Reflective Shadow Maps

Real-Time High-Quality Rendering · Final Project Report

Luis Ángel Larios Cárdenas · *6612279*

Figure 1: The Reflective Shadow Map. From left to right: depth map, world-space positions, world-space normals, flux map, and resulting image rendered with indirect lighting, Percentage-Closer Soft Shadows, and Screen-Space Ambient Occlusion enabled.

Abstract

In this project we have implemented *Reflective Shadow Maps* together with *Screen-Space Ambient Occlusion* and *Percentage-Closer Soft Shadows*. Our approach hinges on the presence of a single directional light for which we create a reflective shadow map, or RSM, a multi-targeted frame buffer object that stores, for each fragment as seen from the light source point of view, the *world-space position*, the *world-space normal vector*, the *flux*, and the *depth*. Figure 1 shows the components of our typical RSM and the final result after the integration of other refinement and performance-enhancing techniques.

Reflective Shadow Maps

The goal of an RSM, as described by Carsten Dachsbacher and Marc Stamminger in [1], is to approximate indirect lighting by sampling neighboring fragments of a target pixel. For this, we require information like the world-space position, normal, and flux of those candidate fragments so that we can determine if they contribute to lighting the pixel currently under consideration, and by how much. If we let $\mathbf{x}$ and $\mathbf{n}$ be the world-space position and normal of a pixel, and suppose we want to compute the irradiance at $\mathbf{x}$ due to a fragment p at position $\mathbf{x}_p$, then [1]:

$$\mathbf{E}_p(\mathbf{x}, \mathbf{n}) = \boldsymbol{\phi}_p \, \frac{\max\left(0, \mathbf{n}_p^T(\mathbf{x} - \mathbf{x}_p)\right) \max\left(0, \mathbf{n}^T(\mathbf{x}_p - \mathbf{x})\right)}{\|\mathbf{x} - \mathbf{x}_p\|^4}$$

where $\boldsymbol{\phi}_p$ is the RGB-vector flux at pixel p, defined for a directional light as the element-wise product of the light color and the material albedo of the model in the scene. In figure 1 you may realize that in this case the flux is basically a flat-shaded image of the 3D objects. The total indirect, one-bounce irradiance at a pixel $\mathbf{x}$ is then given by

$$\mathbf{E}(\mathbf{x}, \mathbf{n}) = \sum_p \mathbf{E}_p(\mathbf{x}, \mathbf{n}) \tag{1}$$

where p is, theoretically, any pixel in *light view* –that is, any fragment receiving energy directly from the light source. In practice, indirect lighting is computed in light projective space, where $\mathbf{x}$ (which is outputted to screen space, as seen from the viewer's perspective) is projected into the shadow map to $\mathbf{s} = (u, v)$; also, as mentioned above, it is necessary to resort to sampling since considering all pixels in the calculation is an extremely expensive task.

We have approximated the theoretical indirect illumination in equation (1) by sampling 151 random, but evenly distributed shadow map pixels around $\mathbf{s}$ via Poisson disks [2]. That is, given a scene-dependent maximum indirect light radius r_{max}, we select the pixel light at position

$$\mathbf{s}_p = \mathbf{s} + r_{max}\xi_1^2\boldsymbol{\xi} \tag{2}$$

where $\boldsymbol{\xi} = (\xi_1, \xi_2)$ is one of the points obtained with Poisson disk sampling (see figure 2). The scaling factor of ξ_1^2 in equation (2) provides a way to assign less contribution to pixel lights coming from the immediate vicinity of $\mathbf{s}$, thus avoiding undesirable "burning" artifacts. Additionally, one can modulate the intensity of the indirect light $\mathbf{E}(\mathbf{x}, \mathbf{n})$ by scaling this result with an appropriate (also scene dependent) constant.

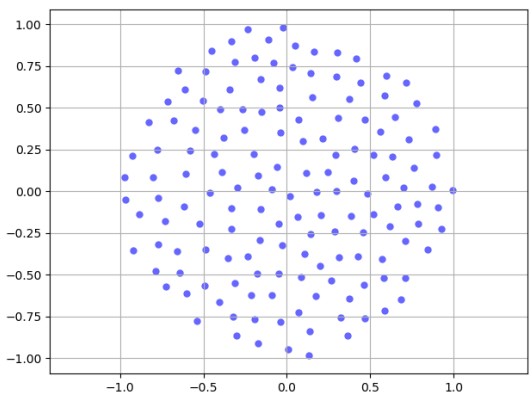

Figure 2: Poisson disk samples within the unit circle.

