Depth of Field Postprocessing For Layered Scenes Using Constant-Time Rectangle Spreading

Todd Jerome Kosloff
Michael Tao
Brian A. Barsky

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2008-187
http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-187.html

December 30, 2008

Depth of Field Postprocessing For Layered Scenes Using Constant-Time Rectangle Spreading

Todd J. Kosloff[*]
University of California, Berkeley
Computer Science Division
Berkeley, CA 94720-1776

Michael Tao[†]
University of California, Berkeley
Computer Science Division
Berkeley, CA 94720-1776

Brian A. Barsky[‡]
University of California, Berkeley
Computer Science Division and School of Optometry
Berkeley, CA 94720-1776

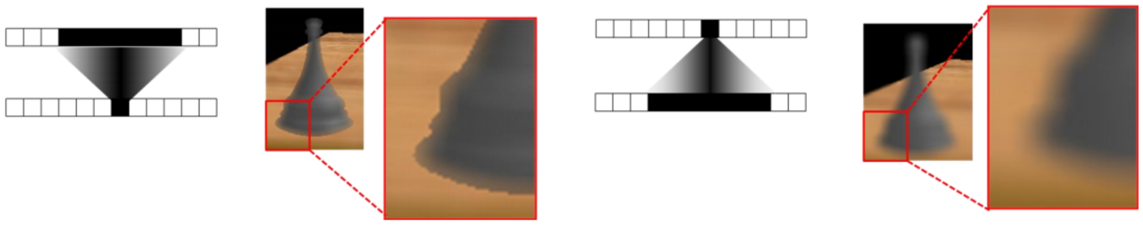
Figure 1: Spreading vs. Gathering. Left: Gathering leads to sharp silhouettes on blurred objects. Right: Spreading correctly blurs silhouettes. This scene uses two layers, one for the background, one for the foreground.

Abstract

Control over what is in focus and what is not in focus in an image is an important artistic tool. The range of depth in a 3D scene that is imaged in sufficient focus through an optics system, such as a camera lens, is called depth of field. Without depth of field, everything appears completely in sharp focus, leading to an unnatural, overly crisp appearance. Current techniques for rendering depth of field in computer graphics are either slow or suffer from artifacts and limitations in the type of blur. In this paper, we present a new image filter based on rectangle spreading which is constant time per pixel. When used in a layered depth of field framework, it eliminates the intensity leakage and depth discontinuity artifacts that occur in previous methods. We also present several extensions to our rectangle spreading method to allow flexibility in the appearance of the blur through control over the point spread function.

Index Terms: I.3.3 [Computer Graphics]: Picture/Image Generation-display algorithms, bitmap and frame buffer operations, viewing algorithms—

1 Introduction

1.1 Background

Control over what is in focus and what is not in focus in an image is an important artistic tool. The range of depth in a 3D scene that is imaged in sufficient focus through an optics system, such as a camera lens, is called depth of field. This forms a swath through a 3D scene that is bounded by two planes that typically are both parallel to the film/image plane of the camera.

Professional photographers or cinematographers often control the focus distance, aperture size, and focal length to achieve desired effects in the image. For example, by restricting only part of a scene to be in focus, the viewer or the audience automatically attends primarily to that portion of the scene. Analogously, pulling focus in a movie directs the viewer to look at different places in the scene, following the point of focus as it moves continuously within the scene.

Rendering algorithms in computer graphics that lack depth of field are in fact modeling a pinhole camera model. Without depth of field, everything appears in completely sharp focus, leading to an unnatural, overly crisp appearance. Techniques for rendering depth of field in computer graphics are well known, but they all are either slow, or suffer from artifacts or limitations.

Distributed ray tracing [9] can render scenes by directly simulating geometric optics, resulting in high quality depth of field effects. However, many rays per pixel are required, leading to slow render times. Post-processing, which adds blur to an image that was rendered with everything in perfect focus, is the alternative for efficiently simulating depth of field.

1.2 Goals

A good depth of field postprocess method should meet the following criteria:

1. Allows the amount of blur to vary arbitrarily from pixel to pixel, as each pixel can lie at a different distance from the focus plane.
2. Achieves high performance, even for large amounts of blur.
3. Allows control over the nature of the blur, by allowing flexibility in choice of the PSF (Point Spread Function).
4. Avoids depth discontinuity artifacts. Some fast methods lead to jarring discontinuities on the silhouettes of blurred objects.
5. Avoids intensity leakage artifacts. Some fast methods lead to a blurred background leaking on top of an in-focus foreground. Intensity leakage is not physically correct and takes away from the realism of the image.

[*]e-mail: koslofto@cs.berkeley.edu
[†]e-mail:ed.grimley@aol.com
[‡]e-mail:barsky@cs.berkeley.edu

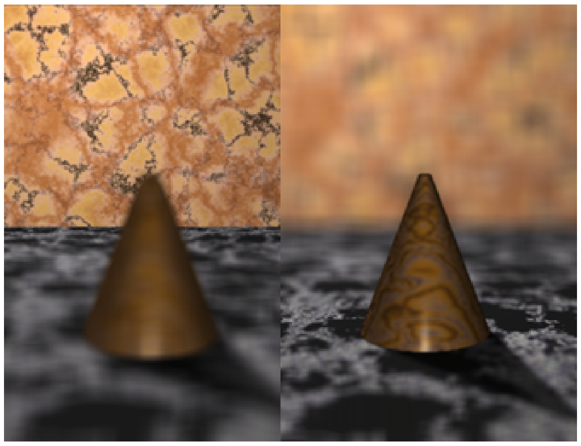

Figure 2: A scene blurred using our basic square spreading method. Left: focus on wall. Right: focus on cone.

