INF584 – Project
Implementing Screen Space Reflections on the GPU

João Baptista de Paula e Silva

1st April, 2020

1 Introduction

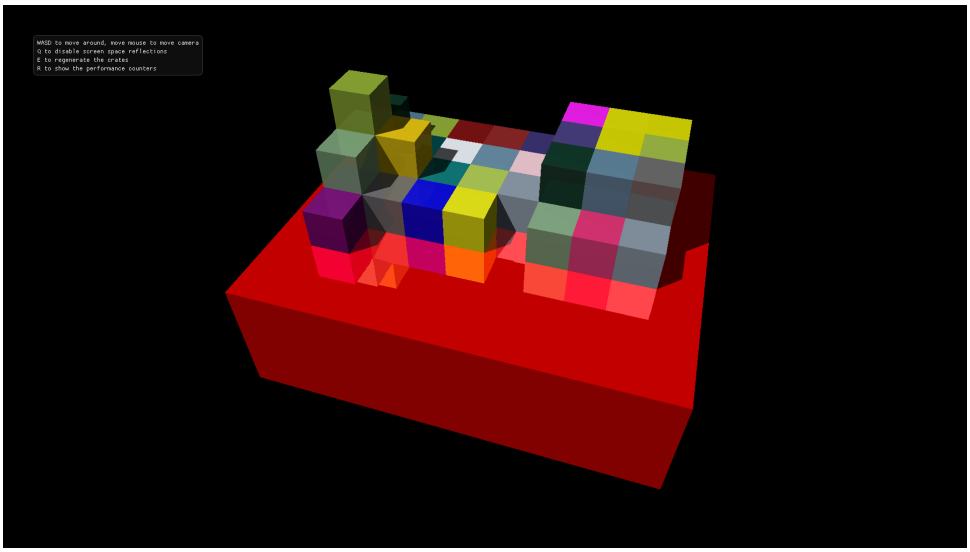


Figure 1: an example of scene rendered with screen space reflections

In photo-realistic graphics rendering, one can say that "the devil is in the details". This is especially true when dealing with indirect and global illumination, of which reflections are a quite important part of. The rendering of reflections is quintessential for the perceived realism of a scene, and this is often sought for the industries that crave for it. Reflections can add a greater visual detail to some materials such as metal and polished stone.

However, correctly computing the reflected color for a given fragment is always a costly operation. Before the arrival of ray tracing hardware in consumer graphics cards, most techniques relied on environment mapping, approximations, or implemented the ray tracing steps using compute shaders, or even required the entire scene to be rendered again – this could yield perfect reflections, but only work for planar reflections.

The technique presented here, *Screen Space Reflections*[1], uses the already-processed rendering data of a frame to search for reflection colors among pixels in the screen. It can provide a very good visual quality, but has some artifacts and fail cases. Since it requires data on screen space, it is often easier to implement on a deferred shading pipeline, but it can also be implemented on forward shading pipelines as a post-processing effect.

2 Technique

The screen-space reflection technique, to work, needs two buffers: a normal buffer and a depth buffer. The normal buffer will have information on the view-space normal coordinate on a specific fragment and the depth buffer will contain information to reconstruct the view-space position on that fragment.

With that information, a reflection ray can be computed by reflecting the normalized view-space position along the normal (the view-space position of the camera, is by definition, zero). The screen is then ray-marched along that ray in screen space, to search for a fragment that would correspond to that position. For this, the depth-buffer is sampled many times and compared with the computed depth of the ray at that fragment. The coordinates of this fragment, if found, are stored and used later on in a final resolution phase.

This texture coordinate data can be combined in many ways, the main use of it is to serve as a "new" light contribution in physically-based rendering, but it can also be combined in other ways, and even sampled in a Monte-Carlo like method to produce reflection data for rough surfaces.

