# A local image reconstruction algorithm for stochastic rendering

Peter Shirley NVIDIA
Timo Aila NVIDIA
Jonathan Cohen NVIDIA
Eric Enderton NVIDIA
Samuli Laine NVIDIA
David Luebke NVIDIA
Morgan McGuire
Williams College and NVIDIA

**Figure 1:** *Left: sixteen sample per pixel stochastic rendering with a pixel box filter. Right: same input samples with new filtering algorithm.*

## Abstract

Stochastic renderers produce unbiased but noisy images of scenes that include the advanced camera effects of motion and defocus blur and possibly other effects such as transparency. We present a simple algorithm that selectively adds bias in the form of image space blur to pixels that are unlikely to have high frequency content in the final image. For each pixel, we sweep once through a fixed neighborhood of samples in front to back order, using a simple accumulation scheme. We achieve good quality images with only 16 samples per pixel, making the algorithm potentially practical for interactive stochastic rendering in the near future.

## 1 Introduction

Hardware rasterization pipelines have been highly successful at rendering complex scenes. However, they have difficulty reproducing the physical camera effects of defocus blur and motion blur. Recent progress has been made in using stochastic techniques for interactive rendering, producing those camera effects by randomly varying center of projection and/or time per sample, as in typical offline rendering systems. Unfortunately, at interactive frame rates, the number of random samples available in the foreseeable future is not sufficient to produce visually smooth images using simple unbiased sample averaging.

If we allow the introduction of some bias, we can reduce noise in the final image with a feature-sensitive blur. Blurring has the effect of spreading information locally over the image, so it increases the total number of samples informing each pixel value. This "denoising blur" is distinct from the physical blur resulting from the optics of a camera, and refers to artificial filters that may be applied in order to reduce high frequency noise.

Our approach is to design a denoising blur method that assumes the input data has been generated via an unbiased physical blurring process. Our goal, then, is to design a filter that reduces visible noise while adding a visually acceptable amount of bias. Other approaches such as bilateral filtering may be used similarly. However, by using depth information from all samples during reconstruction, we can properly handle occlusion effects that often defeat purely image-based methods. Our method processes each pixel independently by depth sorting a fixed number of nearby samples, and then sweeping over them once with a simple accumulation step. The algorithm is simple enough to be amenable to implementation on future graphics hardware. We demonstrate the algorithm on motion blur, defocus blur, and stochastic transparency.

## 2 Related work

Researchers have investigated blurring irradiance in a similar spirit to our bias-versus-noise tradeoff, in various ways. This has been done in world space (e.g. [Ward et al. 1988]) and screen space (e.g. [Kontkanen et al. 2004]). Jensen and Christensen's [1995] screen space algorithm is close to ours in that it used a set of fixed width tiles in screen space, but it did not need to deal with issues of visibility so is fairly different in detail. Diffusion has been used in screen space to lower noise in stochastic rendering (e.g. [Xu and Pattanaik 2005]); however, these techniques have not yet been shown to work over time or for non-illumination noise, and they may not be fast enough for our purposes.

Several researchers have examined reconstruction of stochastic samples as a signal processing problem (e.g. [Egan et al. 2009; Soler et al. 2009]). These techniques are more sophisticated and accurate than ours, but as yet cannot be applied generally when there is both motion and defocus blur. Meyer and Anderson [2006] applied motion blur analysis over multiple frames, but that is difficult unless future motion paths are known. Rushmeier and Ward [1994]


I3D 2011, San Francisco, CA, February 18 – 20, 2011.
© 2011 ACM 978-1-4503-0565-5/11/0002 $10.00

estimated statistical confidence for each pixel and allowed energy to "leak" to neighboring pixels in the presence of an outlier sample. They did not take into account visibility order and their method is thus best suited to Monte Carlo lighting where outliers are common.