In this paper, we describe new image filters that, when used on layered scenes, simultaneously meet all of these criteria.

Spatially varying image filters can typically be characterized as utilizing either spreading or gathering, as will be explained in the next section of this paper. Existing fast blur filters generally use gathering, which is physically incorrect, and leads to noticeable flaws in the resulting image. In this paper, we show that spreading removes these flaws, and can be very efficient. We describe a novel fast spreading blur filter, for use in depth of field for layered scenes. Our method takes inspiration from the summed area table (SAT) [10], though our method differs substantially, in that a SAT is fundamentally a gather method while our method uses spreading. We can characterize the appearance of the blur in an image by the PSF, which describes how a point of light in the scene would appear in the blurred image. Our basic method uses rectangles of constant intensity as the PSF. To allow for more flexible choice of PSFs, we describe two alternate extensions to our method: one that uses constant-intensity PSFs of arbitrary shape, and another that hybridizes any fast blur method with a slower, direct method that allows PSFs of arbitrary shape and intensity distribution. In this way, we achieve a controllable tradeoff between quality and speed.

1.3 Summary of Contributions

This paper makes the following contributions to the problem of depth of field postprocessing of layered scenes.

1. We show that spreading filters do not lead to the depth discontinuity artifacts that plague gathering methods. Note that the use of layers solves the intensity leakage problem when used with either gathering or spreading methods.

2. We show that Crow's summed area table, fundamentally a gather method, can be transformed into a spreading method.

3. We extend the SAT-like method to handle constant-intensity PSFs of arbitrary shape, at modest additional cost.

4. We show how to hybridize any fast spreading filter with arbitrary PSFs according to a controllable cost/quality tradeoff.

2 Depth of Field Postprocessing of Layered Scenes

When blurring an image of a scene using a linear filter, blurred background objects will incorrectly leak onto sharp foreground objects, and sharp foreground objects overlapping blurred background objects will incorrectly exhibit sharp silhouettes. Scofield shows how to solve this problem in the case of axis-aligned planar objects using layering [33]. Each object is placed into a layer, along with an alpha matte. The layers and alpha mattes are blurred using convolution via FFT, and the blurred objects are composited using alpha blending with the blurred alpha mattes. In this paper we describe a novel set of image filters suitable for efficiently extending Scofield's method to nonplanar layers. As a prerequisite to using a layered method, the scene in question must be decomposed into layers. We assume that the decomposition has already been performed, either manually or automatically, and describe how to perform depth of field postprocessing using these layers.

3 Spreading vs. Gathering

The process of convolving an image with a filter can equivalently be described as spreading or gathering. Spreading means each pixel in the image is expanded into a copy of the filter kernel, and all these copies are summed together. Gathering means that each pixel in the filtered image is a weighted average of pixels from the input image, where the weights are determined by centering the filter kernel at the appropriate pixel. When we allow the filter kernel to vary from pixel to pixel, spreading and gathering are no longer equivalent.

In any physically plausible postprocess depth of field method, each pixel must spread out according to that pixel's PSF. Each pixel can have a PSF of different size and shape, so we designed our filter to vary from pixel to pixel. Fast image filters typically only support gathering, so fast depth of field postprocess methods generally use gathering, despite the fact that spreading would produce more accurate results.

We present a depth of field postprocess approach for layered scenes. For example, consider a scene with a foreground layer and a background layer. Both layers are blurred, and are then composited with alpha-blending. Blurring and compositing layers for depth of field postprocessing was first developed by Scofield [33]. However, Scofield blurred via convolution, which limited each layer to only a single amount of blur; this is accurate only for planar, screen-aligned objects. Despite this limitation, Scofield's method is free of depth discontinuity and intensity leakage artifacts. We wish to use spatially-variant blur instead of convolution, to extend Scofield's method to scenes where the layers are not necessarily planar or axis-aligned.

We initially considered using Crow's summed area table (SAT) as the image filter, since SATs run in constant time per pixel and allow each pixel to have an independently chosen filter size. In any given layer, some of the pixels will be opaque, and others will be completely transparent. For the sake of discussion, we ignore semi-transparent pixels, though these are allowed as well. To generate the blurred color for an opaque pixel, we look up the depth value of the pixel, and determine a filter size using a lens model. We then perform a SAT lookup to determine the average color of the filter region. Unfortunately, it is not straightforward to determine a blurred color for a transparent pixel, as transparent pixels correspond to regions outside of any object, and thus do not have depth values. Depth values must be extrapolated from the opaque pixels to the transparent pixels, a process that generally only approximates the correct result, and requires additional computation. Barring such extrapolation, transparent pixels must be left completely transparent. Consequently, pixels outside of any object remain completely transparent, even after blurring, even if said pixels are immediately adjacent to opaque pixels. Visually, this results in a sharp discontinuity in the blurred image, along object silhouettes (Figure 1, left). We refer to these problems as depth discontinuity artifacts. It is important to note that this problem with transparent pixels does not occur with methods that blur by breaking the image into layers which are then approximated with spatially uniform blur [24], [2], as spatially uniform blur does not consider depth values at all, after discretization. However, blurring by discrete depth simply replaces depth discontinuity artifacts with discretization artifacts [5].

