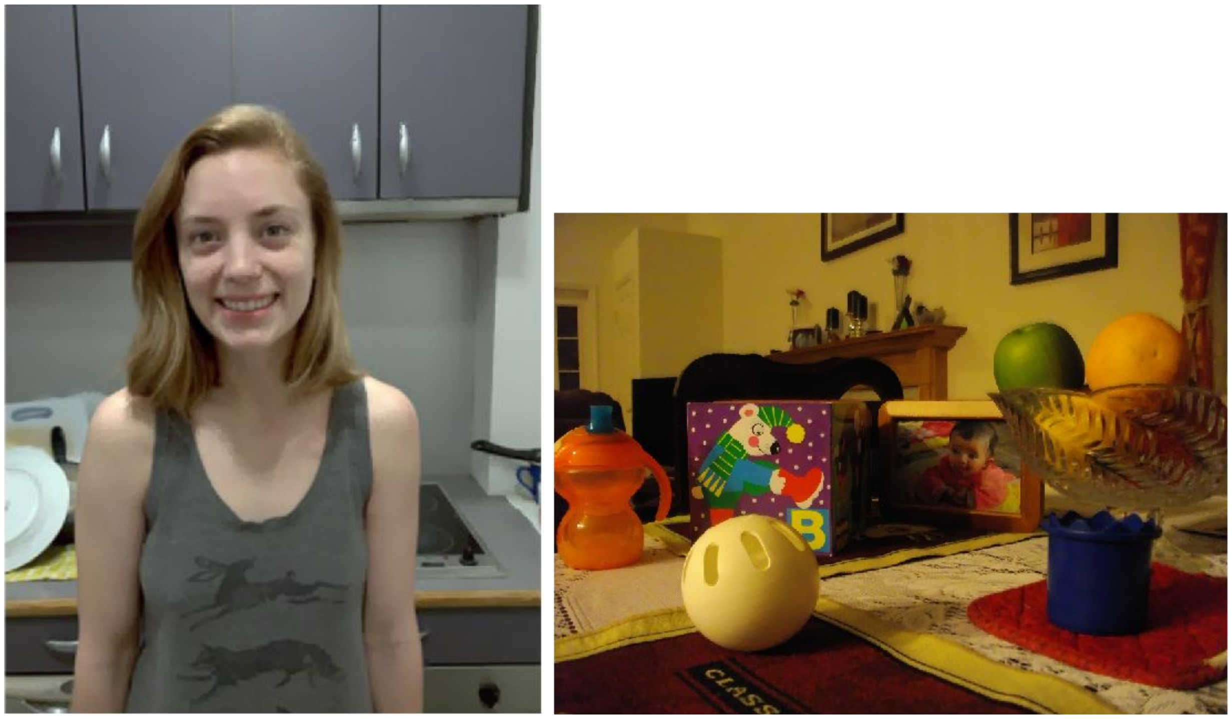
Flash/NoFlash Denoising using guided filter with window size 5x5 and $\sigma_r = 0.1$



Flash/NoFlash Denoising using guided filter with window size 10x10 and $\sigma_r = 0.1$



Flash/NoFlash Denoising using Propagated Image Filter with window size 5x5 and $\sigma_r = 0.1$



Flash/NoFlash Denoising using Propagated Image Filter with window size 10x10 and $\sigma_r = 0.1$

Presentation Link :
https://docs.google.com/presentation/d/17t_5b7Q1fYwKNtRDkpyblQ0u30ZWTurNaNJiNHuWsrE/edit?usp=sharing