Hardware-Accelerated Colored Stochastic Shadow Maps

Morgan McGuire*
NVIDIA Research and Williams College

Abstract

This paper extends the stochastic transparency algorithm that models partial coverage to also model wavelength-varying transmission. It then applies this to the problem of casting shadows between any combination of opaque, colored transmissive, and partially covered (i.e., α-matted) surfaces in a manner compatible with existing hardware shadow mapping techniques. Colored Stochastic Shadow Maps have a similar resolution and performance profile to traditional shadow maps, however they require a wider filter in colored areas to reduce hue variation.

1 Introduction

Translucent materials are visually appealing. Yet designers of interactive programs tend to avoid them because of a lack of rendering algorithms compatible with translucency. This paper presents an efficient and practical method for rendering correct shadows in the presence of colored translucency. It is motivated by a desire to solve the problem of translucent shadowing in a general way that fits the architecture and performance constraints of typical real-time systems like games.

This paper introduces the novel *Colored Stochastic Shadow Map* (CSSM) data structure, which is named both for the fact that it produces the phenomena of colored shadows and for its appearance when visualized (figure 1). It packs into as few as 32 bits per texel, renders at about the same rate as a traditional shadow map, and can accurately simulate shadows between any combination of colored[1] opaque, non-refractive transmissive, and partial coverage (i.e., α-matted) surfaces, including single-scattering particle systems. CSSM requires only one order-independent pass over geometry to generate and has no limit on the number of overlapping translucent layers. It has the nice theoretical property that the primary artifact, stochastic color noise, can be driven arbitrarily low by increasing the resolution, filter radius, and filter shape—practices already in use to mitigate aliasing in traditional shadow maps.

As is often the case in real-time rendering, the competing constraints of space, time, bias, variance, artistic control and generality make it impossible to declare one technique strictly better than another. CSSM is good for cases where multiple colored translucent surfaces may be present in a scene and for which realistic and consistent results are considered important. However, shadowing through translucent surfaces is a complex phenomenon and rendering it correctly is not necessary for all applications. This argument holds for even opaque shadowing—the dark blob under an object is the significant perceptual cue, not the precise shape or shade. In the extreme, simple colored-disk drop shadows may be the best choice for some applications. In others, limiting translucency to a single surface may be appropriate. For example, StarCraft II is a real-time strategy game currently in development by Blizzard. In this game, the camera is nearly orthographic and is always above the action, which is primarily lit by a nearly-white directional light representing the sun. Overlapping translucent shadows are infrequent in this kind of scene, so the developers invented an appropriate shadowing method limited to the first translucent surface from the light [Filion and McNaughton 2009]. CSSM could provide shadowing at about the same performance as that method, but it is not needed here.

In contrast to StarCraft II, consider Marvel Ultimate Alliance II (MUA2), which was developed by Vicarious Visions and published by Activision in 2009. MUA2 contains translucent characters like

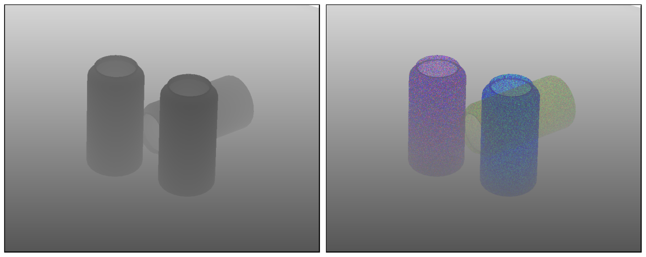
Figure 1: ***Left)*** *Williams 1978 shadow map and* ***right)*** *new Colored Stochastic Shadow Map for the scene shown in figure 4*

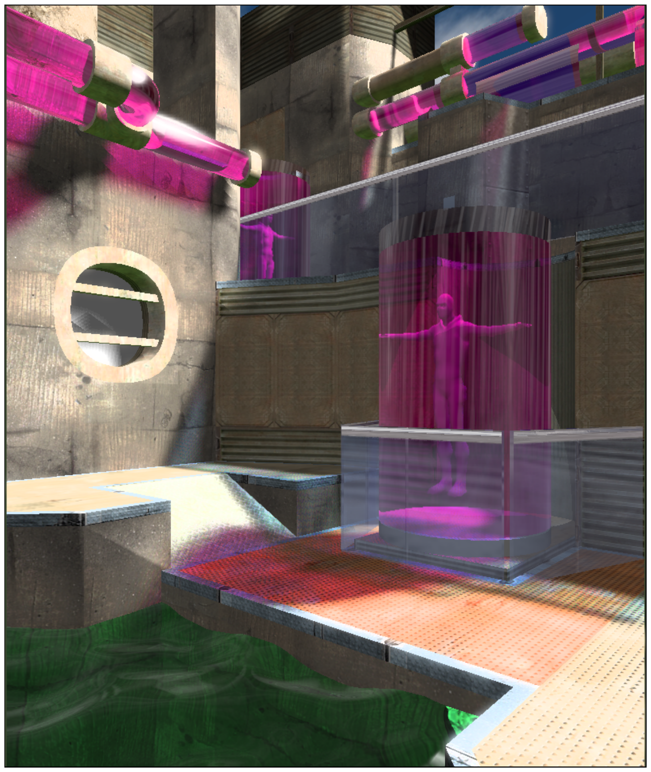
Figure 2: *Game scene with 1M triangles rendered at 1920×1080, 60 Hz with a 2048^2 shadow map on a GeForce GT 280 GPU. CSSM adds 0.1 to 0.9 ms to the opaque shadow render time for this scene, depending on view.*

*e-mail: morgan@cs.williams.edu

NVIDIA Technical Report NVR-2010-003, May 5, 2010.

[1] "Color" is technically a perception, not a physical property. We follow the common substitution of "colored" for "wavelength-dependent."

Iceman and the Invisible Woman, water, colored smoke particle effects, mad science labs with colored glass tanks and fluids, and battles set in science-fiction techno-castles and crystal caves. Primary and shadow translucency were therefore a major challenge in this game. They could never be resolved under previous algorithms to match the artistic vision, so ultimately the developers fell back to scene-specific workarounds and design changes to deliver the game's compelling visuals [Murphy 2009]. This is the kind of application that requires a general method like CSSM for correct simulation of multiple shadow casters.

Beyond games, film micro-polygon renderers like RenderMan often employ shadow maps and already rely on filtering to reduce sampling noise. So that is another domain for which CSSM is appropriate.

