Phenomenological Transparency

Morgan McGuire and Michael Mara

Abstract—Translucent objects such as fog, clouds, smoke, glass, ice, and liquids are pervasive in cinematic environments because they frame scenes in depth and create visually-compelling shots. Unfortunately, they are hard to render in real-time and have thus previously been rendered poorly compared to opaque surfaces.
This paper introduces the first model for a real-time rasterization algorithm that can simultaneously approximate the following transparency phenomena: wavelength-varying ("colored") transmission, translucent colored shadows, caustics, volumetric light and shadowing, partial coverage, diffusion, and refraction. All render efficiently with order-independent draw calls and low bandwidth. We include source code.

Index Terms—transparency, order-independent, refraction, caustic, shadow, particle, volumetric

1 Introduction

Occlusion is the most powerful depth cue, surpassing even perspective and shadows. Partial occlusion due to various forms of transparency gives a strong perception of depth without the limitation of concealing shape. This is one reason that film directors rely heavily on fog, smoke, and glass in cinematic composition. Another reason is that effects such as rolling fog and layered glass refraction create interesting visual complexity.

Beyond just layering, such transparent surfaces present a variety of important phenomena seen in figure 1. These include diffusion by frosted glass and fog, wavelength-varying transmission in colored glass, colored and translucent shadows, multiple layers of refraction, self-shadowing within dense media such as clouds, and relatively bright caustics in shadows.

Many of these phenomena have previously been hard to render efficiently under rasterization. Although there are some good special-case solutions, there is no previous real-time rasterization method that models all of them simultaneously, let alone efficiently. Thus, these important visual elements from film been absent from real-time rendering 3D applications. This absence is particularly unfortunate because depth cues are even more important in an interactive context than in film. Game players, CAD engineers, and 3D artists need to understand depth relationships to perform tasks in 3D. Furthermore, today's stereo screens and head-mounted virtual reality (VR) displays lack the depth cue of focal accommodation. Robust transparency by ray tracing allows ocular vergence to partly compensate for the lack of focus in pre-rendered VR content by presenting multiple viable vergences at each pixel. Our approximation brings complex transparency to real-time VR for the first time (see figure 24).

We introduce a new order-independent transparency (OIT) model for light scattering, and algorithm for rendering with it in real-time on modern GPUs. These emulate the real physical phenomena described in this section through empirically-derived methods. Our model extends previous stochastic and order-independent transparency methods that reduce complex scenes to a single amalgamated layer per pixel. This provides for efficient unordered submission of draw calls, avoids costly primitive and fragment sorting, and limits per-pixel bandwidth and storage requirements. It also has the advantage of smoothly transitioning when primitives abruptly change order, for example, eliminating the "popping" associated with particle systems under ordered transparency. Quantitatively, we demonstrate increased visual effects while achieving a $10\times$ reduction in memory traffic compared to previous colored shadows [1] alone and transparency at a $2\times$ reduction compared to α-only k-buffering [2], [3].

Each approximation that we introduce contains radiometric error, compared to a ray tracer. We contend that for many real-time applications, radiometric error is less important than perceptual error. By emulating physical phenomena, our model significantly improves the perception of transparency in many scenes compared to the alternative of no approximation of those phenomena at all.

This paper contributes new effects and demonstrates real-time performance on modern GPUs and will scale well under bandwidth trends. It extends our previous work [4] on phenomenological transparency with new volumetric results, a solution for screen-space ambient occlusion at the edges of interpolated binary-coverage surfaces such as tree leaves and fences, an unbiased solution for saturation, and new performance optimizations.

2 Related Work

For each of the transparency phenomena that we model, the literature includes many real-time solutions for rendering that phenomenon *in isolation*, typically with more accuracy and higher cost than our model and with greater constraints on the scene. We review some of the most recent below. Our approach is better suited to cases where a coarser approximation is acceptable in exchange for a significant performance increase and reduction in implementation complexity, and integration of all of these phenomena into a unified transparency model.

Sorted transparency operates by the "painter's algorithm," rendering surfaces in back-to-front order. In addition to the sorting and ordered submission costs, it is undesirable because it requires slicing or simply fails when no consistent ordering exists between surfaces. In scenes for which a perfect ordering exists, a two-pass[1]

1. This can be done in one pass for monochrome transmission or on hardware with programmable blending. Fixed-function blending cannot simultaneously modulate and accumulate with λ-varying constants.

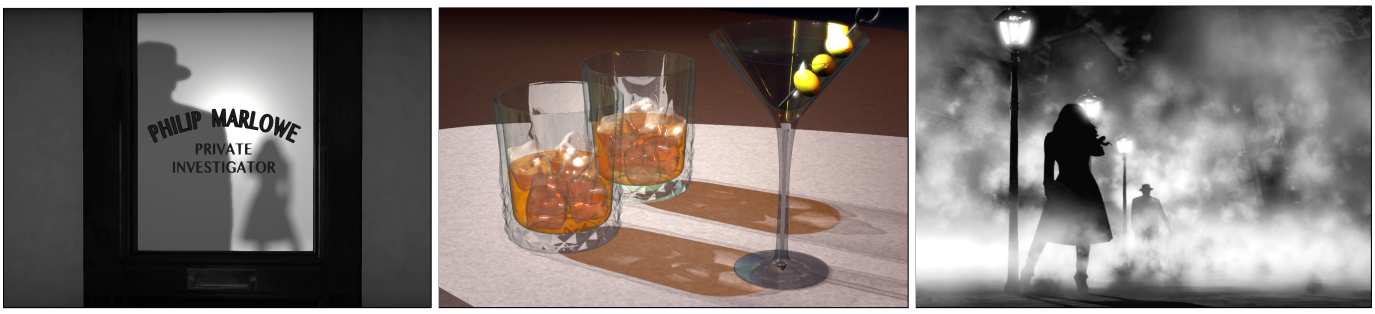


Fig. 1: *Three scenes rendered in real-time by our method, none of which were possible under previous real-time transparency algorithms. Left) Distance-based diffusion by a frosted glass door. Center) Colored transmission with multiple layers of refraction and reflection and colored shadows with caustics. Right) Diffusion and shadowing in a participating medium with partial coverage and varying density. Images © 2015 NVIDIA.*

implementation that first modulates the background by transmission and then adds reflected and emitted light can accurately model non-refractive transmission and partial coverage. We use that as a reference solution for those effects in our results.

2.0.1 k-buffers

Carpenter's [5] A-buffer was the first method for allowing order-independent submission of partial-coverage surfaces to a renderer, albeit at unbounded space and time costs for n surfaces. The Z^3 method [6] stores only bounded k transparent surfaces and merges the other $n - k$ ones. Z^3 hardware has never been built, but a variety of related k-buffer methods were devised for increasingly programmable, real-world GPUs [2], [3], [7], [8], [9].

Parallel to the visible surface k-buffer work are methods for modeling partial coverage in the light's view for shadowing [10], [11], [12] in various cumulative-opacity vs. depth curve representations.

2.0.2 Stochastic Transparency

A stochastic, a.k.a. "screen door" transparency algorithm stores only one layer per sample. These methods are good for primary visibility and shadow partial coverage, but expensive because they require high (e.g., $128\times$ MSAA) sample counts or wide filters [1], [13]. We combine this idea for shadows with the pre-filtering of Variance Shadow Maps (VSM) [14] to solve the performance problem.

2.0.3 Blended Transparency

A key idea in the k-buffer methods is that they all aggregate the $n - k$ overflow fragments. Meshkin [15] proposed a blended transparency method that is essentially a $k = 1$-buffer, aggregating *all* fragments. He redefined the compositing operator to estimate all order-dependent terms as zero. Subsequent work [16], [17], [18] extended this through more accurate approximations and normalization terms. The core blending of our method advances this line of research.

2.0.4 Realistic Scattering

The real-time methods we've described so far were all designed for simple Porter-Duff partial coverage (α-blending). That's a good model for monochrome transmission through a single-scattering medium, and is sufficient for some applications, such as rendering thin smoke and antialiasing opaque edges. However, it fails to model other real-world situations, such as color modulation of objects viewed through colored glass and the blurring seen when an object is in thick fog.

We are aware of no previous real-time diffusion solution, however there is much work on offline diffusion simulation. Nishita et al. [19] derived early models of scattering in participating media at planetary scale. Narasimhan et al. [20], [21] describe how to compute a spatially-varying point spread function for diffusion in participating media based on the Legendre approximation of the Henyey-Greenstein phase function. They note that a *single* Gaussian blur is an insufficient model because it must vary at every pixel and take occlusion into account; we present exactly those extensions. Premože et al. [22] give a derivation for the screen-space Gaussian from physical parameters.

2.0.5 Refraction

The simplest refraction model is to render a dynamic environment map for each object and compute single-surface refraction of distant light. This obviously does not scale to large scenes or multiple refractions. More sophisticated recent methods include models of distant light through rough surfaces [23] and screen-space layered G-buffer ray tracing [24], [25].

2.0.6 Caustics

Caustic methods have been developed for limited circumstances such as flat receivers [26], individual transmissive surfaces [27], and exactly two layers [28]. Wyman et al.'s [29] recent work is extremely accurate for monochrome caustics in isolation.

3 Algorithm

We compute transparent phenomena in three stages. We do not modify opaque surface rendering, which happens between our first two stages and can be via any algorithm: forward-shaded, deferred-shaded, ray traced, etc.

Shadows maps: First, for each light source, we produce a colored stochastic shadow map of transmissive surfaces that is affected by caustics.

OIT: Second (after opaque surfaces have been rendered), we perform an order-independent transparency (OIT) pass. This iterates over all transparent surfaces with a pixel shader that outputs each surface's contribution to accumulated reflected light A_{rgba}, transmission β_{rgb}, diffusion factor D, and refractive displacement δ_{xy} buffers. Section 4 derives these. The appendix details configuring fixed-function blending for the required sums and products. For surfaces such as fog particles that lack fine detail, the second stage can be performed at low resolution. We then upsample the results with a bilateral filter and combine them with full-resolution ones, allowing mixed-resolution results.