Overbeck et al. [2009] created Monte Carlo images directly in wavelet space and then trimmed off wavelet coefficients that appeared to be noise. This smoothed blurry regions and maintained roughly uniform sampling. The method is a sampling and reconstruction framework rather than a pipeline, but it does address similar issues as our method, and the reconstruction could in principle be added to the end of a pipeline with fixed sampling. Such an algorithm might be an interesting alternative to ours.

A variety of algorithms simulate optical blurring from a pinhole camera rendering, but these techniques start with much poorer information than our input data, e.g. missing occluded objects that would be visible to an off-center point on the lens. Some researchers use "taps" at each shading point to determine how to locally blur (e.g., [Scheuermann 2004; Robison and Shirley 2009]). Another approach is to use the depth map to decide how to blend images blurred at several scales (e.g. [Filion and McNaughton 2008]). Researchers have also proposed several depth-of-field approximations for scenes broken into layers (e.g. [Kosloff et al. 2009; Lee et al. 2009]). These techniques attain good results but solve a problem with different input than ours. Point-based methods also attack motion blur (e.g. [Heinzle et al. 2010]) and defocus blur (e.g. [Krivanek et al. 2003]) but these also solve a different and somewhat more challenging problem than ours as many invisible points must be processed.

To our knowledge, filtering stochastic sample buffers at low sampling densities is a largely unexplored area. This may simply be because the offline rendering community has historically been able to take more samples when needed, and only recently has the interactive rendering community been able to take more than one per pixel.

## 3 Algorithm

We start with a buffer of samples with associated color $c_i$, depth, transparency, and motion. The classic reconstruction algorithm sets each pixel to a weighted average of sample colors for all samples within a fixed window in screen space centered at the pixel. Algorithm 1 describes traditional filtering for the case of $N$ samples per pixel and a pixel filter $f$ with diameter $D$. Figure 2 shows a tutorial scene with a reference solution and as rendered by Algorithm 1 using a unit width box filter for $f$.

To gain some intuition about the problem, we first examine altering the algorithm above to make the filter wider for samples representing blurry effects. We could think of this as "splatting" samples with variable width splatting kernels in a point based rendering scheme. We assign a *denoising diameter*, $d_i$, to each sample that is 1.0 for samples where conventional pixel filtering is desired, and

**Algorithm 1** Traditional reconstruction

1: rgb $c = (0, 0, 0)$; // accumulated color
2: float $w = 0$; // accumulated weight
3: **for** each $i$ of $N * D^2$ samples around a pixel **do**
4: $\quad w_i = f(x_i - x_c, y_i - y_c)$; // $(x_c, y_c)$ is pixel center
5: $\quad w$ += $w_i$;
6: $\quad c$ += $w_i * c_i$;
7: **end for**
8: $c_{\text{pixel}} = c/w$; // so that we don't need $w_i$ to sum exactly to 1

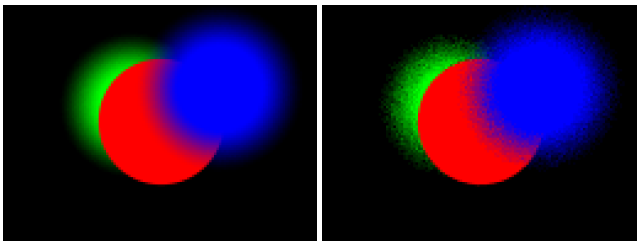

**Figure 2:** *Left: Reference image of three flat-shaded spheres with defocus. Right: a stochastic rendering at 16 samples per pixel.*

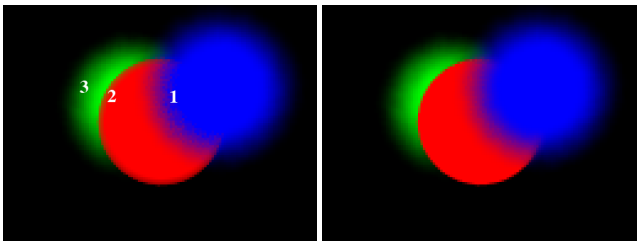


**Figure 3:** *Left: Output of Algorithm 1B with labelled visual artifacts. Right: the result of Algorithm 2.*

larger (wider) for samples where blurring is desired. The resulting Algorithm 1B just requires a change to line 4:

$$w_i = (1/d_i^2) f((x_i - x_c)/d_i, (y_i - y_c)/d_i);$$

The denoising diameter may be any programmer-selected function of sample depth, motion, transparency, texture frequency content, or even artistic variables such as visual importance. A simple choice for $d_i$ is the circle of confusion at the sample's depth.

