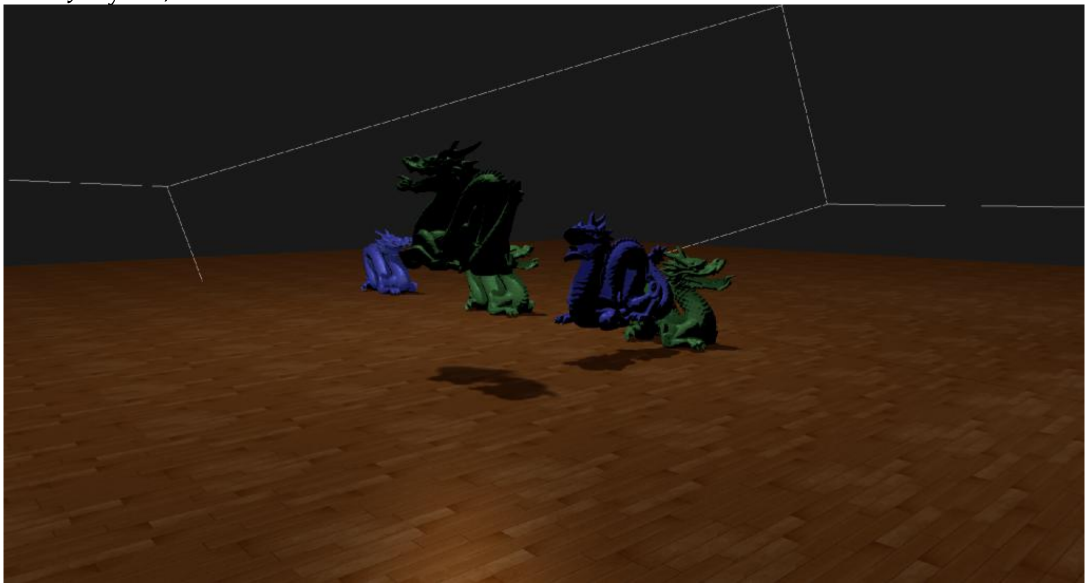
Andrew Pham

Articles on graphics programming

Reflective Shadow Maps

Global Illumination, GPU, Real-Time Rendering, Sampling, Shadow Mapping August 29, 2019July 13, 2020

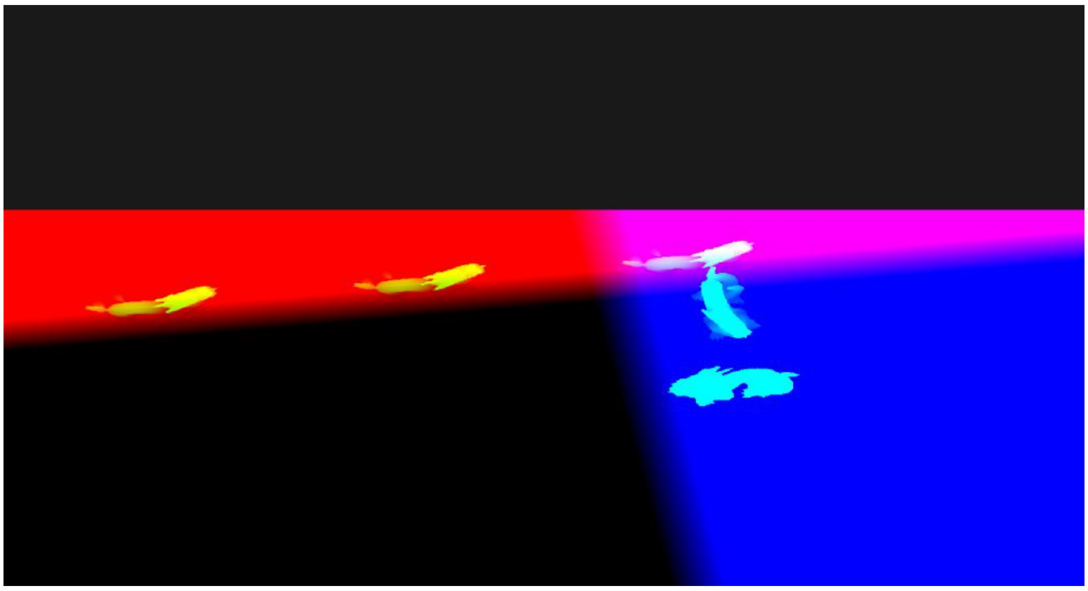


There is a nice extension to shadow mapping that allows for the rendering of plausible indirect illumination. It's not quite as performant as voxel cone tracing, but it introduces some ideas on how we can model the behavior of light.