Resolve: Third, we iterate over pixels in screen space that were affected by transparent surfaces to resolve the output of the second stage into the image of the opaque surfaces in the framebuffer.

4 Scattering Model

4.1 Material Parameters

We model transparent materials with a reflective BRDF (lambertian and microfacet + Fresnel glossy terms), emission term, partial coverage (α), and a BTDF. Our BTDF parameters are refractive index (η), wavelength-varying transmission coefficient at normal incidence ($T(\lambda)$), and a collimation factor ($c \in [0, 1]$). Unit collimation describes an optically homogeneous material like perfect glass that exhibits no multiple-scattering within its volume. Zero collimation describes a dense, isotropic participating medium such as muddy water or fog.

The transmission coefficient $t(\lambda)$ used in shading is $T(\lambda)$ modulated by one minus the BRDF, both evaluated for the actual angle of incidence. Our shading model also uses $\bar{t}$, the mean of $t(\lambda)$ over wavelength.

Both the transmission coefficient and collimation factor assume that light travels a fixed distance through a medium after the surface before exiting; otherwise one would have to apply Beer's law to accumulate their impact per pixel. This is a common real-time rendering assumption that is reasonable for drinking glasses, windows, and particles. It is a poor approximation for something like an ice sculpture that has significantly varying thickness.

Refractive index is specified once per material instead of per texel. We pack the three-channel $T(\lambda)$ and scalar c into an sRGBA8 texture. Because c is encoded in the A channel, unspecified collimation conveniently defaults to $c = 1$ for legacy texture maps.

4.2 Wavelength-Varying Blending

Consider a series of n translucent surfaces ordered from back to front towards the viewer at a pixel and an opaque backing surface that scatters radiance $L(X_0, \lambda)$ towards the camera. (This ordering is only for the derivation. Our algorithm accepts surfaces in any order.)

The outgoing radiance towards the camera at surface point X_i and wavelength λ is the sum of the reflected[2] and transmitted light at the point [30]:

$$L(X_i, \lambda) = \alpha_i \cdot \left[L_{\mathrm{r}}(X_i, \lambda) + t_i(\lambda) L(X_{(i-1)}, \lambda)\right] + (1 - \alpha_i) \cdot L(X_{(i-1)}, \lambda). \quad (1)$$

2. we include emitted light in the "reflected" term to simplify notation

Here, we express the radiance that *would* exist with full coverage and then scale it for the actual coverage α_i to make explicit the blending structure. The transmission coefficient t_i may vary with wavelength, and with orientation due to Fresnel effects. For the moment, we ignore refraction. No direction variable appears because we're explicitly only measuring radiance towards the camera.

The net modulation of the background $L(X_0, \lambda)$ after transmission through, and partial-coverage by, all translucent surfaces is exactly [1]

$$\beta(\lambda) = \prod \left[1 - \alpha_i + \alpha_i t_i(\lambda)\right]. \quad (2)$$

(β for background.) Because β is a product and includes *all* surfaces, order is irrelevant. Since our goal is to aggregate all surfaces into a single-layer, i.e., $(k = 1)$-buffer, for later composition, the net solution must be expressed in the form

$$L(X_n, \lambda) \approx \beta(\lambda) \cdot L(X_0, \lambda) + (1 - \beta(\lambda)) \cdot U(\lambda), \quad (3)$$

where $U(\lambda)$ is a weighted sum of the radiance from the translucent surfaces, whose form is unknown *a priori*.

Note that the weight w_i of the contribution from layer i transmitted through, and partially covered by, surfaces $(i + 1)...n$ is independent of the order of those other layers [15].

If the space between X_i and the camera were filled with a homogeneous participating medium instead of inhomogenous discrete surfaces, then by Beer's law, the radiance $\alpha_i \cdot L_{\mathrm{r}}(X_i, \lambda)$ scattered from surface i that reached the camera would be $w_i = e^{-q|X_i|}$ [17]. We can normalize by total weight to limit the influence of q, since we're only computing color and the magnitude of transmission from all surfaces is known to be $(1 - \beta)$. This gives a solution for an implicit q:

$$U(\lambda) = \frac{A_{\mathrm{rgb}}(\lambda)}{A_{\mathrm{a}}} = \frac{\sum_i w_i \cdot \alpha_i \cdot L_r(X_i, \lambda)}{\sum_i w_i \cdot \alpha_i \cdot (1 - \bar{t_i})} \quad (4)$$

We name the numerator and denominator accumulations in equation 4 $A_{\mathrm{rgb}}(\lambda)$ and A_{a} respectively for later reference in the OIT stage implementation, where they are encoded in an explicit RGBA buffer.

Thus far, our derivation assumed a uniform medium. Since the scene is actually filled with inhomogeneous discrete surfaces, weight w_i should not actually follow from Beer's law but instead some scene-dependent function that is also monotonic and super-linearly decreasing in depth (because it is a product). We use

$$w_i = \min\Big(\max\Big(\big[10 \cdot (1 - 0.99f) \cdot \alpha_i \cdot (1 - \bar{t_i})\big]^3, 0.01\Big), 30\Big) \quad (5)$$

with $f =$ `gl_FragCoord.z`, which is McGuire and Bavoil's [18] equation 10 scaled for an infinite far plane and float16 precision. Note that in this formulation, surfaces observed off the axis of projection have greater distance from the camera than those in the center of the screen. This is a source of error, however, its impact is small because each pixel is normalized independent. The relationships of the z-coordinates of different layers are significant than their absolute values in this equation. The z-coordinates of camera-space points on the same view ray are proportional to their 3D distances from the origin, so the relationships between layers within a pixel are preserved under this optimization.

The sums and products in β and A can all be computed in a single pass over transparent surfaces using multiple render targets

and fixed-function blending. The appendix contains the OpenGL configuration and shaders.

4.3 Multiple Forward Scattering Approximation

4.3.1 Diffusion

Diffusion of the background occurs when transmissive surfaces decollimate light due to repeated scattering, or due to an extremely rough transmissive interface. We explicitly model the former case and also allow artists to approximate the latter.

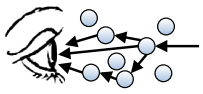


Fig. 2: *Multiple scattering events*

As depicted in Figure 2, multiple scattering in a uniform medium produces a random walk. In screen space, this means that there is a Gaussian distribution of pixel offsets between where light enters and emerges from a surface [22]. The standard deviation of the Gaussian, and thus the diffusion in pixels, decreases with distance from the camera $|z_i|$ (due to perspective) and with collimation c_i.

Following previous work [20], [21], [22], we estimate the total diffusion standard deviation D (in pixels) at each pixel from all foreground surfaces indexed by i, *for an infinitely distant background*, as

$$D = \frac{W}{2\tan(\theta/2)} \sum_i \frac{k_0(1-c_i)\alpha_i}{|z_i|}, \tag{6}$$

where θ is the field of view, W is screen width in pixels, and k_0 is the standard deviation (in meters) of the Gaussian due to a single $c = 0$ surface at a depth of one meter.

What about diffusion of background surfaces at *finite* distances? Intuitively, repeated scattering has the net effect of a collection of concave lenses. The Gaussian kernel point spread function describing the net scattering is the convolution the Gaussian point spread functions of the individual scattering surfaces. The standard deviation of that net kernel is the sum of the standard deviations of the individual Gaussians. Figure 3 depicts the cone of light from background object B and its random-walk dispersion in the medium (B also scatters light that never reaches the camera, e.g., in directions shown in light gray) that gives rise to this Gaussian model [20] at each surface.

Moreover, the amount of defocus from this system increases with the distance from the surface to the background and may be close to zero. That is, the standard deviation of each screen-space Gaussian kernel increases with the distance between the layer that it models and the background. For example, a hand pressed against a frosted glass shower door will be appear sharp, whereas the body behind it is diffused.

Let $z_{B,i}$ be the depth of the camera-space background B as read from the depth buffer, $z_{X,i}$ be the depth of the surface X, and constant k_1 the implied thickness of the scattering surface (we simply use $k_1 = 0.5$m for all scenes).

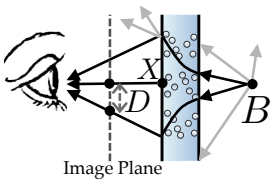


Fig. 3: *Diffusion of the background, B*

Under this compound, convex lens and the previous background model, we approximate the diffusion standard deviation for arbitrary B as:

$$D = \frac{W \cdot k_0}{2\tan(\frac{\theta}{2})} \sqrt{\sum_i \left[\frac{(1-c_i)\alpha_i \left[1 - \frac{1}{1+|z_{X,i} - z_{B,i}|/k_1} \right]}{|z_{X,i}|} \right]^2}. \tag{7}$$

4.3.2 Refraction of Primary Rays

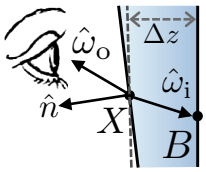


Fig. 4: *Primary ray refraction*

Many real-time renderers approximate refraction by exactly applying Snell's law using a simplified model of the scene: a homogeneous volume of transmissive medium extending backwards to a fixed-depth background plane, as depicted in Figure 4. Some use screen-space ray tracing to compute slightly more accurate refraction at increased cost. Both produce an offset (δ_X, δ_Y) in pixel coordinates by which to distort the background objects already appearing in the frame-buffer.

In the absence of a principled method for combining refraction vectors from overlapping surfaces, we simply blend them. This produces a result that is as exact as previous methods for a single transmissive surface. For multiple surfaces, it is incorrect but captures the displacement phenomenon of refraction.

