Project Report

Geometric Stochastic Image Denoising for Monte Carlo Path Traced Images

Shevlin Bisesar, Mustafa Quraish

Supervisor: Franciso J. Estrada

April 2019

1. Introduction

1.1 Incentive

Path tracing is a method of rendering images that have global illumination. We do this by simulating the path of light from the camera, through the scene and eventually to the light source. The general overview of the Monte Carlo Path Tracing algorithm is to render multiple samples of the image and average them, where one sample of the image is created by simulating the path of one light ray per pixel.

This algorithm relies on every pixel converging to a colour as the number of samples gets higher. However, a result of this is that at lower sample counts, all the pixels may not converge and this can lead to an image looking noisy. Rendering with high sample counts is very expensive, and is not always possible because of the lack of computational power.

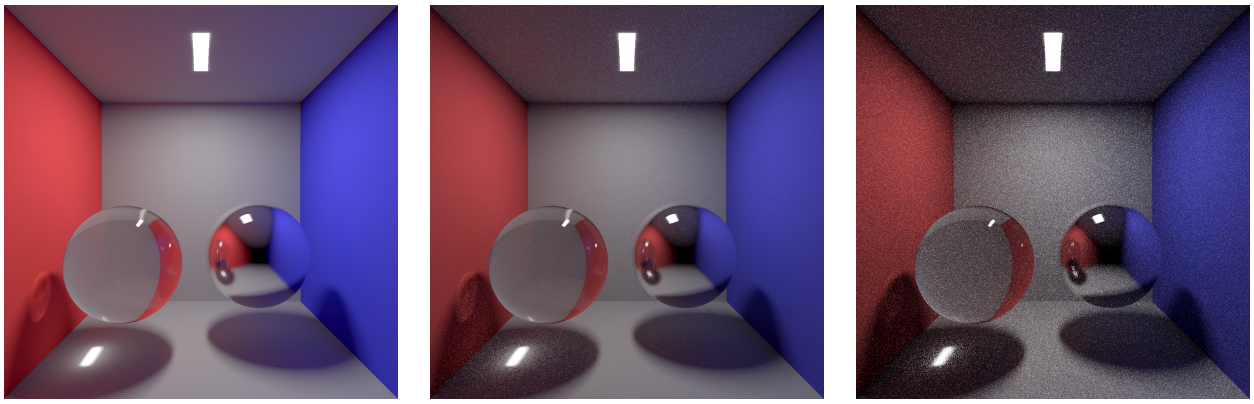


Figure 1.1: The same image rendered with different sample counts (high-to-low)

1.2 Goals

The goal of this project was to create an algorithm that could reduce the noise in path traced images. Traditional denoising algorithms don't have any information about the scene, and denoise naively. We, however, had all the scene information, and wanted to incorporate it

into an algorithm that actually had some idea of what it was trying to denoise. This would potentially help us significantly reduce render times while still retaining quality.

1.3 Related Work

Recently, there has been increasing interest in ray-tracing for video games and movies, and there are already a couple of algorithms that claim to do what we were trying to do. However, most of them seemed to be designed to work with video rather than with still images. This meant that they used hybrid methods, where the image would be rendered using rasterization and ray-tracing, and then both of these would be used reconstruct a noise-free image. They also use information from previous frames, since in video it is usually reasonable to expect little movement between consecutive frames.[2, 3]

These algorithms provide results that are good enough for video playing at multiple frames per second, however they are limited by the time they have to work with each frame. This is not a limiting factor to us, and we didn't want to use rasterization since it doesn't reproduce global illumination, and might cause the result to not look realistic. This was a problem since we were working with still images.

2. Algorithm

For our project, we decided to expand upon the Stochastic Denoising algorithm[1] by adding additional information from the renderer about the scene, and using that information to increase the effectiveness of the denoising.

2.1 Stochasitc Image Denoising Algorithm (SID)

In summary, to denoise an image using this algorithm, we perform a series of random walks on the image beginning at each pixel, and accumulate colours along this random walk. The probability of transitioning from one pixel x_j to another pixel x_{j+1} on a walk that started from the pixel x_0 is defined to be

$$p(x_{j+1}|x_j) = \frac{1}{K} e^{\left(\frac{-d(x_0,x_{j+1})^2}{2\sigma^2}\right)} e^{\left(\frac{-d(x_j,x_{j+1})^2}{2\sigma^2}\right)}$$

where K is a normalizing constant, σ is a scaling parameter, and $d(x_i, x_j)$ is a *difference function* which is used as a measure of (dis)similarity between any two pixels. The original algorithm proposes using the Euclidean distance between the RGB values of the pixels as this difference.