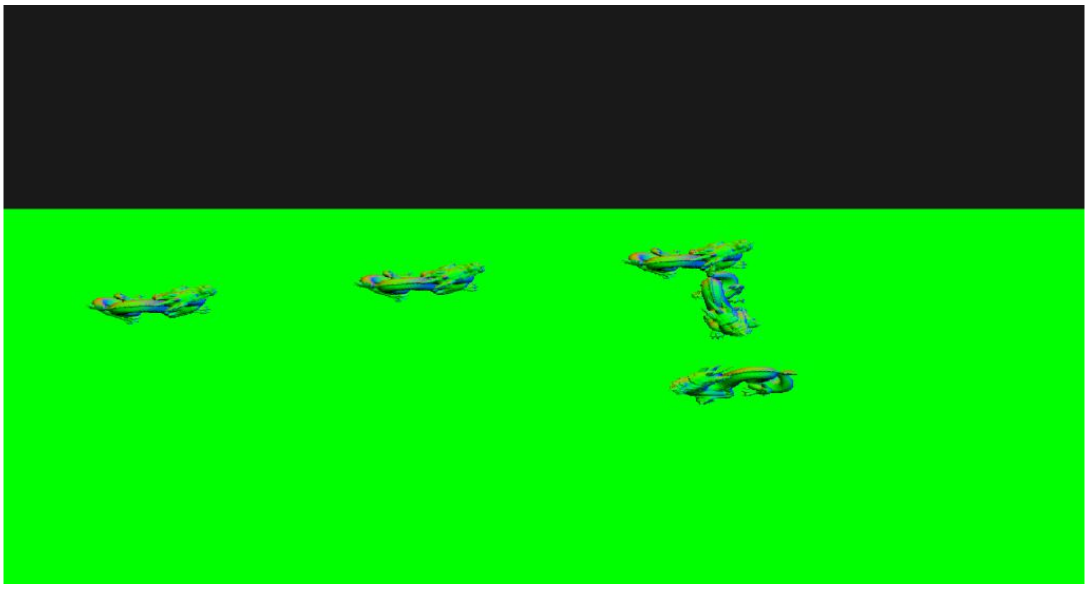
Every pixel in the shadow map is considered as an indirect light source and we sample these pixels to determine the indirect lighting contribution. To create a situation that distills the needed lighting data in each pixel so that we can treat it like an additional light source to our point light, we store additional data across several framebuffer render targets (instead of just the one shadow map) that will be written to during the shadow map pass.

Concretely, each pixel in the shadow map is considered as a small area light source that generates a one-bounce indirect illumination in the scene. The crux idea is that from the point of view of a single light source (i.e., the shadow map), we can determine the surfaces visible from its perspective that cause one-bounce indirect lighting.

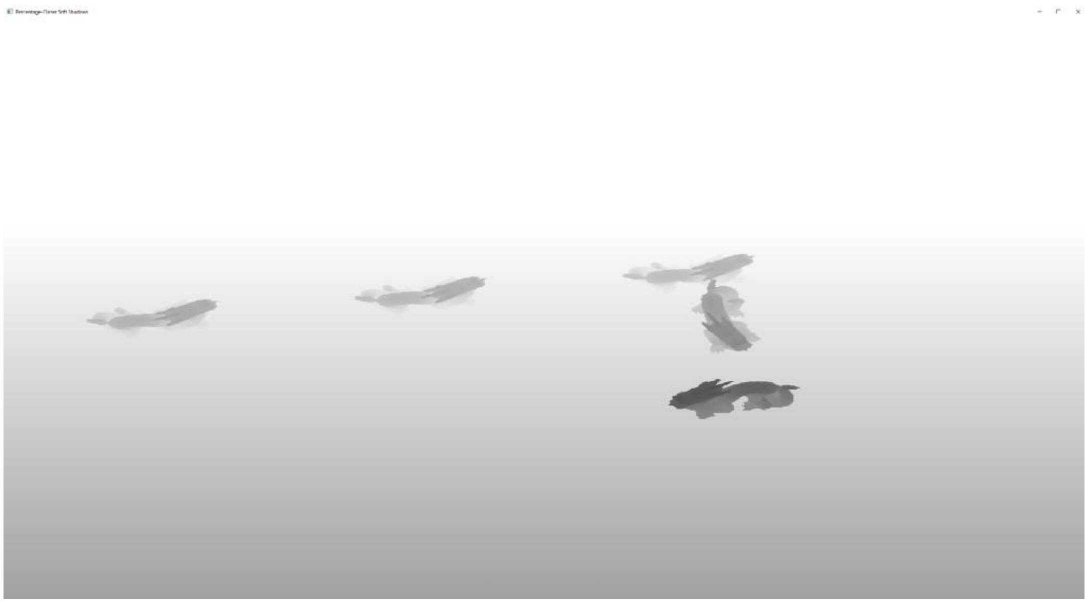
Reflective shadow map (RSM) refers to the collection of data that will be relevant to our determination of global illumination (i.e., RSM manifests as several render targets in our case). It stores four relevant pieces of data: the **depth** value of the pixel or surface (like usual in normal shadow mapping), the **world position**, the **normal**, and the reflected radiant **flux**.