Consider the (camera space) ray to the center of projection from a surface point X with normal $\hat{n}$ and relative refractive index

$$\eta = \frac{\eta_{\text{before}}}{\eta_{\text{after}}}, \tag{8}$$

in direction $\hat{\omega}_o$. By Snell's law, that ray refracted *from* direction

$$\hat{\omega}_i = -\eta\hat{\omega}_o - (\eta - \hat{n} \cdot \hat{\omega}_o + \sqrt{k})\hat{n} \tag{9}$$

if $k = 1 - \eta^2[1 - (\hat{n} \cdot \hat{\omega}_o)^2]$ was non-negative (otherwise, the ray had no transmissive transport). For a background of a plane at fixed depth Δz beyond X, the observed point is given by

$$B = X + \Delta z \frac{\hat{\omega}_i}{\hat{\omega}_i \cdot \hat{z}}. \tag{10}$$

To obtain the refractive pixel offset (δ_X, δ_Y) due to a single layer , we project each B into screen space and subtract the current pixel coordinate. We then average the results from all layers weighted by coverage and opacity (see the implementation for details). Δz is a per-scene parameter; we used $\Delta z = 2$m for all results.

4.3.3 Caustics and Translucent Shadows

In the real world, focusing under refraction redistributes light within the shadowed region, producing lighter and darker areas as shown in Figure 5. We approximate this by varying the density of a stochastic shadow map as follows.

Colored stochastic shadow maps [1] model shadows from surfaces with wavelength-varying transmission using RGB values instead of a single depth map. During shadow map generation, they write the depth to the shadow map for wavelength λ with probability $\rho(\lambda)$ proportional to partial coverage and 1 - transmission. This

works because a shadow map stores surfaces of primary visibility for a light source.

To extend stochastic shadow maps to mimic caustics, we decrease shadowing where light strikes a transmissive surface at normal incidence. We increase shadowing where the refractor is at a glancing angle and thus has a low Fresnel transmission factor.

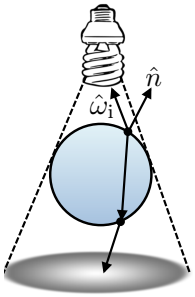


Fig. 5: *Caustic formation*

Specifically, during shadow map rendering, we increase $\rho(\lambda)$ where the magnitude of the dot product of the surface normal $\hat{n}$ and the vector to the light $\hat{\omega}_\mathrm{i}$ is close to 1, and decrease $\rho(\lambda)$ where the dot product is close to zero. We also increase this effect with the relative refractive index η. Including some contrast enhancement and relevant clamping to the unit interval gives our empirically-tuned equation:

$$s = \min(\max((1/\eta - 1)/2, 0), 1) \quad (11)$$

$$g = 2 \cdot \min(\max(1 - |\hat{n} \cdot \hat{\omega}_\mathrm{i}|^{128 \cdot s^2}, 0), 1) - 1 \quad (12)$$

$$\rho(\lambda) = \min(1, (1 + g \cdot s^{0.2}) \cdot \alpha \cdot (1 - t(\lambda))) \quad (13)$$

Intutively, s is the strength of the effect as determined by η, and g remaps the measure of incidence to increase its contrast.

We render one stochastic shadow map per wavelength, or a single map when the scene is known to have no wavelength-varying transmission. Stochastic shadow maps are known to require large filters to suppress noise during shading. In order to reduce the cost of this filtering, we convert the shadow maps to Variance Shadow Maps (VSMs) [14], filter them with a 15×15 tent kernel (in two 1D passes), and downsample by a factor of four in each dimension. The resulting shadow map costs 64 bits per texel per wavelength and need only be sampled once per wavelength during shading to produce noise-free shadows.

To avoid light leaks from the VSMs, we use a separate, leak-free Williams shadow map for opaque shadows cast onto both transparent and opaque surfaces and accept the light leaking for transparent shadows (since the shadow is already translucent).

5 Resolve Stage

The resolve stage comprises optional upsampling followed by a full-screen pass implemented as pixel shading of a screen-space quad, or as a compute shader. It performs the following steps:

5.0.1 Upsample

When accumulating particles at low resolution, we begin the resolve pass by joint-bilateral upsampling the low-resolution buffers as if they were simply colors, using the depth buffer as a key. See Tatarchuk [31] for filtering details. We use a 7×7 tent filter kernel and composite the result of the upsample into the full-resolution buffers using the same blending modes as for the transparency stage (see the appendix), except that the render target with index 1 (RT1) uses `glBlendFuncSeparatei(1, GL_ZERO,GL_SRC_COLOR, GL_ONE, GL_ONE)`.

5.0.2 Refract

For each pixel (x_0, y_0) in the output, we read from the transparent buffers at (x_0, y_0) but sample the background with a filter centered around $(x, y) = (x_0, y_0) + \delta_{\mathrm{xy}}$. Note that transparent surfaces cannot refract one another under this model.

5.0.3 Diffuse

We gather from the background according to a Gaussian point spread function [22] with standard deviation D (this is a single pixel when $D = 0$) for diffusion. Some coefficients in the kernel must also be masked out with zeros to model occlusion by objects that experience less diffusion. We do this by gathering at output pixel (x, y) only from pixels $(x + i, y + j)$ at which $\min(D[x, y], D[x + i, y + j]) \leq ||(i, j)||$. This filter is unfortunately not separable and is thus expensive when D is large. Recognizing that this is similar to depth-of-field (DoF) rendering with the circle-of-confusion parameter replaced by D, we speculate that techniques from DoF methods could accelerate diffusion.

5.0.4 Modulate and Combine

Transmissive modulation of the opaque background as captured in β is exact. However, transmissive surfaces don't modulate each other in the A channels, so sometimes they have an ambiguous appearance where they overlap in the image. For example, a white glossy highlight on a distant glass surface will not be colored in A by a foreground glass object with highly saturated transmission color through which it is observed (see figure 19). To compensate for this, we blend 50% of the hue of β into A when we normalize by the sum of the weights in A_a:

$$U' = \frac{A_\mathrm{rgb}}{A_\mathrm{a}} \left[\frac{1}{2} + \frac{\beta}{2 \max(\max_\lambda(\beta), \epsilon)} \right] \quad (14)$$

We call the term in brackets *self-modulation* because it applies a portion of the β modulation to the transmissive surfaces themselves.

With the background value filtered by diffusion and refraction and A and β sampled from the buffers, we compute U' and then simply apply equation 3 to compute the final pixel radiance.

6 Inferred Ambient Occlusion

Ambient occlusion (AO) is a standard effect in both offline and real-time rendering for contact shadows and darkening environment lighting in locally-concave areas. Most real-time renderers use screen-space AO methods that estimate local occlusion from the depth buffer (e.g., [32]).

Because transparent surfaces typically do not write to the depth buffer, they necessarily cannot *cast* AO under screen-space AO techniques. While this leaves nearby surfaces slightly brighter than they perhaps should be, transparent surfaces would naturally contribute less occlusion than opaque surfaces, so their very nature limits the impact on quality of this omission.

Transparent surfaces also cannot *receive* AO under screen-space AO. That is because those methods compute the AO value at each depth buffer pixel in a deferred rendering pass. They do this so that the noise from stochastic sampling of AO can be smoothed in screen-space before the results are applied in the

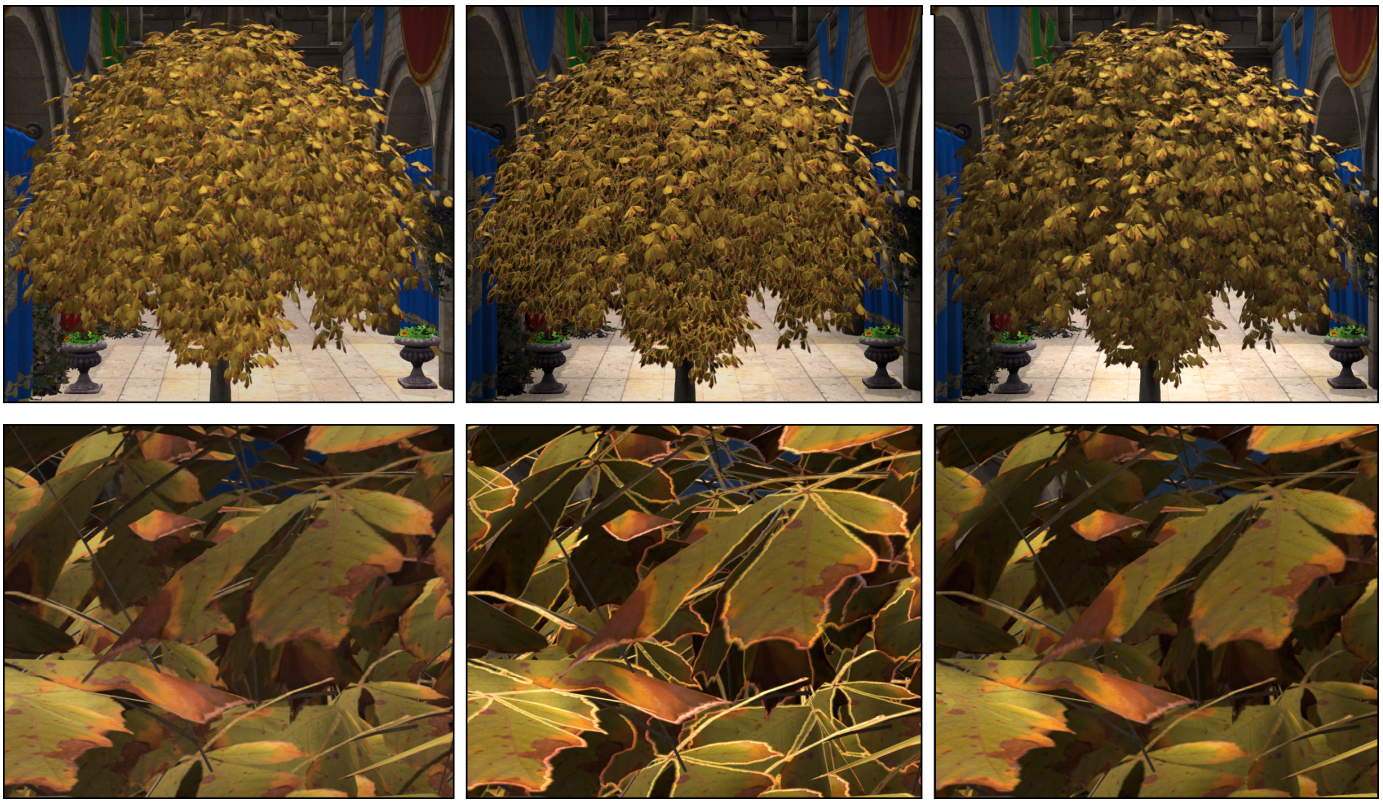


Fig. 6: *From left: Trees without ambient occlusion (AO) appear too flat, AO on opaque surfaces yields incorrect bright outlines, our new inferred AO creates consistent shading. The bottom row enlarges the center area to show detail.*

shading pass; naively running the AO computation at each pixel until convergence would be too slow. For transparent surfaces, the quality impact of not receiving AO is limited, just as it was for not casting AO. The albedo of a transparent surface is often ambiguous to the viewer and the overly bright rendering may go somewhat unnoticed.

