Image Space Gathering

Austin Robison Peter Shirley
NVIDIA Research

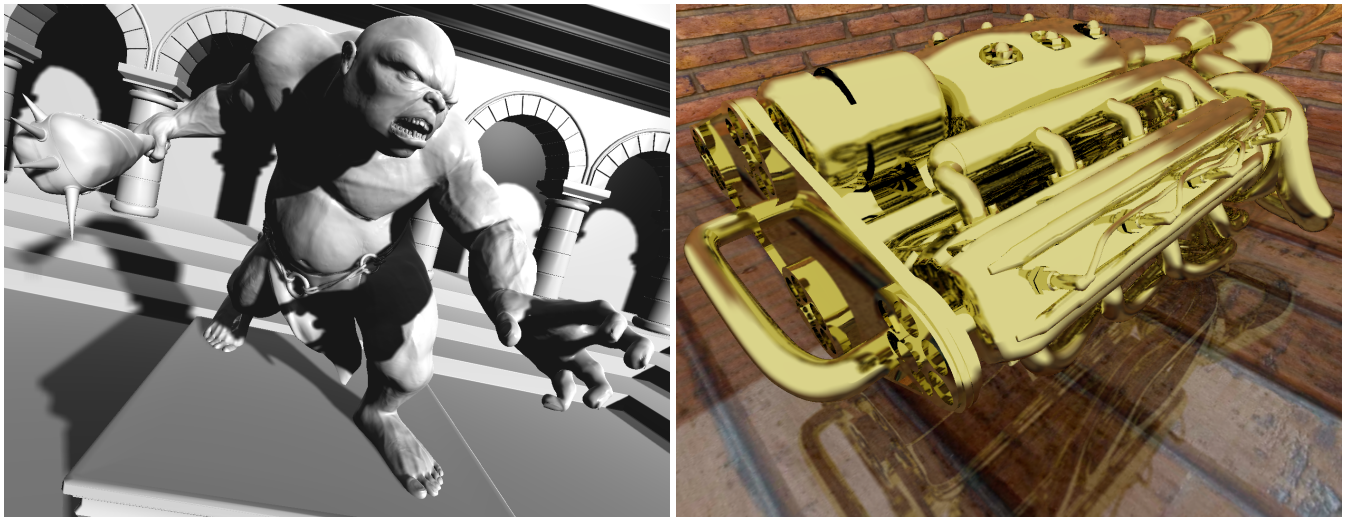

Figure 1: *Left: Soft shadows on Bubs, created from a single shadow ray per pixel. Right: Glossy reflections created from a single reflection ray per pixel. Note the variable glossiness per material; the floor is half the glossiness of the motor and the walls are not reflective at all.*

Abstract

Soft shadows, glossy reflections and depth of field are valuable effects for realistic rendering and are often computed using distribution ray tracing (DRT). These "blurry" effects often need not be accurate and are sometimes simulated by blurring an image with sharper effects, such as blurring hard shadows to simulate soft shadows. One of the most effective examples of such a blurring algorithm is *percentage closer soft shadows* (PCSS). That technique, however, does not naturally extend to shadows generated in image space, such as those computed by a ray tracer, nor does it extend to glossy reflections or depth of field. This limitation can be overcome by generalizing PCSS to be phrased in terms of a gather from image space textures implemented with cross bilateral filtering. This paper demonstrates a framework to create visually compelling and phenomenologically accurate approximations of DRT effects based on repeatedly gathering from bilaterally weighted image space texture samples. These gathering and filtering operations are well supported by modern parallel architectures, enabling this technique to run at interactive rates.

1 Introduction

Some of the most sought after effects in graphics are soft shadows, glossy reflections and depth of field. These effects are accurately computed using distribution ray tracing (DRT) [Cook et al. 1984]. DRT, however, is expensive, requiring tens to hundreds of rays per pixel. Many researchers have attempted to intelligently blur "sharp" images to get the subjective properties of distribution ray tracing while increasing performance. Such a blurring approach will not be able to produce exact results but in some scenes can produce phenomenologically compelling approximations of soft shadows, glossy reflections and depth of field. This paper describes a framework to produce such approximations using only single sampled image space textures and texture sampling. This sampling is used to "gather" a weighted average of sampled values from the texture maps to produce a final image (Figure 1) by approximating a per-pixel variable radius spreading operation.

Image space gathering (ISG) is inspired from the observation in Figure 2 that the rays that might be generated in a distribution ray tracer have corresponding nearby "rays" in an image produced by a traditional z-buffer or Whitted-style ray tracing program. While a correct image can be generated by *gathering* colors or visibility from the rays on the left side of Figure 2 , we only have information from the "rays" on the right images and "gather" from them instead. This is not as accurate as distribution ray tracing but is faster as we can heavily reuse samples. To perform this gather, we first store texture maps in image space containing information about depth, normals, hard shadows, sharp reflections, etc. For each shaded point we then query these textures to determine how far to look for nearby rays (a parameter search), and we then query those rays by point sampling the textures (a gather). The approach is efficient while capturing some of the subjective phenomena we seek such as contact hardening in shadows and reflections.

We believe the main practical contribution of the paper is the glossy reflections algorithm. It is a simple algorithm for an important visual effect and there are currently no viable alternatives to it for non-planar surfaces. The soft shadow algorithm presented also works well, but many competing methods exist. The depth-of-field algorithm presented is not yet a viable competitor to previous techniques and we discuss its limitations.