This paper contributes:

1. Analysis of the different translucency phenomena and their relation to shadows and blending (section 2)
2. Unification of the contributions of partial coverage and transmission models into a probabilistic colored translucency model (section 4.1)
3. The CSSM1 algorithm, which directly samples the shadowing of colored translucent surfaces for arbitrary wavelengths (section 4.2)
4. An optimized CSSM2 algorithm for RGB rendering, which has comparable time and space performance to traditional opaque shadow maps (section 4.3)

2 Translucency Phenomena

Multiple distinct light transport phenomena can produce the common perceptual phenomenon of "translucency." All result in multiple objects along a ray contributing to the radiant flux through a pixel. Real-time approximations of these phenomena are typically built on blending modes in the raster operation stage of the graphics pipeline, which are selected by `glBlendFunc` and `glBlendEq` in the OpenGL API. That commonality leads to a frequent source of error, in that many renderers conflate phenomena with different underlying causes and attempt to use one blending mode to simulate all of them. That source of error has in turn made it challenging to implement correct translucency and translucent shadowing in such renderers.

The following paragraphs describe five distinct phenomena and efficient methods for coarsely approximating them along *eye rays* in OpenGL. This clarifies the scope and terminology of the remainder of the paper, which is concerned with applying these ideas to the related problem of approximating translucent phenomena along *shadow rays*.

Transmission (e.g., by glass) occurs when light is modulated by the *transmission* spectrum of a material that it intersects. For example, this causes the back of a white label on a green wine bottle to appear green when viewed through the bottle. For a transmissive object with uniform material properties, the fraction of light at wavelength λ transmitted through distance d of material is given by $\exp(-4\pi d\kappa_\lambda/\lambda)$, where *extinction coefficient* κ_λ is the imaginary part of its complex index of refraction [Hecht 2002, 128]. The transmission is *non-refractive* if the exitant ray has the same direction as the incident ray, which occurs when the real part, η, of the index of refraction is the same for both the intersected and surrounding media.

One method for approximating non-refractive transmission for scenes with strict depth ordering is as follows. Render surfaces from farthest to nearest. At each, first modulate the previously sampled radiance at each pixel (e.g., using `glBlendFunc(GL_ZERO, GL_SRC_COLOR)`) by the transmission spectrum of the surface, which is all zero for opaque surfaces. Second, add the reflected and emitted radiance from the surface (`glBlendFunc(GL_ONE, GL_ONE)`).

A thin surface has fixed thickness d (at normal incidence), so it is common practice to precompute the net transmission through that thickness at several wavelengths, which we call $\vec{t}$, e.g., with named components $\vec{t} = (\vec{t}_r, \vec{t}_g, \vec{t}_b)$. This is the "source color" for the OpenGL command. For thick transmitters, more sophisticated methods have been developed for efficiently sampling the background color from an offset location to approximate refraction (e.g., [Wyman 2005]), and for computing the varying transmission levels (e.g., [Bavoil et al. 2007]). Note that in the real world, physics constrains all transmissive surfaces to also be specularly reflective to some extent. Transmission always falls off with the angle of incidence according to the Fresnel equations.

Partial coverage (e.g., by a window screen) occurs when a subset of the rays within one pixel's bundle of samples are occluded by a perforated foreground surface or particle set. The fraction of rays that are occluded is denoted by α. Note that at the highest resolution of a model (i.e., level 0 MIP-map) α is ideally either 1 or 0 at every sample. Fractional α arises from taking multiple binary samples per pixel. This is the case for higher MIP levels, `GL_ALPHA_TO_COVERAGE`, and `GL_POLYGON_SMOOTH` rendering.

The observed spectrum of multiple *uncorrelated* partial coverage layers is given by repeated application of Porter and Duff's [Porter and Duff 1984] linear *over* operator: $\alpha F + (1-\alpha)B$. In this equation, F and B are the radiance that would be transported to the viewer from a foreground layer and a background layer in isolation. One method for approximating partial coverage is rendering objects from farthest to nearest with linear radiance interpolation (e.g., using `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)`). Note that a surface can be both transmissive and partially covering. In that case, the observed foreground spectrum contains a term that is a modulated version of the background spectrum.

Emission by a translucent surface occurs when a partial or transmissive surface or medium also emits light. Phosphorescent algae clouds, neon bulbs, and lightning are real-world cases. Science fiction force fields and fantasy magical effects are imaginary ones. One method for simulating this is simple accumulation of radiance at a pixel (e.g., by `glBlendFunc(GL_ONE, GL_ONE)`.)

Bloom and lens flare occur when dispersion and internal reflections within a lens objective cause bright scene points to affect pixels other than those dictated by pinhole projection. Direct simulation of a compound lens as in-scene surfaces tends to be inefficient, so these effects are commonly approximated by post-processing with additive blending (e.g., `glBlendFunc(GL_ONE, GL_ONE)`.)

Motion blur, defocus blur, and antialiasing are cases where samples over multiple dimensions allow multiple scene points to contribute to a sample and therefore can create translucency. Because net radiant flux is the sum over the contribution of each ray, it is mathematically equivalent to the weighted sum provided by partial coverage. These phenomena can therefore be accurately modeled by extending partial coverage by $\alpha' = \alpha * w$, where weight w is an estimate of the product of the fractional of exposure time, projected solid angle, and projected area that the surface covers relative for a pixel, and α is the original partial coverage of the surface. This is an area of significant current research and product development. See Sung et al. [2002] and Barskey and Kosloff [2008] for surveys of various blurring approximations for eye rays, most of which cannot be directly applied to the shadowing.

Our algorithm addresses shadows from non-refractive transmissive and partially covering surfaces, as well as surfaces that are simultaneously transmissive and partially covering. Emission, bloom, and lens flare do not involve an obscuring surface, so they are independent of our algorithm. Our algorithm depends on antialiasing to reduce the variance in stochastic samples, so it naturally supports antialiasing of both eye and shadow rays. We do not address motion and defocus blur, for which there do not yet exist high-quality algorithms for eye rays.

Note that wavelength-independent, non-refractive transmission can be modeled as partial coverage, and some renderers do so. Yet there are very good reasons for modeling partial coverage and transmission with separate parameters, which is why other renderers use that model. These reasons include:

- It is often desirable to model colored transmission, which cannot be done in a system where transmission is modeled by a single coverage α.
- Unlike coverage, transmission is independent of other material parameters. One can tune transmission without affecting the specular highlight or diffuse response, for example. In contrast, partial coverage makes parts of the surface disappear, so if you reduce α, all scattered lighting falls off proportionally. For example, glass modeled with no transmission and $\alpha = 0.1$ cannot produce a strong specular highlight.
- The α-test is used to avoid setting depth buffer values for low-α areas, e.g., cutouts around vegetation and billboards. Yet a 100% transmissive surface is still present, because it may still have emissive and other non-transmissive terms. A $\alpha = 0$ location really is a hole and should not be considered part of the scene.
- For a thin surface, α is independent of viewing angle, but the amount of transmitted light changes significantly due to Fresnel effects.
- For surfaces with partial coverage only, storing α takes 1/3 the space of the equivalent encoding as transmission, and specific texture compression formats are available efficiently for packing α with the diffuse reflectance.