However, a perceptible shading problem occurs on surfaces that have *both* opaque and transparent parts. Consider the foliage pictured in Figure 6. Ambient occlusion is important to the appearance of these tree leaves: the right column rendered with AO looks more realistic than the left column, which we rendered without it.

The interior of each tree leaf has $\alpha = 1$ and renders to the depth buffer. It both casts and receives AO. The edge of each leaf has a partial coverage gradient (due to MIP-mapping of a base binary-coverage mask) that fades out to $\alpha = 0$. This does not render to the depth buffer, so it cannot cast or receive AO under a typical screen-space AO algorithm. As shown in the center column of Figure 6, this means that the edges of tree leaves, which have partial coverage, will be too bright under any transparent rendering strategy. This error occurs under sorted transparency, k-buffers, and weighted-blended compositing because the problem is in the shading, not the compositing.

Our solution is to identify pixels with partial coverage and *infer* the AO value at them from adjacent pixels. For each material texture, we precompute a flag `baseMIPIsBinary` that is true for materials that have no transmission and only binary coverage at the lowest MIP level (highest resolution), and is false otherwise.

In the pixel shader, we compute the screen-space gradient of α using the GPU finite-difference intrinsics. If the gradient is nonnegligible, then we perform a cross-biliateral filter of AO along a gradient ray, using a hyperbolic weight in camera-space z value to diminish the contribution of AO from surfaces with very different depths. When the gradient is negligible, we instead apply this filter in a screen-space box. In practice, this occurs rarely, at isolated pixels adjacent to the $\alpha = 1$ region.

To efficiently a filter a large area, we introduce a sampling stride and also offset the samples in a checkerboard pattern for the box to conceal the sampling pattern. Listing 2 presents the full GLSL pixel shader code for this algorithm.

7 Results

7.1 Evaluation of Isolated Phenomena

Figure 7 shows a grid of two layers of colored transmissive glass bars rendered with three algorithms. Like real glass, these glossy reflect "white", have no diffuse reflection, and transmit the color that they appear. Previous recent OIT methods fail to model transmission, so they approximate colored glass with monochrome partial coverage (left). For this hard test case, our results (center) exactly match two-pass sorted transparency (right).

Figure 8 shows a scene with a row of teapots behind panes of glass. The panes have decreasing collimation from left to right. The top row was rendered with offline path tracing in previous

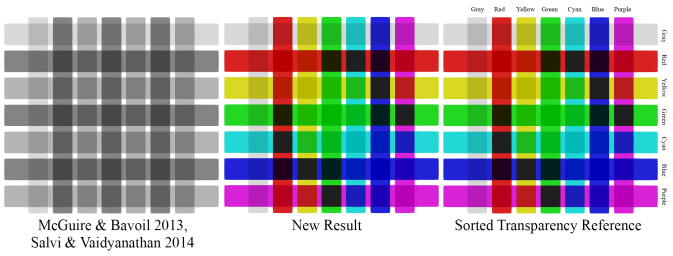


Fig. 7: *Colored transmission through two layers of glass bars.*

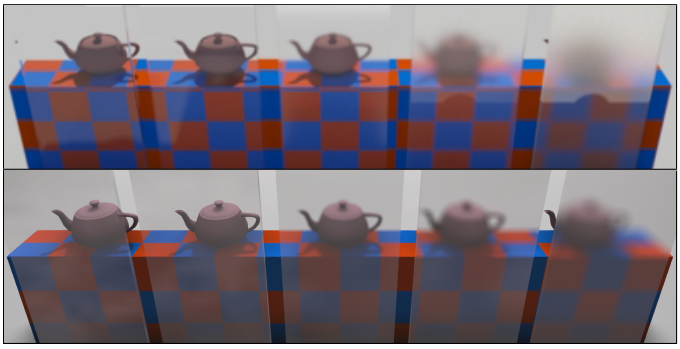


Fig. 8: *Top: Decreasing collimation, rendered by offline path tracing in iRay. Bottom: Our real-time result.* © *2015 NVIDIA*

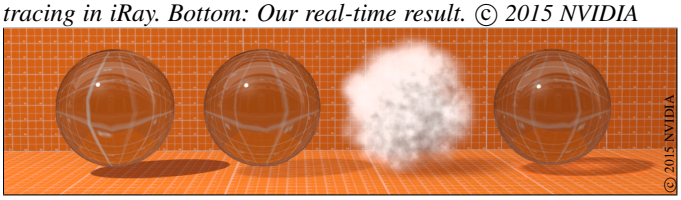


Fig. 9: *Left to right: Williams' opaque shadow map, Enderton et al.'s stochastic shadow map, and our phenomenological caustic stochastic shadow map applied to a cloud (no refraction, no caustic) and glass sphere (producing a refractive caustic).*

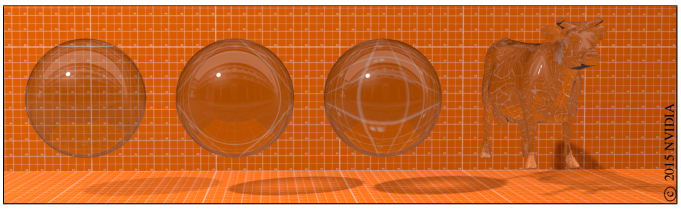


Fig. 10: *Caustic stochastic shadow map shadows for varying refractive indices. From left to right:* η *=1.0 (air), 1.3 (ice), 1.5 (glass), 1.5.*

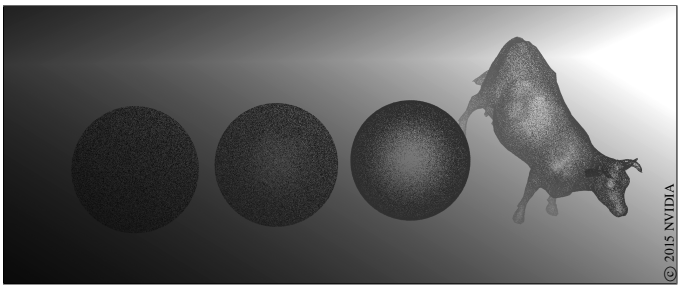


Fig. 11: *Visualization of a caustic stochastic shadow map.*

work by others[3]. We attempted to recreate their scene from that

3. From the iRay developer blog at `http://blog.irayrender.com/post/19731699592/frosted-glass-part-ii`

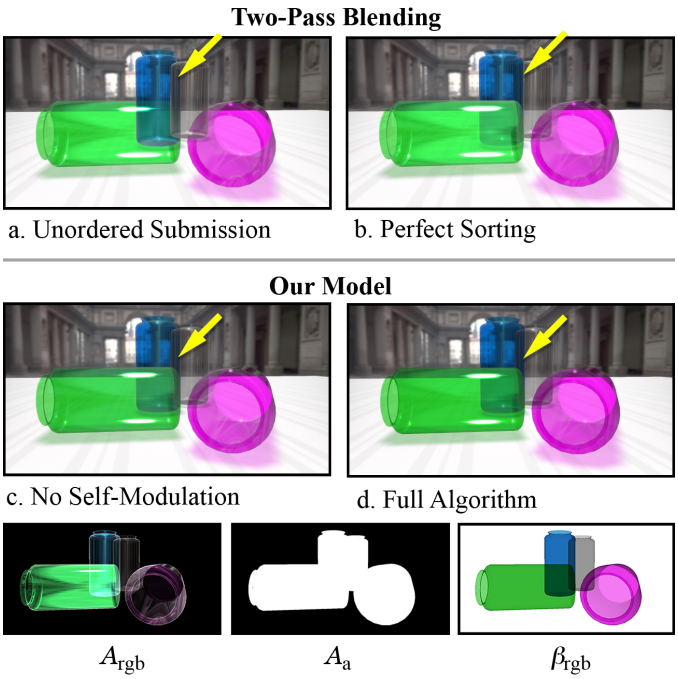


Fig. 12: *Four plastic bottles on a table in the Uffizi rendered by different methods (all using our shadowing). The overlaid yellow arrows denote a challenging area. a) Two-pass blending with unordered submission is obviously incorrect. b) Perfect sorting gives an ideal but result in* $2n$ *draw calls. c) Our method without self-modulation is efficient but has weak coloring in some areas. d) Self-modulation improves overlaps.* © *2015 NVIDIA*

image, and then rendered our real-time result on the bottom, which captures similar phenomena. Note also the decreased diffusion of the checkered box compared to the teapots in both rows, which occurs because the box is pressed against the glass.

Figure 9 compares shadows from three algorithms. The rightmost two objects are our stochastic VSM shadows with caustics. They match the expected phenomena for the cloud (translucency proportional to distance traveled) and the refractive sphere (caustic in the center). The correct darkening of the lower portion of the cloud itself is due to self-shadowing within that medium; the shading normals are identical on all particles.

Figure 10 shows how the caustic approximation varies as expected with η. We use a sphere for clarity and then gives an example of more complex geometry. Figure 11 shows a detail of the caustic stochastic shadow map for this scene before conversion to VSM, where its screen-door nature is apparent. The ground plane is a gradient instead of constant because spotlight is inclined. For another example, refer to the caustics from the glass tumblers and ice cubes in figure 1(center).