Finally, the computed color of a pixel (in our case) is the total contribution of the *Screen-Space Ambient Occlusion* proportional ambient light and the *Percentage-Closer Soft Shadow* factor times the sum of the direct light (due to the *Blinn-Phong Reflectance Model*) and the indirect light due to the RSM.

Implementation and Results

We have implemented RSM together with *Percentage-Closer Soft Shadows* (PCSS) [5] and *Screen-Space Ambient Occlusion* (SSAO) [4] to improve the realism of our scenes. Furthermore, we resorted to *Deferred Rendering* [3] to aid in performance and achieve interactive rates. Our development framework was OpenGL and GLSL 4.1 in a GLFW window system with a true frame buffer size of 1536 by 1536 pixels. In general we followed the next algorithm:

1. Set up a multi-targeted FBO for **RSM**, with textures for *positions*, *normals*, *flux*, and *depths*, which store information for fragments seen by the light source.

2. Set up a multi-targeted FBO for Deferred Rendering (i.e. a **G-Buffer**), with textures for *positions*, *normals*, *albedo and specular shininess*, *positions in projective light space*, and *depths*, which store information for fragments seen by the viewer.

3. Set up a one-targeted FBO for **SSAO**, with a texture to store a negated ambient occlusion factor which is computed in view space.

4. Set up a one-targeted FBO for blurring the negated occlusion factor stored in the SSAO texture.

5. Rendering loop:

   (a) *First pass*: Render scene into the RSM FBO.

   (b) *Second pass*: Render scene into the G-Buffer.

   (c) *Third pass*: Use the G-Buffer to generate the SSAO FBO texture by rendering to a quad in NDC space.

   (d) *Fourth pass*: Blur the SSAO texture by rendering to a quad in NDC.

   (e) *Lighting pass*: Use the RSM, G-Buffer, and blurred SSAO FBOs to compute the final color of the fragments in a quad in NDC.

We should note that for SSAO we employed 48 uniformly-distributed samples (i.e. kernel size) in the *normal hemisphere* of fragments as seen by the viewer. Depending on the scene, we also tuned the radius of the the normal hemisphere and the output intensity of the negated occlusion factor. We found that this kernel size, together with the 151 Poisson disk samples in the RSM and the other 33 Poisson disk samples used for PCSS yielded acceptable results at interactive rates. Figure 3 shows a comparison of "Mercury" in a first plane with and without RSM and SSAO enabled. You may see how the Mexican-pink and blue colored walls are reflected on the gray diffuse material of the 3D model in the right image of that figure. In particular, the "Mercury" scene consisted of 147,024 triangles, achieving 18 FPS with both RSM and SSAO enabled, and 21 FPS when we opted for disabling SSAO.

Another more complex scene is shown in figure 4. This "space" scene consisted of 229,202 triangles, achieving 19 FPS when both RSM and SSAO were enabled, and 22 FPS when we opted for disabling SSAO. As you may note in figure 4, some of the pink color from the right arch is reflected onto the upper body of the olympian statue in the center stage, and the floor texture color bleeds onto the vases and the lower body of the statue.

We should point out that the main contribution of SSAO in our implementation was adding in some more subtleties to our scenes. Particularly, with SSAO the creases and joints in the 3D models became more apparent and gave a sense of more depth, which was not solely attained by using PCSS. You may note this effect in the horizontal ornaments of the colored columns of figure 4. Also, SSAO allowed us to better approximate environment light "in the darkness" of our 3D models. Figure 5 shows a comparison of using and not SSAO in the "space" scene. In this case the camera is facing towards

Figure 3: Comparing "Mercury" without RSM or SSAO (left), and with RSM and SSAO enabled (right).