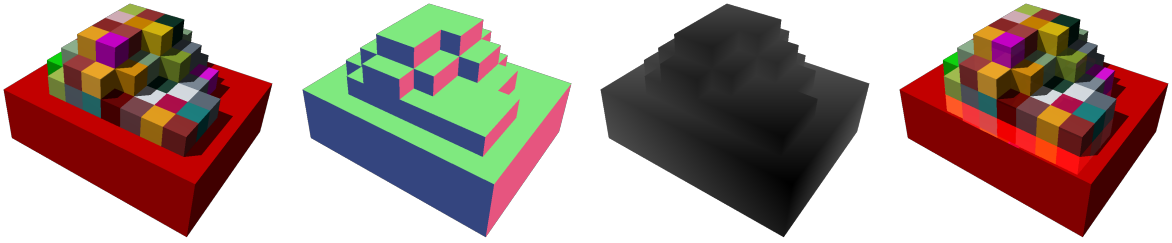


Figure 2: a representation of the components of the reflection technique, from left to right: (a) a illuminated and prepared color buffer, (b) the normal buffer (here storing only two components of the normal) and (c) the depth buffer. The final result is showed in (d).

The effect of the technique can be, at times, very subtle or very evident, as both cases can be seen on the figure above. Besides the blatant reflections on the floor, there are also subtler reflections on the cubes. The reflections could be further perturbed if the cubes used normal maps themselves, in order to perturb the normal map per fragment.

3 Implementation

My small rendering engine consisted five important passes: G-Buffer construction, shadow map creation, illumination, SSR computation and final compositing. Since SSR is a post-processing effect, it lends itself naturally to a deferred rendering pipeline, which is what I use here.

3.1 G-Buffer Construction

Here, four buffers are generated: the color (albedo) buffer, the depth buffer, the normal buffer and a specular/shininess (Blinn–Phong) buffer.

3.2 Shadow Map Creation

A traditional shadow mapping technique is applied, so the shadow map is generated for the single directional map present in the scene. It is interesting to observe that the shadow map is rendered from another perspective (and with an ortographic projection), since it is rendered from the point of view of the light.

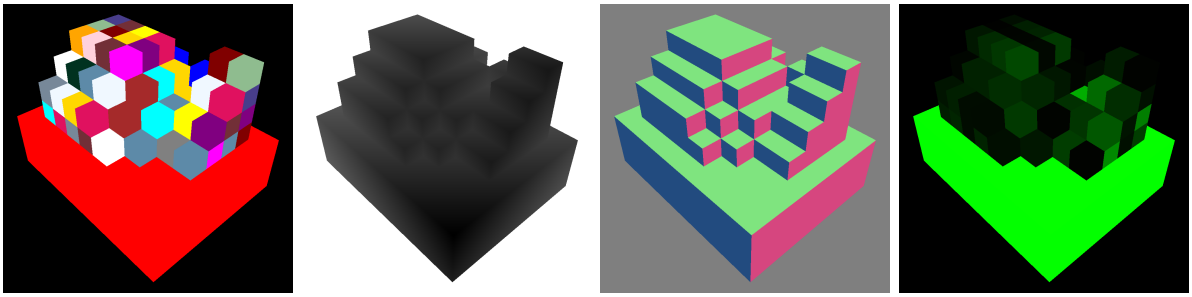


Figure 3: the buffers generated by the construction pass: a color buffer, a depth buffer, a normal buffer and a specular/shininess buffer.

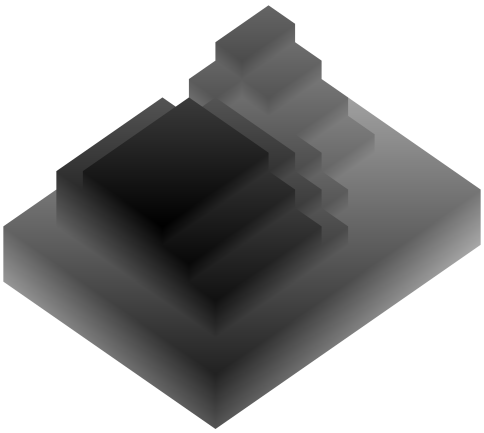


Figure 4: the shadow map generated for a sample scene.

3.3 Illumination

The data from the G-buffer and the shadow map are then combined to calculate the final illuminated pixels, which will then be used on the final compositing pass.