A better option is to use a spreading filter. When an object is blurred by spreading, opaque pixels near silhouettes will spill out into the adjacent region, yielding a soft, blurred silhouette (Figure 1, right). This is a highly accurate approximation to the partial occlusion effect seen in real depth of field. This need for spreading filters motivates the constant-time spreading filter presented in this paper.

4 Previous Work

Potmesil and Chakravarty [29] developed the first ever depth of field rendering algorithm. They used a postprocess approach that employed complex PSFs derived from physical optics. Their direct, spatial domain filter is slow for large blurs.

Cook [9] developed distributed ray tracing, which casts many rays per pixel in order to achieve a variety of effects, including depth of field. Kolb [23] later showed how distributed ray tracing can be used to simulate particular systems of camera lenses, including aberrations and distortions. Methods based on distributed ray tracing faithfully simulate geometric optics, but due to the number of rays required, are very slow. The accumulation buffer [18] uses rasterization hardware instead of tracing rays, but also becomes very slow for large blurs, especially in complex scenes.

Scheuermann and Tatarchuk [32] developed a fast and simple postprocess method, suitable for interactive applications such as video games. However, it suffers from depth discontinuity artifacts, due to use of gathering. They selectively ignore certain pixels during the gathering, in order to reduce intensity leakage artifacts, though, due to their use of a reduced resolution image that aggregates pixels from many different depths, does not eliminate them completely. Their method does not allow for a choice of point spread function; it produces an effective PSF that is a convolution of a bilinearly resampled image with random noise. Our methods allow a choice of PSF, and eliminate the depth discontinuity artifact.

Krauss and Strengert [24] use pyramids to perform fast uniform blurring. By running the pyramid algorithm multiple times at different pyramid levels, they approximate a continuously varying blur. Unlike our method, their method does not provide a choice of PSFs, but rather produces a PSF that is roughly Gaussian.

Bertalmio et al. [7] showed that depth of field can be simulated as heat diffusion. Later, Kass et al. [22] use a GPU to solve the diffusion equation for depth of field in real time using an implicit method. Diffusion is notable for being a blurring process that is neither spreading nor gathering. Diffusion, much like pyramids, inherently leads to Gaussian PSFs.

Mulder and van Lier [26] used a fast pyramid method at the periphery, and a slower method with better PSF at the center of the image. This is somewhat similar to our hybrid method, but we use an image-dependent heuristic to adaptively decide where the high quality PSF is required.

Many other methods exist that achieve fast depth of field, such as [13], [25], [30] and [35], though they all suffer from various artifacts and limitations.

The use of layers and alpha blending [28] for depth of field was introduced by [33]. We also use this layering approach, though our image filters offer significant advantages over those used by Scofield. Barsky showed in [6] and [5] how to use object identification to allow objects to span layers without artifacts at the seams.

Other methods exist as alternates to layers, such as Catmull's method for independent pixel processing [8], which is efficient for scenes composed of a few large, untextured polygons, and Shinya's ray distribution buffer [34], which resolves intensity leakage in a very direct way, at great additional cost. We choose to use layers due to their simplicity and efficiency.

For a comprehensive survey of depth of field methods, please consult Barsky's [3] [4] and Demers' [11] surveys.

The method presented in this paper uses ideas similar to Crow's summed area table [10] originally intended for texture map anti-aliasing. Other constant-time image filters intended for texture map anti-aliasing methods include [14] and [16]. We build our method along the lines of Crow's, as it is the simplest.

The well known Huang's method [21], (cost linear to kernel size), and extended, (cost constant to kernel size), [27] are very fast algorithms to compute the square blur. As the window of the kernel moves to the next pixel, the new pixel values of the window are added to the summation while the old values are subtracted from the summation. By storing the summation of the columns into memory, the process of calculating the blurred value becomes a constant computation. Unfortunately, the limitations of these methods is that the kernel must be rectangular and at a constant size. With these limitations, the Huang's and the extended method are not favorable for depth of field blurring.

Finally, the Fast Fourier Transform (FFT) is a traditional fast method for blurring pictures via convolution, but only applies where the amount of blur throughout an image is constant. Therefore many FFTs are required if we want the appearance of continuous blur gradations. When many FFTs are required, we no longer have a fast algorithm, even if the FFTs are performed by a highly optimized implementation such as FFTW [15]. Therefore, we do not use FFTs in our method.

5 Constant Time Rectangle Spreading

5.1 Motivation For Examining the Summed Area Table

We draw inspiration from Crow's summed area table (SAT). We base our method on the SAT rather than on one of the other table methods, because of the following speed and quality reasons.

Table-based gather methods can be used in a setting where the table is created offline. This means that it is acceptable for table creation to be slow. Consequently, some of these methods do indeed have a lengthy table creation phase. [14], [16]. However, the summed area table requires very little computation to create, and so can be used online [20].

With a SAT, we can compute the average color of any rectangular region. Both the size and location of the region can be specified with pixel-level precision. Other methods, such as [16] and [14], give less precise control.

While summed area tables have beneficial qualities, they can only be used for gathering, requiring us to devise a new method that can be used for spreading. Furthermore, SATs have precision requirements that grow with increased image size. Our rectangle spreading method does not inherit these precision problems, as the signal being integrated includes alternating positive and negative values.