An example using this algorithm is shown in the left of Figure 3. Three visual artifacts are immediately apparent (see labels in the figure):

1. Where there is a blurred and almost opaque object in front of a sharp object, there is obvious noise.
2. Where there is a sharp (in focus) object in front of a blurry object, the blurry object "leaks" on top of the sharp object.
3. Where there are two colors being blurred, there is some low frequency noise.

**Origin of artifact 1.** This artifact occurs where almost all the samples are likely to be from the front blurry object. In the example figure, the blurry blue object contributes with a wide filter ($d_i > 1$), and so each output pixel draws blue samples from a wide neighborhood of input pixels. But because the in-focus red object uses a narrow filter ($d_i = 1$), each output pixel receives red samples from only one input pixel. Suppose the coverage of the front object is 15/16 at 16 samples per pixel. Due to random variation, we might have 0, 1, or 2 samples from the in-focus object contributing to the result, and these variations are visually obvious. In summary, the sharp features are under-sampled.

**Origin of artifact 2.** This artifact has a color shift inside the edge of the middle sphere and occurs because even when all of the samples in the center pixel are in-focus, samples of the blurry background object in neighboring pixels are allowed to influence the pixel value. This artifact is as wide as the largest value of $d_i$.

**Origin of artifact 3.** This artifact occurs because blurring a stochastic image lowers noise amplitude but also moves it to lower spatial frequencies. Humans are more sensitive to contrast fluctuations at middle frequencies that peak at around 5-15 pixels for most display conditions.

Artifact 3 can easily be addressed by using more samples or wider windows, but at the expense of efficiency. An approach by Boulos et al [2007] lessens this artifact by using cooperative patterns where samples in adjacent pixels are negatively correlated. We now develop an algorithm to address the first two artifacts.

### 3.1 Sweep-based algorithm

One approach to addressing artifacts 1 and 2 above would be to develop two approaches, one for each artifact, and combine them based on the demographics of the samples near a pixel. Better would be to find a solution that will work for either case so that hybrid pixels will be handled automatically. First, how would we approach these two cases independently?

**Addressing artifact 1.** When there are just not enough in-focus samples to produce a good image, we need more samples. When there is a blurry object with high opacity obscuring an in-focus object, the in-focus object should be dim, but still in focus. Unfortunately 1-2 samples per pixel is not enough to make the in-focus object look good. If we do not want to take more samples, one alternative is to take samples of the sharp object from neighboring pixels. This blurs the in-focus object, but lowers its noise.

**Addressing artifact 2.** This case is more straightforward: when there are many samples in focus in the center pixel, blurry objects in the background should lose influence. Almost all approximate depth-of-field algorithms somehow deal with this effect.

