preprint (HPG 2010)

Edge-Avoiding À-Trous Wavelet Transform for fast Global Illumination Filtering

Holger Dammertz, Daniel Sewtz, Johannes Hanika, Hendrik P.A. Lensch

Ulm University, Germany

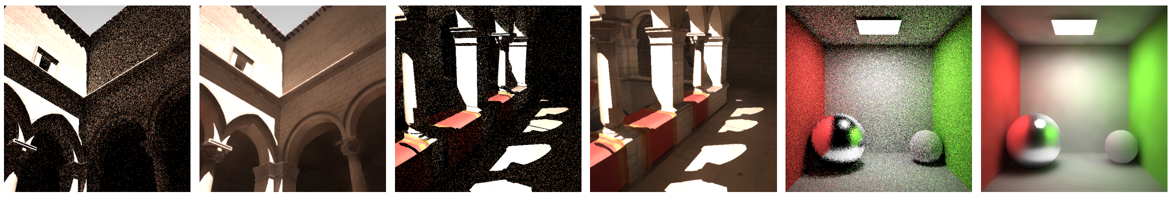

Figure 1: *Using our edge-avoiding À-Trous wavelet transform we filter highly noisy path traced images at interactive rates resulting in smooth indirect illumination while retaining important detail like sharp shadows and hard edges. The images show the (noisy) input into our algorithm and next to them the output we compute.*

Abstract

We present a fast and simple filtering method designed for ray traced Monte Carlo global illumination images which achieves real-time rates. Even on modern hardware only few samples can be traced for interactive applications, resulting in very noisy outputs. Taking advantage of the fact that Monte Carlo computes hemispherical integrals that may be very similar for neighboring pixels we derive a fast edge-avoiding filtering method in screen space using the À-Trous wavelet transform that operates on the full noisy image and produces a result that is close to a solution with many more samples per pixel.

1. Introduction

Monte Carlo-based global illumination ray tracing is one of the most flexible and robust light transport simulation algorithms. Although there exist many different approaches, they all use random path sampling to solve the integral in the rendering equation. Usually hundreds of paths per pixel are necessary to achieve a smooth image. But even modern computers and algorithms can only provide a few paths per pixel in the time-frame needed for interactive applications or a fast preview. Limiting the number of randomly sampled paths results in an image that still contains highly visible noise.

Even in conceptually simple illumination situations like a diffuse uni-colored wall, the noise is visible. But in this case the correct solution varies smoothly from pixel to pixel and the result can be improved by combining several neighboring samples into one integral estimate.

Edge-avoiding wavelets provide an efficient approach for filtering such samples, provided that a good edge-stopping functions is available. But, as we show in Section 3.3, the standard decimated wavelet bases are poorly suited for the Monte Carlo noise and show visible artifacts. On the other hand the bilateral filter avoids these artifacts but can be quite slow for large filter sizes.

Therefore we extend the fast undecimated À-Trous wavelet transform to incorporate multiple edge-stopping functions and show that it can robustly filter highly noisy Monte Carlo global illumination images and produce visually pleasing results at interactive rates. We use a hybrid technique that supplements the output of a CPU based ray tracer with information from additionaly rasterized images. Our GPU shader applies the Edge-Avoiding À-Trous filter combining the different buffers to produce the output image.

Our main contributions in this paper are:

- the interactive edge-avoiding À-Trous filter
- an edge-stopping function (depending on edges in geometry and illumination) for noisy path traced images
- comparison to other (edge and non edge-avoiding) wavelet bases for reconstruction of Monte Carlo images.

This results in the ability to produce path traced global illumination images of visually smooth appearence at interactive rates that retain high detail (like e.g sharp shadows).

1.1. Related Work

Edge-Avoiding Wavelets and the Bilateral Filter Naive computation of the discrete wavelet-transform requires convolutions with increasing filter sizes per level. The *Mallat algorithm* [Mal89] alleviates this problem by introducing *decimation* (i.e. downsampling) and illustrates the relationship to multiresolution analysis. Efficient computation of the undecimated transform has been proposed by [HKMMT89, Mal98] adapting *Fast Filter Transforms* [Bur81] for the use with wavelets. These filters contain a constant number of coefficients on each level of the analysis, by spreading the initial filter by a factor of 2^{level} and filling in zeroes. In the wavelet context this is known as the *algorithme À-Trous* ("with holes"). [HKMMT89] provides a formal analysis of the relationship between the Mallat algorithm and the À-Trous algorithm.

The *bilateral filter* is a non-linear filter, proposed by [TM98] (among others), to smooth images. Since it is computationally expensive, methods for its acceleration have been developed (e.g. [DD02, PD09]). [FAR07] uses the À-Trous algorithm to compute a *fast multiscale bilateral decomposition*. Edge-avoiding wavelets [Fat09] adopt this for the decimated case by extending second-generation wavelets [Swe97] and introducing an edge-crossing function. Edge-avoiding wavelets (computed via the Mallat-algorithm) are the decimated counterpart of the fast bilateral decomposition (computed via the À-Trous algorithm). In contrast to the many specialized and selective wavelet bases, the shape of each edge-avoiding wavelet itself changes data dependently similar to a data dependent windowing in the short time fourier-transform.