5.2 Our Basic Rectangle-Spreading Method

First, we must create an array of the same dimensions as the image, using a floating point data type. We will refer to this as $table$. After initializing the array to zero, we enter Phase I of our method [Figure 4]. Phase I involves iterating over each pixel in the input image, consulting a depth map and camera model to determine how large the circle of confusion is, and accumulating signed intensity markers in the array, at the corners of each rectangle that we wish to spread. Thus we tabulate a collection of rectangles that are to be summed together to create the blurred image.

At this point, the array contains enough information to construct the blurred image, but the array itself is not the blurred image, as only the corners of each rectangle have been touched.

To create the blurred image from the array, we need Phase II of our method [Figure 5]. Phase II is similar to creating a summed

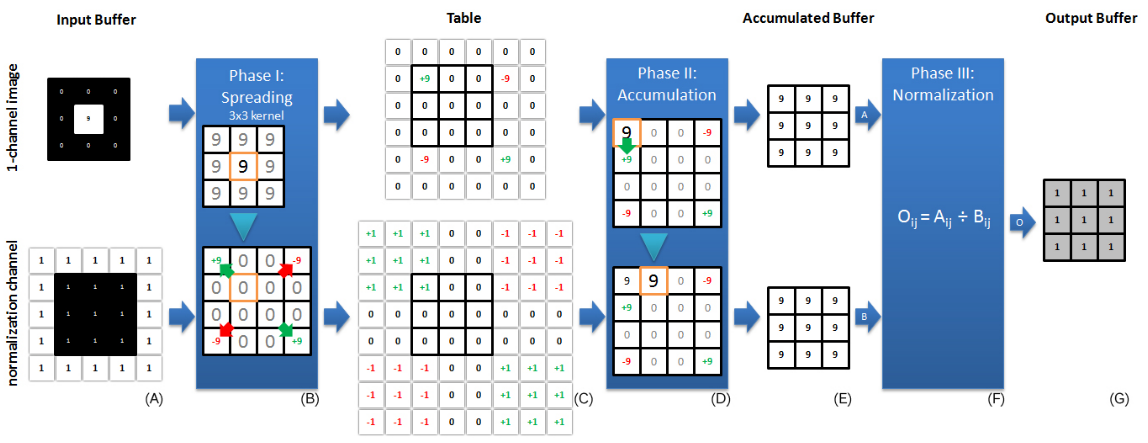

Figure 3: For every image input, there are two input buffers (A)- the image channel as an input and the normalization channel, which is the size of the image and contains pixels of intensity 1.0. In Phase I (B), for each pixel, we read the value of the pixel and spread the positive and negative markers accordingly on to the table (c). In Phase II (D), there is an accumulator (shown in orange) that scans through the entries of the tables from left to right, top to bottom. The accumulated buffer (E) stores the current state of the accumulator. Phase II is equivalent to building a summed area table of the table (c). In phase III (F), for each entry at i and j, we divide the image's accumulated buffer by the normalization channel's accumulated buffer to obtain the output of the image (G).

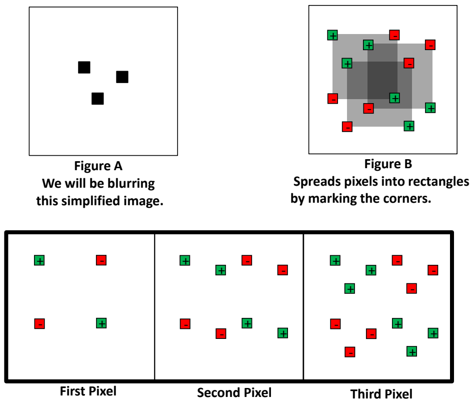

Figure 4: Phase I: Accumulating

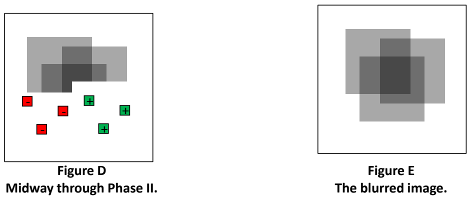

Figure 5: Phase II: Filling In

area table from an image. However, the input to Phase II is a table of accumulated signed intensities located at rectangle corners, whereas the input to a summed area table creation routine is an image. Furthermore, the output of Phase II is a blurred image, whereas the output of a summed area table creation routine is a table meant to be queried later.

A summary of the entire blur process is illustrated in Figure 3.

```
//This code is for a single color channel,
//for simplicity.
//In a real implementation, R,G,B and A channels
//must all be blurred by this same process.
//Fast rectangle spreading: Phase I.
float table[width][height];
float area;
//zero out table (code omitted)
//S is the input image (S stands for sharp)
for(int i=0; i<width; i++)
for(int j=0; j<height;j++)
{
    int radius = get_blur_radius(i,j);
    float area = (radius*2+1)*(radius*2+1);

    table[i-radius][j-radius]+= S[i][j] / area;
    table[i+radius][j-radius]-= S[i][j] / area;
    table[i-radius][j+radius]-= S[i][j] / area;
    table[i+radius][j+radius]+= S[i][j] / area;
}

//Fast rectangle spreading: Phase II.
float accum;
float I[width][height]; //I is the blurred image
for(int y=1; y<height;y++)
{
    accum = 0;
    for(int x=0; x<width; x++)
    {
    accum += table[x][y];
    I[x][y] = accum + I[x][y-1];
    }
}
```