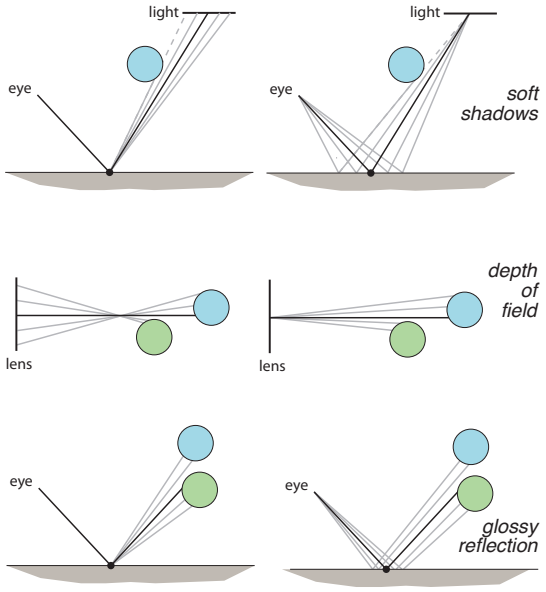

Figure 2: *Left: the rays that might be sent in a distribution ray tracer. Right: the "rays" that are sent in a traditional z-buffer render. While only the central ray is guaranteed to be in both ray sets, our goal is to gather information from the right rays that can give an estimate of the correct gather performed when the left rays are computed.*

2 Related Work

The most popular methods that compute glossy reflections, depth of field and soft shadows simultaneously are DRT and the accumulation buffer [Haeberli and Akeley 1990]. These require either many samples per pixel or many images to be generated. Other techniques have computed individual effects with "fat" cone tracing or pyramid rays which has proven problematic to apply to complex scenes [Amanatides 1984; Genetti et al. 1998]. Precomputed radiance transfer methods have been used to simulate glossy reflections and soft shadows, but these methods are best suited to fairly diffuse effects so are complimentary to our algorithm which works well with more high frequency phenomena [Kautz et al. 2005].

There are a plethora of approximate shadow algorithms; the most effective are surveyed by Bavoil [2008]. Of those, percentage-closer soft shadows (PCSS) is, in our opinion, the most successful in capturing the subjective properties of soft shadows [Fernando 2005]. PCSS uses two passes at each shaded point. The first pass is a point-based search over a large radius in the shadow map to determine what blurring radius is appropriate. The second pass is a blur implemented by point sampling the shadow map within that blurring radius. At each of those sample points a shadow map visibility query is made to determine if that sample should be counted, effectively implementing a bilateral filter [Paris et al. 2007] to prevent image space artifacts. Our shadowing algorithm is closely related to PCSS, but its details are different due to our shadow map being defined in image space. The lack of light space information makes preventing artifacts more challenging.

There have been many methods that take a sharp raster image and blur it to simulate depth of field. These fall roughly into two categories as surveyed in the recent paper by Lee et al. [2008]: post-filtering from a single layer or post-filtering from multiple layers. Methods that "splat" each pixel with its own circle of confusion do not neatly fit into this scheme but can be thought of as a layer per pixel as they must be depth sorted. A recent example of such an approach is presented by Kosloff et al. [2009] who also articulate why a scattering or spreading method has intrinsic accuracy advantage over gathering, while not mapping as well to modern hardware. None of these methods both operate from a single layer and allow an out of focus object in the foreground to blur across an in focus object. Our method does achieve that, gaining more accuracy than other single layer methods while retaining most of their simplicity.

Many methods compute glossy reflections from environment maps which is not the problem we are targeting. There are relatively few methods for creating glossy reflections of local objects. Arvo computes analytic results when visibility is known [Arvo 1995] but his method has not been shown to scale well to more complex environments. Landis [2002] blurs specular maps to achieve a glossy look, and shows results that are sufficiently plausible to be composited into film frames. Hensley et al. [2005] have used fast summed area tables to compute glossy reflections. Their method can create appealing contact hardening but it is currently limited to planar surfaces. Kulikova et al. [2001] blur specular reflections but do so with a constant width blur so do not produce contact hardening. Shah et al. [2007] use brickmaps to prefilter the geometry to decrease the needed sampling rate for glossy reflection and achieve excellent results, but their speed is not yet near interactive. Yu et al. [2008] achieve glossy reflections with contact hardening by using geometry images for their objects so they can do their blurring in an approximate object space. Their method works well, but places constraints on the model representation we would like to avoid.

One of the problems with gathering in screen space is preserving discontinuities in the image, such as depth edges, as well as preserving correct behavior on curved surfaces. To overcome this in a blurring kernel we must use a signal-dependent kernel function; one that relies on the image being blurred. Bilateral filtering provides a framework to do that by adding a *range function* to deal with kernel dependencies on the image [Paris et al. 2007]. We use a *cross* bilateral filter (also sometimes called a *joint* bilateral filter) where the filter's range function uses a different image than the image being filtered. In rendering, the bilateral filter has been used effectively for upsampling an illumination field [Sloan et al. 2007], and we will use it in a similar way but with specialized range functions and importance sampling depending upon the effect we seek to achieve.

3 Algorithm

Our algorithm is a post-process that operates on images produced by any rendering technique that can compute sharp shadows and reflections. We assume that a rendering system has created several images that will be consumed to produce the final composited image; these images may include normal maps, depth maps, screen space shadow maps, sharp specular reflection maps, world space position maps, etc. These images are then combined using a two-phase cross bilateral filter during a traditional shading pass. The goal is to simulate a spreading operation with a per-pixel dependent radius using only two gathering steps. We also do not need the resolution of the input images to match the resolution of the final image, enabling subsampling of the phenomenon we intend to blur if desired.