The bilateral filter has been extended for different applications to use more information in the edge-stopping function. The *trilateral filter* [CT05] uses an adaptive neighborhood function and the image gradient. The *joint* and *cross bilateral filter* [PSA*04, ED04] use the color information from multiple images taken under different lighting conditions to combine their freguency components. [SGNS07] uses bilateral upsampling from a low resolution low frequency light buffer for *real-time soft global illumination*. But a low resolution buffer lacks the positional precision necessary to decide on which side of an edge a sample lies thus removing the possibility of retaining high resolution details like sharp shadows. For this reason our *edge-avoiding À-Trous* filter operates on a full resolution buffer. It combines edge-stopping functions from multiple input images (noisy ray tracing image, normal buffer, position buffer) and is efficiently computed on the GPU.

Fast Ray Tracing The starting point of our algorithm is a ray tracer that we use for the light transport simulation. In recent years several approaches for fast ray tracing have been developed. The fastest class of algorithms exploits coherence for tracing primary and shadow rays [Wal04]. They are however limited to very basic light transport simulation. For full global illumination light transport simulation as described in the next paragraph, no such coherence exists. Two recent approaches to accelerate this kind of computation are to reintroduce coherence by reordering [SBB08] or to ignore coherence and build faster data-structures [DHK08, EG08]. [Tsa09] combines both approaches.

Global Illumination Real time display of smooth global illumination solutions can be achieved using precomputation. The most prominent algorithms are Precomputed Radiance Transfer [SKS02] and Meshless Radiosity [LZT*08]. Based on interactive ray tracing on distributed systems and interleaved sampling in [WKB*02] an approach for interactive global illumination was presented that uses the discontinuity buffer [Kel98] to avoid filtering across edges.

Another approach builds on the Instant Radiosity [Kel97] algorithm that approximates the global illumination with many point lights [DS06, LSK*07, NW09]. [RGK*08] also approximate the visibility of the point light sources. These methods, like most GPU methods, have restrictions like the shape of the light sources and lack the generality of a full ray tracing solution.

Finally the advancement in GPU computing power allows to implement previous offline algorithms like photon mapping [Jen96] or final gathering in realtime [FD09, ML09, REG*09, WWZ*09].

There exist a vast mount of general (offline) techniques for Monte Carlo global illumination. Recently the use of wavelets in the context of adaptive sampling has proven to be useful [ODR09]. We discuss more of the related Monte Carlo work in Section 2.1.

1.2. Overview

Our method takes a noisy Monte Carlo path traced full resolution image and produces a smooth output image by taking into account the input image itself as well as a normal and a position buffer. Using the ray traced image as input allows to retain high frequency detail like sharp shadows that are not available in the other buffers. The resulting image shows sharp edges around geometry borders and successfully smoothes the illumination where needed. Using a general path tracing algorithm allows for various different light transport effects like glossy materials and different light source types and shapes (e.g. simple point lights, arbitrary (deformed) area lights, image based lighting), without special handling.

Section 2 describes stages of our algorithm and gives the necessary theoretical background. Section 3 discusses our implementation with optimizations and presents the performance and results.

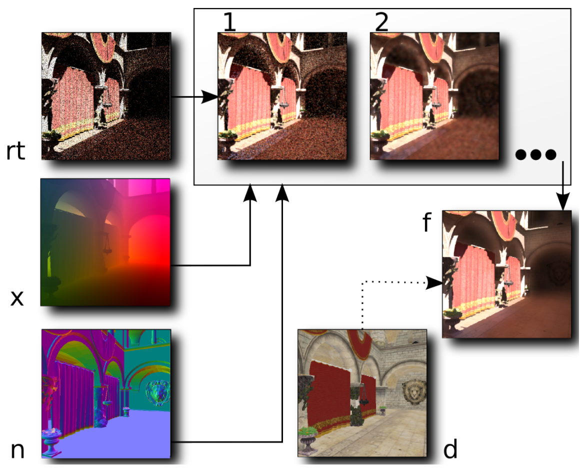

Figure 2: rt), n) and x) *are the input buffers (Ray Tracing, Normal, World Space Position).* 1), 2) *show 2 levels of our edge-avoiding À-Trous filter.* f) *is the final output image after tone mapping.* d) *shows the optional diffuse buffer*

2. Edge-Avoiding À-Trous Filtered Path Tracing

Figure 2 illustrates the basic steps of our algorithm. The input is the path-traced (noisy) image (rt buffer), containing in the simplest case both direct and indirect illumination. Additionally, the edge-stopping function of the À-Trous filter takes into account the normal and the position buffer. The number of applications of the À-Trous filter is determined by the total desired filter size (in our case 5 iterations i.e. 80×80).

In the absence of edges (e.g. all buffers uni-colored) it applies an even smoothing with increasing filter size per iteration. In the presence of edges the influence of neighboring samples is diminshed by the edge-stopping function. This is closely related to the bilateral filter and was used with the À-Trous algorithm in [FAR07] and with decimated wavelet bases in [Fat09].

In the following sections we will lay out the theory of the individual parts of the algorithm and in Section 3 we will describe the implementation specific details and compare various options for performance improvement.

2.1. Monte Carlo Light Transport

The Monte Carlo Method for light transport solves the integral of the rendering equation [Kaj86] by constructing random paths that connect the camera with the light sources in the scene. This is the most general known solution for the full rendering equation and is well studied in literature [Vea97, PH04]. One problem with Monte Carlo image synthesis is the high variance that is visible even in simple scenes. To reduce this variance quadratically more samples are needed for a linear improvement.