A third alternative is to represent both transmission and coverage using a wavelength-varying α and no separate transmission term. RenderMan uses this representation, which is simple and elegant, but may be better suited to offline than real-time rendering. For surfaces with partial coverage but no translucency, storing three α values sacrifices some performance benefits, such as stochastic transparency with a single test and packing all parameters into an optimized diffuse-plus-coverage compressed RGBA format. It also makes it hard to model the aforementioned Fresnel and specular highlight effects, which affect transmission but not coverage. The CSSM algorithm discussed in this paper can be applied to either the RenderMan representation, or a representation with scalar coverage and a separate colored transmission texture map.

3 Related Work

Enderton et al. [2010] introduced several methods for increasing the effectiveness of screen-door translucency for partial coverage for both eye and shadow rays. This is the inspiration for CSSM. We extend their single-pass, partial-coverage shadowing method to colored translucency and then optimize it for RGB wavelengths and low sampling rates. We focus on shadowing instead of eye rays for two reasons. Viewers are more tolerant of blurry shadows than a blurry view, so one can do more filtering there. Shadow translucency is a more significant problem than eye ray translucency for games because far-to-near rendering of convex parts is inconvenient but sufficient for correct translucency in the camera's view but does not solve translucent shadowing.

Lokovic and Veach [2000] created a *deep shadow map* that stores every translucent fragment overlapping a pixel as a linked list or array. This can be used to produce ideal shadowing. Various methods have since been developed for constructing and applying this data structure for real-time rasterization rendering. These use clever GPGPU methods but are ultimately limited by the fact that the structure inherently requires unpredictable space and time per pixel. CSSM can be viewed as a stochastic equivalent of a deep shadow map that fits within the existing rendering pipeline.

Gosselin et al. [2004] basically computes a projective texture for the light based on transmissive surfaces in the scene. A closely related technique was used in the Starcraft II video game [Filion and McNaughton 2009]. That technique augments a traditional shadow map with a color buffer. The shadow depth map is computed solely from opaque surfaces. The shadow color buffer is the product of all transmissive surfaces closer to the light than the opaque depth, as seen by the light. The limitations of these methods are that they cannot cast shadows *on* transmissive surfaces, cannot cast proper shadows on participating media like fog and smoke, are incorrect for more than one layer of transmission, and cannot model partial coverage receivers or casters.

Dachsbacher and Stamminger's [2003] similary-named *translucent shadow maps* (TSM) are unrelated to CSSM. TSM are primarily for modeling subsurface scattering, not transmission and partial coverage of discrete surfaces.

Opacity shadow maps [Kim and Neumann 2001], Occupancy maps [Sintorn and Assarsson 2009] and Fourier opacity maps [Jansen and Bavoil 2010] both form low-frequency representations of a transmissive volume. These are well-suited to hair and dense participating media (like smoke), with uniform spectral response and no discrete surfaces. CSSM produces more noise and is inefficient for such materials, but can more accurately and efficiently represent layered discrete surfaces. Figure 7 contains depth-slicing artifacts from directly applying CSSM to particle-system smoke. This is a case where one would prefer some new extension of Fourier opacity maps to colored translucency.

Mitchell [2004] describes an extremely practical method based on work by Dobashi et al. [2002] that is today employed by games for simulating single-scattering in participating media. His method renders a traditional shadow map from opaque objects only and then fills the scene with hundreds of translucent fog planes that receive the shadowing. This produces compelling light shafts, which can have color if the light has a projective texture or the fog planes have colored texture. Under Mitchell's original method, translucent objects (and the fog itself) cannot cast shadows, however, his method can be trivially extended to use the new CSSM as shown in figures 3 and 6.

4 Algorithms

4.1 Combining Coverage and Transmission

Let the following probabilistic events be defined at the incidence of a photon of wavelength λ and a surface that lies within a triangle:

A = "The photon hits the triangle surrounding the surface"
S = "The photon hits the surface itself"
T = "The photon is transmitted through the surface"

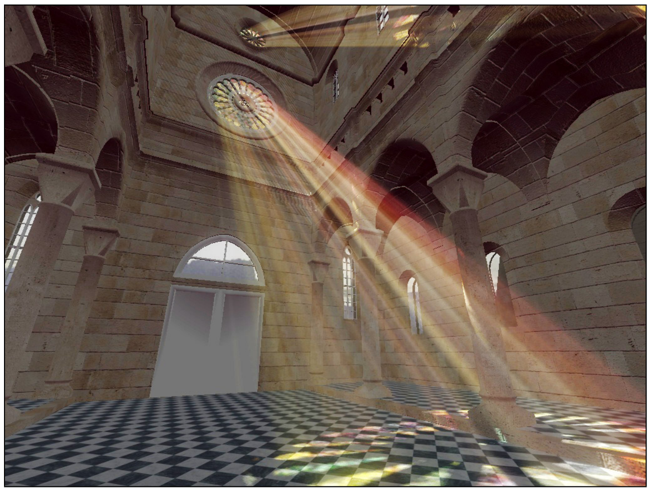

Figure 3: *Light shafts rendered by combining CSSM with Mitchell's method [2004]. The light source is a distant white spotlight representing the sun; colors arise from the translucent shadows cast by the stained glass windows.*

An example of the distinction between a surface and a triangle is an object like a tree leaf modeled with a triangle larger than the leaf and the exterior region trimmed away with a region of $\alpha = 0$, that should be considered "not present." Partial coverage is a statistical representation of this for surfaces like window screens where the holes are spread throughout the triangles.

Let the probability that a photon strikes the surface, given that the photon hit the triangle bounding the area spanned by the surface, be $\mathrm{P}(S \mid A) = \alpha$.

Let the probability that a photon at wavelength λ is transmitted through a surface, given that it hit the surface, be $\mathrm{P}(T \mid S) = \vec{t}_\lambda$. For example, the some surfaces might be modeled as:

Material	α	$\vec{t}_r$	$\vec{t}_g$	$\vec{t}_b$
Green glass	1.00	0.1	0.9	0.1
"Clear" nylon screen	0.25	0.5	0.5	0.5
Brick	1.00	0.0	0.0	0.0
Black nylon screen	0.25	0.0	0.0	0.0

Both $\vec{t}$ and α can be texture mapped across a mesh.

The net probability of a photon incident on the triangle being absorbed or reflected conveniently reduces to:

$$\begin{aligned}\vec{\rho}_\lambda &= \mathrm{P}(\bar{T} \mid A) = 1 - \mathrm{P}\left(\left[(S \cap T) \cup \bar{S}\right] \mid A\right) \\ \vec{\rho}_\lambda &= (1 - \vec{t}_\lambda)\alpha \qquad (1)\end{aligned}$$

In other words, $\vec{\rho}_\lambda$ is the fraction of light at each wavelength that hits the surface and is not transmitted, which is the constant we require for colored stochastic shadow casting.

4.2 General Algorithm (CSSM1)

Given the derivation from section 4.1, we simply extend stochastic transparency shadow maps [Enderton et al. 2010] to include non-refractive colored transmission. We call this algorithm CSSM1. It requires an array of shadow maps, one for each wavelength (e.g., three for RGB.)