The first filtering phase is the *parameter search*. This is used to query the images for information to set the support of the second filtering phase, the *gather*. Both of these phases use cross bilateral information to produce the desired phenomenological result from only image space information. For example in the soft shadows case, we want the edges of our hard shadows to be blurred both inside the shadow boundary as well as outside of the shadow bound-

ary. By splitting the filtering into two phases, we enable the edges of sharp images to be blurred in both directions. The purpose of the parameter search is to look across these boundaries to set the support of the gather kernel, enabling blurring across these boundaries, not just within them.

To keep the running time of the algorithm independent of the support of the filter kernels we employ a point-sampled Monte Carlo method to evaluate the kernels' integration that is, itself, independent of input image resolution. We will first discuss the details of these filters and how we importance sample them, followed by a discussion of tuning the range and spatial weighting functions of the filter for each target effect.

From here onward we will assume that our images are parameterized by 2D texture coordinates in the range $[0, 1]$. Our filtering takes place at the point at 2D image space texture coordinate $\vec{p}_0$. The cross bilateral filter for that point is:

$$g(\vec{p}_0) = \frac{1}{k} \int \underbrace{r(\vec{p}, \vec{p}_0)}_{\text{range fn.}} \underbrace{w(\vec{p}, \vec{p}_0)}_{\text{weighting fn.}} \underbrace{I(\vec{p})}_{\text{image fn.}} \, dA, \tag{1}$$

where $\vec{p}$ is a point in screen space (the integration variable), I is the image we are blurring (e.g., opacity values for shadows), and k enforces a weighted average:

$$k = \int r(\vec{p}, \vec{p}_0) \; w(\vec{p}, \vec{p}_0) \, dA. \tag{2}$$

Here the range function is used to set the "relevance" of the point at $\vec{p}$ to the point $\vec{p}_0$. The spatial weighting function can vary with $\vec{p}_0$ (e.g., it may be related to the reflectance function BRDF at that point) but it does not depend on any values other than those at $\vec{p}_0$. This enables different blurring behavior on a per-pixel basis, for example, enabling differing amounts of glossiness in the reflections of different rendered materials. While we could evaluate the integrals above by summing over individual pixels, instead we view the images as continuous functions (i.e., a texture map with an interpolation rule) and point sample them using Monte Carlo integration. If we use uniform density random variables to sample the domain of these integrals this yields the Monte Carlo approximation g':

$$g'(\vec{p}_0) = \frac{\sum_{i=1}^{N} r(\vec{p}_i, \vec{p}_0) w(\vec{p}_i, \vec{p}_0) I(\vec{p}_i)}{\sum_{j=1}^{M} r(\vec{p}_j, \vec{p}_0) w(\vec{p}_j, \vec{p}_0)}. \tag{3}$$

To be a traditional Monte Carlo estimator, the numerator and denominator must have independent points to avoid bias, and M and N need not be equal. We could also importance sample with a density proportional to w which would likely lower error for large numbers of samples [Ernst et al. 2006]. However, our main artifact is geometric undersampling so we instead use a uniform density to spread the sample points, and thus the gaps between points, more evenly. We can make several alterations to that basic method to improve efficiency:

- use stratified or QMC samples [Dutré et al. 2006];
- use the same samples for the numerator and denominator;
- have the sums start at $i = 0$ thus using the point $\vec{p}_0$ being processed.

The last two items add bias that vanishes in the limit, and thus they make our technique a "consistent" Monte Carlo method [Arvo and Kirk 1990]:

$$g'(\vec{p}_0) = \frac{\sum_{i=0}^{N} w(\vec{p}_i, \vec{p}_0) r(\vec{p}_i, \vec{p}_0) I(\vec{p}_i)}{\sum_{i=0}^{N} w(\vec{p}_i, \vec{p}_0) r(\vec{p}_i, \vec{p}_0)}. \tag{4}$$

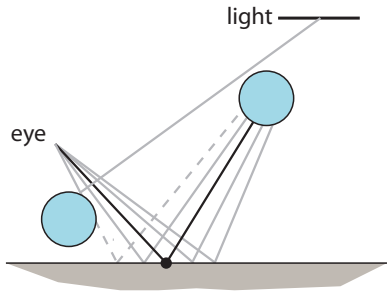

Figure 3: *Because some of the image space samples near the shaded point may be uncorrelated with the rays that we are trying to approximate, we need the range function to downweight or discount such points. In this example, the bottom sample represents a ray that is dissimilar from the ray we want to approximate (the dashed line) so it will be discounted from the filter's integral.*

3.1 Soft Shadows

The basic idea of the algorithm is to approximate the rays we would send in distribution ray tracing with a set of rays we trace in a traditional Whitted-style ray tracer as shown in Figure 2. This will not give us a correct answer, but can give subjectively pleasing results for most scenes. We assume that we have a shadow map stored in image space. This map stores, at each pixel, the distance from the shaded point to the nearest occluder as well as a bit indicating if the pixel is in shadow. That map could be produced, for example, by a ray tracer or by a z-buffer with irregular sampling as introduced independently by Aila and Laine [2004] and Johnson et al. [2005]. To drive our algorithm we make the same assumption as PCSS – the light, receiver and occluder are parallel and we can approximate the size of the penumbra using similar triangles. From this we will compute a final shadow opacity value that can be used in subsequent shading.