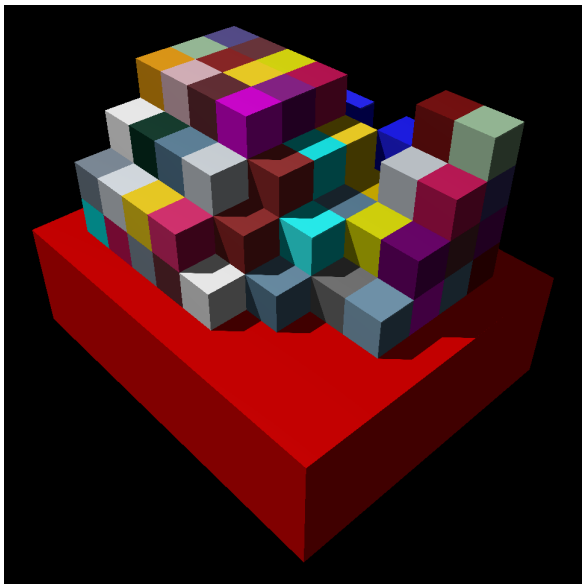


Figure 5: the illuminated pixels, used on the final composite pass alongside the buffers generated by the Screen Space Reflections.

3.4 Screen Space Reflections

This is the most relevant part of the scene. As explained, the normal buffer and the depth buffer are used in order to reconstruct the position, $\vec{p}$, of the fragment and the normal, $\hat{n}$ pointing out of it in view-space. We assume here that $\hat{n}$ is normalized. The reflected ray can then be created using the formula:

$$\hat{r} = \hat{I} - 2\left(\hat{n} \cdot \hat{I}\right)\hat{n}, \quad \text{where } I = \frac{\vec{p}}{\|\vec{p}\|}$$

The incident ray is the ray going from the view-space origin to the position vector. After that, given the projection matrix P, we can construct the screen-space homogeneous vectors:

$$\vec{p_s} = P\begin{bmatrix}\vec{p}\\1\end{bmatrix} \quad \text{and} \quad \vec{r_s} = P\begin{bmatrix}\hat{r}\\0\end{bmatrix}$$

Those homogeneous vectors can be used for ray marching, since the ray in screen space will be represented by $\vec{q_s} = \vec{p_s} + t\vec{d_s}$.

Now, the trick here is that it is not trivial to march in homogeneous coordinates such that our step consists exactly of a fragment (or of a number of fragments) in a direction. The system of equations we need to solve is:

$$\begin{cases} \left|\frac{x_{p_s}+tx_{d_s}}{w_{p_s}+tw_{d_s}} - \frac{x_{p_s}}{w_{p_s}}\right| \leqslant \frac{2}{W} \\ \left|\frac{y_{p_s}+ty_{d_s}}{w_{p_s}+tw_{d_s}} - \frac{y_{p_s}}{w_{p_s}}\right| \leqslant \frac{2}{H} \end{cases}$$

Where W and H are the width and height of the buffer, respectively. The 2 factor here is due to the fact that the screen space coordinates range from -1 to 1. Solving this system, the largest positive t that satisfies this equation is:

$$t = \frac{w_{p_s}}{M - w_{d_s}}, \quad \text{where } M = \max\left\{\frac{W}{2}\left|x_{d_s} - w_{d_s}\frac{x_{p_s}}{w_{p_s}}\right|, \frac{H}{2}\left|y_{d_s} - w_{d_s}\frac{y_{p_s}}{w_{p_s}}\right|\right\}$$

We can scale t to "march" approximately a number greater than 1 of pixels in the buffer. This can be used as a coarse step to find a fragment whose depth is greater than the depth of the marching ray, which means there is an intersection in that interval.

The exact pixel where the intersection happens can then be computed using a simple binary search. The coordinates of this fragment are stored in a buffer, alongside a visibility factor that is computed using the fragment's specular and shininess values.