Figure 19 shows overlapping colored transmission of plastic (i.e., nonrefracting) bottles under four methods. The yellow arrows point to one challenging area. Subfigure (a) shows what two-pass sorted blending produces with *unordered* primitive submission; the performance is good and quality is unacceptable. Subfigure (b) shows that $2n$ draw calls for n objects with perfect sorting yield ideal quality, albeit at low performance. Subfigure (c) shows our algorithm with the self-modulation term disabled; some areas of colored overlap give an ambiguous appearance. Subfigure (d) shows our final result, in which β modulates A during the resolve

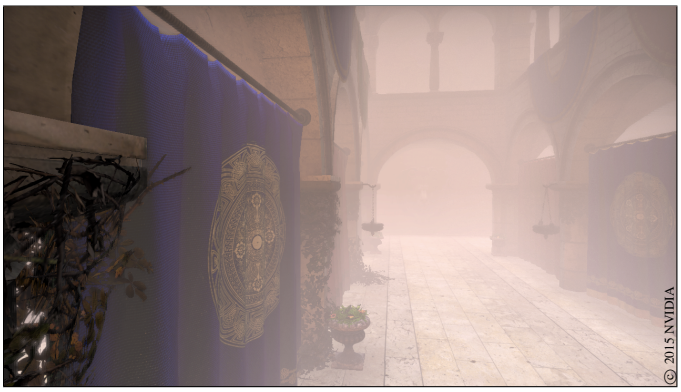


Fig. 13: *Fog with uniform ambient lighting rendered by compositing a single quad before the camera that itself models ray-marched OIT fog for the entire volume.*

Fig. 14: *Volumetric light shafts, backscattering, and shadows rendered by compositing a single quad before the camera that itself models ray-marched OIT fog for the entire volume.*

pass using equation 14. This result is qualitatively close to optimal (b) in terms of disambiguating overlaps, but slightly increases color saturation because surfaces must also modulate their own reflections. The bottom row shows the intermediate buffers.

7.2 Volumetric Effects

Our method naturally produces many volumetric lighting phenomena. Unbounded layers of participating medium can be accommodated via explicit particles or ray marching.

Figure 12 shows uniform fog in Sponza under ambient lighting. We rendered the fog as a single full-screen quad with a pixel shader that ray-marched from the camera to the opaque surfaces depth buffer. At each iteration, the shader applied in- and out-scattering via the Henyey-Greenstein phase function and accumulated the results using our weighted blending instead of ordered compositing. Of course, there is a good closed-form approximation for this case. We use the more general algorithm involving ray marching to demonstrate that the results appear as expected in this simple test, which would help address concerns of banding artifacts or bias. We now move on to apply the same method applied to more challenging illumination scenarios.

Because our method relies on strictly linear blending, all of the fog results are accumulated in pixel shader registers and then written once to the framebuffer. This gives the same result as rendering thousands of translucent planes in the scene, but requires only the blending bandwidth of a single surface. Because the algorithm is order-independent, we can compute the fog's contribution independently (and at lower resolution) than glass and partial coverage surfaces in the same scene. There is also no limitation that the fog be uniform–we chose that for this experiment solely to demonstrate the result in the simplest scenario. Any procedural or voxel-based density method can be applied.

Figure 13 (modeled after a figure by Hillaire [33]), shows the dramatic effects of light shafts, backscattering, and volumetric shadows that naturally arise when applying ray marching and our accumulation method in the presence of a participating medium.

Figure 14 shows several views of the more complex San Miguel scene in the presence of fog. Note that the volumetric transparency effects integrate seamlessly with the extensive partial-coverage filtering of foliage and the glass light bulbs, oil lamp flues, and wine glasses.

Figure 15 shows a particle-system ball of smoke over a ground plane, with a cube partly inside the smoke. The left image shows translucent stochastic-variance shadows cast on the ground. We disabled self-shadowing on the smoke itself, so it appears uniform in tone. When we re-enable self-shadowing on the right, the smoke correctly darkens away from the light source and gains visual weight.

Figure 16 shows a similar scene from the top. In this case, we elevated the cube. On the left the cube is shown floating away from the smoke, and on the right it is shown casting a volumetric shadow on and through the smoke due to the Williams shadow map. The smoke is simultaneously shadowing itself.

Figure 17 shows a more realistic scene, in which clouds of water vapor overlap each other and a skybox. They receive both direct and environment lighting, and are dark on the bottom from self-shadowing. The center cloud also receives a shadow from the left cloud, which partly occludes the sun.

7.3 Complex Scenes with Simultaneous Phenomena

The left and right subimages of figure 1 show diffusion in complex noir scenes. Figures 21a and 21b visualize the diffusion channel D for these. Both scenes exhibit diffusion increasing with depth, but for different reasons. In the first scene, objects become more diffused as the distance $|z_{X,i} - z_{B,i}|$ from the single frosted glass surface increases. The door frame and text have $D = 0$ because they are in front of the glass. The man, woman, and far wall are progressively distant behind the glass and therefore increasingly diffused.

In the second scene, objects become more diffused as the thickness of the medium increases because there are more surfaces i in the summation. The woman is not very diffused because very little fog is between her and the camera. The man deep in fog is more diffused and the buildings are blurred beyond recognition. As the lamp posts recede into the fog, they become more diffused.

Figure 18 shows a red and clear glass chess set in a living room. On the left is two-pass blending applied to objects. Because all chess pieces with the same material are a single mesh, there is no correct sorting order. Furthermore, shadows don't capture transparency and there is no refraction. Our result in the center fixes these

Fig. 15: *Volumetric lighting effects under single-plane fog in San Miguel interacting with glass (lamps and wineglasses) and partial coverage (leaves).*

problems and is rendered in a single draw call for all transparent surfaces. A visualization of the intermediate buffers is on the right.

Figure 20 shows a CAD model of an automobile engine rendered in two orientations by our method. The orientation is clear in each case due to weighted blending. This is a challenging case because the surfaces are in close proximity and all have partial coverage.

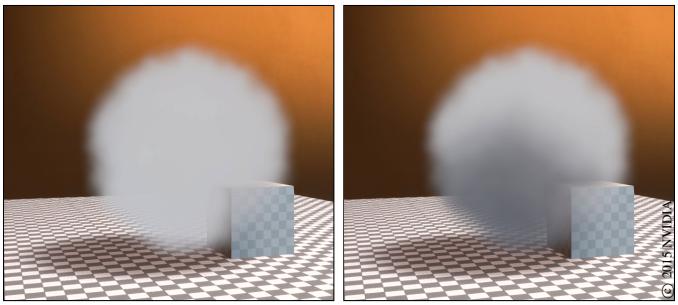


Fig. 16: *Particle system of 1000 particles without (left) and with (right) stochastic-variance shadow map volumetric self-shadowing. The self-shadowing creates smooth darkening away from the light with density.*

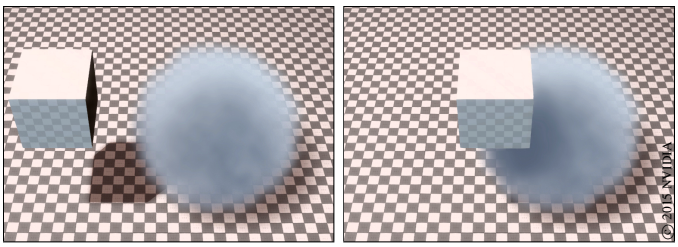


Fig. 17: *A cube casts a volumetric Williams shadow cast on a particle system (right). The particle system shadows itself and the ground using stochastic-variance shadow mapping.*

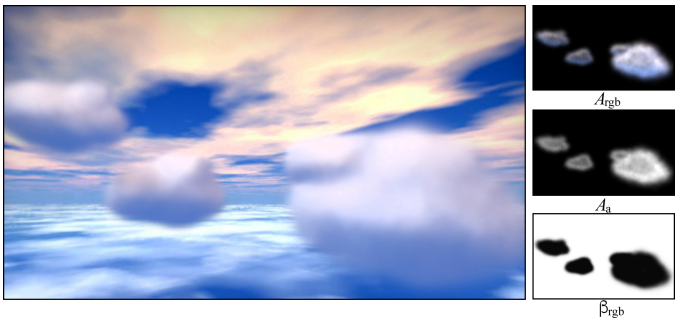


Fig. 18: *Particle system clouds against a skybox. Note the self-shadowing and the shadow cast on the center cloud by the left-most one above it.*

Figure 22 compares hair under three methods. We include this for completeness, but do not claim it is a strongly-motivating case for our method. A head of hair has a very narrow depth range, so all weights w_i are nearly uniform across it. Our model doesn't produce a result substantially better than unsorted blending. Because hair is a special case in which a uniform partial-coverage material quickly saturates to full opacity and in many cases covers a minority of the screen, we propose using a k-buffer or other bandwidth-intensive method (see Yuksel and Tariq's survey [34]) to render hair to a small off-screen texture and then compositing that as a single aggregate surface under our model. This would improve quality while still integrating with a general method.

Figure 23 shows two versions of a scene with overlapping transmissive glass and partial-coverage smoke. The left image uses a 4×4 subsampling for the smoke buffers. Because the transparent pass rendering time is dominated by the smoke, this gives nearly a $16\times$ bandwidth reduction and proportional speedup. Glass is at at full resolution in both images. While some fine detail is of course lost due to subsampling, note that opaque and transparent objects meet at clean edges due to the bilateral upsampling and that in the final images, areas where the glass and smoke overlap are nearly identical even though smoke was processed separately on the left.

We also implemented our algorithm for the Oculus VR SDK. Figure 24 shows the head-mounted display (HMD) image with precorrection for its optical distortion. The overlaid yellow arrows mark a red chess knight in the background seen directly by the left eye and observed through the foreground clear piece by the right. Evaluating recent transparency methods for HMDs is future work, however we report that the ability to "focus" one's vergence on either the foreground or the background knight felt surprisingly like true focal accomodation to us. Here and in the noir scenes, we had a much stronger sense of depth and presence than when we removed the fog or turned the glass objects opaque–those changes made the HMD's fixed focal plane more apparent and made depths more ambiguous.