5.3 Normalization

The blurred image is the sum of many rectangles, meaning each output pixel is the sum of a variable number of rectangles. The above code listing will lead to the pixels receiving more squares appearing too bright, and pixels receiving too few squares appearing too dim. We fix this by adding a fourth channel, which we will use for normalization. We spread rectangles of unit intensity into the normalization channel, and divide through by the normalization channel at the end. Note that overlapping PSFs of disparate sizes will have appropriate relative contributions to the final image, as intensities were divided through by area during the initial spreading phase. The alpha channel is not normalized, to preserve semi-transparent silhouettes (partial occlusion). In place of normalization, the alpha channel is biassed slightly to prevent dimming, and clamped at 1.0.

5.4 Non-Integer Blur Sizes

In scenes with continuously varying depth levels, we will often have pixels whose blur value is not an integer. If we were to simply use squares whose radius is the closest integer to the desired blur value, visible discontinuities would appear in the blurred image. Fortunately, we can easily approximate fractional blur sizes by spreading two integer sized rectangles, one slightly larger than the other, with weights dependent on the size of the desired rectangle.

5.5 Borders

As is typical in blur algorithms, our methods require special consideration near the borders of the image. When a pixel near the border is spread into a PSF, the PSF may extend beyond the borders of the image. We handle this case by padding the image by a large enough amount to ensure that the PSFs never extend beyond the borders of the padded image. The padding is cropped away before the image is displayed.

5.6 GPU Implementation

We have developed a DirectX 10 implementation of fast rectangle spreading to achieve real-time performance. The implementation is straightforward.

1. Phase I

   To accumulate corners, each corner is rendered as a point primitive. To avoid transferring large amounts of geometry, the points are generated without the use of vertex buffers, via the vertex ID feature of DirectX 10. A vertex shader maps the vertex id to pixel coordinates and appropriate signed intensities. To cause the signed intensities to accumulate rather than overwrite one another, alpha blending is used, configured to act as additive blending.

2. Phase II

   Any GPU implementation of SAT generation could be used for the integration step. Currently we are using the recursive doubling approach [20].

Our GPU implementation achieves 45 frames per second with two layers at 800x600 on an ATI HD4870. Numerous low-level optimizations should raise performance even higher. At the moment our method is about twice as expensive as a SAT. We suspect this is due to the serialization that inherently must take place when multiple pixels spread corners to the same point. As future work, we plan to mitigate the serialization problem by reordering the spreading to reduce contention.

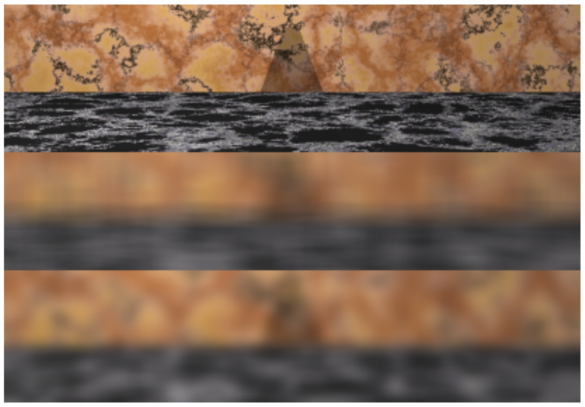

Figure 6: Top: Original image. Middle: blurred with our basic rectangle spreading method. Bottom: blurred using bilinear PSFs via repeated integration. Observe that the bilinear method appears significantly smoother.

6 Arbitrarily Shaped Constant Intensity PSFs

Although our rectangle spreading method is fast and simple, we require a different method for applications that mandate circular or hexagonal PSFs, which are necessary to mimic the shape of particular camera diaphragms.

There exists a simple modification to our basic method that allows for arbitrarily shaped PSFs with constant intensity. We modify Phase I such that, for each scanline, we write markers wherever the PSF boundary intersects the scanline. In Phase II, each scanline is integrated in a one dimensional fashion. These changes are very simple to implement, but the enhanced generality comes at a cost: blurring now has a cost proportional to the circumference of the PSF, whereas our basic method had a fixed cost for any size PSF. Note that direct blurring has a cost related to the area of the PSF, so a cost related to the circumference is a significant improvement. Also note that a cost proportional to the circumference is optimal for PSFs that have arbitrary, per-pixel shapes.

7 Hybridizing With Arbitrary PSFs

7.1 Hybrid Method Concept

Sometimes we want to blur with a PSF that cannot be approximated as either a rectangle or as a constant-intensity shape, even with arbitrary shape. For example, PSFs derived from physical optics can have complex diffraction fringes. Highly complex PSFs are best dealt with directly. This is extremely expensive, but it turns out not to be necessary to use an accurate PSF at every pixel. Rather, we can utilize a hybrid method that uses accurate PSFs only in high contrast portions of the image. Low contrast portions of the image can be blurred using our fast rectangle spreading method, and the resulting artifacts will be difficult to see. See Figure 7 for an example.

7.2 Hybrid Method Details

First we must determine the contrast of each pixel. Our contrast measure is the absolute difference between a pixel's intensity and the average intensity of that pixel's neighborhood. The size of the neighborhood is the size of the PSF for that pixel. We use a summed area table to efficiently compute the average intensity over the neighborhood. This contrast measure is simple and works reasonably well, but we emphasize that our hybrid approach could be used with other contrast measures, as well as frequency measures, if this is found to produce better results.