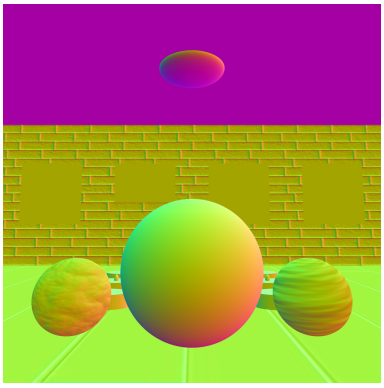
This algorithm is able to average out the colours of localized regions as random walks stay in a similar neighbourhood because of the way probabilities are assigned. It also is able to maintain edges of objects in the images since the probability of transitioning relies on pixels being similarly coloured. While this works well with photographs, it doesn't do as well when dealing with noisy renders.

2.2 Additional Data

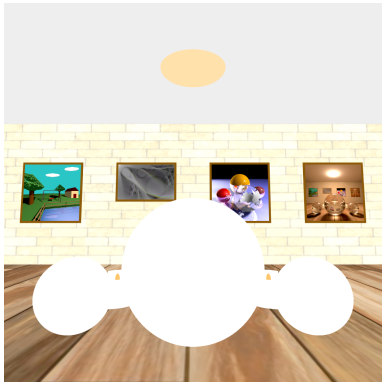
We felt that this algorithm had potential for what we were trying to accomplish, and we could incorporate the scene information as part of the distance function that would be more representative of how similar two pixels should actually be.

The additional geometric information that we will be using include the normal maps, true colour maps, material maps, shadow maps, and depth maps. All of these maps should

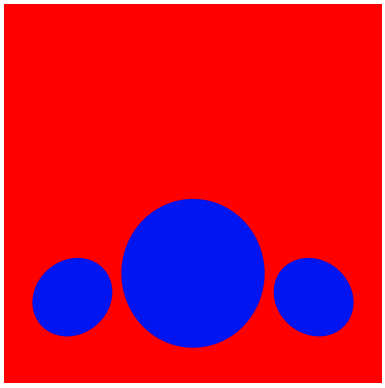
be easy enough to generate from any rendering engine, and will allow the random walks to rely on the similarity of the objects in the scene rather than just the colour of the pixels in the final render, which may be an outlier.



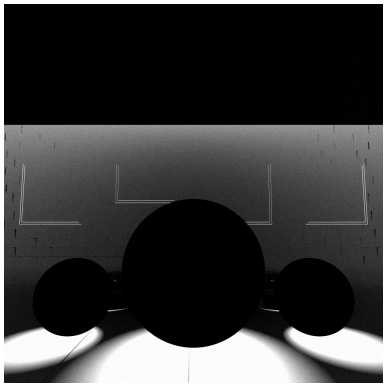
(a) Normal Map



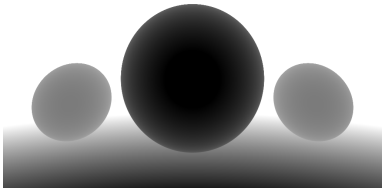
(b) True Colour Map



(c) Material Map



(d) Shadow Map



(e) Depth Map

(a) The normal map stores data about the surface normal of the corresponding object at every pixel. Each pixel's *RGB* values represent a unit vector which is the average surface normal for all the samples taken on that image from the Path Tracer. A unit vector (x, y, z) is stored as the RGB triplet $(\frac{x+1}{2}, \frac{y+1}{2}, \frac{z+1}{2})$. If the vectors obtained for two pixels are similar this would mean that they likely share similar lighting.

(b) The true colour map stores the object colours without accounting for any lighting effects (Colour bleed, caustics, etc.). If two pixels share similar colours on this map it means that it is likely that the lighting effects on these pixels would generate similar colours.

(c) The material map stores values relating to the diffuse, reflective and refractive properties of an image. If an object in one pixel has a high reflective value and another pixel has a high diffuse value, the way light will behave on these two objects is very different and

thus these two pixels are less likely to be in the same random walk. The RGB values of the colour represent the diffuse, reflective and refractive components.

(d) The shadow map stores a value for each pixel representing how exposed the corresponding object is to a light source. This map is used to prevent having pixels in shadow and pixels not in shadow being used in the same random walk.