The world positions map.



The world space normals map from the perspective of our point light.



The depth values buffer (i.e., the canonical shadow map).

Flux defines the radiant energy per unit time (i.e., brightness), which I define as diffuse color times light color in the shadow map pass. In Carsten Dachsbacher's Reflective Shadow Maps paper (http://www.klayge.org/material/3_12/GI/rsm.pdf), Dachsbacher defines reflected flux as the flux through the pixel times the reflection coefficient of the surface. It is generally constant for a uniform parallel light (this maps well to how I set up my point light to display an orthographic projection from its perspective), so its definition matches what I have.

The reflected flux map. Note that it looks like an unshaded image.

I should also note that the positions map is optional because we can calculate the world position using the depth buffer and the inverse view projection matrix.

With all of our variables, we can define the irradiance at surface point x with normal n due to pixel light p as

$$E_p(x,n) = \Phi_p \frac{\max\{0, \langle n_p | x - x_p \rangle\} \max\{0, \langle n | x_p - x \rangle\}}{||x - x_p||^4}$$

From this point, the indirect lighting contribution of the sample pixel p follows, followed by the total indirect lighting on the given pixel or surface point (noting that we can determine radiance from the calculated irradiance).

Note also that basing irradiance in terms of the radiant flux allows us to forgo integrating the area of the light as part of our calculations. What the equation tells us is that the spatial configuration described by the pair of normals has an impact on the resulting indirect lighting contribution. Additionally, larger distances between pixels attenuate the result. Of course, all of this is controlled by the amount of flux. Clearly, the N dot L factors are indicative of a diffuse reflectance scenario; thus, right off the bat, this equation can be quite limiting (i.e., we can't handle non-diffuse reflectors).

Concretely, the total indirect lighting on a given pixel is defined as

$$E(x,n) = \sum_{\text{pixels} p} E_p(x,n)$$

There is one issue with indiscriminately sampling points around a given pixel—Dachsbacher does not handle occlusion for indirect light sources well at all. Dachsbacher goes into detail regarding this issue but, needless to say, the summation above is a bit of an approximation. It makes the trade-off of actually having indirect lighting effects at the expense of rendering a potentially inaccurate and brighter result than we need. That being said, it is possible to apply ambient occlusion techniques in addition to what's being described here, and I typically prefer it when occlusion and other factors are readily baked into the global illumination calculations.

```
glBindFramebuffer(GL_FRAMEBUFFER, renderTargets->_FBO);
normalMap = renderTargets->genAttachment(GL_RGB16F, GL_RGB, GL_FLOAT, true);
fluxMap = renderTargets->genAttachment(GL_RGB16F, GL_RGB, GL_FLOAT, true);
shadowMap = renderTargets->genAttachment(GL_R16F, GL_RED, GL_FLOAT, true);
GLenum bufs[4] = { GL_COLOR_ATTACHMENT0, GL_COLOR_ATTACHMENT1, GL_COLOR_ATTA
glDrawBuffers(4, bufs);
```

Also, here's how we render to our render targets:

```
#version 460 core
layout (location = 0) out vec3 Position;
layout (location = 1) out vec3 Normal;
layout (location = 2) out vec3 Flux;
layout (location = 3) out float Depth;
const vec3 LIGHT_COLOR = vec3(0.3f);
in vec3 WorldPos;
in vec2 TexCoords;
in vec3 WorldNormal;
uniform sampler2D gDiffuseTexture;
void main()
{
    Position = WorldPos;
    Normal = normalize(WorldNormal);
    Flux = texture(gDiffuseTexture, TexCoords).rgb * LIGHT_COLOR;
    Depth = gl_FragCoord.z;
}
```