Now that we have the contrast, we apply a threshold to determine whether to use a full-quality PSF, or whether to use a fast square in-

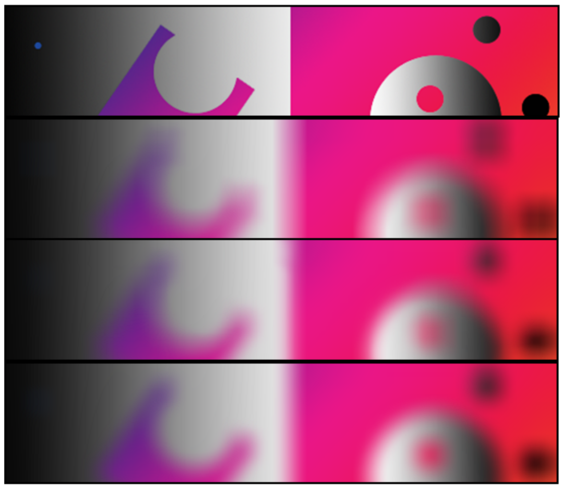

Figure 7: Top: image with high and low contrast regions. 2nd from top: image blurred via square spreading. 2nd from bottom: image blurred with a smooth PSF. Bottom: The image blurred with our hybrid method. Notice that the hybrid image looks similar to the high quality PSF image, but took much less time to blur.

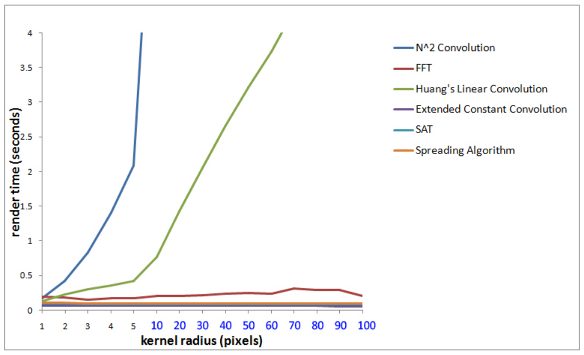

Figure 8: Our constant-time rectangle spreading filter compares competitively with other fast box filters while improving the quality for depth of field. Tests are done on a 1680 x 1050 image, Intel Core 2 Quad Q9450 processor, 4 GB RAM.

stead. The threshold is a user-definable parameter that trades off quality for speed. A key contribution of this hybrid is that the speed/quality tradeoff can be finely tuned. Where contrast is very low, this method degenerates into our basic fast square approach. Where contrast is very high, it becomes a direct filter that uses precisely the true PSF. The threshold can be set anywhere along this continuum, making this method useful for a range of applications. We find that we can raise the threshold until our method takes only 30 percent the time of the direct method, before significant quality loss occurs. Such fine-grained control over filter approximation is a key contribution of our hybrid method.

8 Performance Comparison

Our basic method uses constant-intensity rectangular PSFs, so we compare against other methods that use constant-intensity rectangular PSFs, including $O(N^2)$ spatial-domain convolution, SATs, and Huang's methods, both basic [21] and extended [27]. See Figure 8. For completeness, we also compare against fast convolution via FFT. These comparison are for software implementations, as we don't yet have complete GPU implementations for some of these methods. Our method performs the same as SATs, much faster than

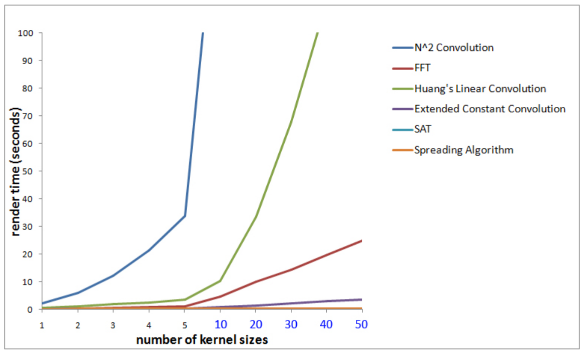

Figure 9: Our constant-time rectangle spreading filter compares competitively with the summed area table algorithm. To perform a spatially varying blur with a spatially uniform method, on the other hand, requires multiple executions, once for each kernel size. This leads to performance hits proportional to the number of different kernel sizes. Our spreading algorithm remains constant time as the number of kernel sizes increases because the algorithm requires only one execution. The first kernel radius starts with radius 5 and each kernel after the first the radius increments by 2. Tests are done on a 1680 x 1050 image, Intel Core 2 Quad Q9450 processor, 4 GB RAM.

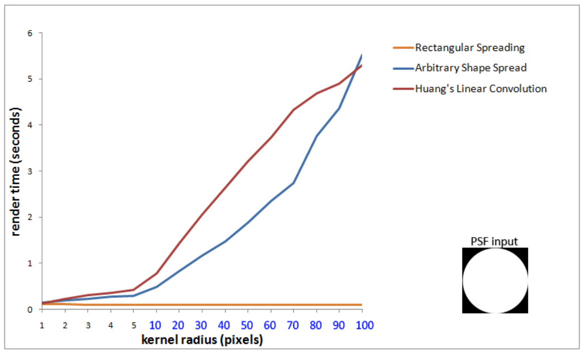

Figure 10: In this example, a constant intensity circle shape was used. The arbitrary shape spreading is linear to the circumference of the circle. Although the timings are significantly different at higher kernel radii, the method competes competitively with the Huang's linear convolution. Tests are done on a 1680 x 1050 image, Intel Core 2 Quad Q9450 processor, 4 GB RAM.