**The key idea in this paper.** At first glance the proposed solutions to the two artifacts seem at odds: "blur sharp objects when there are blurry objects in front" versus "turn off blurry objects when there are sharp objects in front." However, if you approach each of these with weights rather than binary decisions they can be made similar. This gives rise to the key idea in this paper: *The filter used for each sample should be a combination of its denoising filter and the denoising filters of the samples in front of it*. A sharp object in isolation will tend to use a narrow filter, but as more blurry samples are in front of it, it will tend to use a wider filter. A blurry background object will tend to use a wide filter, but as more samples in front of it use a narrow filter, it will tend to use a narrow filter. Algorithm 2 shows an approach that has these properties. Here $D$ is the width of the wide filter, and the narrow filter is the default filter used in unblurred regions. When all the samples have the same diameter, the algorithm behaves as desired and either a narrow or wide blurring diameter is used for all samples. When we have the case of blurry samples in front of sharp samples, a filter with more wide weight is used for the sharp samples as desired. When some of the center pixels are sharp (use $f_n$) then the contribution of blurry background pixels is diminished. Figure 3 shows the output of Algorithm 2. The flaw of Algorithm 2 is that the binary "if" in line 10 can produce a visual break in objects that go in and out of focus. This can be solved by blending between the "if" clauses in the spirit of MIP-maps, or by adding clauses for additional intermediate scales, e.g. for filter widths of 1, 3, 5, and 7 pixels. In practice we have found both approaches to work well. See Appendices for source code.

For depth of field, the same reconstruction can be computed more efficiently, with no sorting and less computation. Depth of field separates samples into three layers: a widely filtered (out of focus) layer near the camera, then a narrowly filtered (in focus) layer near the focal plane, and a widely filtered layer farther away. The boundary layers are at constant depths, the depths where the circle of confusion crosses the threshold value. We observe that the output of Algorithm 2 is unaffected by permuting samples within a layer. Thus we can loop over the unsorted samples, accumulating the sums of $f_n$, $f_w$, $f_n * c_i$, and $f_w * c_i$ separately for the samples in each layer, and finally combine these sums front-to-back to arrive at the identical pixel value as before. For blending, samples near a layer boundary may contribute proportionally to both layers. Efficiently identifying layers in the presence of motion blur or transparency is a topic for future research.

**Algorithm 2** New reconstruction

```
1:  sort all samples that can influence this pixel, in depth
2:  rgb c = (0, 0, 0);
3:  float w = 0;
4:  float a_n = 0; // accumulated narrow contribution
5:  float a_w = 0; // accumulated wide contribution
6:  for each sample i in front-to-back order do
7:      compute d_i from z_i or other inputs
8:      f_n = f(x_i − x_c, y_i − y_c);
9:      f_w = (1/D^2) * f((x_i − x_c)/D, (y_i − y_c)/D);
10:     if d_i < D then
11:         w_i = (1 − a_w) * f_n + a_w * f_w;
12:         a_n += (1 − a_w) * f_n;
13:     else
14:         w_i = (1 − a_n) * f_w + a_n * f_n;
15:         a_w += (1 − a_n) * f_w;
16:     end if
17:     w += w_i;
18:     c += w_i * c_i;
19: end for
20: c_pixel = c/w;
```

## 4 Results

We developed the algorithm in a simple stochastic ray tracer and then ported it to three existing stochastic rasterizers used for different projects. Figures 1 and 4 were generated using the system described by McGuire et al. [2010]. The noise is somewhat higher than in the other figures due to the underlying regular sampling of that system. This emphasizes that our algorithm is agnostic about the sampling pattern used to generate the image, but the quality of the results does depend on the sampling patterns used. These pictures used 1x1 and 5x5 tiles and blending between the "if" statements (see Appendix A for the code segment).

Figure 5 was generated in a general software stochastic rasterizer similar in spirit to that of Fatahalian et al. [2009]. It shows an extreme case that is difficult for algorithms that use a pinhole camera, and something of a worst case for our algorithm: an in-focus object of high saliency behind a fuzzy object. Notice the face of the fairy is blurry compared to the reference solution; this is a conscious choice of our algorithm in preference to noise. This figure also shows a variation that biases the transition in favor of small scales when the scales are mixed. This emphasizes that our method has the flexibility for artistic control.

Figure 6 shows a stochastic transparency image generated by the system described by Enderton et al. [2010] where each sample's "width" increases with transparency. Again this is something of a difficult case for our algorithm, because the correct tradeoff of blurring versus noise is not as obvious a choice when transparency is not an intrinsically a blurry effect. Thus our method is probably best suited to handling transparency in effects such as smoke.