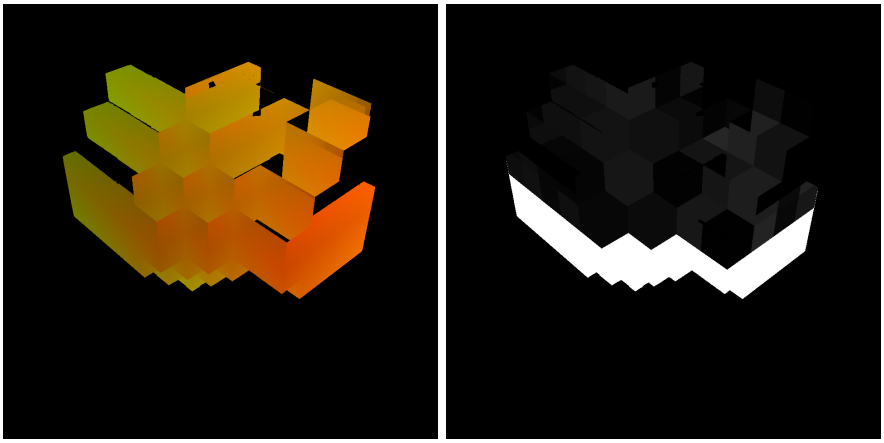


Figure 6: the buffers constructed by the SSR pass.

3.5 Final Compositing

Finally, with the SSR texture coordinate map and the visibility map generated, this pass combines the original illumination buffer and the fragments fetched on it based on the texture coordinate map. After that, the user interface is drawn on top of it.