The algorithm has two parts: colored shadow map generation (for shadow rays) based on ρ, and the net illumination $\vec{X}$ based on that shadow map (for eye rays) and light color $\vec{L}$ to be applied at each fragment. We call the second *shadowedLightColor*; it is what PCF sampling does for traditional opaque shadow maps. For the CSM1 algorithm, these parts are:

```
CSSM1 ALGORITHM

generateShadowMap():
  1. For each wavelength $\lambda$:
     (a) Bind and clear depth texture shadow[$\lambda$]
     (b) Set the projection matrix from the light's viewpoint
     (c) Render all surfaces; discard fragments with $\vec{\xi}_\lambda < \vec{\rho}_\lambda$
  2. Return the shadow array

shadowedLightColor():
  1. Let $\vec{s}_{xyz}$ be the projected shadow map texture coordinate
     and depth value (as specified by GLSL shadow2D)
  2. For each wavelength $\lambda$:
     (a) Let $X_\lambda = 0$
     (b) For each sample offset $\Delta$ (of $n$ total):
           i. $\vec{X}_\lambda$ += (texture2D(shadow[$\lambda$], $\vec{s}_{xy} + \vec{\Delta}$).r $> \vec{s}_z$)
     (c) $\vec{X}_\lambda = \vec{L}_\lambda \vec{X}_\lambda / n$
  3. Return $\vec{X}$
```

In this pseudocode, $\vec{\xi}$ is a vector of uniformly distributed random numbers on [0, 1], which we compute by a hash of the fragment's world-space position, following Enderton et al. Let the boolean→real mapping of the greater-than comparison be: false→0, true→1. The `texture2D` function corresponds to the GLSL 1.50 texture sampling function. The sample and compare can be replaced with the `shadow2D` function, which is incorporated into the overloaded `texture` function for `sampler2DShadow` arguments under GLSL 3.30. We present an explicit depth comparison here to set up the later derivation of the CSSM2 algorithm.

ShadowedLightColor must be applied in the context of some other algorithm for rendering translucent surfaces with correct eye ray visibility, e.g., the painter's algorithm, depth peeling, or an order-independent transparency method.

There are three drawbacks to the CSSM1 algorithm. The first is that it must render and sample multiple shadow maps. This increases the shadow map generation time, memory space, and shadow bandwidth required when shading proportionally. The multiple passes during shadow map generation are particularly troubling because for texture-mapped transmission and partial coverage, the same texels must be fetched for each wavelength.

The second drawback is that it may require more shadow samples when shading than a traditional shadow map to produce pleasingly smooth results. This is because of the variance inherent in the stochastic sampling during shadow map generation and is inherited from stochastic transparency, which also requires many samples per pixel. The third drawback is that because each wavelength's stochastic samples are independent, there can be color noise as well as intensity noise. In practice, we observe that in practice this is no worse than the intensity noise and is ameliorated by the same level of sampling during shading.

4.3 Efficient Algorithm for the RGB Case (CSSM2)

The CSSM2 algorithm is a time and space optimization of CSSM1 for the common case of RGB wavelength samples. To avoid rendering three shadow maps, the CSSM2 algorithm packs three depth buffers into a single color texture. This immediately yields a 3x performance increase for shadow map generation. It also saves bandwidth and instructions, increases coherence, and allows vectorization in both shadow map generation and fragment shading.

The challenge is encoding depth values in color channels without losing the hierarchical and early-z tests and basic depth-test functionality, which are tied to depth textures under current GPUs and APIs. Our approach is to retain a temporary depth buffer for opaque surfaces and use min-blending of color channels to simulate a depth test for translucent surfaces. Many real-time systems render all shadow maps to textures before all visible surfaces to allow multiple lights in each shading pass. The downside of this approach is that all shadow maps must be resident simultaneously, and on consoles texture memory is fairly limited. Fortunately, the CSSM2 data structure is just the color texture; the depth texture is only needed to construct the color texture. Thus the memory for single depth texture may be shared among all lights.

The CSSM2 algorithm is:

```
CSSM2 ALGORITHM

generateShadowMap():
  1. Set the projection matrix from the light's viewpoint
  2. Bind and clear the depth buffer and shadow color buffer

  3. Disable color write, enable depth write
  4. Render all opaque surfaces

  5. Enable color write, disable depth write
  6. Copy depth to all color channels by rendering a large quad

  7. Set MIN blending
     (i.e., glBlendEq(BLEND_MIN); glBlendFunc(ONE, ONE))
  8. Render all translucent surfaces; let each fragment's color be
     $\max(z,\ (\vec{\xi} > \vec{\rho}))$, where $z$ is the fragment's depth value
     (i.e., glFragCoord.z)
  9. Return the shadow color buffer texture

shadowedLightColor():
  1. Let $\vec{s}_{xyz}$ be the projected shadow map texture coordinate
     and depth value
  2. Let $\vec{X} = \vec{0}$
  3. For each sample offset $\vec{\Delta}$ (of $n$ total):
       (a) $\vec{X}$ += (texture2D(shadow, $\vec{s}_{xy} + \vec{\Delta}$).rgb $> \vec{s}_z$)
  4. Return $\vec{X}\vec{L}/n$
```

CSSM2 addresses the primary drawback of CSSM1 because it eliminates the triple-shadow map and per-wavelength loops. For scenes that can be rendered with 10-bit shadow map depth precision, the CSSM2 algorithm requires only 2/3 the memory of CSSM1 because it packs into 30 bits per pixel using the OpenGL `GL_RGB10` texture format, versus three `GL_DEPTH16` textures. That is fairly limited depth precision, although it is reasonable for scenes with limited vertical range and overhead lights (e.g., both the StarCraft II and Marvel Ultimate Alliance games previously mentioned). We rendered all results in this paper with `GL_RGB16F` textures, which we recommend for general scenes, and observed no performance difference from the 50% higher bandwidth.

4.4 Choosing the $\vec{\Delta}$s

As with traditional shadow maps, a regular block of $\vec{\Delta}$-offsets is inferior to a distributed pattern [Reeves et al. 1987]. A regular block makes adjacent pixels statistically dependent, which leads to low-frequency noise in light space. In the case of CSSM, that noise manifests as color splotches in shadows.

Designing a shadow reconstructions filter for a very low sample count is something of a black art because theoretical signal processing considerations become swamped by the particulars of human perception, the scene texture, artifacts from other effects, and the characteristics of specific noise functions. We informally investigated n-rooks, box, disk, and random striated filters, then selected and tuned a box-plus-cross-shaped filter for its empirical performance and aliasing characteristics. We report that filter here and observe that it gives a reasonably low variance and consistent shadow term estimate at high performance, but make no quantitative claims about its variance reduction properties. We suggest as future work that a better filter could further improve image quality.