Figure 25 shows the San Miguel scene, which has significant partial coverage at foliage silhouettes as well as glass transmission on windows, lamps, and wine glasses. Note that only the silhouettes of leaves and not the interiors appear in the OIT buffers–those surfaces render in the opaque rendering pass where $\alpha = 1$.

Table 1 gives the run time for each stage of our algorithm on an NVIDIA GeForce 980 GPU at 1920×1080 resolution. We render shadow maps at 4096^2 resolution and filter them down to 1024^2

Fig. 19: *A glass chess set rendered with two draw calls and naive blending vs. our method in a single draw call for all transmissive surfaces. The δ_{xy} term is visualized as RGB color $((\delta_x + 1)/2, (\delta_y + 1)/2, 0)$. Arrows point to some errors in the naive rendering. Our result corrects the ordering and captures the phenomena of colored shadows with caustics, refraction, and colored transmission.* © *2015 NVIDIA*

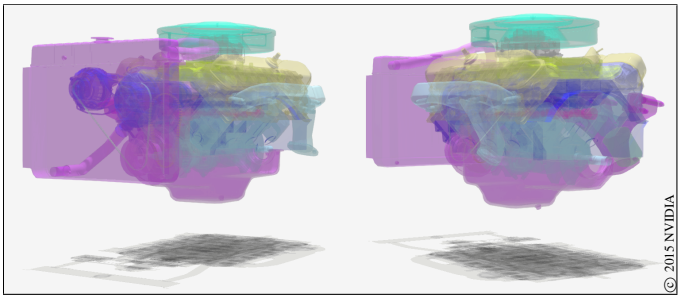
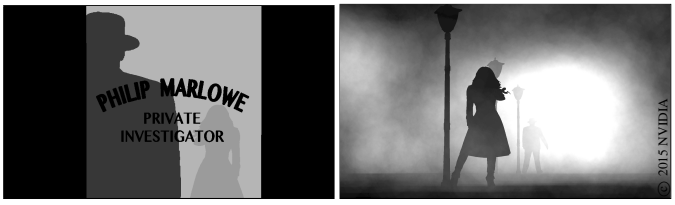


Fig. 20: *CAD-style rendering of a car engine with many closely-packed partial-coverage surfaces, in two orientations.*



(a) *One layer of glass* **(b)** *Dense fog*

Fig. 21: *Diffusion channel D for scenes from figure 1.*

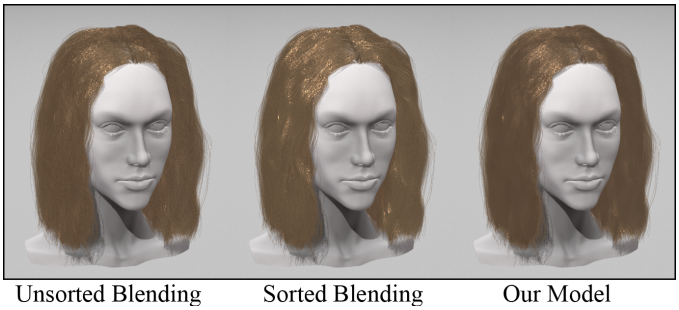


Fig. 22: *Heads with 50k $\alpha = 0.2$ hairs totaling 24M polygons each, rendered under three transparency strategies with stochastic shadow maps for self-shadowing.* © *2015 NVIDIA*

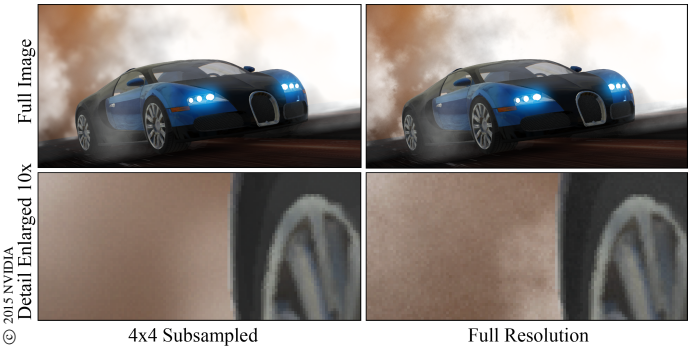


Fig. 23: *Overlapping glass and smoke at varying resolutions.*

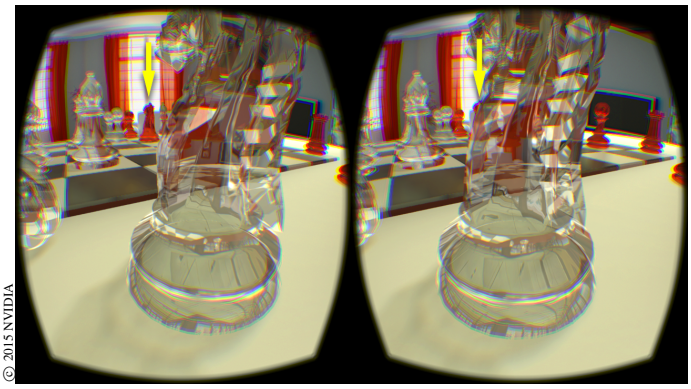


Fig. 24: *Chess scene rendered in a head-mounted VR display.*

TABLE 1: *Run time in milliseconds for each stage of our algorithm.*

Scene	Figure	1. Shadow	2. OIT	3. Resolve
Door	1 left	0.01 + 1.55	0.04	2.37
Glasses	1 center	0.73 + 4.65	1.53	0.09
Foggy Night	1 right	0.37 + 1.55	4.32	4.02
w/ 4×4 sub.	-	0.37 + 1.55	0.37	5.01
Chess	18	0.24 + 4.65	6.21	0.09
Engine	20	0.48 + 1.55	10.0	0.07
Hair (24 Mtris)	22	28.13 + 1.55	19.21	0.06
Car	23 right	0.68 + 1.55	2.91	0.11
w/ 4×4 sub.	23 left	0.68 + 1.55	0.25	1.10
San Miguel	25	5.07 + 1.55	0.54	0.09

VSMs. The first number in the Shadow column is the geometry-dependent depth map generation time. The second number is VSM creation time, which depends only on the number of lights and wavelengths.

The Order-Independent Transparency (OIT) stage 2 is dominated by the cost of shading the surfaces themselves–the weighting and refraction calculations in listing 1 add only about 25 arithmetic operations and one texture fetch. Recall that the Hair scene has 24M sub-pixel triangles and thus a high render time.

The Resolve stage 3 is extremely fast when there is no upsampling

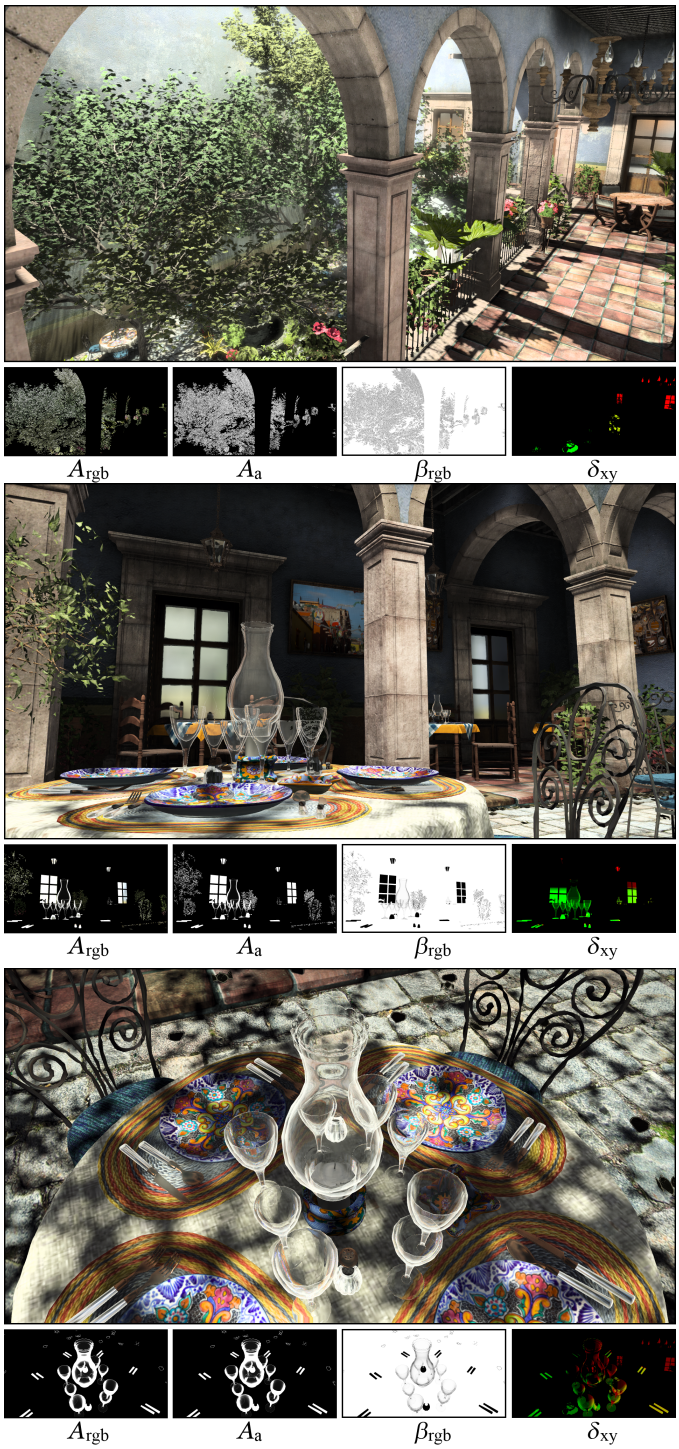


Fig. 25: *Several views in San Miguel and intermediate buffers. Note overlapping partial coverage (leaves) and transmissive and refractive (glass) surfaces.* © *2015 NVIDIA*

or diffusion. Subsampling smoke and fog produces better than an order of magnitude throughput increase at the OIT stage, with the cost of a fixed ≈ 1 ms overhead for upsampling in the Resolve stage. Diffusion running time is linear in the number of non-zero elements of the point-spread kernels ($\propto D^2$).