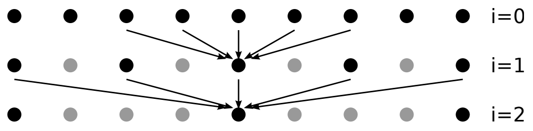

Figure 3: *Three levels of the À-Trous wavelet transform. Arrows indicate pixels that correspond to non-zero entries in the filter h_i and are used to compute the center pixel at the next level. Gray dots are positions that the full undecimated wavelet transform would take into account but that are skipped by the À-Trous algorithm.*

In our system we use a path tracer (PT) with next event estimation as our basic image formation algorithm. This path tracer starts by tracing rays from the camera into the scene as a random walk. At each interaction point a connection to a randomly chosen position on a light source is made. If it is not occluded the contribution of this path is added to the image accumulation buffer.

In the next section we will describe a smoothing process to reduce the variance of the final ray traced image. This is motivated by the following observation: The incident irradiance at a single point on a surface is described by the integral over the hemisphere. Under interactive or real-time constraints a path tracer can only trace a single path per pixel thus estimating the integral with a single sample only. But if neighboring hemispheres are similar one would expect similar integrals. Therefore the smoothing tries to average samples with a similar hemisphere.

2.2. Edge-Avoiding À-Trous Filter

In this section we will first describe the (unweighted) À-Trous wavelet transform and then extend it to incorporate our edge-stopping function. Each iteration of the À-Trous algorithm (with increasing step width) corresponds to computing one more level of the wavelet analysis which also corresponds to doubling the filter size.

[Bur81] shows that wide gaussian filters can be well approximated by repeated convolution with *generating kernels*. [HKMMT89] uses this construction to compute the discrete wavelet transform, known as the *algorithme À-Trous* :

1. At level $i = 0$ we start with the input signal $c_0(p)$
2. $c_{i+1}(p) = c_i(p) * h_i$, where $*$ is the discrete convolution. The distance between the entries in the filter h_i is 2^i.
3. $d_i(p) = c_{i+1}(p) - c_i(p)$,
   where d_i are the detail or wavelet coefficients of level i.
4. if $i < N$ (number of levels to compute):
   increment i, go to step 2
5. $\{d_0, d_1, ..., d_{N-1}, c_N\}$ is the wavelet transform of c.

The reconstruction is given by

$$c = c_N + \sum_{i=N-1}^{0} d_i \tag{1}$$

Filter h is based on a B_3 spline interpolation $(\frac{1}{16}, \frac{1}{4}, \frac{3}{8}, \frac{1}{4}, \frac{1}{16})$ [Mur97]. At each level $i > 0$ the filter doubles its extent by filling in 2^{i-1} zeros between the initial entries. Thus the number of non-zero entries remains constant. (see Figure 3)

Edge-avoiding filtering is achieved by introducing a data-dependent weighting function. We extend the intensity-based edge-stopping function of the bilateral filter to combine multiple edge-stopping functions. The discrete convolution in step 2 becomes

$$c_{i+1}(p) = \frac{1}{k} \sum_{q \in \Omega} h_i(q) \cdot w(p,q) \cdot c_i(p) \tag{2}$$

with weight function w, pixel positions p and q and Ω the positions of non-zero coefficients in h_i.
k is the normalization factor

$$k = \sum_{q \in \Omega} h_i(q) \cdot w(p,q) \tag{3}$$

and w are the combined edge-stopping functions from the raytraced input image (rt), normal buffer (n) and position buffer (x)

$$w(p,q) = w_{rt} \cdot w_n \cdot w_x \tag{4}$$

$$w_{rt}(p,q) = e^{\left(-\frac{||I_p - I_q||}{\sigma_{rt}^2}\right)} \tag{5}$$

where I_p and I_q are the color values of the rt buffer at positions p and q. w_n and w_x are computed in the same way with their own σ_n and σ_x respectively.

We implement this filter in GLSL (see the end of the paper for source code) passing the stepwidth (i.e. number of zeroes between filter entries), filter h, parameters σ_{rt}, σ_n, σ_x as uniforms and images rt, n, x as textures.

Since it is hard to make assumptions about the noise contained in the raytraced image, we choose our inital σ_{rt} to include variations of the scale of the maximum intensity of the raytraced image. Note that the intensities can be bigger than 1 in our full-hdr environment. Thus the edge-stopping function depends at the first level on w_n and w_x only. At each pass we set σ_{rt}^i to $2^{-i}\sigma_{rt}$ thus allowing for smaller illumination variations to be smoothed.

Edges that are preserved by our edge-stopping function are still present at coarser levels of the transformation. Therefore we discard the finer levels of the wavelet transform, rendering step 3 of the À-Trous algorithm unnecessary, and directly use level N as output image.

3. Implementation Details and Results

We implemented the edge-avoiding À-Trous filter in GLSL. After computing the input buffers (rt, normal, position) the filter is applied multiple times to the rt buffer. Each pass uses the previously smoothed result as input. All our buffer textures use a 16 bit floating point format.

The normal and position buffers are rasterized with OpenGL using frame buffer objects and passed to the smoothing shader via textures. Even though these buffers could easily be computed using the ray tracer we conserve bandwidth by computing them directly on the GPU. We directly display the final smoothed rt output buffer, or in the case of deferred shading, multiply the result with an OpenGL generated diffuse color buffer.

As ray tracer we use a rather slow but very flexible CPU implementation. As Table 1 shows this is the bottleneck of our system and could be improved by using a more specialized ray tracer. Nevertheless we achieve interactive frame rates with our system.