(e) The depth map stores values for each pixel representing how far the corresponding object is from the virtual camera. If one object is very close and another object is very far, it's likely that they have very different lighting and thus they shouldn't be part of the same random walk.

2.3 Difference Function

Given any two pixels x_i and x_j in the image, we used all of our additional information from the maps above to define the following 8 features:

$F_1(x_i, x_j)$ - Euclidean dist. between the RGB values of the input image
$F_2(x_i, x_j)$ - Euclidean dist. between the RGB values of the true colour
$F_3(x_i, x_j)$ - Dot product of the two normal vectors
$F_4(x_i, x_j)$ - Absolute diff. between the diffuse component
$F_5(x_i, x_j)$ - Absolute diff. between the reflective component
$F_6(x_i, x_j)$ - Absolute diff. between the refractive component
$F_7(x_i, x_j)$ - Absolute diff. between the shadow map values
$F_8(x_i, x_j)$ - Absolute diff. between the depth map values

The Euclidean RGB distance made the most sense for difference between colour since it was what the original SID algorithm was based on, and we know it performs really well.

In order to compare the normals, a dot product was used. Note that this is because our Path Tracer assumes a cosine BRDF, however, this may need to be changed depending on how the image was rendered for best results.

The depth, shadow and material features use a simple absolute difference of values to compare two pixels. Depending on how these values are scaled by the engine, or what the user feel is more important, the relevant features could be squared.

We defined our distance between function to be a linear combination of these features

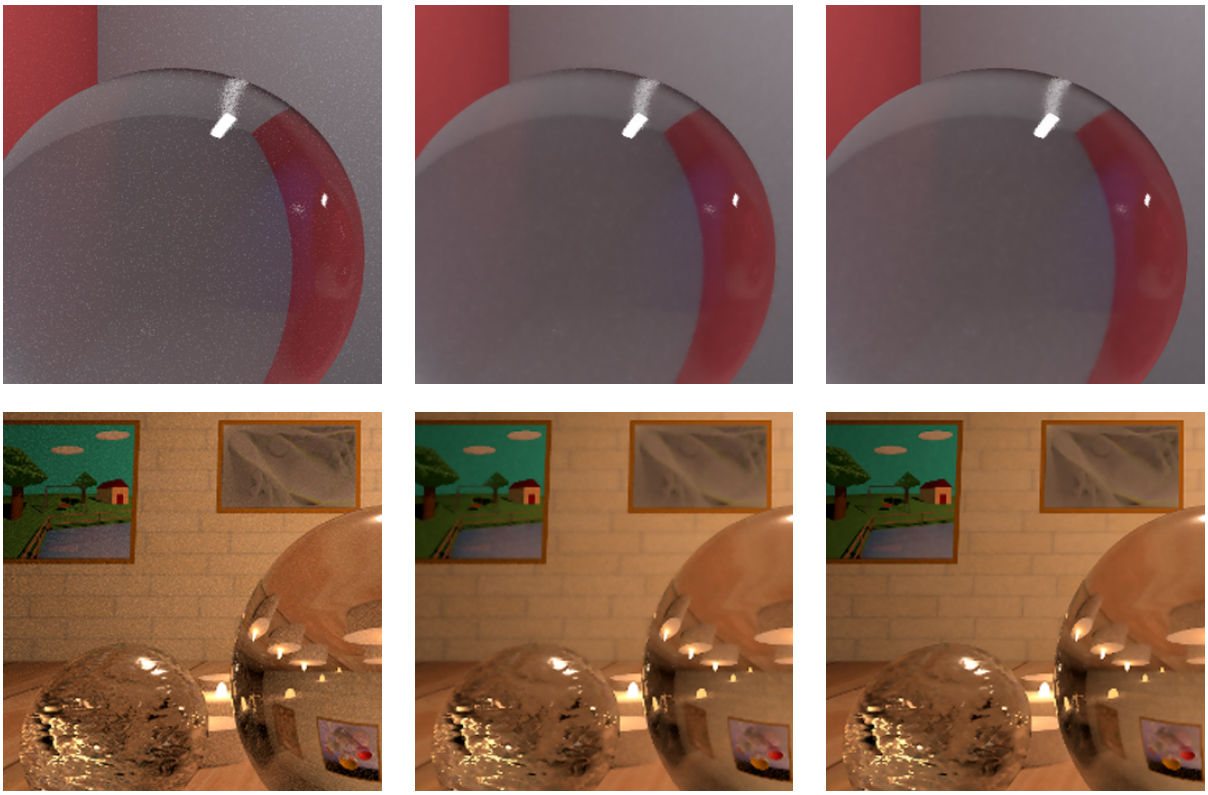
$$d(x_i, x_j) = \sum_{k=0}^{8} W_k F_k(x_i, x_j)$$

where W_k are the corresponding weights. We used $[2, 2, 3, 1, 2, 1, 0.5, 2]$, however these may need to be tweaked for optimal results based on the rendering engine.