The CSSM2 filter contains 13 single taps placed relative to the center, in texels, at

$$\vec{\Delta} = \vec{x} + \vec{\delta}(\vec{s}_{xy}) \tag{2}$$

$$\vec{x} \in \{(0,0), (\pm 3, \pm 3), (\pm 4, 0), (0, \pm 4), (\pm 7, 0), (0, \pm 7)\} \tag{3}$$

The micro-offset $\vec{\delta}$ provides jittering. It ensures that single-pixel noise appears instead of large texel blocks when a shadow map texel projects to multiple screen pixels. This is a commonly observed technique in games that is an alternative to bilinear interpolation as a texture magnification method for shadows. We sought to mimic a similar effect from the Futuremark Games Studio title Shattered Horizon, and chose the particular jitter function

$$\vec{\delta}(\vec{s}_x y) = \frac{[(5\vec{s}_{xy}) \bmod (2,2)] - (1,1)}{6\left(\left|\left|\frac{\partial \vec{s}_{xy}}{\partial x}\right|\right|_1, \left|\left|\frac{\partial \vec{s}_{xy}}{\partial y}\right|\right|_1\right)}, \tag{4}$$

in which $||\cdot||_1$ denotes Manhattan distance. The strange denominator arises because the Manhattan distance of a spatial derivative is supported by specific OpenGL/DirectX API calls and GPU hardware that provide derivative estimates by finite differences across sets of four pixels. This noise function is a simplified version of a more sophisticated one described by Mittring [2007] that was used on Crytek's Crysis 2. The filter gave results roughly comparable to a 9-tap bilinear filter of diameter five texels for traditional shadow maps. The CSSM2 filter needs to be wider than the bilinear filter to reduce stochastic variance because it cannot average four values per tap using hardware PCF sampling.

5 Graphics API Considerations

Like traditional shadow maps, the CSSM algorithm only depends on some high-level features of a renderer and is therefore largely independent of the implementation API. Nonetheless, the design of a specific API can affect the implementation complexity and constant-factor performance.

5.1 Hardware Anti-Aliasing

Many renderers use multi-sample antialiasing (MSAA) to shade only once per fragment but sample visibility at multiple locations, which improves the quality of antialiasing without incurring a proportional cost. Compared to traditional shadow maps, there is no new interaction with MSAA when shading visible surfaces. However, when generating the shadow map one could theoretically leverage MSAA to increase performance. For example, the CSSM can be rendered at 1/16 resolution with 16 MSAA samples per pixel, which yields equivalent coverage at reduced rendering cost. To ensure that the stochastic masking is performed per sample and

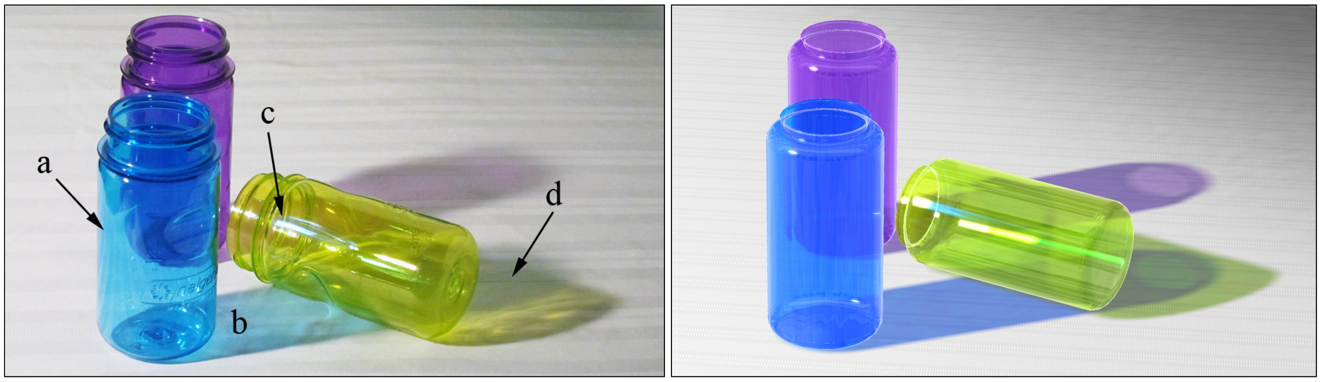

Figure 4: ***Left)*** *Photograph and* ***right)*** *image rendered by our algorithm, demonstrating target transmissive shadowing phenomena it correctly simulates, including:* ***a)*** *directly illuminated transmissive surfaces receive no shadowing;* ***b)*** *surfaces indirectly illuminated by transmitted light exhibit shadows matching the product of the transmitter's and receiver's spectra, which also leads to* ***c)*** *colored highlights; and* ***d)*** *the shadows of multiple transmitters are the product of all the transmitters' spectra with the receiver's spectrum.*

not per pixel, replace the per-fragment discard decision with a per-coverage-mask element decision (by writing to `gl_SampleMask[]` in OpenGL 4.0/DirectX 10.1).

5.2 Percentage Closer Filtering Optimizations

Using *percentage-closer filtering* (PCF) [Reeves et al. 1987; Fernando 2005], NVIDIA GPUs will average the result of four depth tests if a single shadow comparison (`shadow2D`) is made to the point between four texels in a depth map. This allows those GPUs to issue fewer texture fetch instructions in the high-level shading language, which may lead to performance gains depending on the low-level architecture. Because `shadow2D` is only defined for the red channel of a depth texture, CSSM2 cannot use this instruction. Thus its memory performance may be lower per filter tap on pre-2010 NVIDIA cards, including the Playstation 3.

GPUs by other vendors, including the Xbox 360 ATI GPU, do not support PCF and thus CSSM2 has the same memory behavior as a traditional shadow map on them. Newer GPUs support the DirectX 11 and OpenGL 3.3 four-texel fetch instruction, which allows the texture fetch for percentage-closer style filtering to be issued efficiently across all vendors. CSSM2 should have the same memory performance as a traditional shadow map if implemented with this instruction. The CSSM2 results in this paper were rendered under OpenGL 3.0 with one fetch per instruction, i.e., equivalent to the Xbox 360's limitations, and are thus are a conservative performance estimate.

For programmer convenience and potential extended architectural support, I propose an extension to GLSL that (1) defines `shadow2D` for `sampler2D` as well as `sampler2DShadow` types, and (2) defines PCF filtering for all color channels.