If not otherwise noted, performance measurements and comparisons were computed on an Intel(R) Core(TM)2 Duo CPU (E6850) @ 3.00GHz with an NVIDIA GeForce GT 9500 GPU.

Deferred Shading Deferred shading [Hag04] postpones the illumination computation at each pixel until a final rendering pass per light source. This pass takes a position, normal and diffuse buffer as input. We can apply this to our approach by using the ray tracer to compute the incident illumination at each visible pixel without the final material evaluation. This light buffer can then be smoothed using our À-Trous filter and be applied by multiplying it with the diffuse buffer. This allows to retain high detail in textures that would otherwise be blurred but can only be applied to diffuse surfaces.

3.1. GPU Accelerated Implementation and Results

Table 1 shows the performance of each individual part of our system for all scenes depicted in Figure 12. This measurement was done at a resolution of 512×512 with a single path per pixel and one indirect bounce. For each path, 4 rays are traced (primary and secondary plus two shadow rays). The measurements were averaged over 16 iterations for the shown camera perspective. The À-Trous filter was applied five times. As expected the À-Trous filter performance is independent of the input data and very fast.

The filter can be adapted to different kinds of applications by specifically extending it to evaluate more input weight buffers. This is illustrated in Figure 4 for zero to three weights. The impact on the performance is shown in Table 2. This table also shows the performance of the À-Trous GPU filter when changing the resolution. Figure 5 shows a sharp shadow of a pole being preserved correctly while smoothing the other parts.

The time spent on the OpenGL rendering varies extensively with scene complexity and number of textures. Depending on application and input data it might be more ef-

Figure 4: *À-Trous filter with increasing number of edge weights: unfiltered rt buffer (input), no additional weights: ("pure" À-Trous), rt buffer only: σ_{rt} (bilateral approx.), two buffers: σ_{rt}, σ_n and all three buffers: $\sigma_{rt}, \sigma_n, \sigma_x$ (our result)*

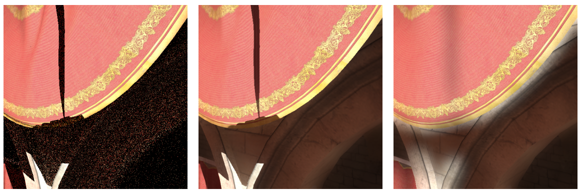

Figure 5: *From left to right: Close up of the rt-input of a pole shadow in the sponza scene. Result with our presented weighting approach. Result without rt buffer: the shadow is blurred.*

Scene	RT (ms)	GL (ms)	Upload (ms)	À-Trous (ms)	FPS
Box	307.6	2.9	5.8	78.5 (5.6)	3.2
Sponza	835.2	35.5	4.4	78.6 (5.6)	1.13
Sibenik	510.2	8.5	5.2	78.6 (5.6)	1.9
Outdoor	316.4	2.6	6.3	78.6 (5.6)	3.0

Table 1: *Individual timings for the test scenes shown in Figure 12 at a resolution of 512×512. RT is the total time to produce the input ray tracing buffer, GL the time to generate the position, normal and diffuse buffer using OpenGL. Upload is the time it takes to copy the ray tracing buffer from main memory to the GPU. À-Trous gives the time of five repeated applications of our filter on a GT 9500. The number in brackets is the time on an NVIDIA GTX 285. FPS are the final frames per second achieved by our testing system.*

Res.	512×512			1024×1024			1920×1080		
# Iter	1	5	10	1	5	10	1	5	10
# W									
0	0.3	1.5	2.9	1.0	5.0	11.3	1.9	9.6	22.4
1	0.5	2.4	4.7	1.8	8.7	17.2	3.5	16.9	33.6
2	0.8	3.5	6.9	2.7	12.9	26.6	5.0	25.1	52.1
3	1.2	5.6	11.0	4.2	20.9	41.6	8.2	42.6	86.0

Table 2: *Time in ms to compute the À-Trous filter on a NVIDIA GTX 285 at different resolutions (Res.). $5 \cdot 2^{\#Iter}$ corresponds to the total filtersize and # W is the number of weights. The original À-Trous algorithm uses 0 weights. Our implementation uses 3 weights. See Figure 4 for results with fewer weights.*

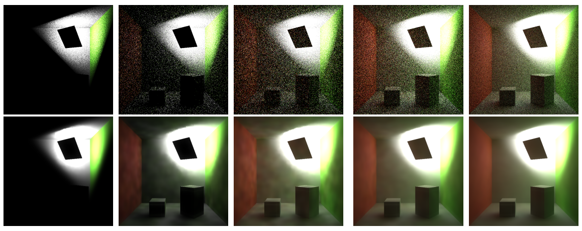

Figure 6: *The top row shows the input, the bottom row the output. The first three columns show increasing number of bounces from 0 (direct light) to 1 and 5. The last two columns show the improvement when tracing more than one path per pixel (4 and 16 paths).*

ficient to produce the required normal, position and diffuse buffer with the ray tracer and upload them to the GPU.

Increasing Bounces and Samples per Pixel The number of bounces can easily be increased. While slowing down the ray-tracing it allows for a more correct global illumination solution. The first three images in Figure 6 show results with more bounces. To reduce the variance in the input buffer, more than one path per pixel can be traced. This is illustrated in the last two image columns in Figure 6 with 4 and 16 paths per pixel.

Changing Scene Parameters Since we use a general ray tracer it is easy to change for example the glossyness of materials or the size and shape of light sources. The effects of this are illustrated in the Figures 7 and 8.