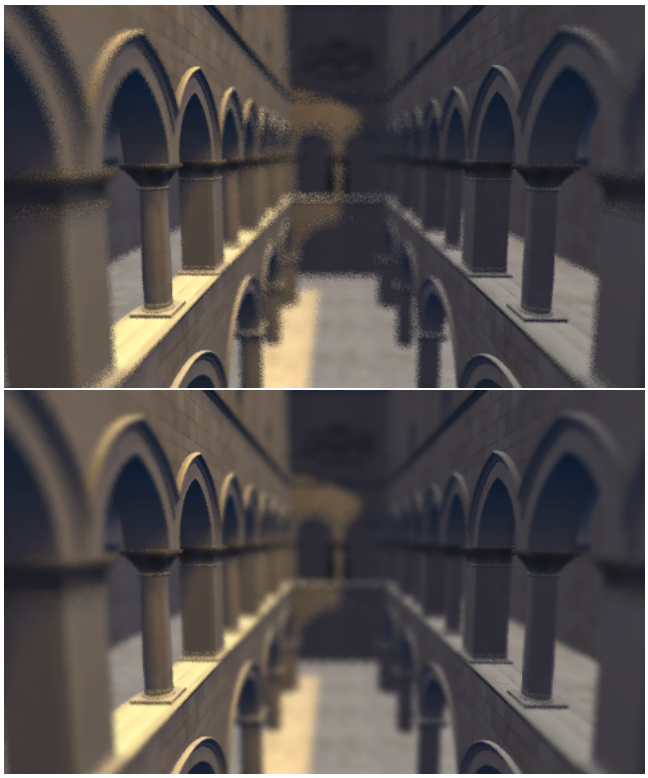

**Figure 4:** *Cathedral with (top) 1x1 box filter and (bottom) the new algorithm. 16 samples per pixel.*

**Figure 5:** *In reading order: Pinhole camera, 16 spp box filtered, new algorithm, reference solution, closeup of new method, closeup of new method with each $w_i$ raised to the power 1.3 to reduce blurring. Inset magnified region includes the fairy's ear.*

## 5 Discussion

We have presented a simple algorithm for selectively adding blur to reduce noise in stochastic renderings. We have shown results for motion and defocus blur as well as stochastic transparency. The algorithm is designed for interactive stochastic rasterization, and may also be useful for preview for film rendering.

Many avenues remain for future work. One is to explore how artistic control should be exposed in the transition function. Another is to determine whether special purpose hardware is needed to process the sample buffer, or whether it is sufficient to supply a programmable sample resolve. We have noted that sometimes blur is better than noise, but exactly when this is true is not well understood, and a psychophysics model could be beneficial. Finally, we note that we have presented an existence proof of a simple and efficient algorithm for reconstructing pixels from stochastic samples; this is a surprisingly sparsely explored area and there may be much better algorithms yet to be discovered.

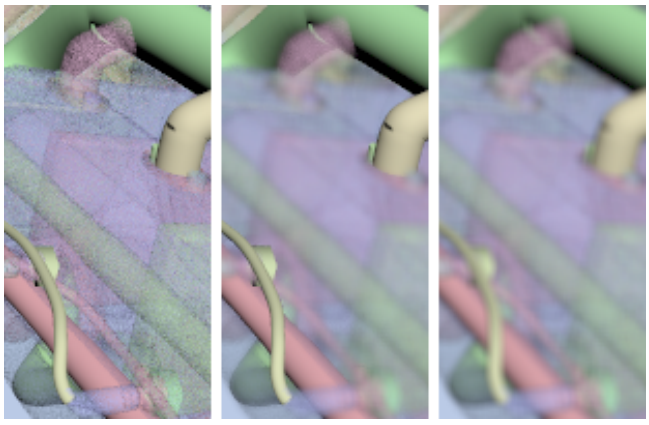

**Figure 6:** *Stochastic transparency rendering with the same 16 samples per pixel filtered in three ways. Left: One pixel box filter is sharp but noisy. Right: Five pixel box filter is blurry everywhere. Center: The new algorithm is sharp in opaque regions but blurry in noisy regions.*

## References


BOULOS, S., EDWARDS, D., LACEWELL, J. D., KNISS, J., KAUTZ, J., SHIRLEY, P., AND WALD, I. 2007. Packet-based Whitted and distribution ray tracing. In *Proc. GI '07*, 177–184.