First we need to estimate the distance from the point being shaded to the virtual area light. We do this via a parameter search that is analogous to the blocker search in PCSS. We use a user-defined set of samples within a radius to search for potential occluders from the point being shaded. By changing the distribution of these samples it is possible to simulate differently shaped virtual area lights. The shape of the virtual area light is the convolution of a point and the shape of the spatial weighting function in the filter, for example, a disk of samples will simulate a spherical light while a rectangular array of samples will simulate a rectangular area light.

The parameter search radius, R_p is given by:

$$R_p = L / z_{\text{eye}}, \tag{5}$$

where L is a scaling factor representing the size of the virtual area light and z_{eye} is the eye space depth value of the shaded point used to project the light radius into screen space.

The values the parameter search are filtering are distances to occluders; the final result of the filtering will be a weighted average of the distances found in the search. Unlike PCSS, however, we only have screen space information and thus make use of a range function to discount samples that have low correspondence with the rays we are approximating, as seen in Figure 3. We use Equation 4 with a constant weighting function and the following range function:

$$r(\vec{p}_i, \vec{p}_0) = e^{-l^2/\sigma} \cdot \text{inShadow}(\vec{p}_i), \tag{6}$$

where

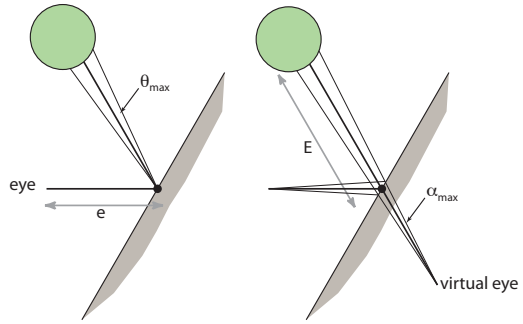

Figure 4: *Left: the information for a correct reflection is drawn from a cone with angle θ_{max}. Right: to approximate the left we gather information from within a cone starting at the reflection of the eye with an inner angle chosen to create the same cross section near the primary reflected object.*

$$l = \text{length}(\vec{w}(\vec{p}_i) - \vec{w}(\vec{p}_0)) \tag{7}$$

Here, $\vec{w}(\vec{p})$ is a lookup into a texture of world space positions for the screen space point $\vec{p}$. Effectively the parameter search range function is a gaussian that gives a similarity metric between two world space points, which is tunable by the σ parameter, and throws out filter taps that are not in shadow.

Given the estimated distance to the occluder by the parameter search, d, we can calculate the screen space shadow penumbra radius, R_s, by:

$$R_s = L\frac{d}{D-d} \tag{8}$$

where D is the distance from the shaded point to the light. Given the penumbra radius, we can project to screen space to calculate the support, R_g, of our gather filter:

$$R_g = R_s / z_{\text{eye}} \tag{9}$$

The gather then uses the same spatial and range weighting functions as the parameter search but with radius R_g and operating over the bits that signal a sample being in or out of shadow. We then arrive at an estimate the opacity of the shadow at the point being shaded. Figure 7 shows a model with varying L demonstrating variable penumbra size. Figure 8 shows a comparison with a ground-truth image.

3.2 Glossy Reflection

To simulate glossy reflection, we will need an input reflection map, consisting of reflected colors as well as the distance to that reflected object from the shaded point. A glossy reflection is a result of convolution over the hemisphere of incident directions using the BRDF ρ for the blurring kernel. This kernel can also be expressed in screen space. As shown in Figure 4, we first assume that the BRDF is zero outside of a cone with angle θ_{max}. The simulated BRDF may be zero inside of the cone as well; this angle serves as a bound to the maximum glossiness at the shaded point (and may vary from point to point).

Our parameter search, in order to capture the blurring of the reflection outside the boundary of the image of the reflected object, must search over the angle θ_{max}. The parameter search support, R_p, is just the ratio of θ_{max} and the field of view of the camera. (Note that the layout of the parameter search samples may change its aspect ratio to match the aspect ratio of the camera to maintain the shape of the simulated BRDF.)

Similar to soft shadows, our parameter search for glossy reflections operates over the image of distances to reflected objects, within radius R_p, to find a single distance value for the shaded point. We again use Equation 4 with a constant spatial weighting function and the following range function:

$$r(\vec{p}_i, \vec{p}_0) = e^{-\frac{a \cdot l^2 + b \cdot n^2}{\sigma}} \tag{10}$$

Where

$$l = \text{length}(\vec{w}(\vec{p}_i) - \vec{w}(\vec{p}_0)) \tag{11}$$

$$n = 1 - \vec{N}(\vec{p}_i) \cdot \vec{N}(\vec{p}_0) \tag{12}$$

Again, $\vec{w}(\vec{p})$ represents a texture of world space positions, a, b and σ are tuning parameters and $\vec{N}(\vec{p})$ is the corresponding texture of surface normals. This gives us a similarity metric that is sensitive to both the proximity of two points as well as the similarity of their surface normals.

Given the distance to a reflected object from the parameter search, we must now compute the support of the gather kernel. Looking at Figure 4 we can find the following trigonometric equivalence:

$$\tan\alpha = \frac{E \tan\theta_{\text{max}}}{e + E} \tag{13}$$

Where e is the distance from the eye to the shaded point, E is the distance from the shaded point to the reflected object and θ_{max} is our maximum BRDF angle from above. From this we can compute the angle we wish to sample over and project it into screen space by dividing by the camera's field of view. This gives us the screen space support of our gather kernel, R_s. To account for the tightening of BRDFs at grazing angles, we multiply this radius by the cosine of the incident angle to obtain our final gather support:

$$R_g = R_s \cos\theta_{\text{incident}} \tag{14}$$

The gather then operates over the image of reflected colors and uses the same cross bilateral filter function, with Equation 10 as the range function and a spatial weighting function that corresponds to the projection of the shape of the simulated BRDF inside of the cone into screen space. This can, for example, be constant, a gaussian falloff or a more complicated function to simulate other BRDFs.