3. Results

3.1 Strengths

The results seemed promising. Our new algorithm seems to outperform the original in every regard. We get better edge separation between objects of similar colour due to the additional information we have. In general, we had less noisy images without having to turn up the σ parameter as high, which also meant that our image was less blurry.



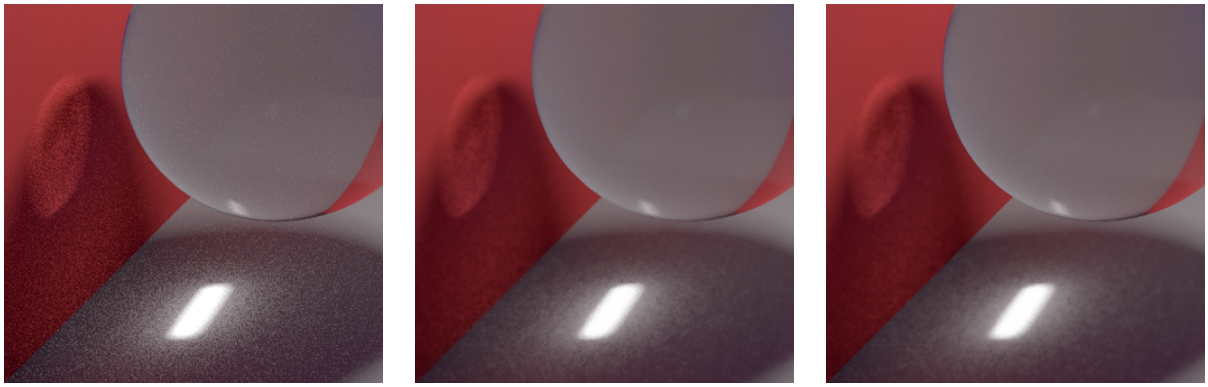
(a) Input (b) Regular SID (c) Geometric SID

Figure 3.1: Denoised images using SID and Geometric SID

In the Path Tracing algorithm, the caustics are noisy (for smaller sample counts) due to the fact that Path Tracing starts tracing light rays from the camera. Due to the nature of the Monte Carlo algorithm, the chances of having the a ray of light bounce off a surface, go

through a refractive object, and hit a light source are very small. Even with techniques such as photon mapping, it is not possible to get completely clean (and realistic) caustics with limited photon counts.

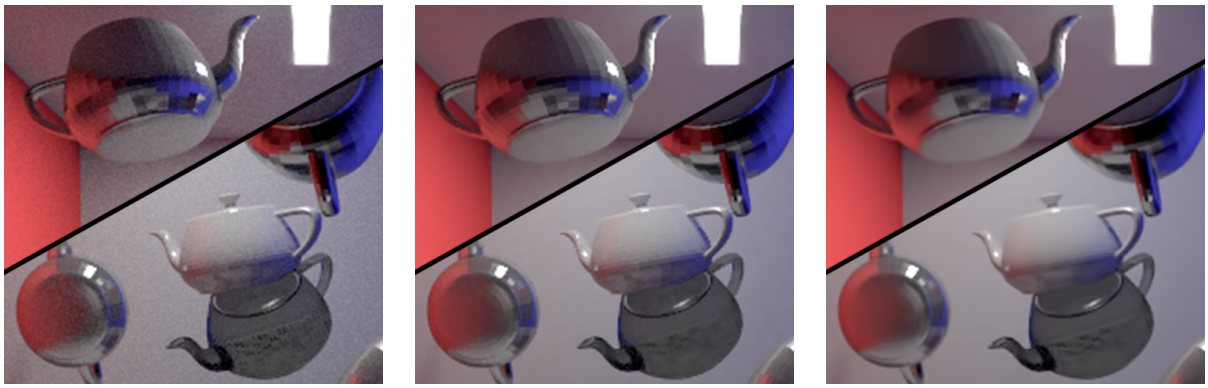
In the Stochastic Denoising algorithm, when cleaning up these areas we only look at the difference in pixel colours, and since these caustic areas are very noisy our random walks are not able to average out the colour well. In contrast the geometric stochastic denoising algorithm takes into account that these surfaces are similar for other reasons (that aren't the colour) and allows us to blend the colours together better.



