Real-time Rendering of Infinity Mirrors

Zhou Rui

November 26, 2022

1 Introduction

A set of infinity mirrors (Figure 1) consists of several mirrors facing towards each other in a parallel alignment. Each of the mirrors shows a sequence of smaller and smaller reflected images that seemingly extends to infinity. This illusion is caused by light beams bouncing back and forth for an infinite number of times between the reflective surfaces.

Due to the nature of this phenomenon, ray tracing is a natural option to render a scene with infinity mirrors. However, while ray tracing can achieve lifelike rendering results, its rendering efficiency is largely constrained by the computational resources of a device. Therefore, it is not a good approach when we aim to achieve real-time rendering on a wide range of devices with varying hardware configuration.

This project proposes a real-time rendering approach for infinity mirrors from a rasterization perspective. It mitigates the artifacts of rendering infinity mirrors in rasterization by increasing the number of reflections between the two mirrors. Furthermore, by controlling the degree of light attenuation in each time of reflections, the effect produced by a small number of reflections is already comparable to the truth. Finally, there is a balance between the rendering effects and the rendering efficiency. The demonstration is available here.

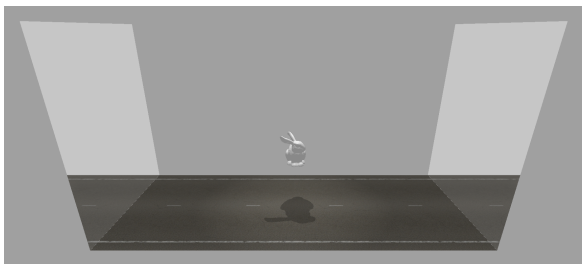

Figure 1: The physical arrangement of infinity mirrors.

2 Problem Definition

There are two mirrors (Mirror 1 and Mirror 2) placed in front of each other with an object in between (Figure 1). The objective is to produce authentic visual effects of infinite reflections when we look into a mirror. However, as Figure 2 shows, a rasterization approach can only render two reflections in a mirror. This artifact is due to the recursive rendering pattern of using environment mapping to handle mirror reflections (Algorithm 1).

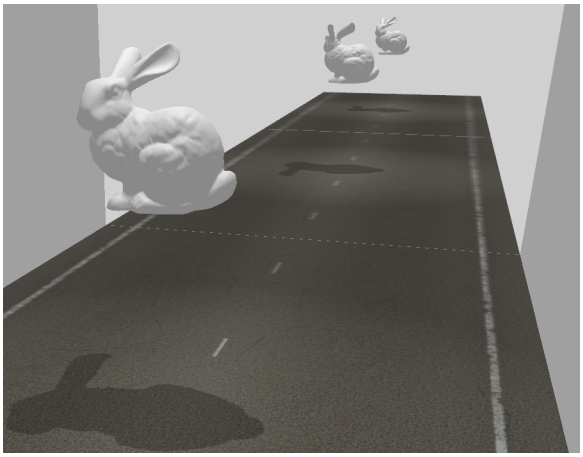

Figure 2: Naive environment mapping does not produce infinity mirrors.

Algorithm 1 Mirror Reflection

```
procedure MirrorTexture(mirror M)
    Set a camera C at M with the same orientation.
    Denote {M_i} as the set of mirrors passing z-test in the perspective of C.
    Denote P as the fragment shader output of C.
    if {M_i} ≠ ∅ then
        M.visible = False
        for m ∈ {M_i} do
            T ← MirrorTexture(m)
            apply T to P
        end for
    end if
    return P
end procedure
```

In the scenario of Figure 1, since the texture of Mirror 1 is given by an auxiliary camera looking out of it, the camera captures Mirror 2 of unknown texture, then it proceeds to find the texture of Mirror 2 when **without** Mirror 1. Therefore, light beams only bounce between the two mirrors twice

and there is no reflection receding into infinity in the mirrors (Figure 2). Neglecting Mirror 1 in the perspective of Mirror 2 breaks down the infinite loop of reflections. However, we cannot use Mirror 1 to texture Mirror 2. Otherwise, the original rendering target (the texture of Mirror 1) becomes the source texture for Mirror 2, which results in infinite recursion.

3 Methodology

This section presents the two techniques that help to forge infinity mirrors in real-time rendering.