3.2. Optimizations

In this section we discuss some possibilities to speed up a full rendering system that uses our filter by specializing the rendering algorithm.

Sub-Sampling the image While it would be possible to use a dense low resolution ray tracing buffer, one of the main advantages of our À-Trous filtering would be lost. Namely that in the full resolution edge buffer each sample location knows

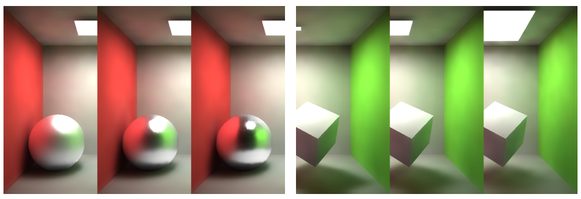

Figure 7: *Varying the glossiness of the sphere and the size of the light source.*

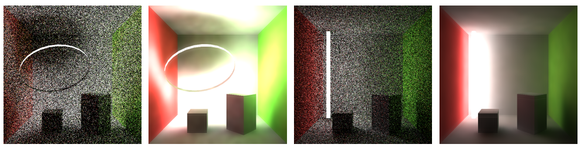

Figure 8: *Arbitrarily shaped light sources. The images show the rt input buffer and the filtered output with two bounces.*

on which side of the edge it lies. We can however reduce the number of rays by sampling only every $n-$th pixel. Now the first steps of the filtering are related to an in-painting problem. This of course increases the variance of the final image even more and some features like hard shadows may be lost. Since in our system the major bottleneck is the ray tracer we get an almost linear speedup with the reduction of traced rays. Figure 9 shows results for sub-sampling. Note that for correct in-painting, the rt buffer should not be used as edge weight for the first iterations but only when the image is fully in-painted. For 2×2 sub-sampling this is after one iteration, for 4×4 after two.

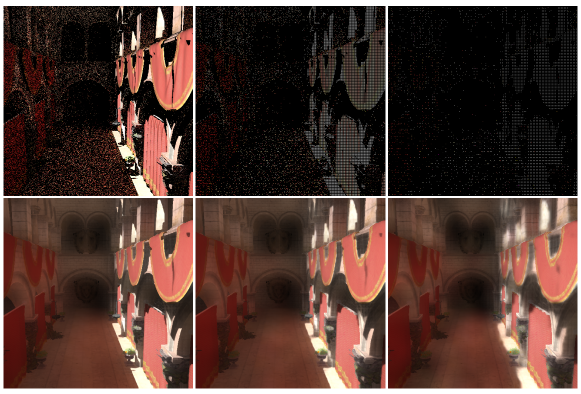

Figure 9: *Sub-sampling the rt buffer. The top row shows the input using one sample per pixel* (1.1 *fps), one sample every* 2×2 *pixels* (3.8 *fps) and one sample every* 4×4 *pixels* (11.0 *fps). The bottom row shows the result after filtering where the loss of detail is clearly visible.*

Avoiding Primary Rays Depending on bandwidth usage one can avoid tracing primary rays by downloading the depth or position buffer from the GPU. This might give a speedup when the path depth is low and the bandwidth is not a problem. In our case the resulting render times were slower by about 15%. This of course changes when a GPU ray tracer is used. In that case all data would already by available on the GPU.

Splitting Direct Illumination When the flexibility of arbitrary formed light sources is not needed the direct light computation can also be performed on the GPU using e.g. shadow maps or shadow volumes. This optimization is related to the previous one but now the first shadow ray can also be omitted. In our case the speed-up was almost a factor of 2 when using a single point light source in the sponza scene with one indirect bounce.

3.3. Comparison to Other Wavelet Bases

As compared to standard denoising techniques our method allows to discard the detail coefficients instead of shrinking them. In a high variance ray traced image a single bright pixel in a dark region may indicate a large smooth patch with the same mean irradiance. Such a pixel would result in a detail coefficient of very large magnitude which would not be sufficiently smoothed by wavelet shrinkage.

Figure 10 shows the comparison of our edge-avoiding À-Trous filter to other bases. As can be seen in the standard decimated wavelet transform (using CDF(2,2) and RedBlack wavelets [UB98]) filtering out detail coefficients of large magnitude leads to visible artifacts resulting from the decimation. The edge avoiding versions suffer from the same artifacts. See for example the EAW CDF(2,2) image where it fails to preserve the two edges between the light source and the wall.

Adaptive wavelet rendering [ODR09] takes advantage of the hierarchical nature of decimated wavelet bases. They adaptively refine the wavelet representation and sample the support of the wavelets. While it would be interesting to carry over this approach to undecimated bases, this is not straight forward because there is no implicit hierarchy anymore.

It would also be interesting to apply our filtering technique to other global illumination algorithms like micro-rendering [REG*09] or image space photon mapping [ML09].