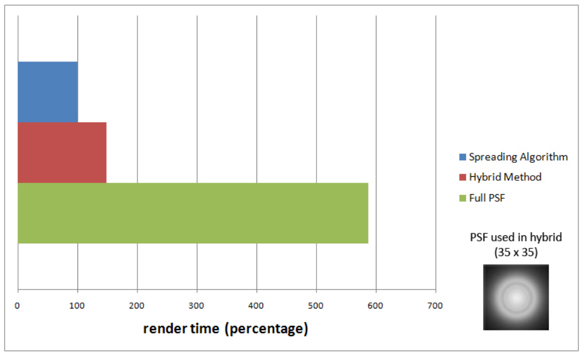

Figure 11: In this example, the hybrid used 1761271 fast rectangular PSFs and 127747 slow detailed PSFs. As we can see, the hybrid method takes only 49 percent more time than the spreading algorithm while the using all PSF on the same image takes nearly 487 percent more time than the spreading algorithm. Through the image comparisons in figure 7, we can see that the image quality does not have a noticeable degradation. Tests are done on a 1680 x 1050 image, Intel Core 2 Quad Q9450 processor, 4 GB RAM.

direct convolution, much faster than Huang's linear-time method, significantly faster than the FFT method, and about 20 percent slower than Huang's constant time filter. Note that Huang's method as well as FFT convolution require the kernel to be the same at all pixels, but ours allows the kernel to vary.

Spatially uniform blurring methods can be used in the nonuniform case by repeated application with each different filter required. This increases cost, but it is conceivable that the simplicity of uniform blurring outweighs this added cost. We show this not to be the case in Figure 9.

Our arbitrary-outline method has a cost proportional to the perimeter of the PSF. We compare the performance of this method with other methods in Figure 10.

Finally, we compare the performance of our hybrid method with the fast square blur, and with the slow detailed PSF method in Figure 11.

9 Limitations and Future Work

Some scenes cannot be decomposed into layers, because objects can span many depth levels. Our simple treatment of layers only applies to scenes where objects do not straddle layers. This precludes self-occluding objects. The method of Krauss and Strengert [24] allows objects to straddle layers by applying a careful weighting to the hide the seam at layer boundaries. We believe that our fast spreading filter could be used with Krauss and Strengert's layering framework, to allow our method to work with arbitrarily complex scenes. This method would often need fewer layers than Krauss and Strengert's method, because we need additional layers only when there is additional depth complexity, whereas they need additional layers whenever there are additional amounts of blur, whether or not there is any depth complexity.

Just as Crow's SATs can be extended to use arbitrary-order polynomial kernels via Heckbert's repeated integration scheme [19], our rectangle spreading method can also be extended in a similar way. In Phase I, the weights developed for Heckbert's scheme are accumulated, much as rectangle corners are accumulated in our rectangle spreading method. In Phase II, N integration steps are performed, for polynomials of order N. We will describe the extension to polynomial kernels in full detail in a paper currently under preparation.

Finally, the contrast measure in our hybrid method is a heuristic that seems to work well. Further work is required to determine more precisely when we can get away with simple PSFs, and when more complex PSFs are required.

10 Conclusion

In this paper we have shown that Scofield's method of convolving and compositing planar axis-aligned layers can be extended to non-planar layers with great efficiency, using constant-time spreading filters. We show that spreading filters are necessary to avoid artifacts, and we show how to construct GPU-capable constant-time spreading filters operating on the same underlying principles as the SAT. We also show how to extend our basic method from constant-intensity rectangles to constant-intensity PSFs of arbitrary shape, at modest additional cost. Finally, we describe a hybrid method for efficiently introducing highly detailed PSFs via a controllable cost/quality tradeoff.

In conclusion, our spreading filters enable artifact-free depth of field effects for layered scenes, with image quality comparable to expensive $O(N^2)$ filters, with performance approaching that of traditional constant-time filters. We believe that due to the configurable cost/quality tradeoff of our approach, our methods will be useful both in interactive applications such as video games, as well as production quality rendering.

Acknowledgements

The authors wish to thank A, B, C. This work was supported in part by a grant from XYZ.

References

[1] Gpubench: http://graphics.stanford.edu/projects/g pubench/.

[2] B. A. Barsky. Vision-realistic rendering: simulation of the scanned foveal image from wavefront data of human subjects. In *APGV '04: Proceedings of the 1st Symposium on Applied perception in graphics and visualization*, pages 73–81, New York, NY, USA, 2004. ACM.

[3] B. A. Barsky, D. R. Horn, S. A. Klein, J. A. Pang, and M. Yuf. Camera models and optical systems used in computer graphics: Part i, image based techniques. In *Proceedings of the 2003 International Conference on Computational Science and its Applications (ICCSA'03)*, pages 246–255, 2003.

[4] B. A. Barsky, D. R. Horn, S. A. Klein, J. A. Pang, and M. Yuf. Camera models and optical systems used in computer graphics: Part ii, image based techniques. In *Proceedings of the 2003 International Conference on Computational Science and its Applications (ICCSA'03)*, pages 256–265, 2003.