3.3 Depth of Field

We devoted considerably more time to trying to make a good DOF algorithm than to shadows and reflections, but in the end it is the worst of the three algorithms in our opinion. Its weakness is for objects between the camera and the front focal plane. We believe this is due to the great difference between the rays needed for the correct calculation and the rays we have. This can be seen subjectively in Figure 2. The figure also implies the results might be better for objects past the front focal plane, and indeed the algorithm performs well in that region.

To aid our intuition about depth of field, we deal with its effects in image coordinates. Fundamentally, a lens causes the image of a point to project to an area of the screen called its circle of confusion (CoC). In world space, the circle of confusion changes linearly with distance from the lens, first shrinking as it approached the front focal plane, and then reaching zero (perfect focus) at that focal plane, and then increasing linearly with distance from that plane. In screen space, the radius c of that circle of confusion is:

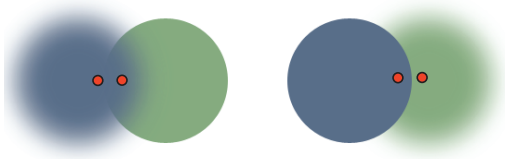

Figure 5: *In each of the pairs of spheres, the two red points would sample different spheres in the unblurred image. When shading at those points the pair on the left should get assigned similar colors, while the pair on the right should not. This example shows why depth ordering must affect our range function.*

$$c(\vec{p}_i) = \frac{|z(\vec{p}_i) - z_f|}{z(\vec{p}_i)} c_\infty \tag{15}$$

Where z is the depth lookup texture of the shaded point, z_f is the depth to the front focal plane, and c_∞ is the screen space CoC radius for a point at $z = infinity$.

Again, we proceed by first using a parameter search (with radius c_∞) to sample the circles of confusion around a shaded point followed by a gather to perform the blurring based on the computed CoC radius of the parameter search. We now need to design range functions that will capture the depth of field effect. For the case on the left of Figure 5, one possible regime is to take a parameter search and use the average of the circles of confusion we encounter for the second pass. This would work reasonably except for the case of an in focus object being shaded with an out of focus object behind it. This is because an out of focus object should blur across an in focus object, but that is not the case when an out of focus object is in back. So for the top right case we should use a gather with a small gather radius, while for the others we can use a uniform average of the samples we find. This suggests a range function inspired by Riguer et al.'s [2004] rule for handing an analogous case for blurring (as opposed to the parameter search we are doing):

$$r(\vec{p}_i, \vec{p}_0) = \begin{cases} c(\vec{p}_i), & \text{if } z(\vec{p}_i) > z(\vec{p}_0), c(\vec{p}_i) > c(\vec{p}_0); \\ c_\infty, & \text{otherwise.} \end{cases} \tag{16}$$

This main problem with this rule is that when the sampled points are all on the same surface as the center point there is a transitional artifact. That artifact has its roots in a related case for point based rendering where points should be splatted in a way that depends on whether they are on the same surface [Gross and Pfister 2007]. We deal with this by employing the *smoothstep* function to soften the transition between the two cases in the conditional above.

Once we have a radius, we can simply gather from the new region with the exception of the right case of Figure 5. Here we do not want the in-focus foreground object to bleed over the blurry background object. To accomplish that we again use a variant of the method of Riguer et al. [2004]:

$$r(\vec{p}_i, \vec{p}_0) = \begin{cases} c(\vec{p}_i), & \text{if } z(\vec{p}_i) < z(\vec{p}_0), c(\vec{p}_i) < c(\vec{p}_0); \\ c_\infty, & \text{otherwise.} \end{cases} \tag{17}$$

During the gather, changing the spatial weighting function can change shape of the final blur and can be used to simulate lens bokeh effects.

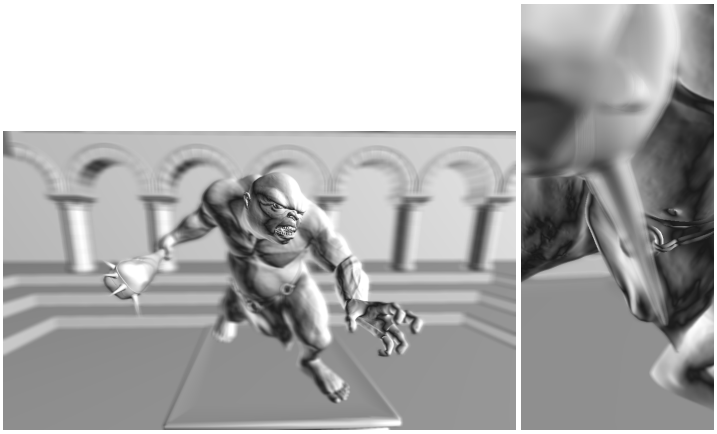

Figure 6: *Left: Bubs rendered with depth of field using single sampled color and depth buffers. Right: Depth of field artifact from missing information behind the spike. Notice that you cannot see the in focus ring in the background through the blurred foreground object.*

The main limitation of this technique is that when a foreground object blurs over an in focus background object we are missing information. A true distribution ray tracer sends rays "around" a foreground object because the source of the camera rays are from an area lens, not a single point as in our case. Because of this, large circles of confusion in the foreground exhibit artifacts in areas where this information is missing from our single sampled images, as seen in Figure 6.

