

Real-Time Mirage Reflections Using Screen Space Ray Tracing

M. Tarik Tamyurek*
Electrical and Computer
Engineering Department
UC Santa Barbara



Figure 1: *Mirage reflections approximated using perspective-correct DDA line rasterization algorithm*

ABSTRACT

The mirage effect is an important visual aspect of scenes in hot weathers. It gives the sense of high temperature in a scene. However, it is generally ignored or forgotten in video games. To render a more photorealistic high temperature scene, this effect can be included in the rendering pipeline without causing too much performance loss. In this project, a mirage effect is approximated and implemented in real-time using screen space ray tracing techniques. A perspective-correct DDA line rasterization algorithm is used, which efficiently uses the GPU resources. After implementing the algorithm, the performance reduced from 115 FPS to only 97 FPS on a Nvidia GTX 850M graphics card.

Keywords: Screen Space Reflections, Global Illumination

1 INTRODUCTION

A mirage is an optical phenomenon in which the light refracts when traveling through layers of air with different temperatures and gives the illusion of displaced image of objects far away or the sky. This effect can be easily observed outside in hot weathers, where the displaced image of the sky becomes visible on the hot sands of the desert or the hot asphalt of a country road. This effect generally ends up by giving the illusion to people if there is a lake in the distance. This is due to the reflection of the sky color, which also gives the color we see when we look to actual lakes.

This effect generally takes place in images that are taken in these kind of high temperature scenes. However, in terms of video games and real time applications this phenomenon is generally ignored or forgotten. Though, this is such an important effect that it can actually give the sense of high temperature in a scene. This gives a total different look to hot summer scenes when compared to plain low temperature scenes.

Modelling and simulating a mirage can be pretty painful. The temperatures and the refractions of the layers of air as well as the movement of the air particles should be modelled. Also, ray tracing through this kind of scene would make it very hard to achieve real time in this application. Therefore, to achieve real time, this effect should be approximated.

Most of the time this effect occurs in far distances and the reflection that is produced is generally not a perfect mirror reflection. Also, the sources of the mirage reflections can generally be found inside the frame of the image. Therefore, mirage reflections can be achieved using some approximations and screen space reflection (SSR) techniques.

In this project, an efficient GPU screen-space ray tracing technique is used, which is based on the perspective-correct DDA line rasterization algorithm with some extended features. [McGuire et al. 2014]

Using render elements of a scene to apply ray tracing limits the complexity of scene. For example, the depth buffer gives a great approximation of the scene geometry. Depth buffers also allow ray tracing along fully dynamic scenes. Screen space ray tracing by iterating over a depth buffer can be achieved by using either linear 3D ray marching, or linear 2D iteration.

Linear 3D ray marching is a widely used method for SSR. The ray moves step by step in the world coordinates in the reflection direction and its position in every iteration is projected into screen space and compared with the depth of the corresponding pixel. If the ray goes behind the depth value, the ray is considered as hit at that point and the color values at the corresponding pixel is retrieved from the previously rendered frame.

*e-mail: tamyurek@ucsb.edu

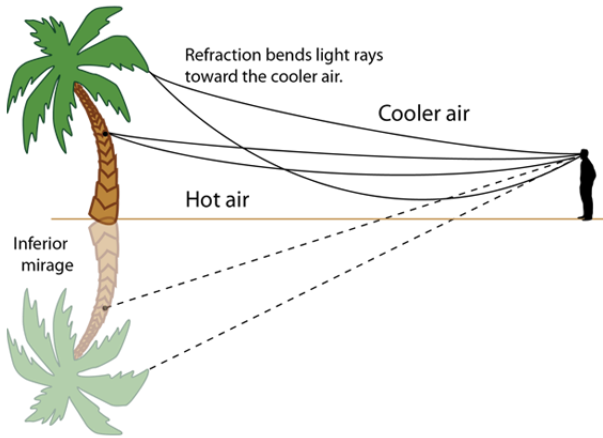


Figure 2: Illustration of the concept behind a mirage



Figure 3: Real photographs of an inferior mirage

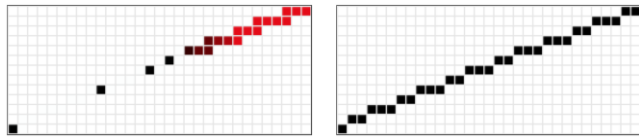


Figure 4: Linear 3D ray marching (left) misses some pixels while oversampling some of others. Linear 2D iteration (right) moves through each adjacent pixel by sampling for only once.

However, in 3D ray marching, some of the consecutive steps may project to the same pixel, which may end up by some of the pixels being sampled multiple times, while leaving some of the pixels missed. Therefore, linear 2D iteration is a more efficient way to trace ray in screen space. By using 2D ray marching, each pixel is sampled only once and in a consecutive order, and reduces GPU resource consumption. [McGuire et al. 2014]

2 IMPLEMENTATION

The project was implemented using OpenGL. First of all, the whole scene is rendered using the Blinn-Phong reflectance model. During the first geometry render pass, all necessary geometry buffers are also rendered and stored in the memory as textures. These G-Buffers are passed to the deferred shader, where the screen space ray tracing is handled.

The G-Buffers include the depth buffer, view-space position buffer, view-space normal buffer, roughness buffer and the color frame buffer, which are also shown in Figure 5.