[5] B. A. Barsky, M. J. Tobias, D. Chu, and D. R. Horn. Elimination of artifacts due to occlusion and discretization problems in image space blurring techniques. In *Graphical Models 67(6)*, pages 584–599, 2005.

[6] B. A. Barsky, M. J. Tobias, D. R. Horn, and D. Chu. Investigating occlusion and discretization problems in image space blurring techniques. In *First International Conference on Vision, Video, and Graphics*, pages 97–102, 2003.

[7] M. Bertalmio, P. Fort, and D. Sanchez-Crespo. Real-time, accurate depth of field using anisotropic diffusion and programmable graphics cards. In *Proc. 2nd International Symposium on 3D Data Processing, Visualization and Transmission 3DPVT 2004*, pages 767–773, 6–9 Sept. 2004.

[8] E. Catmull. An analytic visible surface algorithm for independent pixel processing. In *SIGGRAPH 1984 Conference Proceedings*, pages 109–115. ACM Press, 1984.

[9] R. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. In *ACM SIGGRAPH 1984 Conference Proceedings*, pages 137–145, 1984.

[10] F. C. Crow. Summed-area tables for texture mapping. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 207–212, New York, NY, USA, 1984. ACM Press.

[11] J. Demers. *GPU Gems*, pages 375–390. Addison Wesley, 2004.

[12] P. Dubois and G. Rodrigue. An analysis of the recursive doubling algorithm. In *High Speed Computer and Algorithm Organization*, pages 299–305. 1977.

[13] P. Fearing. Importance ordering for real-time depth of field. In *Proceedings of the Third International Computer Science Conference on Image Analysis Applications and Computer Graphics*, volume 1024, pages 372–380. Springer-Verlag Lecture Notes in Computer Science, 1995.

[14] A. Fournier and E. Fiume. Constant-time filtering with space-variant kernels. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 229–238, New York, NY, USA, 1988. ACM.

[15] M. Frigo and S. G. Johnson. The design and implementation of fftw3. *Proceedings of the IEEE 93 (2) Special Issue on Program Generation, Optimization, and Platform Adaptation*, 93(2):216–231, Feb. 2005.

[16] C. Gotsman. Constant-time filtering by singular value decomposition. In *Computer Graphics Forum*, pages 153–163, 1994.

[17] S. Greene. Summed area tables using graphics hardware. Game Developers Conference, 2003.

[18] P. Haeberli and K. Akeley. The accumulation buffer: hardware support for high-quality rendering. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 309–318, New York, NY, USA, 1990. ACM.

[19] P. S. Heckbert. Filtering by repeated integration. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 315–321, New York, NY, USA, 1986. ACM Press.

[20] J. Hensley, T. Scheuermann, G. Coombe, M. Singh, and A. Lastra. Fast summed-area table generation and its applications. In *Eurographics 2005*, 2005.

[21] T. Huang, G. Yang, and G. Yang. A fast two-dimensional median filtering algorithm. In *In IEEE Transactions on Acoustics, Speech, and Signal Processing v. 27*, pages 13–18, 1979.

[22] M. Kass, A. Lefohn, and J. Owens. Interactive depth of field. In *Pixar Technical Memo 06-01*, 2006.

[23] C. Kolb, D. Mitchell, and P. Hanrahan. A realistic camera model for computer graphics. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 317–324, New York, NY, USA, 1995. ACM.

[24] M. Kraus and M. Strengert. Depth of field rendering by pyramiadal image processing. In *Computer Graphics Forum 26(3)*, 2007.

[25] J. Krivanek, J. Zara, and K. Bouatouch. Fast depth of field rendering with surface splatting. In *Computer Graphics International 2003*, 2003.

[26] J. Mulder and R. van Lier. Fast perception-based depth of field rendering. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 129–133, 2000.

[27] S. Perreault and P. Hebert. Median filtering in constant time. In *IEEE Transactions on Image Processing 16(9)*, pages 2389–2394, 2007.

[28] T. Porter and T. Duff. Compositing digital images. In *ACM SIGGRAPH 1984 Conference Proceedings*, pages 253–259, New York, NY, USA, 1984. ACM.

[29] M. Potmesil and I. Chakravarty. Synthetic image generation with a lens and aperture camera model. In *ACM Transactions on Graphics 1(2)*, pages 85–108, 1982.

[30] P. Rokita. Generating depth-of-field effects in virtual reality applications. In *IEEE Computer Graphics and Applications 16(2)*, pages 18–21, 1996.

[31] T. Scheuermann and J. Hensley. Efficient histogram generation using scattering on gpus. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 33–37, New York, NY, USA, 2007. ACM.

[32] T. Scheuermann and N. Tatarchuk. Advanced depth of field rendering. In *ShaderX3: Advanced Rendering with DirectX and OpenGL*, 2004.

[33] C. Scofield. 2 1/2-d depth of field simulation for computer animation. In *Graphics Gems III*. Morgan Kaufmann, 1994.

[34] M. Shinya. Post-filtering for depth of field simulation with ray distribution buffer. In *Proceedings of Graphics Interface '94*, pages 59–66. Canadian Information Processing Society, 1994.

[35] T. Zhou, J. X. Chen, and M. Pullen. Accurate depth of field simulation in real time. In *Computer Graphics Forum 26(1)*, pages 15–23, 2007.