6 Results

6.1 Quality

The side-by-side comparisons of real photographs and images rendered with CSSM in figures 4 and 5 demonstrate that the algorithm is able to simulate the kinds of colored translucency phenomena observed in the real world. (All result images were rendered with CSSM2, which produces identical results to CSSM1 with the same filter.) The rendered images are not intended to match the photographs exactly, since the model geometry and materials are only

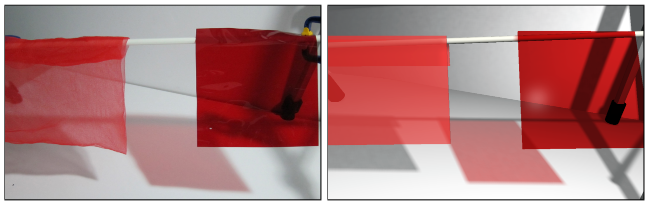

Figure 5: ***Left)*** *Photograph of a red scarf and red theatre gel demonstrating different colored translucency shadow phenomena that can be simulated by our method. The scarf's appearance arises from partial coverage (α) by opaque "red" threads. The gel's appearance is due to preferential transmission of "red" light. Note the difference in shadow color.* ***Right)*** *A similar virtual scene rendered with a Colored Stochastic Shadow Map.*

rough approximations and the bottles in figure 4 create some caustics that CSSM does not simulate.

We note one interesting artifact in the *photographs*: despite being captured with a midrange (Cannon S90) camera under about 40W of incident illumination and filtered down to HD resolution, they exhibit about as much noise as the rendered images.

Figure 6 demonstrates the correctness of CSSM in comparison to previous algorithms, which are denoted by abbreviated citations. The scene contains a blue crystal statue, pierced by a beam of light from a high window. The scene is filled with low-coverage, highly reflective particles that do not cast shadows themselves. These are rendered as full-screen textured quads that fill the view frustum, following Mitchell [Mitchell 2004]. This causes the light shaft to be visible. The shaft should be white before it strikes the statue and blue afterward. Note that the first transmissive surface seen by the light is the window glass, not the statue. Image (a) shows the result produced by End10 [Enderton et al. 2010], in which the shaft remains colorless despite the blue transmitter because that algorithm cannot represent colored transparency. The Fil09 [Filion and McNaughton 2009] result (not shown) has the same artifact for this scene because it samples the window color and not the statue color. Image (b) shows that Gos04 [Gosselin et al. 2004] incorrectly colors the entire shaft blue. That is because that algorithm propagates transmissive colors all of the way back to the light, as if there were a colored gobo in front of it. Image (c) is the CSSM result. CSSM can represent spectrum varying with distance from the light, so the shaft is correctly blue on the lower-left and white in the upper right.

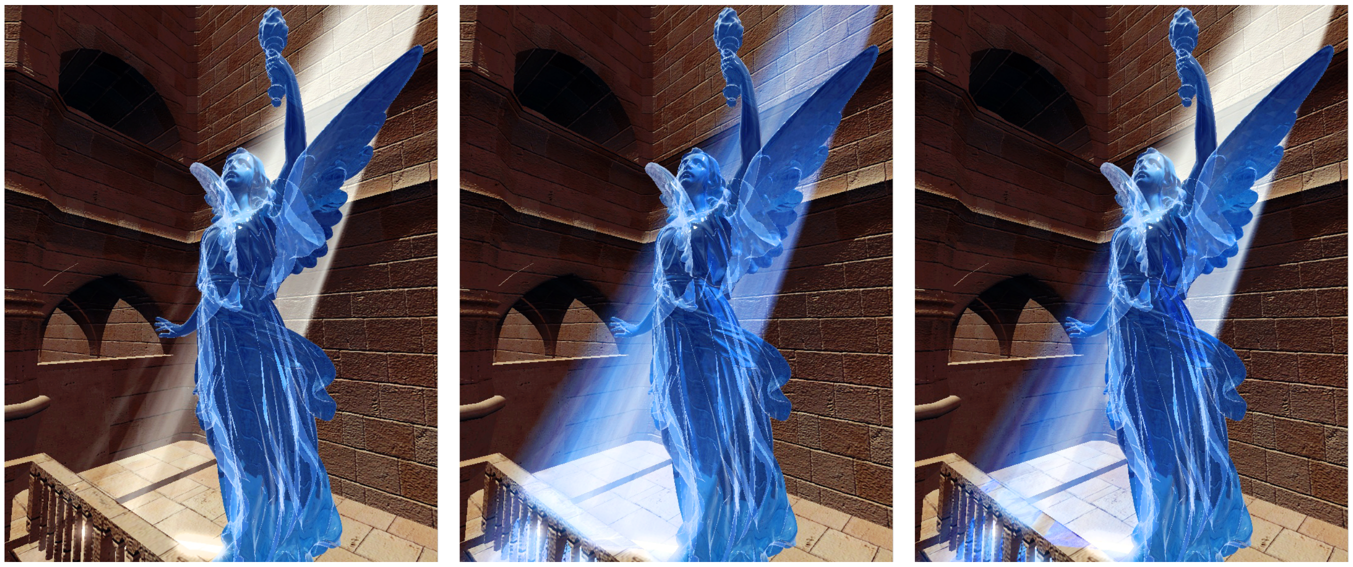
a) End10/Fil09: no color b) Gos04: wrong color c) CSSM: correct distance-varying color

Figure 6: *A blue crystal angel (Stanford's low-polygon "Lucy") statue in a shaft of light in the Sibenik cathedral.*

Figure 7 shows a scene containing dense fog, for which translucent self-shadowing is important. (This image is an homage to a similar figure *without* colored translucency by Lokovic and Veach [Lokovic and Veach 2000].) This scene is modeled as two opaque vertical white pipes, two transmissive orange pipes, three opaque cyan pipes, and a particle system of opaque, partially-covering fog modeled with texture-mapped billboards. Note the colored and opaque shadows cast through the smoke. Also note the self-shadowing of the smoke, causing it to darken near the bottom. The white vertical pipes are also darker near the ground because of shadowing from the smoke. Banding artifacts on the cyan pipes occur because the particles are billboards. The soft particle method is one algorithm (that we did not implement) that can be used to conceal this artifact.

6.2 Performance

We intentionally selected simple scenes for quality evaluation so that noise and color interactions would be visible. For performance evaluation we selected four scenes with varying levels of complexity: the game scene shown in figure 2, from both a typical viewpoint and the worse viewpoint we could find for CSSM2, the Sibenik and Sponza benchmark models by Marko Dabrovic, and the Postsparkasse model (figure 8) by Christian Bauer that contains a two-layer glass ceiling and glass floor. The latter three models were downloaded from `http://hdri.cgtechniques.com/~sibenik2/`. The worst case viewpoint for CSSM2 overhead on the game scene was where the camera was located so that all surfaces were in shadow.

We evaluated five algorithms. We consider the Wil78 [Williams 1978] algorithm for opaque shadows a baseline, since most game developers use some variation of it for opaque shadows today. Any practical translucent shadow algorithm must not be significantly more expensive than this for deployment on current hardware for interactive applications. The Gos04 and End10 algorithms generate incorrect results for overlapping and colored surface, as previously demonstrated. However they are known to have good performance characteristics and are therefore algorithms one would consider in practice, especially for an application that generally could work within their limitations. CSSM1 and CSSM2 are the new algorithms presented in this paper.