4 Results

We implemented our algorithm on an NVIDIA Quadro FX 5800 using OpenGL, Cg and a preliminary version of the NVIDIA Interactive Ray Tracing API (NVIRT) [Robison 2009]. For a sampling pattern we used regular (grid) samples passed, on the fly, through the square to disk mapping of Shirley and Chiu [2005]. Our specular reflection and shadow maps were generated with NVIRT and stored in OpenGL textures. The basic rendering flow was to first rasterize ray requests into a texture using OpenGL (world space positions, normals, etc.) then map that texture to NVIRT. These ray requests are rasterized simultaneously using multiple render targets. NVIRT then takes the ray requests and casts shadow or reflection rays, storing the results into an output buffer that has been mapped from OpenGL. The final pass is a standard OpenGL rendering pass, where the results of the ray tracer are fed to Cg shaders as the inputs to the algorithm.

As an example, the soft shadow case needs only rasterize world space positions as ray requests: NVIRT traces a ray with an origin equal to the input world space position towards the light source. Multiple light sources can be handled by tracing multiple sets of rays from a single request image towards different lights and storing them into separate output buffers. For glossy reflection, we also need a texture of normals, which could be generated from triangle face normal or procedurally; this technique is fully compatible with bump or normal mapping techniques. The depth of field implementation does not use NVIRT and exists as a simple Cg shader. The Cg shaders themselves are very simple texturing lookup and filtering functions, simply sampling the specified images and weighting those samples, much like PCSS or any type of point-sampled blur.

The running time of the algorithm is roughly proportional to the number of samples taken in the parameter search and gather, as

212k tri, 1.8 ms base
49.7 ms preprocess

	9	25	49	81
9	8.2	14.2	26.3	38.2
25	16.9	19.9	32.4	44.7
49	22.2	31.6	40.9	53.5
81	36.1	42.4	51.7	64.3

322k tri, 3.6 ms base
477.5 ms preprocess

	9	25	49	81
9	46	79	128	193
25	84	118	170	238
49	139	174	226	295
81	212	246	299	369

1.8M tri, 0.3 ms base
8.4 ms preprocess

	9	25	49	81
9	3.8	6.9	21.7	35.4
25	6.6	9.1	24.8	38.6
49	10.1	11.9	28.9	42.9
81	12.9	14.7	31.6	45.4

Table 1: *Total time in milliseconds for rendering the final pass for various sampling rates at* 1024×768*. The columns vary in the number of samples taken in the parameter search and the rows vary in the number of samples taken for the gather. The baseline time represents the final shading pass without ISG. The preprocess time includes generating ray requests and invoking NVIRT for soft shadows and glossy reflections.*

summarized in Table 1. The number of samples required is dependent upon the application's tolerance for undersampling artifacts. Figure 11 shows a comparison between an undersampled and oversampled application of the algorithm. The tuning parameters for the bilateral filters' range functions do not require much tuning and we used values near 1×10^{-3} scaled by the inverse size of the scene for σ.

5 Discussion

We have presented a framework for approximating distribution ray tracing effects by gathering from image space textures. The main artifacts of ISG are visible in the figures and temporal artifacts are minimal. Any aliasing that is present in the input images is generally visible in the output of this algorithm, but given smooth input, the output will also be smooth. Additionally, these three techniques can also be used simultaneously, as shown in Figure 9.

From the data in Table 1 it is clear that this technique is not fast enough for real-time use on current hardware; however, it still provides interactive framerates with high numbers of samples per shaded point and high image quality. Additionally, this algorithm can be used in off-line post-processing or compositing tools that can tolerate the frame to frame latency for improving the image quality of single sampled rendered images. The running time is bounded by the number of texture accesses that occur during the processing of a fragment; you can see in the data that in the glossy reflections case having the range function depend upon values of the normal texture, and incurring twice as many texture reads because of that, has a substantial impact on performance compared to the soft shadows or depth of field cases.

Note that we have used ray tracing to generate some of our textures, and rasterization for others; the textures can come from any method that can generate the required data so the method is not tied to any particular visibility paradigm. We now summarize the approach's limitations and potential areas for extensions.

Figure 7: *Fern model with varying penumbra sizes.*

Limitations. As with any sample-based method, undersampling will cause visible artifacts, and it is straightforward to construct scenes with small bright objects that are arbitrarily hard to sample accurately. Our method aims to produce phenomenologically correct effects, not to match ground truths, and thus can produce incorrect but plausible results that are unsuitable for some applications. The textures we gather from contain only information directly visible to the eye point so it misses objects that are barely hidden but would contribute to the image produced by a distribution ray tracer. This is most noticeable with the depth of field technique.

Future Work. Anisotropic reflection as well as grazing reflection elongation could be added by using an elongated shape for the gather region. Additionally, the use of different sampling patterns or shifted/rotated patterns could improve the results for some scenes or applications.

The number of gathering samples could be made adaptive based on the results of the parameter search to provide uniform sampling rates and motion blur might be simulated by adding a texture with velocity information. Gaining more information about occluded geometry with a technique such as depth peeling could also improve the results.