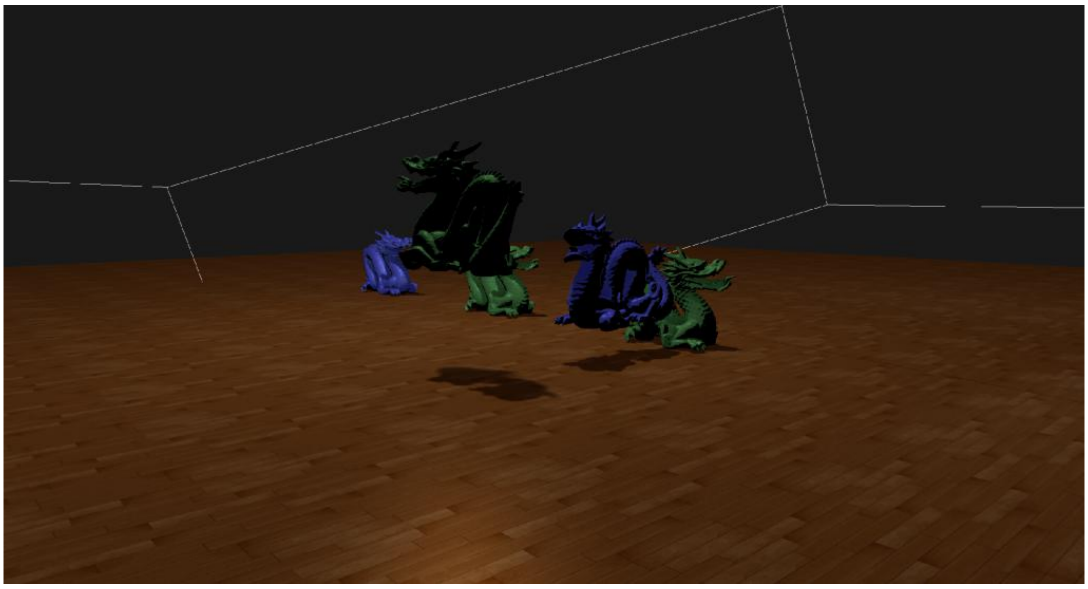
The only question that remains is what pixels do we sample in screen space to determine a reasonable value for indirect lighting? Obviously, it's inefficient to sample all of them. Dachsbacher makes the assertion that distances between pixels in the shadow map are a good approximation of the actual distances in world space and that indirect lighting will tend to draw from samples that are close to the pixel in question. Looking at two pixels, if the depth values differ significantly, then the sampling correction understates the difference in world space to a large enough extent that inaccuracies in the rendering could occur.

In any case, using screen space as a proxy for world space simplifies things greatly. And while Dachsbacher goes into detail on a sampling technique tailored to our method, I simply went ahead and used Poisson sampling in a manner that enforced temporal coherency, similar to how I used it to determine percentage-closer soft shadows. The point of the offset correction (see below) determined by the sample is to recalibrate decreasing sample densities further out from the given pixel so that sample contributions are at least somewhat uniform. Dachsbacher describes this as "importance sampling."

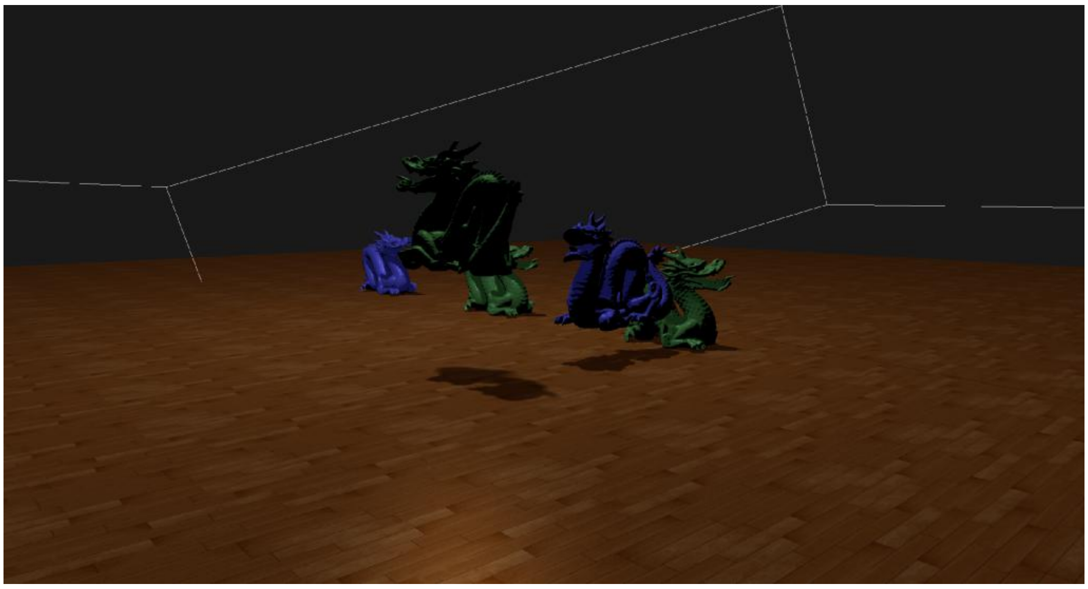
Without further ado, here's my shader function detailing the complete determination of indirect lighting on a pixel:

```
vec3 calcIndirectLighting()
{
    vec3 indirectLighting = vec3(0.0f);
    for (int i = 0; i < NUM_SAMPLES; ++i)
    {
        vec2 offset = vec2(
            ROTATION.x * POISSON_DISK[i].x - ROTATION.y * POISSON_DISK[i].y
            ROTATION.y * POISSON_DISK[i].x + ROTATION.x * POISSON_DISK[i].y
        vec2 texCoords = LIGHT_SPACE_POS_POST_W.xy + offset * TEXEL_SIZE *
        vec3 vPLWorldPos = texture(gPositionMap, texCoords).xyz;
        vec3 vPLWorldNormal = texture(gNormalMap, texCoords).xyz;
        vec3 flux = texture(gFluxMap, texCoords).xyz;
        vec3 lightContrib = flux * max(0.0f, dot(vPLWorldNormal, fs_in.Worl
        lightContrib *= offset.x * offset.x;
        indirectLighting += lightContrib;
    }
    return clamp(indirectLighting, 0.0f, 1.0f);
}
```