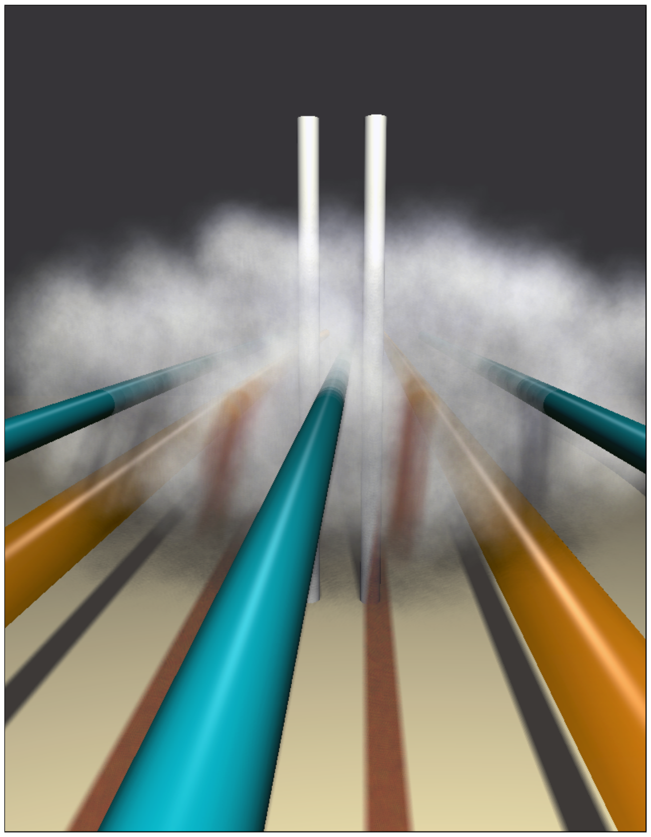
Figure 7: *Particle-system smoke casting and receiving shadows with CSSM. The orange pipes are transmissive, the smoke has partial coverage.*

We used the same reconstruction filter for Wil78, Gos04, End10, and CSSM1. This circle-plus-point-shaped filter contained 9 bilinear taps placed at the center of and at 45° intervals around a 2-texel radius disk. For CSSM2 we used the filter described in section 4.4. This is because CSSM2 is unable to perform bilinear filtering, so it requires more filter taps than the other algorithms to produce good results. CSSM2 with the 9-tap filter is faster and produces noisier results; the other algorithms on the CSSM2 filter are slower and give slightly blurrier results. All depth values maps were encoded in 16-bit floating point (per channel, for CSSM2) and the Gos04 color map was at 8-bits per channel.

Table 1 summarizes the render time, in milliseconds, of shadow map generation and actual shading for each algorithm when run on an NVIDIA GeForce GT 280 GPU under Windows Vista using OpenGL 2.0. All timings were computed with the `GL_TIMER_QUERY` extension, which enables accurate and asynchronous evaluation of the time for commands to propagate through the GPU pipeline. The right-most column of the table lists the overhead in milliseconds for CSSM2 compared to Wil78. This is the per-frame cost of adding colored translucent shadows to a typical existing rendering engine.

Beware that render times for complex scenes are affected by many factors beyond per-texel computation. These factors include memory and branch coherence, cache hit rate, the pipeline impact of texture and shader changes, and the sharing of units between vertex and pixel processing. Thus in some cases an algorithm that performs strictly more computation may still have higher performance, e.g., Gos04 compared to Wil78 on the game scene.

Camera and light viewpoints also can affect performance by about 10%, as demonstrated by the game/typical vs. game/worst results. We observed similar behavior for all scenes; this is a representative result. Note that the worst case viewpoint yielded higher net performance for CSSM2 than the typical case. It is the "worst case" because Wil78 was disproportionally faster for that viewpoint.

In general, CSSM2 maintains performance close to that of the previous algorithms, yet it is able to also correctly model the colored translucent shadows. CSSM2 is two to three times faster than CSSM1, which demonstrates that the optimizations in its design successfully reduced most of the overhead of managing three shadow maps simultaneously.

		Opaque		*Translucent*			*CSSM2 -*
		Wil78	*Gos04**	*End10**	*CSSM1*	*CSSM2*	*Wil78*
Game (typical)	Generate	7.6	7.0	7.4	27.4	7.7	
1096 kTri	Apply	8.4	8.6	7.1	9.8	8.4	
	Total	**16.0**	**15.6**	**14.5**	**37.2**	**16.1**	**0.1**
Game (worst)	Generate	6.6	6.8	6.5	28.2	6.5	
1096 kTri	Apply	7.6	7.5	6.0	8.4	8.6	
	Total	**14.2**	**14.3**	**12.5**	**36.6**	**15.1**	**0.9**
Sibenik	Generate	2.6	2.3	3.1	22.9	3.2	
80 kTri	Apply	4.6	4.0	4.7	4.7	4.6	
	Total	**7.2**	**6.3**	**7.8**	**27.6**	**7.8**	**0.6**
Sponza	Generate	3.3	4.1	3.3	13.5	4.2	
66 kTri	Apply	4.3	3.2	3.1	6.4	4.1	
	Total	**7.6**	**7.3**	**6.4**	**19.9**	**8.3**	**0.7**
Postsparkasse	Generate	2.6	4.8	3.4	21.1	4.3	
267 kTri	Apply	6.8	7.1	6.8	8.9	8.3	
	Total	**9.4**	**11.9**	**10.2**	**30.0**	**12.6**	**3.2**

Table 1: *Rendering time in milliseconds for the shadow map generation and lighting application for various algorithms at 1920×1080 for scenes containing four unshadowed lights, hemisphere ambient, and one shadow casting light. The right column shows the overhead of CSSM2 over traditional shadow maps. * Gos04 and End10 generate incorrect results for these scenes.*

7 Discussion

We expect that developers would like accurate colored shadows, but are only willing to add them if the incremental cost over opaque shadows is fairly low. We have shown that at the same resolution as a traditional shadow map, CSSM adds at most a few milliseconds to a high-resolution frame render time. However, the CSSM shadows are slightly blurrier than opaque ones because they use a wider filter to reduce the hue variance in colored shadows. This can be addressed by increasing shadow map resolution. The cost of a shadowing algorithm is subjective because the impact of blur, noise, lack of color, render time, and texture map space depend on the viewer and the application.

Note that the tradeoff of noise versus blur versus resolution is less significant for shadow rays than for eye rays. This is why CSSM looks reasonable with many fewer samples than one would need for stochastic transparency of eye rays. For static lights and objects, shadow noise is in world-space, so it blends with texture noise. For dynamic lights and objects the shadows are in motion, so noise is less perceptible. Overblurring shadows to reduce variance and aliasing is often acceptable because that also approximates shadowing from an area source or diffusion inside a translucent surface (at least, viewers often interpret the images that way). We cannot apply the CSSM reconstruction filter directly to colored stochastic transparency for eye rays because they would blur edges in the image itself, which is not an acceptable artifact.