(a) Input (b) Regular SID (c) Geometric SID

Figure 3.2: We get smoother caustics compared to the regular SID algorithm

The added features result in excellent edge detection and preservation. For example, when the surface normal vectors of two pixels differ slightly, it means that both pixels are probably on the same object and that this object has a curve to it. However, if the normal vectors are very different there is a good chance that we have moved to a different object. Even compared to state of the art denoising algorithms such as BM3D[4], our algorithm preserves edges better since we simply have a lot more information to work with.



(a) Input (b) Geometric SID (c) BM3D

Figure 3.3: The edges are preserved by our algorithm

Path traced images at a high sample count are able to generate an image that follows

a Gaussian noise model. Because of this images rendered for a longer time are able to get denoised very accurately and look almost perfect. Because the transitional probabilities from pixel p_i going to some pixel p_j are normalized, when all possible p_j's are very similar the slightest difference can lead to a large change in probability. This allows us to retain a high level of detail in the images.

(a) Groundtruth (b) Geometric SID (c) BM3D

Figure 3.4: BM3D oversmoothing the image, losing some of the fine details in the bricks

3.2 Weaknesses

An inherent weakness of our algorithm (and also SID) is that it assumes the noise in the image follows a Gaussian distribution. For photographs taken with a camera, this is usually the case. However, when working with renders at a low sample count, the pixels have not begun to converge to their actual colour, what we encounter is actually salt-and-pepper noise (Figure 1.1 illustrates this). This noise does not follow a Gaussian distribution, and because of this when we try to denoise it, our algorithm gives us a "splotchy" effect.

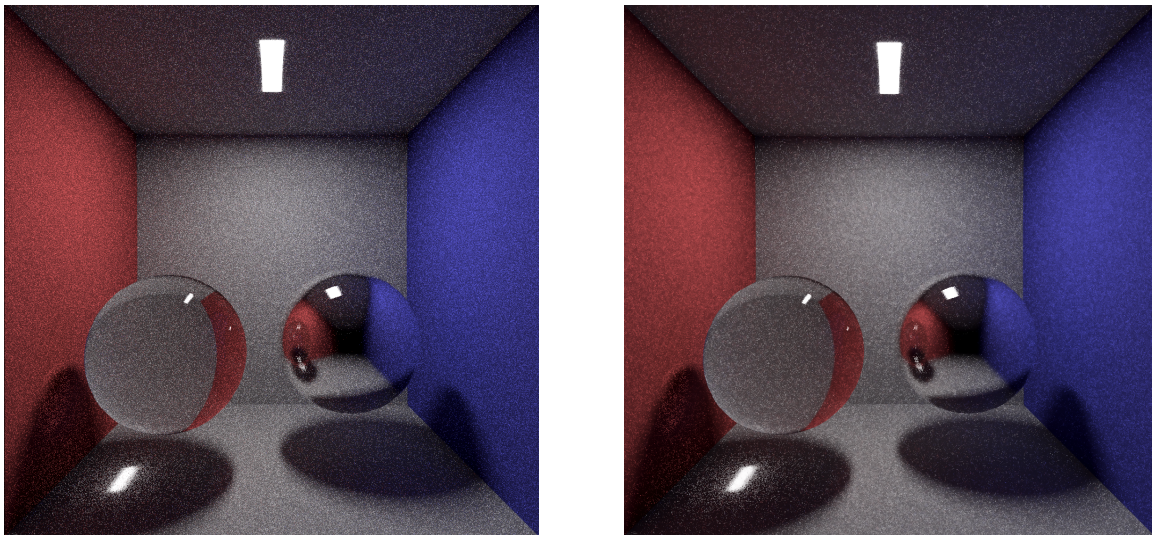
(a) Input (b) Geometric SID (c) BM3D

Figure 3.5: "Splotchy" effect when denoising on an image with a low sample count

The reason this happens is because the salt-and-pepper noise makes it harder for the random walks to average out local regions smoothly. All of the pixels on the same smooth

surface may have very different colours yet they are all counted because they share similar geometric properties, and this is what causes the splotchy effect on what should be smooth surfaces. BM3D gives us more uniform colours since it does not assume such a noise distribution, however, we can see that this comes at a cost of possibly losing details in other areas of the image. (Figure 3.5)

Outlier pixels (pixels with drastically different colours) also contribute to this effect as they can heavily influence the colour of a denoised pixel by only being in a few of the random walks. Similarly when an outlier pixel is being denoised, it will always favour other outlier pixels for it's random walks causing it to still be an outlier after denoising.