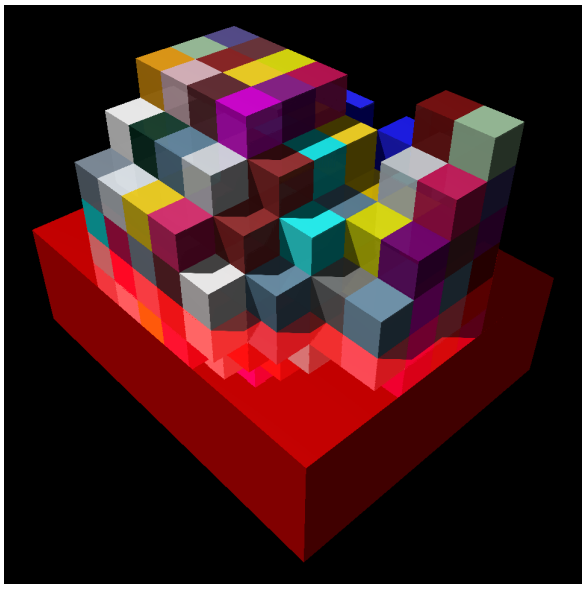
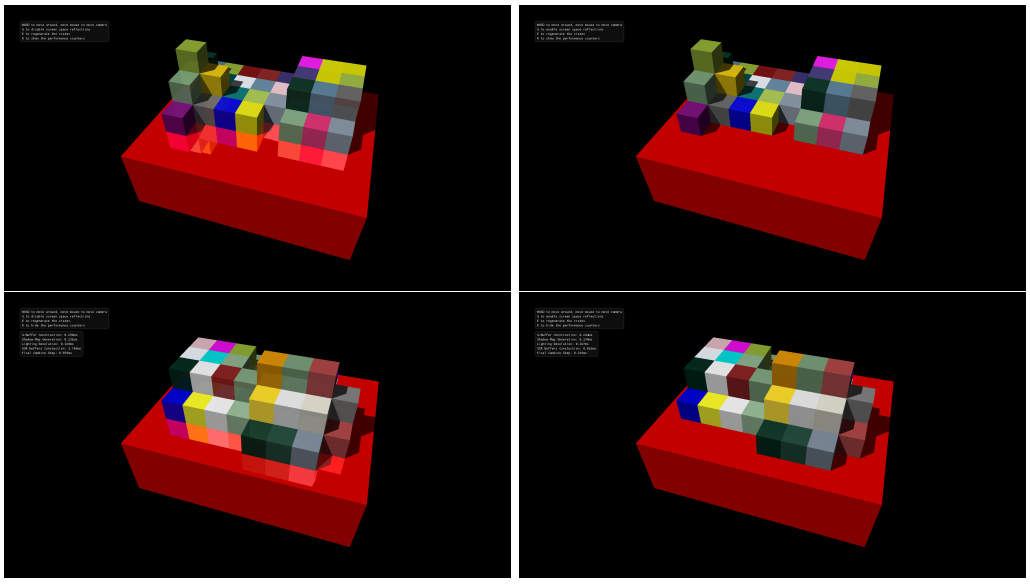
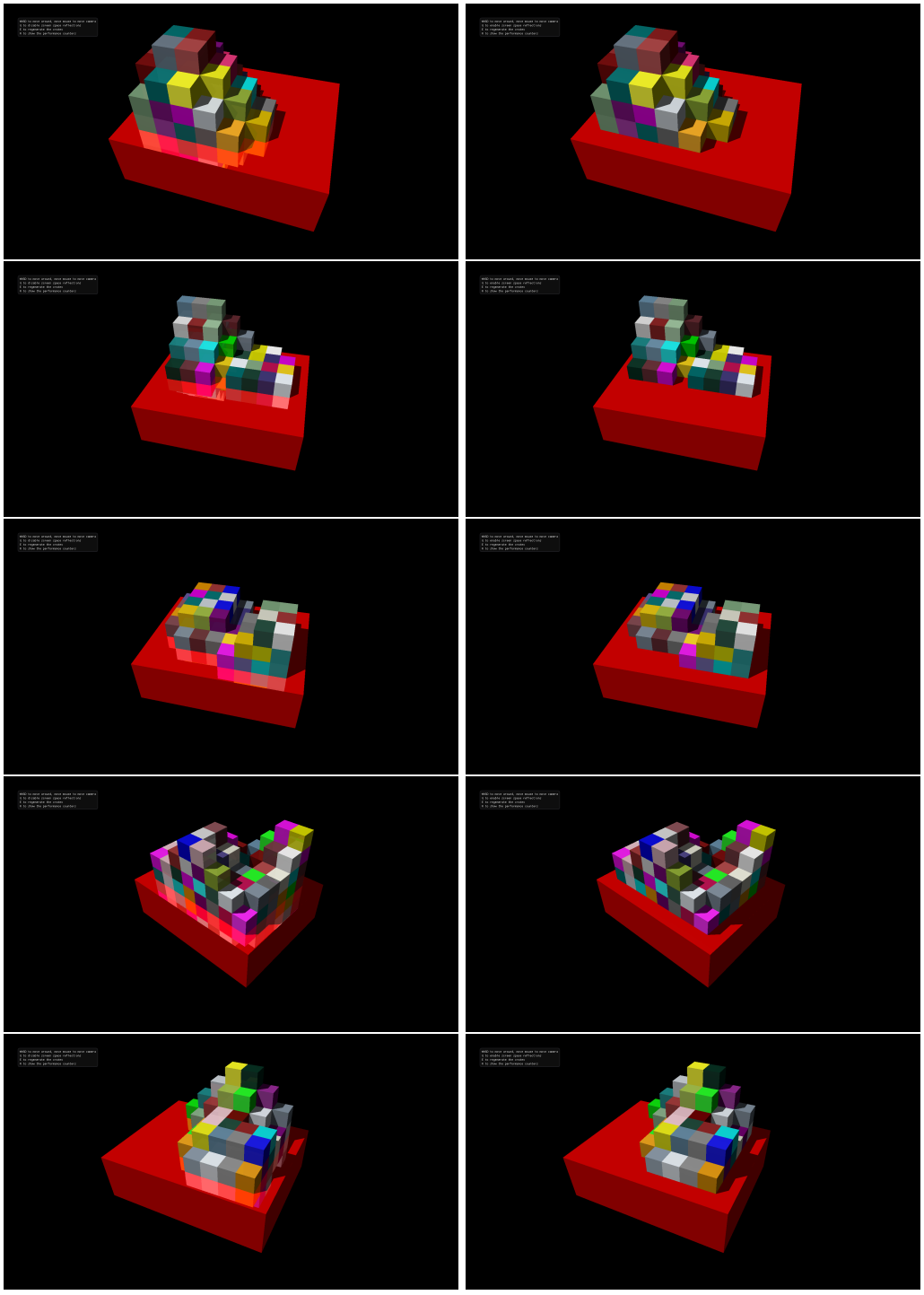


Figure 7: the illuminated pixels, as well as the reflection data derived from the texture coordinate and visibility buffers from the last pass.