EGAN, K., TSENG, Y., HOLZSCHUCH, N., DURAND, F., AND RAMAMOORTHI, R. 2009. Frequency analysis and sheared reconstruction for rendering motion blur. *ACM Trans. Graph. 28,* 3.

ENDERTON, E., SINTORN, E., SHIRLEY, P., AND LUEBKE, D. 2010. Stochastic transparency. In *Proc. I3D*, 157–164.

FATAHALIAN, K., LUONG, E., BOULOS, S., AKELEY, K., MARK, W., AND HANRAHAN, P. 2009. Data-parallel rasterization of micropolygons with defocus and motion blur. In *Proc. HPG*, 59–68.

FILION, D., AND MCNAUGHTON, R. 2008. Starcraft II: Effects and techniques. In *SIGGRAPH Course Notes, Advanced Real-Time Rendering in 3D Graphics and Games*, 133–164.

HEINZLE, S., WOLF, J., KANAMORI, Y., WEYRICH, T., NISHITA, T., AND GROSS, M. 2010. Motion blur for EWA surface splatting. In *Proc. Eurographics*, 733–742.

JENSEN, H., AND CHRISTENSEN, N. 1995. Optimizing path tracing using noise reduction filters. In *Proceedings of WSCG*, vol. 1995.

KONTKANEN, J., RÄSÄNEN, J., AND KELLER, A. 2004. Irradiance filtering for Monte Carlo ray tracing. *Monte Carlo and Quasi-Monte Carlo Methods 2004*, 259–272.

KOSLOFF, T., TAO, M., AND BARSKY, B. 2009. Depth of field postprocessing for layered scenes using constant-time rectangle spreading. In *Proc. GI '09*, 39–46.

KRIVANEK, J., ZARA, J., AND BOUATOUCH, K. 2003. Fast depth of field rendering with surface splatting. In *Proc. CGI*, 196–201.

LEE, S., EISEMANN, E., AND SEIDEL, H. 2009. Depth-of-field rendering with multiview synthesis. *ACM Transactions on Graphics (TOG) 28*, 5, 1–6.

MCGUIRE, M., ENDERTON, E., SHIRLEY, P., AND LUEBKE, D. 2010. Hardware-accelerated stochastic rasterization on conventional GPU architectures. In *Proc. HPG*, 77–89.

MEYER, M., AND ANDERSON, J. 2006. Statistical acceleration for animated global illumination. *ACM Trans. Graph. 25*, 3.

OVERBECK, R., DONNER, C., AND RAMAMOORTHI, R. 2009. Adaptive wavelet rendering. *ACM Trans. Graph. 28*, 5.

ROBISON, A., AND SHIRLEY, P. 2009. Image space gathering. In *Proc. HPG*, 91–98.

RUSHMEIER, H., AND WARD, G. 1994. Energy preserving non-linear filters. In *Proc. SIGGRAPH*, 131–138.

SCHEUERMANN, T. 2004. Advanced depth of field. In *GDC*.

SOLER, C., SUBR, K., DURAND, F., HOLZSCHUCH, N., AND SILLION, F. 2009. Fourier depth of field. *ACM Trans. Graph. 28*, 2.

WARD, G., RUBINSTEIN, F., AND CLEAR, R. 1988. A ray tracing solution for diffuse interreflection. In *Proc. SIGGRAPH*, 85–92.

XU, R., AND PATTANAIK, S. 2005. A novel Monte Carlo noise reduction operator. *IEEE CG&A 25*, 2, 31–35.