8 Discussion

8.1 Impact in the Context of GPU Architecture

We described a method that extends the previous weighted-blended transparency methods with several new effects and demonstrated their visual impact and performance. Moreover, this research targets not just high performance today, but scaling with longer-term trends in hardware architecture.

Our method requires only fixed-function blending during framebuffer writes. Thus, it avoids the pixel-synchronization overhead of programmable blending on new GPUs. This is important because synchronization points *limit* parallelism, and scaling to upcoming 4k TVs and head-mounted displays will require *more* parallelism than even today's best GPUs. In the short term, avoiding synchronization also allows implementation on widely deployed DX11/GL4-class GPUs, including consoles.

Memory bandwidth is a limiting constraint on high-performance rendering [35]. That is because off-chip DRAM and its physical interface have changed relatively little over the past 50 years. Hence, bandwidth increased slowly compared to the Moore's Law pace of arithmetic operations. In fact, the open secret of early GPUs was that their advantages over CPUs were large texture caches and wide memory interfaces, and not flop/s.

NVIDIA's new 2016 Pascal GPU architecture stacks DRAM on top of the processor for a one-time higher-bandwidth interface [36]. However, because every surface on the package is now occupied, we will never see another comparable increase under DRAM with metal interconnects. Thus, while we predict a brief upset for a few months, in the long term, increased performance will be achieved only by decreasing memory traffic to take advantage of the steady arithmetic scaling offered by massive parallelism.

Our full algorithm requires only 112 bits/pixel of framebuffer write bandwidth and storage during the key Transparency stage 2. For α-only scenes, this drops to 88 bits/pixel. In comparison, Salvi and Viadyanathan's method [3] with a $(k = 4)$-buffer for α requires 256 bits/pixel and Vasilakis and Fudos's method [2] requires between 128 and 320 bits/pixel in the worst case. Thus, our approach has strictly higher throughput for this stage by up to $3.6\times$ for α-only, and our full model achieves a 2-3$\times$ increase vs. previous α-only methods.

For casting transparent shadows on all scene elements, McGuire and Enderton's [1] method requires around 176 texture fetches for the 15⊠15 tent filter that eliminates stochastic noise. By combining their method with VSM [14], we reduced that to a single fetch per shaded fragment. Combined with the geometry-independent prefiltering process, this is about a $15\times$ net bandwidth reduction for the scenes in our experiments.

8.2 Limitations and Future Work

Despite their run-time costs and limitations, we find the quality of special-purpose subsurface scattering (e.g., [37]) and k-buffer hair methods compelling. We desire a unified method that combines all of the effects and performance of our method with these.

We've shown how to make screen-space ambient occlusion work with partial coverage, and how to integrate screen-space volumetric ray marching with transparency. A good future step is exploration of the interaction between transparency and other screen-space effects such as reprojection antialiasing, reflection, motion blur, and depth of field.

A rasterization renderer with our method and various screen-space effects is about an order of magnitude faster for transparency than a comparable ray tracer. This performance is too significant to forego today, but at some point ray tracing will likely be preferred to reduce the cost of authoring software and content around the approximation errors. The natural questions for further research are what new algorithms and architectures would enable this inflection point, and since there is unlikely to be an instantaneous change, how they can smooth the transition from an all-rasterization to a hybrid-ray tracing pipeline.

REFERENCES

[1] M. McGuire and E. Enderton, "Colored stochastic shadow maps," in *Proc. of I3D*. ACM, February 2011, pp. 89–96. [Online]. Available: http://research.nvidia.com/publication/colored-stochastic-shadow-maps

[2] A. A. Vasilakis and I. Fudos, "K+-buffer: Fragment synchronized k-buffer," in *Proc. of I3D*. ACM, 2014, pp. 143–150.

[3] M. Salvi and K. Vaidyanathan, "Multi-layer alpha blending," in *Proc. of I3D*. ACM, 2014, pp. 151–158. [Online]. Available: http://doi.acm.org/10.1145/2556700.2556705

[4] M. McGuire and M. Mara, "A phenomenological scattering model for order-independent transparency," in *I3D 2016*, February 2016, p. 10. [Online]. Available: http://graphics.cs.williams.edu/papers/TransparencyI3D16

[5] L. Carpenter, "The A-buffer, an antialiased hidden surface method," in *Proc. of SIGGRAPH*. ACM, 1984, pp. 103–108. [Online]. Available: http://doi.acm.org/10.1145/964965.808585

[6] N. P. Jouppi and C.-F. Chang, "Z^3: an economical hardware technique for high-quality antialiasing and transparency," in *Proc. of Graphics Hardware*. ACM, 1999, pp. 85–93. [Online]. Available: http://doi.acm.org/10.1145/311534.311582

[7] L. Bavoil, S. P. Callahan, A. Lefohn, J. L. D. Comba, and C. T. Silva, "Multi-fragment effects on the GPU using the k-buffer," in *Proc. of I3D*. ACM, 2007, pp. 97–104.

[8] M. Maule, J. Comba, R. Torchelsen, and R. Bastos, "Hybrid transparency," in *Proc. of I3D*. ACM, 2013, pp. 103–118. [Online]. Available: http://doi.acm.org/10.1145/2448196.2448212

[9] M. Salvi, J. Montgomery, and A. Lefohn, "Adaptive transparency," in *Proc. of HPG*. ACM, 2011, pp. 119–126. [Online]. Available: http://doi.acm.org/10.1145/2018323.2018342

[10] T. Lokovic and E. Veach, "Deep shadow maps," in *Proc. of SIGGRAPH*. ACM, 2000, pp. 385–392. [Online]. Available: http://dx.doi.org/10.1145/344779.344958

[11] E. Sintorn and U. Assarsson, "Hair self shadowing and transparency depth ordering using occupancy maps," in *Proc. of I3D*. ACM, 2009, pp. 67–74. [Online]. Available: http://doi.acm.org/10.1145/1507149.1507160

[12] J. Jansen and L. Bavoil, "Fourier opacity mapping," in *Proc. of I3D*. ACM, 2010, pp. 165–172. [Online]. Available: http://doi.acm.org/10.1145/1730804.1730831

[13] E. Enderton, E. Sintorn, P. Shirley, and D. Luebke, "Stochastic transparency," in *Proc. of I3D*. ACM, 2010, pp. 157–164. [Online]. Available: http://doi.acm.org/10.1145/1730804.1730830

[14] W. Donnelly and A. Lauritzen, "Variance shadow maps," in *Proc. of I3D*. ACM, 2006, pp. 161–165. [Online]. Available: http://doi.acm.org/10.1145/1111411.1111440

[15] H. Meshkin, "Sort-independent alpha blending," March 2007, perpetual Entertainment, GDC Session. [Online]. Available: http://twvideo01.ubm-us.net/o1/vault/gdc07/slides/S3721i1.pdf

[16] L. Bavoil and K. Myers, "Order independent transparency with dual depth peeling," NVIDIA, Tech. Rep., 2008.

[17] K. Kluczek, "Efficient rendering of intersecting soft particles," September 2012, talk at WGK, Poland.

[18] M. McGuire and L. Bavoil, "Weighted blended order-independent transparency," *JCGT*, vol. 2, no. 2, pp. 122–141, December 2013. [Online]. Available: http://jcgt.org/published/0002/02/09/

[19] T. Nishita, "Light scattering models for the realistic rendering of natural scenes," in *Proc. of E.G.* Eurographics, 1998, pp. 1–10.

[20] S. G. Narasimhan and S. K. Nayar, "Shedding light on the weather," in *Proc. of CVPR*. IEEE, 2003, pp. 665–672. [Online]. Available: http://dl.acm.org/citation.cfm?id=1965841.1965928

[21] S. G. Narasimhan, R. Ramamoorthi, and S. K. Nayar, "Analytic rendering of multiple scattering in participating media," Columbia University, Tech. Rep., 2004.

[22] S. Premože, M. Ashikhmin, J. Tessendorf, R. Ramamoorthi, and S. Nayar, "Practical rendering of multiple scattering effects in participating media," in *Proc. of EGSR*. Eurographics, 2004, pp. 363–374. [Online]. Available: http://dx.doi.org/10.2312/EGWR/EGSR04/363-374

[23] C. de Rousiers, A. Bousseau, K. Subr, N. Holzschuch, and R. Ramamoorthi, "Real-time rendering of rough refraction," *IEEE Trans. on Vis. and Comp. Graph.*, Feb 2012. [Online]. Available: http://graphics.berkeley.edu/papers/Rousiers-RTR-2012-02/

[24] M. McGuire and M. Mara, "Efficient GPU screen-space ray tracing," *JCGT*, vol. 3, no. 4, pp. 73–85, December 2014. [Online]. Available: http://jcgt.org/published/0003/04/04/

[25] P. Ganestam and M. Doggett, "Real-time multiply recursive reflections and refractions using hybrid rendering," *Vis. Comput.*, vol. 31, no. 10, pp. 1395–1403, Oct. 2015. [Online]. Available: http://dx.doi.org/10.1007/s00371-014-1021-7

[26] C. Yuksel and J. Keyser, "Fast real-time caustics from height fields," *Vis. Comput.*, vol. 25, no. 5-7, pp. 559–564, Apr. 2009. [Online]. Available: http://dx.doi.org/10.1007/s00371-009-0350-4

[27] M. A. Shah, J. Kontinnen, and S. N. Pattanaik, "Caustics mapping: An image-space technique for real-time caustics." *IEEE Trans. Vis. Comput. Graph.*, vol. 13, no. 2, pp. 272–280, 2007.

[28] C. Wyman, "An approximate image-space approach for interactive refraction," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 1050–1053, 2005. [Online]. Available: http://doi.acm.org/10.1145/1073204.1073310

[29] C. Wyman and G. Nichols, "Adaptive caustic maps using deferred shading," *Computer Graphics Forum*, 2009.