4 Results

The following figures demonstrate the SSR technique on a variety of scenes, with comparisons to the scene rendered with it off. It is interesting to observe that the reflections are not just present on the floor, but also on small portions of the walls and floors on the cubes, a task which would require plenty of re-renderings of the scene in order to achieve with traditional planar reflections. The latter would be impossible to achieve on curved and/or rounded surfaces without proper ray tracing, but this technique provides a good approximation.





Figures 1, 8-20: some examples of scenes generated and rendered with and without the SSR technique.

As for performance, the technique was tested on the author's personal computer, which has a nVidia GeForce GTX 1050 graphics card. As it can be seen, the SSR pass is the most expensive of them all, taking nearly 4 times the time of each one of the passes, and contributing to 50% of the rendering time of the scene. Still, it provides a really good reflection quality while not being that heavy on the performance.

5 Limitations

This technique, using the illumination buffer itself to compute the reflection color, will only be able to derive the reflected surface of rays that are actually present on the screen and facing towards it. If a reflection ray ends up hitting a surface oriented behind the view or goes away from the screen, no information can be derived, which can cause visual artifacts when rendering the screen.

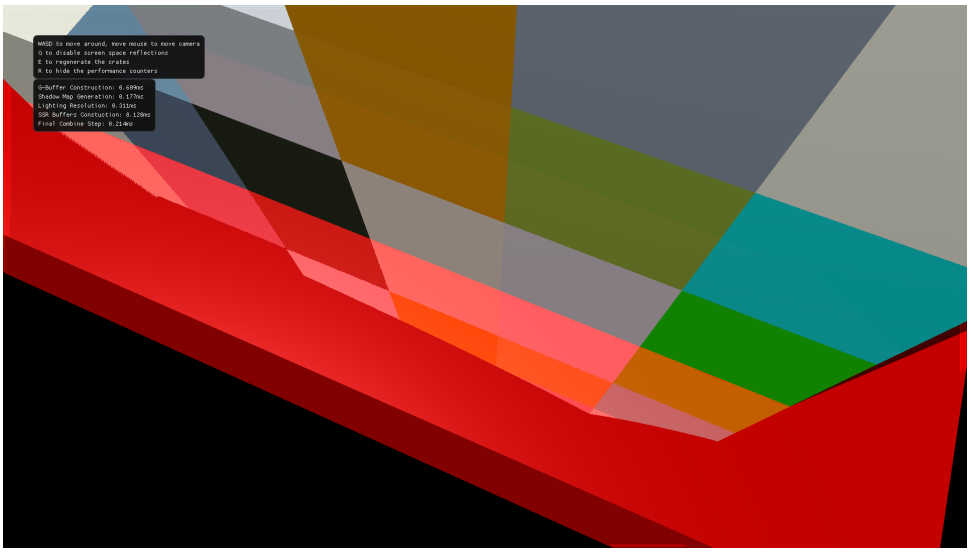


Figure 21: the reflection of the upper cubes on the floor is cut off short because there is not enough information on the frame buffer to render it entirely. Also, with more rays missing, the pass takes much more time to finish.

The technique is, though, a primer on actual ray tracing, which can be prohibitively expensive to achieve on graphics cards without a dedicated hardware for it. It can provide good visual quality for the places it is applicable.

6 Conclusion

Screen Space Reflections is a very powerful and efficient technique to enable the rendering of surface reflections on the screen, which can increase a lot its perceived realism. It works by using the framebuffer's already computed data in order to retrieve the colors for the reflections. However, since it works on screen space, it cannot derive the color of off-screen surfaces or surfaces that point away from the camera, and is best combined with other reflection techniques, such as reflection probes.

References

[1] *"Screen Space Reflections"*, `http://www.cse.chalmers.se/edu/year/2017/course/TDA361/Advanced%20Computer%20Graphics/Screen-space%20reflections.pdf`.