(a) Low Sample (b) Low Sample Denoised

Figure 3.6: The denoised image on the right has retained many of the outlier pixels

3.3 Comparing Performance against other algorithms

We rendered 2 different scenes at low, medium and high sample counts, and then denoised these images using Regular SID, Geometric SID and BM3D. These were then compared to ground-truth images (rendered at a few hundred thousand samples) using the SSIM Index for image quality assessment[5]. One of the scenes was a simple Cornell box with no textures, and the other was a more complex scene with textures, normals, and multiple light sources (Both these scenes are illustrated in Figure 3.1). These results are below.

Overall, we did better than the original algorithm in all cases, but did not get better results than BM3D. However, we believe that some features of the denoised images are more preferable from our algorithm (especially in renders with low sample counts). While BM3D does remove more noise, this comes at a cost of blurring out some details.

Scene 1

	Low	Medium	High
Input	0.1314	0.6564	0.8432
SID	0.2769	0.8975	0.9623
GSID	0.2936	0.9197	0.9757
BM3D	0.2323	0.8530	0.9646

Scene 2

	Low	Medium	High
Input	0.4667	0.6794	0.8674
SID	0.7735	0.9076	0.9574
GSID	0.7935	0.9198	0.9678
BM3D	0.8184	0.9204	0.9707

Table 3.1: SSIM index comparison of different denoising algorithms on renders with different sample counts. Values closer to one mean that the image is similar to the ground truth image.

Another reason we think it does worse on Scene 2 is because of the textures. Even though we do account for the true texture color for every pixel, because of the way we have defined our weight function, if all the other features are really similar we still take jumps across colours. This, however, could probably be fixed by carefully tuning the weights, or using some other function instead of a linear combination. The problem does not seem to be caused by anything fundamental to the algorithm, unlike the issue with salt-and-pepper noise.

4. Conclusion

4.1 Summary

Overall we were able to improve the effectiveness of the Stochastic Image Denoising algorithm on Path Traced images, and even come close to - if not surpass in some aspects - the state of the art denoising algorithms like BM3D. By adding in extra information that can easily be generated by a rendering engine we were able to account for various factors in the difference between two pixels. This leads to an effective denoising algorithm that could be used to generate a clean image out of a noisy render.

4.2 Potential Further Work

In order to account for the outliers that are present in lower sample count renders it maybe possible to add in outlier detection as a pre-procesing step. Once we have detected these noisy pixels, they could be accounted for in the weight function appropriately, and potentially get rid of the splotchy effect. We had some initial luck detecting these outliers at the beginning of our project, but never implemented this as some careful thinking would be needed to account for them properly.

In order to account for the random noise model of very noisy renders, we could potentially use Image Pyramids and Image Blending [6]. The idea behind this is to run our denoising filter on the image at full resolution, than we would subsample this denoised image to half it's resolution and run the filter again. We would repeat this subsample denoising process for some depth (depth 3 would be 3 levels of subsampling and denoising for example) and then we would use an image blending algorithm to reconstruct our final image using all of the denoised images. By doing this our denoise filter could travel further along image, and let us account account for larger regions of a smooth surface, since our current random walks are only in a very local neighbourhood.

Bibliography

[1] Francisco J. Estrada, David Fleet, and Allan D. Jepson, *Stochastic Image Denoising*, British Machine Vision Conference, BMVC 2009.

[2] Michael Mara, Morgan McGuire, Benedikt Bitterli,and Wojciech Jarosz, *An Efficient Denoising Algorithm for Global Illumination*. Proceedings of High Performance Graphics, July 2017.

[3] Christoph Schied, Anton Kaplanyan, Chris Wyman, Anjul Patney, Chakravarty R. Alla Chaitanya, John Burgess, Shiqiu Liu, Carsten Dachsbacher, Aaron Lefohn, and Marco Salvi, *Spatiotemporal Variance-Guided Filtering: Real-Time Reconstruction for Path-Traced Global Illumination*, Proceedings of HPG 2017.

[4] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. *BM3D image and video denoising software*, Website.

[5] Zhou Wang, A. C. Bovik, H. R. Sheikh, E. P. Simoncelli, *Image quality assessment: From error visibility to structural similarity*, IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600-612, Apr. 2004. Website.

[6] Alexei Efros, *Image Pyramids and Blending*, Cernegi Mellon University, Computational Photography Fall 2005