3.4. Limitations

Highly complex scenes are problematic for our smoothing algorithm since one sample per pixel (rt buffer as well as position and normal buffer) no longer suffice to capture the necessary information. This results in the loss of details and

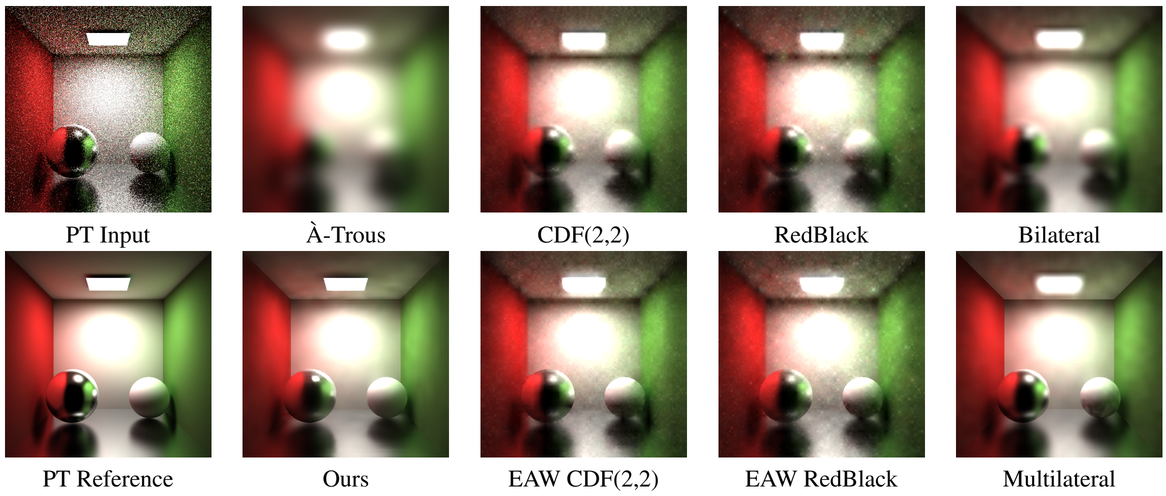

Figure 10: *Comparison to other wavelet bases for filtering noisy Monte Carlo images. The first row shows the result of filtering with the standard wavelet transform. The second row shows the edge avoiding versions (using our edge-stopping function). We also include the results of the bilateral and multilateral filter, where the multilateral version is the extension of the standard bilateral filter with our multiple edge-stopping function. Applying this filter through standard convolution however only allows to use fixed σ.*

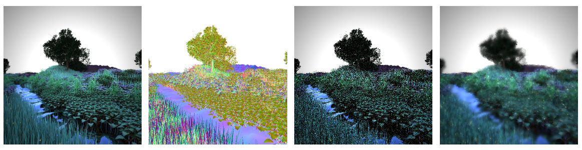

Figure 11: *Failure case for our algorithm: A nature scene with 20 Million triangles. Due to many complex shadow casting objects the illumination variance is very high and a single sample per pixel is insufficient. Additionally there is severe aliasing in the normal buffer.*

smoothing for example over shadows or object boundaries. Both problems are illustrated in Figure 11.

Another problem is that due to the edge-stopping function the À-Trous wavelet transform is no longer energy conserving. While this is acceptable in some situations it has to be considered for high quality offline rendering.

4. Conclusion and Future Work

In this paper we presented a novel filtering technique for highly noisy Monte Carlo global illumination images that can be used in interactive applications. By extending the À-Trous wavelet transform to incorporate an edge stopping function taking multiple buffers into account, we can successfully eliminate the noise while preserving sharp details.

Since our approach is very general in future work we plan to apply it to more advanced Monte Carlo global illumination algorithms like bi-directional path tracing and to investigate how a more sophisticated edge-stopping function could improve the results even further. Another interesting extension would be to incorporate adaptive sampling to avoid sampling in areas where the À-Trous filtering already gives a good result or to increase sampling in problematic regions . A possibility to fix the energy conservation would be to use an additional channel in the wavelet transform and post-normalize [Fat09] the output buffer. Using a ray tracer that operates fully on the GPU is likely to provide a significant speedup for a rendering system using our filter.

5. Acknowledgments

This work has been partially funded by the DFG Emmy Noether fellowship (Le 1341/1-1).

GLSL À-Trous fragment shader

```
uniform sampler2D colorMap, normalMap, posMap;
uniform float c_phi, n_phi, p_phi, stepwidth;
uniform float kernel[25];
uniform vec2 offset[25];

void main(void) {
 vec4 sum = vec4(0.0);
 vec2 step = vec2(1./512., 1./512.); // resolution
 vec4 cval = texture2D(colorMap, gl_TexCoord[0].st);
 vec4 nval = texture2D(normalMap, gl_TexCoord[0].st);
 vec4 pval = texture2D(posMap, gl_TexCoord[0].st);

 float cum_w = 0.0;
 for(int i = 0; i < 25; i++) {
  vec2 uv = gl_TexCoord[0].st + offset[i]*step*stepwidth;

  vec4 ctmp = texture2D(colorMap, uv);
  vec4 t = cval - ctmp;
```