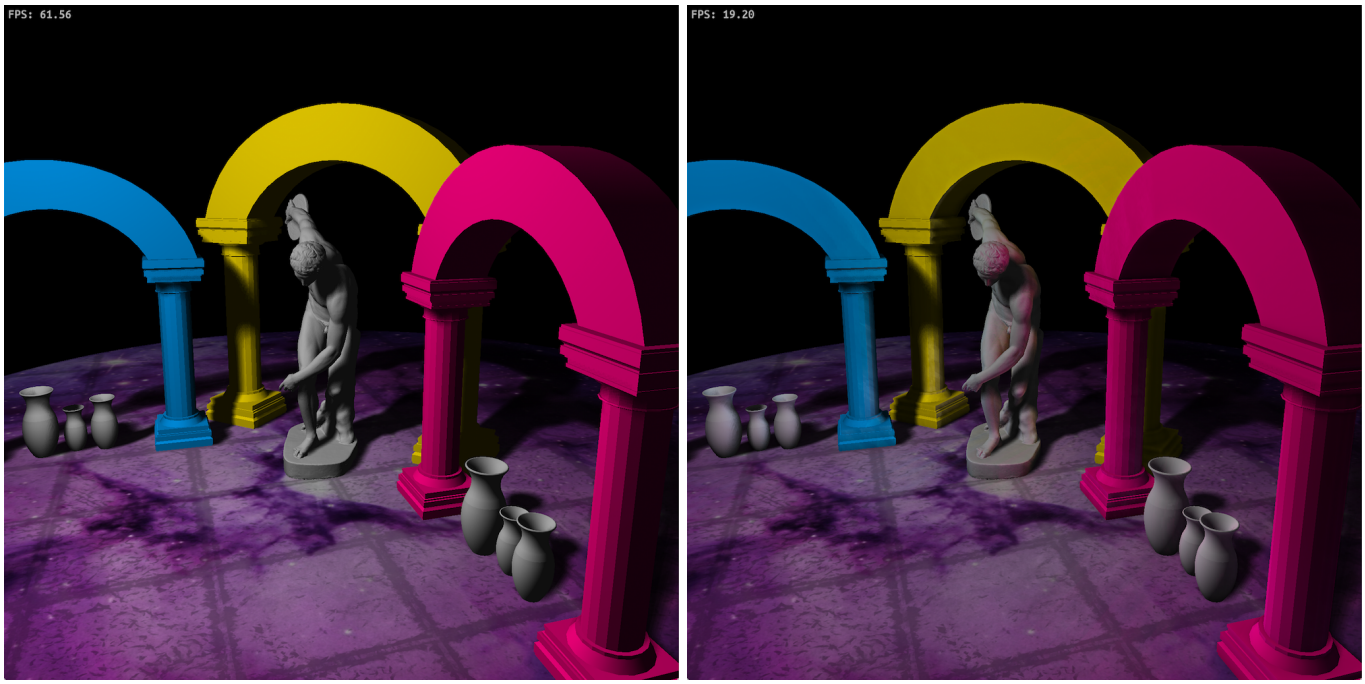

Figure 4: Comparing the "space" scene without RSM or SSAO (left), and with RSM and SSAO enabled (right).

one region of the scene that is in shadows, and, despite that, when SSAO is enabled, we can still see the joints of the horizontal ornaments at the base and top of the yellow arch.

We encourage you to check out these results in the accompanying **YouTube video**.

Learning Experience

In this project I learned how to allocate, fill, and use OpenGL multi-targeted *frame buffer objects* (FBOs) and their textures to implement deferred rendering, screen-space ambient occlusion, and reflective shadow maps. With deferred rendering I was able to shade high-polygon-count geometries that would have been impos-