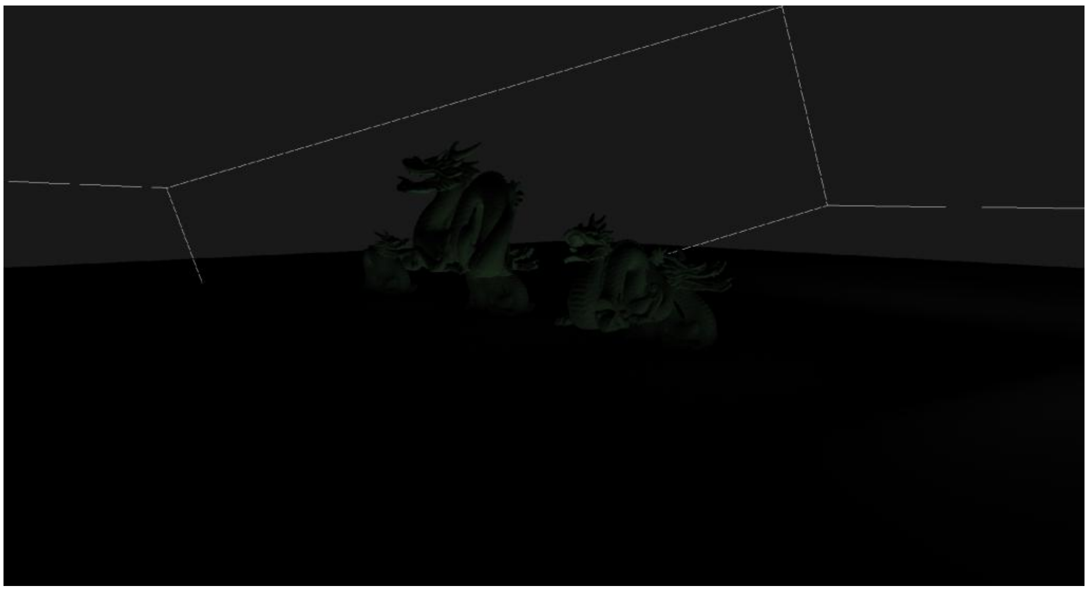
This is simply added to the normal direct lighting calculations. I find that a tweakable kernel size of 143.36 works well with my setup, but you're going to want to experiment with a few numbers to see what works best for you.



RSM ON



RSM OFF



The diff

Generally speaking, I don't recommend a cache-inefficient sample-based implementation of global illumination. While percentage-closer soft shadows and filtering emphasize similar concepts, they at least offer sizable quality improvements to any given scene—enough to really consider the trade-off of decreases in performance. Dachsbacher does at least offer a solution to address the performance issues, but developers are increasingly turning to techniques like voxel cone tracing.

Resource

Carsten Dachsbacher's Reflective Shadow Maps (http://www.klayge.org/material/3_12/GI/rsm.pdf) Paper

Tagged:
Global Illumination,
GPU,
Real-Time Rendering,
Sampling,
Shadow Mapping