Significance and Novelty. We believe the most novel and important technique in this paper is the handling of glossy reflections as we are unaware of viable alternatives that are both efficient and work on general models. The soft shadow algorithm we present is closely related to PCSS, but, unlike PCSS, can be used in conjunction with ray traced shadows. Whether this is significant will depend on how the future development community weighs hard to quantify tradeoffs in efficiency, artifacts, algorithmic complexity and implementation difficulty present when comparing light space and image space shadow algorithms. Our depth of field technique is an extension of Riguer et al.'s work to allow blurred foreground objects but suffers from major artifacts; we believe its subjective quality suffers more from the lack of occluded information than is the case for soft shadows or glossy reflections, but is useful when the accuracy of multilayer techniques is not needed.

Acknowledgements

Thanks to the creators and distributers of the models used for examples including Ryan Vance, Thomas Luft, Juan Carlos Silva, Alvaro Luna Bautista, Joel Andersdon, and the Stanford repository, to Randy Fernando and Louis Bavoil for many helpful discussions, to the anonymous reviewers for their detailed and constructive comments, and to Annen et al. [2008] for the use of content from Figure 7 of their paper.

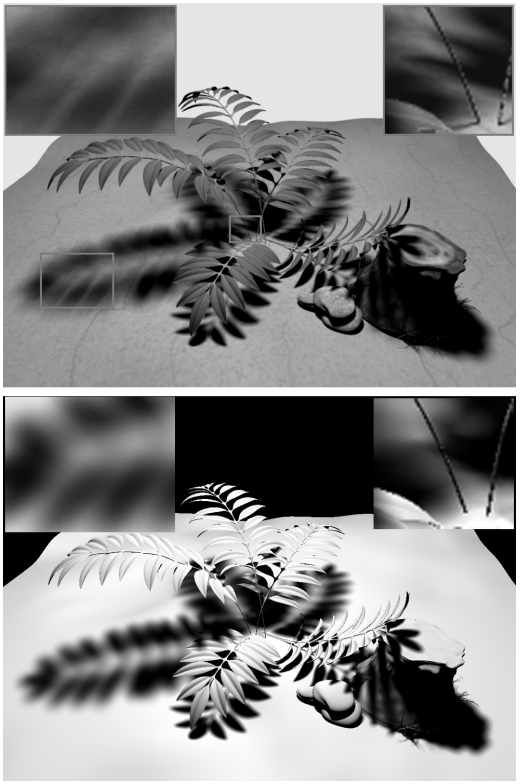

Figure 8: *Top: ray traced ground truth image from Annen et al. [2008]. Bottom: our technique with an untextured model with the same viewpoint and approximate penumbra size.*

References

AILA, T., AND LAINE, S. 2004. Alias-free shadow maps. In *Proceedings of Eurographics Symposium on Rendering*, 161–166.

AMANATIDES, J. 1984. Ray tracing with cones. In *Proceedings of SIGGRAPH*, 129–135.

ANNEN, T., DONG, Z., MERTENS, T., BEKAERT, P., SEIDEL, H., AND KAUTZ, J. 2008. Real-time, all-frequency shadows in dynamic scenes. In *Proceedings of SIGGRAPH*.

ARVO, J., AND KIRK, D. 1990. Particle transport and image synthesis. In *Proceedings SIGGRAPH*, 63–66.

ARVO, J. 1995. Applications of irradiance tensors to the simulation of non-Lambertian phenomena. In *Proceedings of SIGGRAPH*, 335–342.

BAVOIL, L. 2008. Advanced Soft Shadow Mapping Techniques.

COOK, R. L., PORTER, T., AND CARPENTER, L. 1984. Distributed ray tracing. In *Proceedings of SIGGRAPH*, 137–145.

DUTRÉ, P., BALA, K., AND BEKAERT, P. 2006. *Advanced Global Illumination*. AK Peters, Ltd.

ERNST, M., STAMMINGER, M., AND GREINER, G. 2006. Filter Importance Sampling. In *IEEE Symposium on Interactive Ray Tracing*, 125–132.

FERNANDO, R. 2005. Percentage-closer soft shadows. In *SIGGRAPH Sketch*.