Today, CSSM just manages to hold the stochastic noise to an acceptable level with low overhead compared to traditional shadow maps. Assuming that GPUs continue to increase in raw processing power and bandwidth, in the near future this will likely be so negligible that it will make sense to always use stochastic shadowing. In general, we suspect that stochastic techniques for rasterization like stochastic transparency and CSSM offer so many advantages that they will become widespread. Stochastic methods have long dominated ray tracing because they allow phenomena to combine naturally, rather than requiring special purpose "effects." This reduces the software engineering burden and artifacts of combining phenomena. Motion blur, defocus, and translucency are three phenomena that are currently hard to simulate well under rasterization, yet they are all trivial when implemented stochastically. Of these, translucency for shadows offers the best performance because it can undersample visibility, but we believe that the others will also be viable in the near future as well.

Acknowledgements

Thanks to Eric Enderton (NVIDIA) for his input on the algorithm design and its presentation in this paper.

References

BARSKY, B. A., AND KOSLOFF, T. J. 2008. Algorithms for rendering depth of field effects in computer graphics. In *ICCOMP'08: Proceedings of the 12th WSEAS international conference on Computers*, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 999–1010.

BAVOIL, L., CALLAHAN, S. P., LEFOHN, A., COMBA, JO A. L. D., AND SILVA, C. T. 2007. Multi-fragment effects on the gpu using the k-buffer. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, 97–104.

DACHSBACHER, C., AND STAMMINGER, M. 2003. Translucent shadow maps. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 197–201.

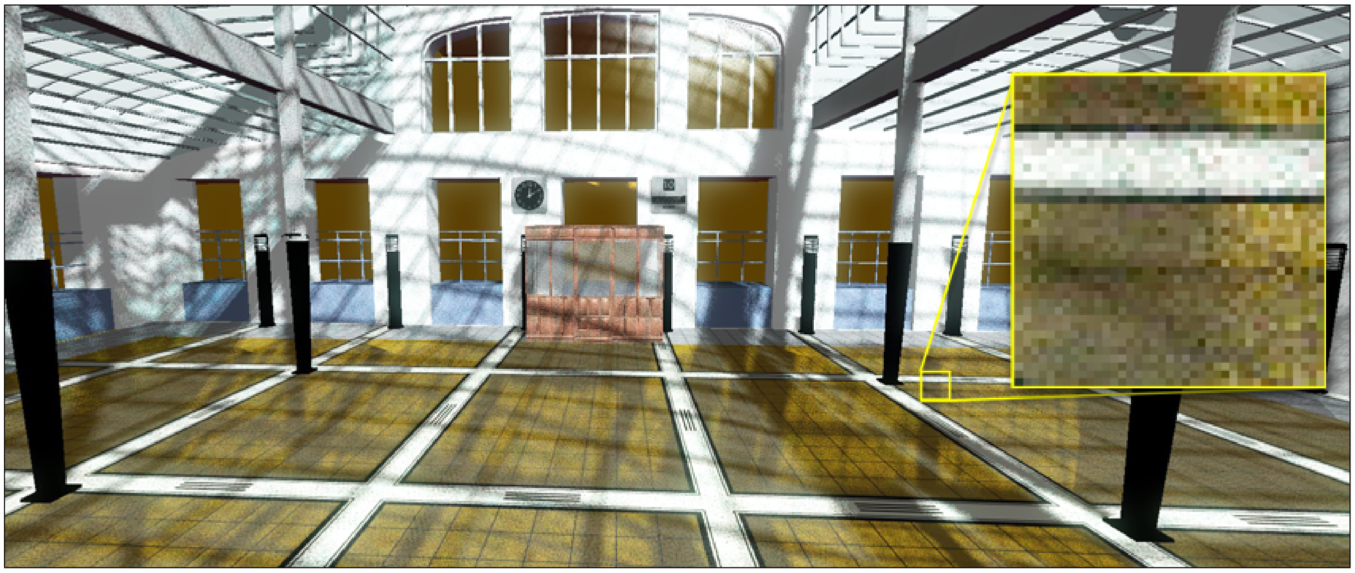

Figure 8: *The Austrian Postsparkasse building contains two layered glass roofs, a glass floor, and multiple windows. All surfaces are thus within two or three translucent shadows. The inset shows stochastic sampling noise scaled up 10x. This is a worst-case scene for noise because there is no surface texture.*

DOBASHI, Y., YAMAMOTO, T., AND NISHITA, T. 2002. Interactive rendering of atmospheric scattering effects using graphics hardware. In *HWWS '02: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 99–107.

ENDERTON, E., SINTORN, E., SHIRLEY, P., AND LUEBKE, D. 2010. Stochastic transparency. In *I3D '10: Proceedings of the 2010 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA.

FERNANDO, R. 2005. Percentage-closer soft shadows. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches*, ACM, New York, NY, USA, 35.

FILION, D., AND MCNAUGHTON, R., 2009. StarCraft II effects and techniques. SIGGRAPH 2009 Real-Time Rendering Course, Natalya Tatarchuk, moderator.

GOSSELIN, D., SANDER, P. V., AND MITCHELL, J. L. 2004. *Real-Time Texture-Space Skin Rendering*. Charles River Media, Inc., Rockland, MA, USA, ch. 2.8, 171.

HECHT, E. 2002. *Optics*. Addison-Wesley. 4th Edition.

JANSEN, J., AND BAVOIL, L. 2010. Fourier opacity mapping. In *I3D '10: Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, 165–172.

KIM, T.-Y., AND NEUMANN, U. 2001. Opacity shadow maps. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, Springer-Verlag, London, UK, 177–182.

LOKOVIC, T., AND VEACH, E. 2000. Deep shadow maps. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 385–392.

MITCHELL, J. L. 2004. *Light Shaft Rendering*. Charles River Media, ch. 8.1, 573–588.

MITTRING, M. 2007. Finding next gen: Cryengine 2. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, ACM, New York, NY, USA, 97–121.

MURPHY, S., 2009, September. Personal communication with the Senior Lead Artist, Vicarious Visions.

PORTER, T., AND DUFF, T. 1984. Compositing digital images. In *Computer Graphics (Proceedings of SIGGRAPH 84)*, 253–259.

REEVES, W. T., SALESIN, D. H., AND COOK, R. L. 1987. Rendering antialiased shadows with depth maps. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 283–291.

SINTORN, E., AND ASSARSSON, U. 2009. Hair self shadowing and transparency depth ordering using occupancy maps. In *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games*, 67–74.

SUNG, K., PEARCE, A., AND WANG, C. 2002. Spatial-temporal antialiasing. *IEEE Transactions on Visualization and Computer Graphics 8*, 2, 144–153.

WILLIAMS, L. 1978. Casting curved shadows on curved surfaces. In *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 270–274.

WYMAN, C. 2005. Interactive image-space refraction of nearby geometry. In *GRAPHITE '05: Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, ACM, New York, NY, USA, 205–211.