[30] A. Glassner, "Interpreting alpha," *JCGT*, vol. 4, no. 2, pp. 30–44, May 2015. [Online]. Available: http://jcgt.org/published/0004/02/03/

[31] N. Tatarchuk, C. Tchou, and J. Venzon, "Mythic science fiction in real-time: Destiny rendering engine," July 2013, sIGGRAPH Advances in Real-Time Rendering in Games Course.

[32] M. Mara, M. McGuire, D. Nowrouzezahrai, and D. Luebke, "Deep g-buffers for stable global illumination approximation," in *HPG*, June 2016. [Online]. Available: http://graphics.cs.williams.edu/papers/DeepGBuffer16

[33] S. Hillaire, "Towards unified and physically-based volumetric lighting in frostbite," 2015, in SIGGRAPH Advances in Real-Time Rendering Course.

[34] C. Yuksel and S. Tariq, "Advanced techniques in real-time hair rendering and simulation," in *SIGGRAPH Courses*. ACM, 2010, pp. 1–168. [Online]. Available: http://doi.acm.org/10.1145/1837101.1837102

[35] T. Whitted, "Disaggregated graphics: Rich clients for clouds," June 2010, keynote at HPG.

[36] J.-H. Huang, "Keynote," March 2015, GPU Technology Conference, San Jose, CA.

[37] J. Jimenez, D. Whelan, V. Sundstedt, and D. Gutierrez, "Real-time realistic skin translucency," *CG&A*, vol. 30, no. 4, pp. 32–41, 2010.

APPENDIX: IMPLEMENTATION DETAILS

A complete C++ and OpenGL implementation of the techniques described in this paper is available in the open source G3D Innovation Engine 10.01 at `http://g3d.cs.williams.edu`.

```
void computeOutput(vec3 L_r, float alpha, vec3 t,
  float c, float eta, vec3 X, vec3 n, out vec4 A,
  out vec3 beta, out float D, out vec2 delta) {

  float netCoverage =
    alpha * (1.0 - dot(t, vec3(1.0/3.0)));
  float tmp = (1.0 - gl_FragCoord.z * 0.99) *
    netCoverage * 10.0;
  float w = clamp(tmp * tmp * tmp, 0.01, 30.0);

  float z_B = depthToZ(texelFetch(depthBuffer,
    ivec2(gl_FragCoord.xy), 0).r, clipInfo);

  vec2 refractPix = refractOffset(n, X, eta);
  const float k_0 = 120.0 / 63.0, k_1 = 0.05;

  A = vec4(L_r * alpha, netCoverage) * w;
  beta = alpha * (vec3(1.0) - t) * (1.0 / 3.0);
  D = k_0 * netCoverage * (1.0 - c) *
    (1.0 - k_1 / (k_1 + X.z - z_B)) / abs(X.z);
  delta = refractPix * netCoverage * (1.0 / 63.0);

  D *= D; // Store D^2, variance, during summation
  if (D > 0.0) D = max(D, 1.0 / 256.0);
}
```

Listing 1: *Pixel shader outputs for Transparency stage 2.*

8.2.0.1 Render Targets: Configure four OpenGL framebuffer render target (RT) attachments as specified in table 2 for the OIT stage. All targets use the additive blend *equation* `glBlendEq(GL_ADD)`. For the blending *function*, RT1 uses `glBlendFuncSeparatei(1, GL_ZERO, GL_ONE_MINUS_SRC_COLOR, GL_ONE, GL_ONE)` and RT0 and RT2 use `glBlendFunc(GL_ONE, GL_ONE)`.

On platforms without separate alpha blending, move D to the third channel of RT2. Doing so intuitively packs all multiple scattering terms into RT2, but costs another 16 bits because a modern GPU will align each 24-bit texture to 32 bits per pixel. For platforms without SNORM formats, scale and bias RT2.

Particle systems often contain many surfaces that each have low coverage (α), which raises the issue of underflow in the Transparency stage. We round non-zero values less than $1/255$ up to $1/255$ in the D channel, which creates slight over-diffusion but avoids underflow. We find that underflow in β is often imperceptible.

TABLE 2: *Framebuffer layout. (1/63) factors give roughly $\frac{1}{2}$-pixel precision on refraction and diffusion.*

	Encoding	Format	Bits/Pix	Clear Value
RT0	(A_r, A_g, A_b, A_a)	RGBA16F	64	(0, 0, 0, 0)
RT1	$(\beta_r, \beta_g, \beta_b, D^2/63^2)$	RGBA8	32	(1, 1, 1, 0)
RT2	$(\delta_x, \delta_y)/63$	RG8_SNORM	16	(0, 0)

```
// Sample AO at a pixel offset from the current one.
// Return bilaterally filtered AO and weight
vec2 sampleAO(vec2 offset, float z1) {
   ivec2 C = clamp(ivec2(gl_FragCoord.xy + offset),
       ivec2(0), AOTexSize - 1));
   float d = fetch(depthBuffer, C, 0).r;
   float z2 = reconstructCSZ(d, projectionConstants);
   float AO = fetch(AOTexture, C).r;
   // Hyperbolic weighting
   float weight = 0.01 / (0.01 + abs(z1 - z2));
   return vec2(AO * weight, weight);
}
...
if ((alpha > 0) && (alpha < 1) && baseMIPIsBinary) {
  vec2 gradient = vec2(dFdx(alpha), dFdy(alpha));
  float L = length(gradient);
  vec2 sum = vec2(0);
  if (L < 1e-4) { // sample a square
     const int R = 3; const float stride = 2;
     for (int dy = -R; dy <= R; ++dy) {
       float dxShift = ((abs(dy) & 1) - 0.5);
       for (int dx = -R; dx <= R; ++dx)
          sum += sampleAO(vec2(dx + dxShift, dy) *
              stride, csZ);
     }
   } else { // sample a line
     vec2 dir = gradient / L;
     const int R = 12, stride = 3;
     for (float t = 0; t < R; ++t)
        sum += sampleAO(dir * (t * stride), csZ);
   }
   AO = clamp(sum.x / max(0.01, sum.y), 0, 1);
} else {
   // No occlusion for other transparents
   AO = 1;
}
```

Listing 2: *Ambient occlusion inference pixel shader.*

8.2.0.2 Pixel Shader: In our initial experiments, we used the standard G3D surface pixel shaders for the Transparent stage, with the output changed to write to the A, β, D, δ buffers instead of a single color framebuffer using the code from listing 1. Position `X` and normal `n` are in camera space. Variable names match section 4. Listing 3 is the GLSL implementation of the resolve stage.

```
#version 410
// Typical GLSL preprocessor and math helpers from
// http://g3d.cs.williams.edu
#include <compatibility.glsl>
#include <g3dmath.glsl>

uniform sampler2D ATex, BDTex, deltaTex;
uniform sampler2D bkgTexture;

out Color3 result;

// Pixels per unit diffusion std dev
const float PPD = 200.0;
const int maxDiffusionPixels = 16;

#define fetch(a, b) texelFetch(a, b, 0)

void main() {
  vec2 bkgSize = textureSize(bkgTexture, 0).xy;
  int2  C = int2(gl_FragCoord.xy);
  vec4  BD = texelFetch(BDTex, C, 0);
  vec3  B = BD.rgb;
  if (minComponent(B) == 1.0) {
      // No transparency
      result = fetch(bkgTexture, C).rgb;
      return;
  }
```

```
  float D2 = BD.a * square(PPD);
  vec2 delta = fetch(deltaTex, C).xy * 0.375;

  vec4 A = fetch(ATex, C);
  // Suppress under- and over-flow
  if (isinf(A.a)) A.a = maxComponent(A.rgb);

  if (isinf(maxComponent(A.rgb)))
      A = vec4(isinf(A.a) ? 1.0 : A.a);

  // Self-modulation
  A.rgb *= vec3(0.5) + max(B, vec3(1e-3)) /
      max(2e-3, 2 * maxComponent(B));

  // Refraction
  vec3 bkg = vec3(0);

  if (D2 > 0) {             // Diffusion

    C += int2(delta * bkgSize);
    // Tap spacing
    const float stride = 2;
    // Kernel radius
    int R = int(min(sqrt(D2), maxDiffusionPixels) /
        float(stride)) * stride;

    float weightSum = 0;
    for (vec2 q = vec2(-R); q.x <= R; q.x+=stride) {
     for (q.y = -R; q.y <= R; q.y+=stride) {
      float radius2 = dot(q, q);

      if (radius2 <= D2) {
       int2 tap = C + ivec2(q);
       float t = fetch(BDTex, tap).a;
       float bkgRadius2 = t * PPD * PPD;

       if (radius2 <= bkgRadius2) {
         // Disk filter (faster, looks similar)
         float w = 1.0 / bkgRadius2 + 1e-5;

         // True Gaussian filter
         //float w=exp(-radius2 / (8*bkgRadius2)) /
         // sqrt(4 * PI * t);

        bkg += w * fetch(bkgTexture, tap).rgb;
        weightSum += w;
    }}}}
    bkg /= weightSum;
  } else {
    // No diffusion (+ fractional refraction)
    bkg = textureLod(bkgTexture,
      delta + gl_FragCoord.xy / bkgSize, 0).rgb;
  }

  result = bkg * B + (vec3(1) - B) * A.rgb /
     max(A.a, 0.00001);
}
```

Listing 3: *Pixel shader for Resolve stage 3 with common, constant-time implementation optimizations.*

Acknowledgements

We thank David Luebke, Dan Evangelakos, and our colleagues in the architecture group at NVIDIA for their assistance; anonymous reviewers for their suggestions; Cem Yuksel for the hair model; and Guillermo Llaguno for San Miguel. Other scene elements were purchased from TurboSquid.

Morgan McGuire Dr. McGuire is a professor of Computer Science at Williams College and researcher at NVIDIA. He is the author or coauthor of *The Graphics Codex*, *Computer Graphics: Principles and Practice* 3rd edition, and *Creating Games*. He received his degrees at Brown University and M.I.T.

Michael Mara Michael Mara is a graduate student in the Stanford University Computer Science department and researcher at NVIDIA.