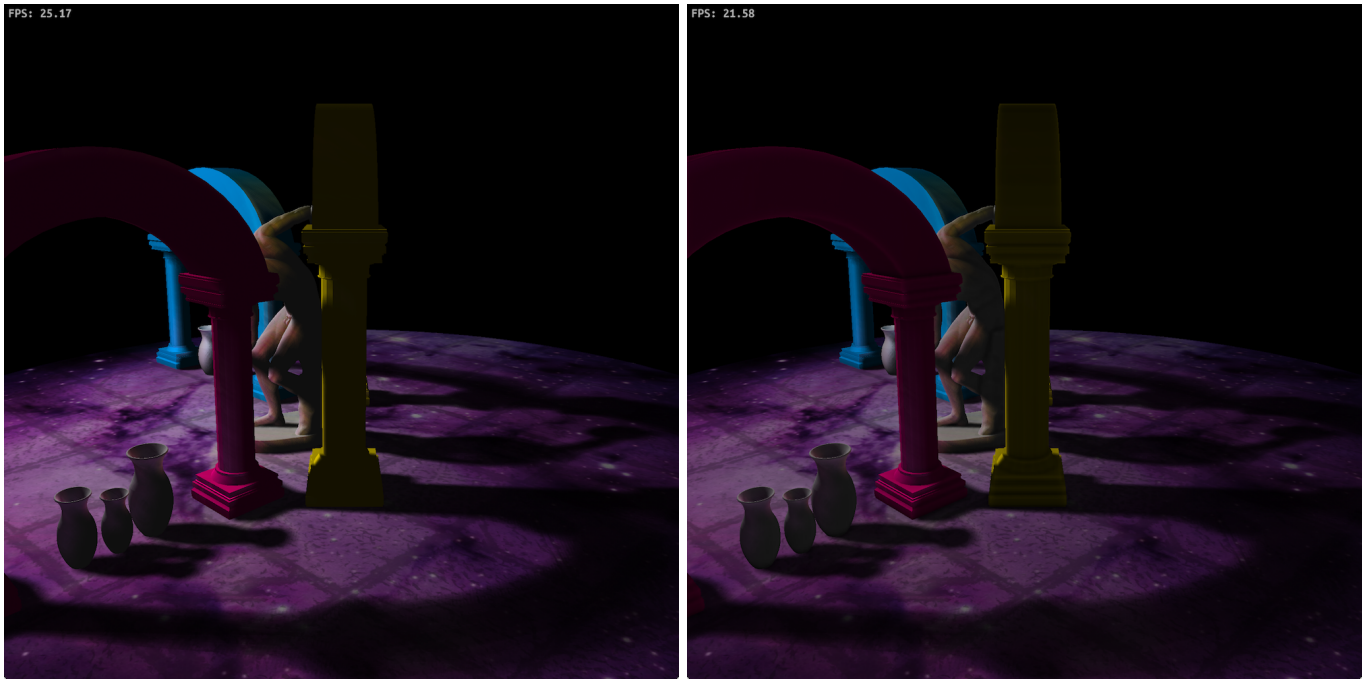

Figure 5: Comparing the effect of SSAO in the dark regions of the "space" scene: without SSAO (left), and with SSAO enabled (right).

sible to deal with at interactive rates during the lighting phase with RSM and SSAO enabled. According to [3] and [4], the G-Buffer is the basis for these screen-space techniques and to optimize the execution of the otherwise costly operations in the fragment shaders. On this regard, understanding the G-buffer was fundamental in the implementation and outcomes of the current project.

Moreover, I reiterated the necessity for tuning parameters in order to achieve realistic results as reported in the original paper(s) and tutorials. This parameterization was scene dependent, and it also heavily relied on the device I used to program the application. In this particular implementation I worked on a macOS system with retina display (AMD Radeon Pro 555X with 4GB VRAM), and the resolution of the outputted GLFW window contained twice the number of pixels than a normal screen (tested on a late 2013 iMac). Because of this, an originally expected window of 768 by 768 pixels unfolded into 1536 by 1536 pixels, which was a large area to fill in. Thanks to deferred rendering, I kept such a large screen quad and yet achieved between 18 and 22 FPS for the results presented in the previous sections. These findings are also important if one considers that for each pixel we visited 48 samples for SSAO, 16 samples for blurring the negated SSAO occlusion factor, 151 samples for RSM, and 33 samples for PCSS. Regardless, for future work, I would like to learn and investigate more about OpenGL optimization techniques, "early culling", and the *simple* and *easy* bilinear interpolation between FBO textures that the authors in [1] employed to speed up indirect light computations for scenes with flat surfaces. Even when we did not follow their approach, we still reduced the amount of calculations for RSM by testing fragments' depth in the lighting pass in order to discard those not associated to any geometry (i.e. their depth value from the G-buffer was 1.0) –of course this worked better for the "space" scene, but not for the closed-up scene illustrated in figure 3.

References and Tutorials

[1] Carsten Dachsbacher, and Marc Stamminger. *Reflective Shadow Maps*. Proceedings of the 2005 Symposium on Interactive 2D Graphics and Games, April 03 - 06, 2005. Washington, D.C. 2005.

[2] Christian at ScyPython. *Poisson disc sampling in Python*. ScyPython Blog. April 22, 2017. https://scipython.com/blog/poisson-disc-sampling-in-python/

[3] Joey De Vries. *Deferred Shading*. Learn OpenGL Tutorials. https://learnopengl.com/Advanced-Lighting/Deferred-Shading

[4] Joey De Vries. *SSAO*. Learn OpenGL Tutorials. https://learnopengl.com/Advanced-Lighting/SSAO

[5] Kevin Myers, Randima (Randy) Fernando, and Louis Bavoil. *Integrating Realistic Soft Shadows into Your Game Engine*. NVIDIA Whitepaper. http://developer.download.nvidia.com/whitepapers/2008/PCSS_Integration.pdf