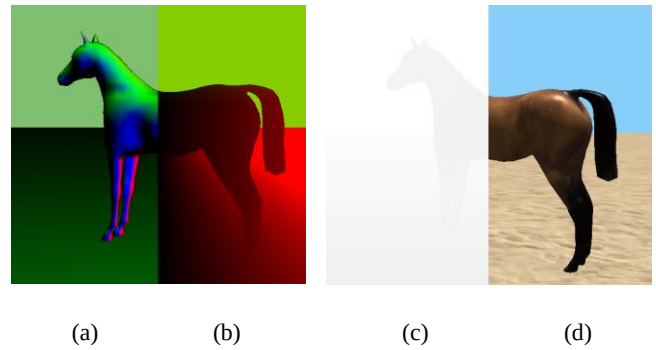


Figure 5: G-Buffers; (a) normal buffer, (b) position buffer, (c) depth buffer, (d) color buffer.

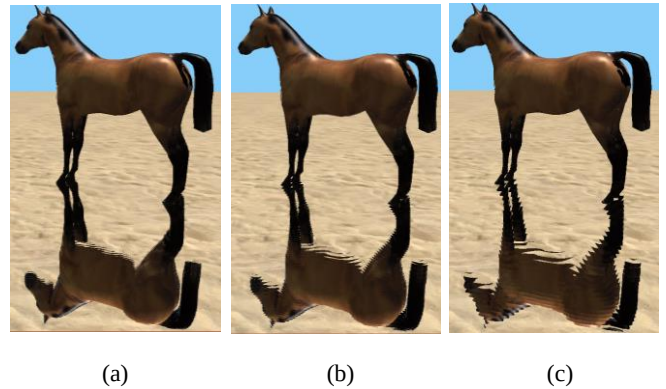


Figure 6: At 200 maximum iteration limit the FPS count on a Nvidia GTX 850M GPU are (a) 20 FPS at 5 pixel stride, (b) 36 FPS at 10 pixel stride, and (c) 49 at 20 pixel stride.

After the scene is rendered for the first pass and the necessary buffers are passed to the second shader, the view-space position is projected into screen space and points to a pixel. The ray is traced along the camera space direction until the end point. In the algorithm, the derivatives are constant over the x-axis. Therefore, the ray iterates over the line by going step by step on the x-axis. The iteration continues either until the ray hits a point, or maximum iteration number is reached. If a hit occurs, the algorithm returns the pixel coordinates of the hit point, and the color value in the frame buffer is retrieved from that hitpoint. If the ray doesn't hit anything and exceeds the limits, it doesn't overwrite the pixel in the frame buffer. The new image is rendered on a quad which covers the screen.

Finally, in order to get the reflections to more closely resemble a mirage, limitations, approximations and hacks are added to the effect. A shimmering effect is added by modulating the y-direction of the reflection vector of the fragments by multiplying with a bunch of sinusoids, which are seeded by time and texture coordinates. Also, the reflections are blurred using lower mipmap levels instead of applying any post processing. The mipmap level depends on the roughness value of the fragment.

Since the mirage reflections mostly appear when viewing at a low angle, and at a large distance, ray tracing is restricted depending on the angle of reflection and its distance to the camera.

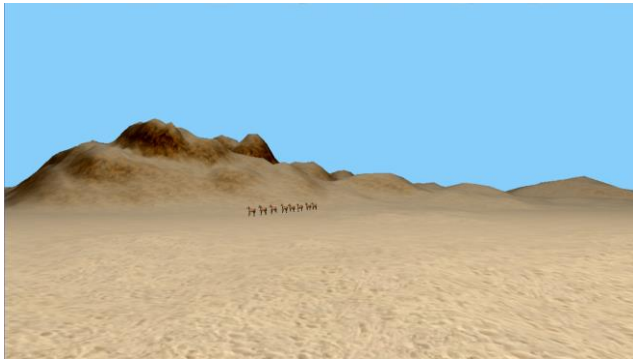


Figure 7: A desert scene rendered with Blinn-Phong Reflectance model without mirage effect



Figure 8: The same desert scene with the added mirage reflections

3 RESULTS

The perspective-correct DDA line rasterization algorithm proposed by McGuire et al. was implemented using OpenGL interface and GLSL shader language. The mirage reflections are successfully implemented using this algorithm. Compared to 3D linear ray marching, the 2D iteration is much more efficient. Because the ray takes steps by traversing over adjacent pixels, sampling them one at a time, while 3D ray marching misses or oversamples some of the pixels. Also, since a large desert scene is used, the 3D ray marching requires much more steps than 2D iteration, which reduces the performance. If the step size is increased in 3D ray marching similar performances is achieved. However, the quality reduces and becomes worse than 2D iteration.

In 2D iteration, the pixel stride can also be adjusted to increase performance, while reducing the quality. Some of the results are given in Figure 6. Since the mirage reflections appear in the distance, high pixel stride is sufficient to obtain high quality mirage reflection effect with high performance, which approximately around 97 FPS was achieved on an Nvidia GTX 850M GPU. When the reflections are disabled, the performance increases to 115 FPS. Therefore, the performance loss is not too much.

REFERENCES

- McGUIRE, M., MARA M., COLLEGE W. 2014. Efficient GPU screen-Space ray tracing. *Journal of Computer Graphics Techniques (JCGT)*, vol. 3, no. 4, 73-85
- McGUIRE, M. 2014. Screen Space Ray Tracing. <http://casual-effects.blogspot.com/2014/08/screen-space-ray-tracing.html>
- PEARCE, W. 2015. Screen Space Glossy Reflections. <http://roar11.com/2015/07/screen-space-glossy-reflections/>
- FOTIA, I., 2017. Screen Space Reflections on Epsilon Engine. <http://imanolfotia.com/blog/update/2017/03/11/ScreenSpaceReflections.html>
- ClayGL Advanced Renderer. Retrieved from: <https://github.com/pissang/claygl-advanced-renderer/blob/master/src/SSR.glsl>