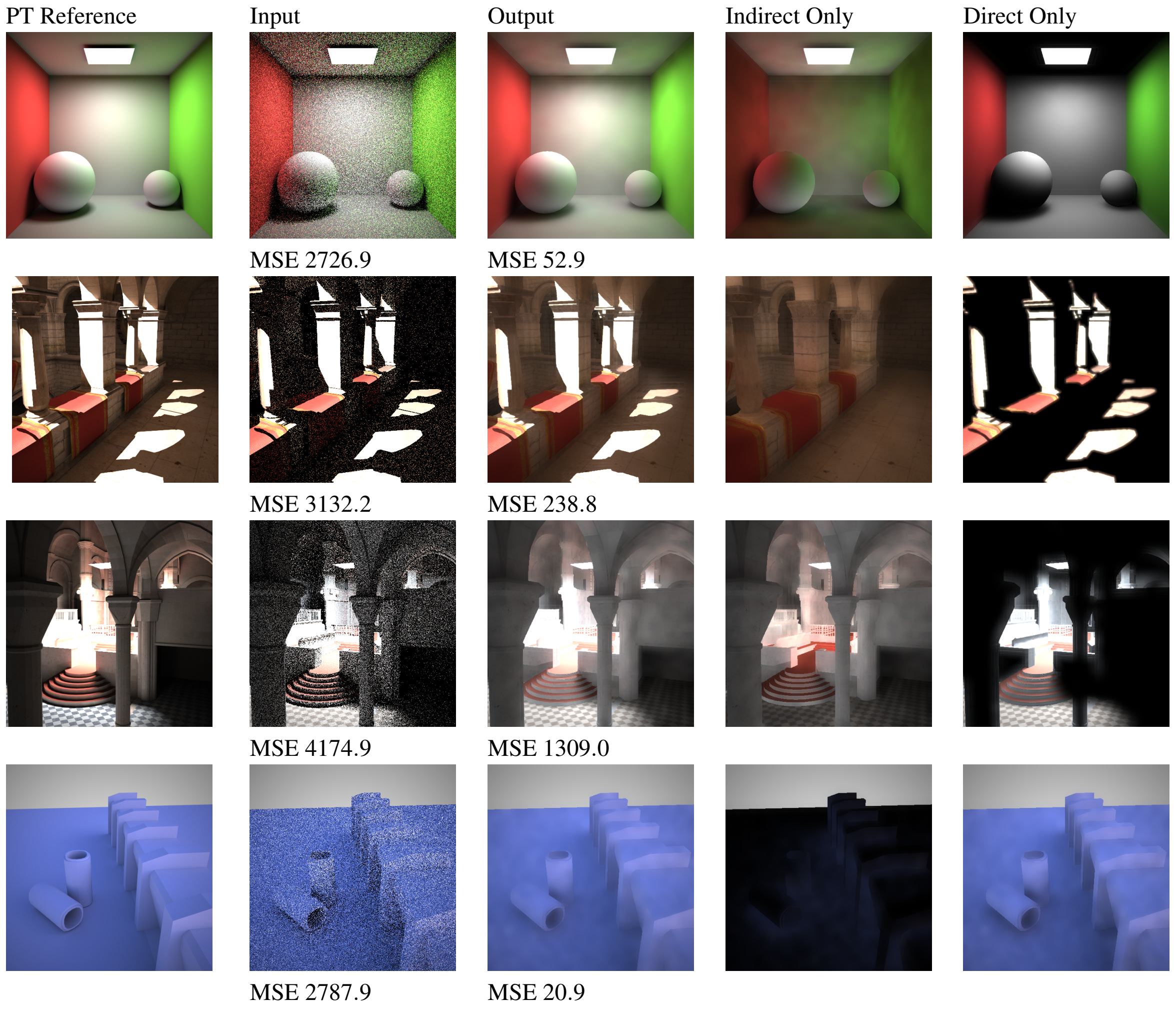

Figure 12: *Test scenes for our filtering algorithm. Each image was computed at a resolution of* 512 × 512 *with one path per pixel and one indirect bounce. The reference image was computed with* 4096 *paths per pixel. MSE shows the mean square error to the reference solution. The last row shows an outdoor scene illuminated with a diffuse skylight.*

```
  float dist2 = dot(t,t);
  float c_w = min(exp(-(dist2)/c_phi), 1.0);

  vec4 ntmp = texture2D(normalMap, uv);
  t = nval - ntmp;
  dist2 = max(dot(t,t)/(stepwidth*stepwidth),0.0);
  float n_w = min(exp(-(dist2)/n_phi), 1.0);

  vec4 ptmp = texture2D(posMap, uv);
  t = pval - ptmp;
  dist2 = dot(t,t);
  float p_w = min(exp(-(dist2)/p_phi),1.0);

  float weight = c_w * n_w * p_w;
  sum += ctmp * weight * kernel[i];
  cum_w += weight*kernel[i];
 }
 gl_FragData[0] = sum/cum_w;
}
```

References

[Bur81] Burt P. J.: Fast filter transform for image processing. *Computer Graphics and Image Processing 16*, 1 (1981), 20 – 51.

[CT05] Choudhury P., Tumblin J.: The trilateral filter for high contrast images and meshes. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses* (2005), p. 5.

[DD02] Durand F., Dorsey J.: Fast bilateral filtering for the display of high-dynamic-range images. In *SIGGRAPH* (2002), pp. 257–266.

[DHK08] Dammertz H., Hanika J., Keller A.: Shallow bounding volume hierarchies for fast SIMD ray tracing of incoherent rays. In *Computer Graphics Forum (Proc. 19th Eurographics Symposium on Rendering)* (2008), pp. 1225–1234.

[DS06] Dachsbacher C., Stamminger M.: Splatting indirect illumination. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games* (2006), pp. 93–100.

[ED04] EISEMANN E., DURAND F.: Flash photography enhancement via intrinsic relighting. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers* (2004), pp. 673–678.

[EG08] ERNST M., GREINER G.: Multi bounding volume hierarchies. In *Proc. 2008 IEEE/EG Symposium on Interactive Ray Tracing* (2008), pp. 35–40.

[FAR07] FATTAL R., AGRAWALA M., RUSINKIEWICZ S.: Multiscale shape and detail enhancement from multi-light image collections. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers* (2007), p. 51.

[Fat09] FATTAL R.: Edge-avoiding wavelets and their applications. *ACM Trans. Graph. 28*, 3 (2009), 1–10.