3.1 Auxiliary reflections

As mentioned in Section 2, the failure to render infinity mirrors lies in the broken reflection loop. Then, increasing the number of reflections between the two mirrors is at the heart of tackling the root cause. According to the recursive flow in Algorithm 1, each mirror only adds to one reflection. Therefore, some imaginary mirrors should be added to increase the total number of reflections.

To add n additional reflections, we need to initialize n auxiliary mirrors and label them as mirrors $\{1, 2, \ldots, n\}$. The mirrors are then divided into two groups by their labels' parity. The two groups will overlap with Mirror 1 and Mirror 2, respectively. Or equivalently, for each additional reflection, we initialize an auxiliary mirror at the other side opposite to the previous one. In this way, the scene in Mirror 1 or 2 is the result of reflections through a rendering queue of $n + 2$ mirrors.

Figure 3 shows the effect after having 4 more reflections (cf. Figure 2). By having a small number of auxiliary reflections, it already approximates infinity mirrors quite well.

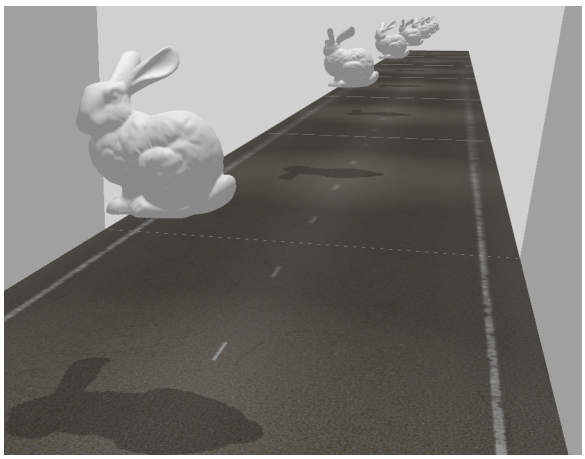

Figure 3: $n = 4$ additional reflections.

Even though increasing the number of auxiliary reflections is effective, it is also computationally expensive. As Table 1 shows, the processing time increases exponentially as the number of auxiliary reflections increments. Therefore, a large number of reflections should be avoided. Because as the number increments, there is a larger number of shrinking images stacked at the end of the sequence, providing diminishing improvement while requesting exponentially heavier computation.

No.	0	1	2	3	4
ms/frame	25	61	142	435	1330

Table 1: Relationship between number of auxiliary reflections and processing time per frame, tested on Safari.

3.2 Light attenuation

Following the idea in Subsection 3.1, we further introduce light attenuation to the reflections. For the reflected images in the mirror, their light intensity decays as the light travel further away. The light intensity of the n-th reflection is proportional to α^n, where $\alpha \in [0, 1]$ is a user-specified parameter. By doing this, with even less reflections, we can still give a sense of infinitely receding reflections (Figure 4).

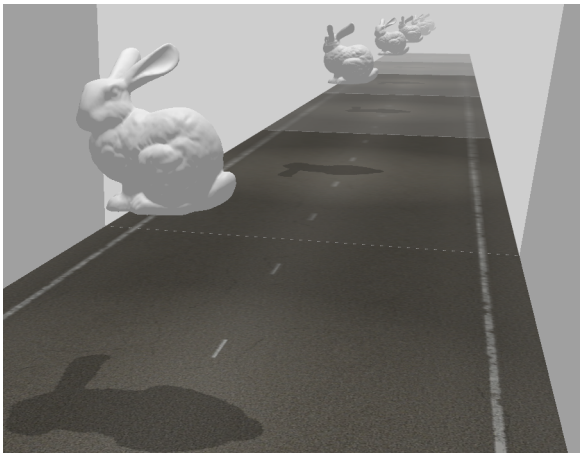

Figure 4: $n = 3$ additional reflections with $\alpha = 0.7$.

4 Conclusion

In this report, we proposed a way to render infinity mirrors in real time. Adopting a rasterization perspective, we started from analysing the failure of rendering infinity mirrors with naive environment mapping. Then, it leads to the customized solution of deploying auxiliary reflections. In addition, light attenuation is introduced to approximate infinity mirrors with even less reflections. We observed that 3 or 4 additional reflections with a small degree of light attenuation already achieves lifelike visual effects, while the scene can be rendered in web browsers supporting WebGL with decent real-time frame rate.