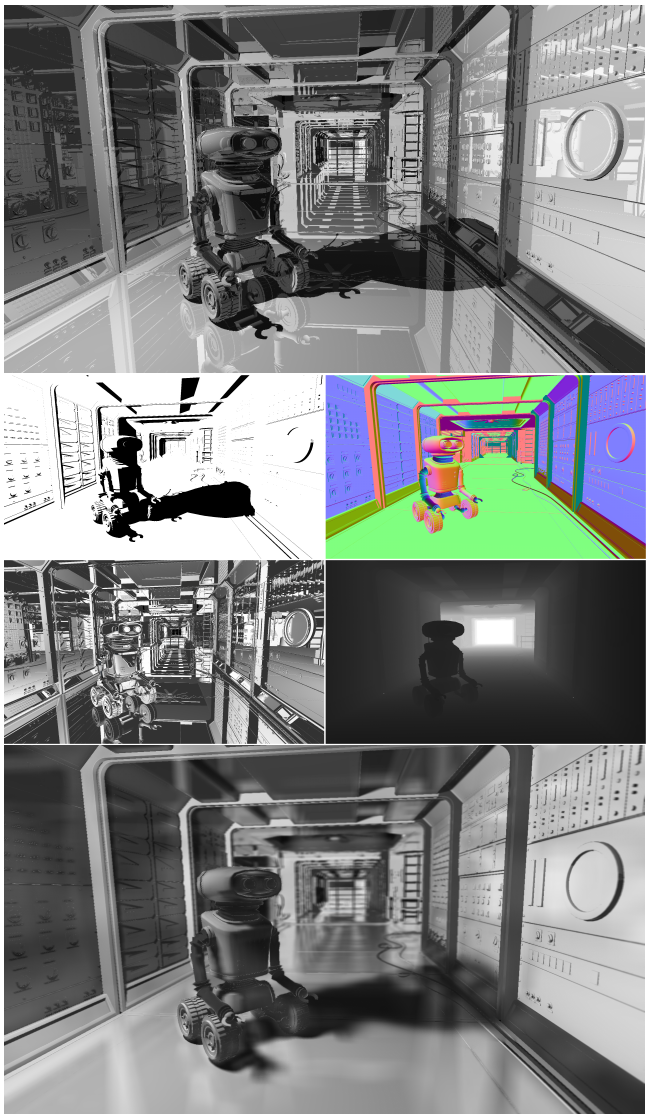

Figure 9: *Top: A conventionally rendered image with "sharp" shadows, specular reflections and focus. Center: a set of textures that are a natural byproduct of rendering the top image. (Clockwise from upper left: screen space shadow map, normal map, depth map and reflection map.) Bottom: an image computed by gathering values from the center texture images.*

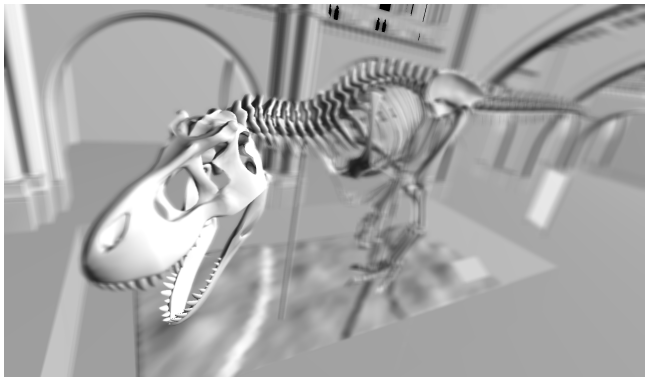

Figure 10: *A model rendered with the depth of field technique.*

Figure 11: *Top: 9 parameter search samples and 9 gather samples. Bottom: 81 parameter search samples and 81 gather samples. Notice the undersampling artifacts present in the top image, both in the detection of blur radii and within the blurs themselves. Please see the supplementary materials for images with other sampling rates.*

Figure 12: *Left: No ISG, this is the unblurred reflection map. Middle: ISG reflections with a small glossiness factor. Right: ISG reflections with a large glossiness factor.*

GENETTI, J., GORDON, D., AND WILLIAMS, G. 1998. Adaptive Supersampling in Object Space Using Pyramidal Rays. *CGF 17*, 1, 29–54.

GROSS, M., AND PFISTER, H. 2007. *Point-based Graphics*. Morgan Kaufmann.

HAEBERLI, P., AND AKELEY, K. 1990. The accumulation buffer: hardware support for high-quality rendering. In *Proceedings of SIGGRAPH*, 309–318.

HENSLEY, J., SCHEUERMANN, T., COOMBE, G., SINGH, M., AND LASTRA, A. 2005. Fast Summed-Area Table Generation and its Applications. In *Computer Graphics Forum*, vol. 24, Blackwell Synergy, 547–555.

JOHNSON, G. S., LEE, J., BURNS, C. A., AND MARK, W. R. 2005. The irregular z-buffer: Hardware acceleration for irregular data structures. *ACM Trans. Graph. 24*, 4, 1462–1482.

KAUTZ, J., LEHTINEN, J., , AND SLOAN, P. 2005. Precomputed radiance transfer: theory and practice. In *SIGGRAPH course notes*.

KOSLOFF, T., TAO, M., AND BARSKY, B. 2009. Depth of field postprocessing for layered scenes using constant-time rectangle spreading. In *Proceedings of Graphics Interface*.

KULIKOVA, A., DMITRIEV, K., AND MOSCOW, R. 2001. Fuzzy reflections rendering. In *The 11-thInternational Conference on Computer Graphics and Computer Vision, August/September*.

LANDIS, H. 2002. Production-ready global illumination. *Siggraph Course Notes 16*.

LEE, S., KIM, G., AND CHOI, S. 2008. Real-Time Depth-of-Field Rendering Using Point Splatting on Per-Pixel Layers. *IEEE Transactions on Visualization and Computer Graphics* (September/October).

PARIS, S., KORNPROBST, P., TUMBLIN, J., AND DURAND, F. 2007. A gentle introduction to bilateral filtering and its applications. In *SIGGRAPH Course Notes*.

RIGUER, G., TATARCHUK, N., AND ISIDORO, J. 2004. Real-Time Depth of Field Simulation. *ShaderX 2*, 529–556.

ROBISON, A., 2009. Interactive Ray Tracing on the GPU and NVIRT Overview, February. `http://www.nvidia.com/research`.

SHAH, A., RITTER, J., KING, C., AND GRONSKY, S. 2007. Fast, soft reflections using radiance caches. In *SIGGRAPH Sketch*.

SHIRLEY, P., AND CHIU, K. 2005. A Low Distortion Map Between Disk and Square. *Graphics Tools: The Jgt Editors' Choice*.

SLOAN, P., GOVINDARAJU, N., NOWROUZEZAHRAI, D., AND SNYDER, J. 2007. Image-Based Proxy Accumulation for Real-Time Soft Global Illumination. In *Pacific Graphics*, 97–105.

YU, X., WANG, R., AND YU, J. 2008. Interactive Glossy Reflections using GPU-based Ray Tracing with Adaptive LOD. In *Pacific Graphics*.