[FD09] FABIANOWSKI B., DINGLIANA J.: Interactive global photon mapping. *Computer Graphics Forum 28*, 4 (2009), 1151–1159.

[Hag04] HAGREAVES S.: Deferred Shading, Game Developers Conference, 2004.

[HKMMT89] HOLSCHNEIDER M., KRONLAND-MARTINET R., MORLET J., TCHAMITCHIAN P.: *A real-time algorithm for signal analysis with the help of the wavelet transform*. Springer-Verlag, 1989.

[Jen96] JENSEN H. W.: Global illumination using photon maps. In *Rendering Techniques '96 (Proc. of the Seventh Eurographics Workshop on Rendering)* (1996), pp. 21–30.

[Kaj86] KAJIYA J. T.: The rendering equation. *SIGGRAPH Comput. Graph. 20*, 4 (1986), 143–150.

[Kel97] KELLER A.: Instant radiosity. *Proc. of SIGGRAPH '97* (1997), 49–56.

[Kel98] KELLER A.: *Quasi-Monte Carlo Methods for Photorealisitic Image Synthesis*. PhD thesis, Universität Kaiserslautern, 1998.

[LSK*07] LAINE S., SARANSAARI H., KONTKANEN J., LEHTINEN J., AILA T.: Incremental instant radiosity for real-time indirect illumination. In *Proceedings of Eurographics Symposium on Rendering 2007* (2007), Eurographics Association, pp. xx–yy.

[LZT*08] LEHTINEN J., ZWICKER M., TURQUIN E., KONTKANEN J., DURAND F., SILLION F. X., AILA T.: A meshless hierarchical representation for light transport. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers* (2008), pp. 1–9.

[Mal89] MALLAT S.: A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 11* (1989), 674–693.

[Mal98] MALLAT S.: *A Wavelet Tour of Signal Processing*. Academic Press, 1998.

[ML09] MCGUIRE M., LUEBKE D.: Hardware-accelerated global illumination by image space photon mapping. In *HPG '09: Proceedings of the Conference on High Performance Graphics 2009* (2009), pp. 77–89.

[Mur97] MURTAGH F.: Multiscale transform methods in data analysis.

[NW09] NICHOLS G., WYMAN C.: Multiresolution splatting for indirect illumination. In *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games* (2009), pp. 83–90.

[ODR09] OVERBECK R. S., DONNER C., RAMAMOORTHI R.: Adaptive wavelet rendering. In *SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers* (2009), pp. 1–12.

[PD09] PARIS S., DURAND F.: A fast approximation of the bilateral filter using a signal processing approach. *International Journal of Computer Vision 81*, 1 (2009), 24–52.

[PH04] PHARR M., HUMPHREYS G.: *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., 2004.

[PSA*04] PETSCHNIGG G., SZELISKI R., AGRAWALA M., COHEN M., HOPPE H., TOYAMA K.: Digital photography with flash and no-flash image pairs. *ACM Trans. Graph. 23*, 3 (2004), 664–672.

[REG*09] RITSCHEL T., ENGELHARDT T., GROSCH T., SEIDEL H.-P., KAUTZ J., DACHSBACHER C.: Micro-rendering for scalable, parallel final gathering. In *SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers* (2009), pp. 1–8.

[RGK*08] RITSCHEL T., GROSCH T., KIM M. H., SEIDEL H.-P., DACHSBACHER C., KAUTZ J.: Imperfect shadow maps for efficient computation of indirect illumination. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers* (2008), pp. 1–8.

[SBB08] SOLOMON BOULOS I. W., BENTHIN C.: Adaptive ray packet reordering. *Symposium on Interactive Ray Tracing 0* (2008), 131–138.

[SGNS07] SLOAN P.-P., GOVINDARAJU N. K., NOWROUZEZAHRAI D., SNYDER J.: Image-based proxy accumulation for real-time soft global illumination. In *PG '07: Proceedings of the 15th Pacific Conference on Computer Graphics and Applications* (2007).

[SKS02] SLOAN P.-P., KAUTZ J., SNYDER J.: Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), pp. 527–536.

[Swe97] SWELDENS W.: The lifting scheme: A construction of second generation wavelets. *SIAM J. Math. Anal. 29*, 2 (1997), 511–546.

[TM98] TOMASI C., MANDUCHI R.: Bilateral filtering for gray and color images. In *ICCV '98: Proceedings of the Sixth International Conference on Computer Vision* (Washington, DC, USA, 1998), IEEE Computer Society, p. 839.

[Tsa09] TSAKOK J. A.: Faster incoherent rays: Multi-bvh ray stream tracing. In *HPG '09: Proceedings of the Conference on High Performance Graphics 2009* (2009), pp. 151–158.

[UB98] UYTTERHOEVEN G., BULTHEEL A.: The Red-Black wavelet transform. In *Signal Processing Symposium (IEEE Benelux)* (1998), pp. 191–194.

[Vea97] VEACH E.: *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University, 1997.

[Wal04] WALD I.: *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Saarland University, 2004.

[WKB*02] WALD I., KOLLIG T., BENTHIN C., KELLER A., SLUSALLEK P.: Interactive global illumination using fast ray tracing. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering* (2002), pp. 15–24.

[WWZ*09] WANG R., WANG R., ZHOU K., PAN M., BAO H.: An efficient gpu-based approach for interactive global illumination. In *SIGGRAPH '09: ACM SIGGRAPH 2009 papers* (2009), pp. 1–8.