## A Two Scale Code Sample

This algorithm was used for Figure 1 (cars) and blends between two scales: 1x1 and 5x5 blocks. Here $N$ is the number of samples in the widest tile, and *ns* is the number of samples in a pixel. As a default, *denoisingblur* is the maximum of the circle of confusion diameter and, if motion blur is to be smoothed, the length of the projected motion vector in pixels.

```
float box(float x, float radius) {
    return fabs(x) < radius ? 0.5/radius : 0; }

// process samples front-to-back
qsort(s, N, sizeof(*s), compare_sample_z);
float cov_1 = 0.0;
float cov_5 = 0.0;
float cov = 0.0;
rgb sum(0,0,0);
for (int i = 0; i < N; i++) {
    // user programmable "denoising" blur diameter
    float D = s[i].denoisingblur;
    float w_1 = (1/ns)*box(s[i].x,0.5)*box(s[i].y,0.5);
    float w_5 = (1/ns)*box(s[i].x,2.5)*box(s[i].y,2.5);
    float w;
    float blend = clamp((D-1)/5, 0, 1);
    w      =    blend  * (w_5*(1 - cov_1) + w_1*cov_1);
    cov_5 +=    blend  * (w_5*(1 - cov_1));
    w     += (1-blend) * (w_1*(1 - cov_5) + w_5*cov_5);
    cov_1 += (1-blend) * (w_1*(1 - cov_5));
    cov += w;
    sum += w * s[i].color;
}
image->setPixel(x, y, (1/cov)*sum);
```

## B Four Scale Code Sample

When the block sizes are 1x1 and 7x7, there is a visible jump between the scales, so adding the intermediate scales 3x3 and 5x5 is an improvement. This algorithm was used for Figure 5 (fairy) where the extreme depth of field made noise more obvious and wider maximum blurring more beneficial. Blending between the scales as in Appendix A could be done, but we have not found it to be necessary in practice.

```
qsort(s, N, sizeof(*s), compare_sample_z);
float cov_1 = 0.0;
float cov_3 = 0.0;
float cov_5 = 0.0;
float cov_7 = 0.0;
float cov = 0.0;
rgb sum(0,0,0);
for (int i = 0; i < N; i++) {
    float D = s[i].denoisingblur;
    float w_1 = (1/ns)*box(s[i].x,0.5)*box(s[i].y,0.5);
    float w_3 = (1/ns)*box(s[i].x,1.5)*box(s[i].y,1.5);
    float w_5 = (1/ns)*box(s[i].x,2.5)*box(s[i].y,2.5);
    float w_7 = (1/ns)*box(s[i].x,3.5)*box(s[i].y,3.5);
    float w;
    if (D > 7) {
        w =      w_7*(1 - cov_1 - cov_3 - cov_5) +
                 w_1*cov_1 + w_3*cov_3 + w_5*cov_5;
        cov_7 += w_7*(1 - cov_1 - cov_3 - cov_5);  }
    else if (D > 5) {
        w =      w_5*(1 - cov_1 - cov_3 - cov_7) +
                 w_1*cov_1 + w_3*cov_3 + w_7*cov_7;
        cov_5 += w_5*(1 - cov_1 - cov_3 - cov_7);  }
    else if (D > 3) {
        w =      w_3*(1 - cov_1 - cov_5 - cov_7) +
                 w_1*cov_1 + w_5*cov_5 + w_7*cov_7;
        cov_3 += w_3*(1 - cov_1 - cov_5 - cov_7); }
    else {
        w =      w_1*(1 - cov_3 - cov_5 - cov_7) +
                 w_3*cov_3 + w_5*cov_5 + w_7*cov_7;
        cov_1 += w_1*(1 - cov_3 - cov_5 - cov_7); }
    cov += w;
    sum += w * s[i].color;
}
image->setPixel(x, y, (1/cov)*sum);
```