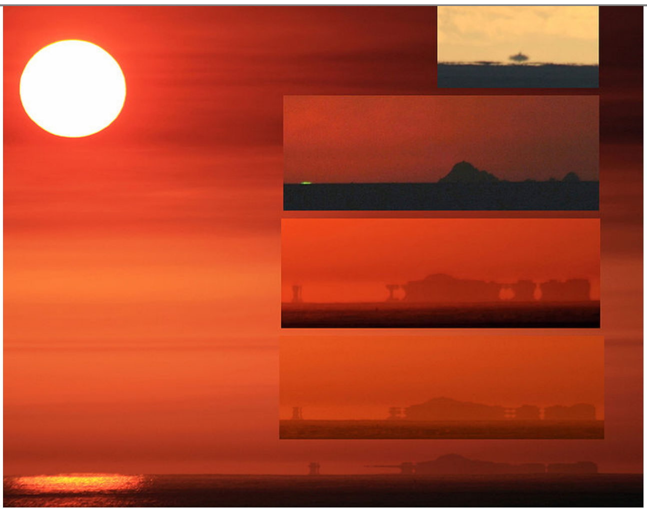
Two-Pass Realtime Rendering of Mirage Effects

Justin Tolmar White & Michael "Z" Goddard

Abstract

Mirages are difficult to simulate in real time because they are instances of refraction. This is a shame, because a mirage effect is an easy way to convey a sense of temperature in a scene. Our approach details a simple, fast method to render believable mirage effects . This approach can be easily added to realtime applications that wish to use this effect.

Introduction



Images of Farallon Islands showing inferior mirage, no mirage, and superior mirage effects. [1]

Refraction in real time rendering has always been a tricky subject. As the mirage is an example of refraction it has to deal with many of the problems. Rasterization by default expects linear lines from the scene to the eye and does not account for refracted lines. As refracting the pixel locations during rasterization would be prohibitively expensive most refraction is performed after the general scene has been rendered. During this second pass we need to take into account the difference in refraction to be able to create both inferior mirages, which show an object below its true position, and superior mirages, which show an object above its true position.

To accommodate, we provide a two pass rendering technique to create a simplified mirage effect.

Our Approach

1. Anna Zinkova. http://en.wikipedia.org/wiki/File:Farallon_Islands_at_inferior_mirage_no_mirage_and_superior_mirage.jpg

Our approach is a simplification of Wyman's approach in An Approximate Image-Space Approach for Interactive Refraction. In a typical refraction problem, rays of light refract twice: once as they enter the refractive geometry and again as they leave. The distance that a ray travels inside the refractive surface determines a displacement in the ray's exiting path. To handle this, each refractive surface needs to be rendered twice: both for back-facing and front-facing polygons.

Because the cause of a mirage is the refraction caused by a temperature gradient, it would seem natural to model the heated region and refract as we enter and leave it. However, due to varying temperature gradients, using this approach is much more complicated than refracting into solid geometry.

Instead of modeling temperature throughout the scene, we model only the temperature gradients. By assuming that temperature on either side of a gradient is constant, we eliminate the need to undo our refraction when leaving the region's geometry, allowing us to handle the entire mirage process with only one additional rendering pass.

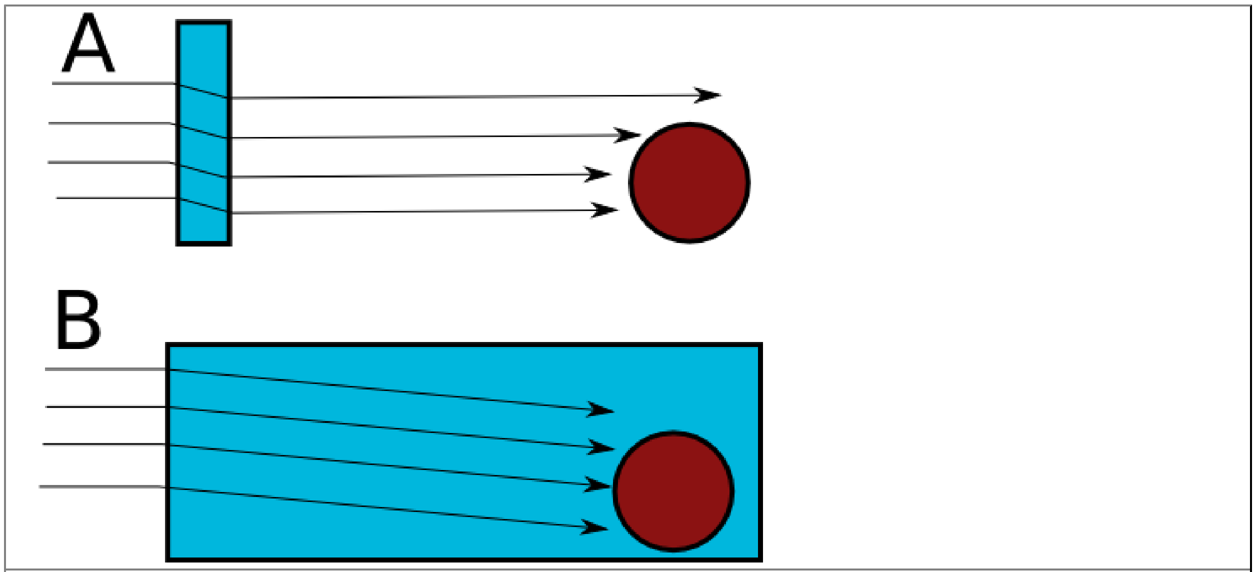


Figure n: in A an object is viewed through refractive geometry. Light is refracted twice before hitting the image. In B the object is inside the refractive geometry, meaning light must only refract once.

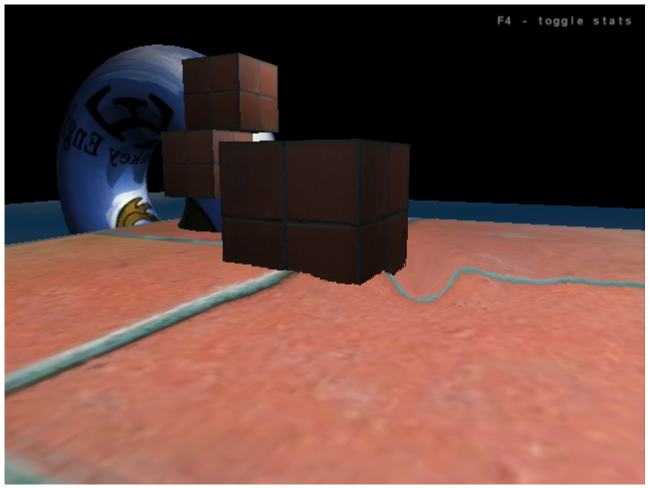
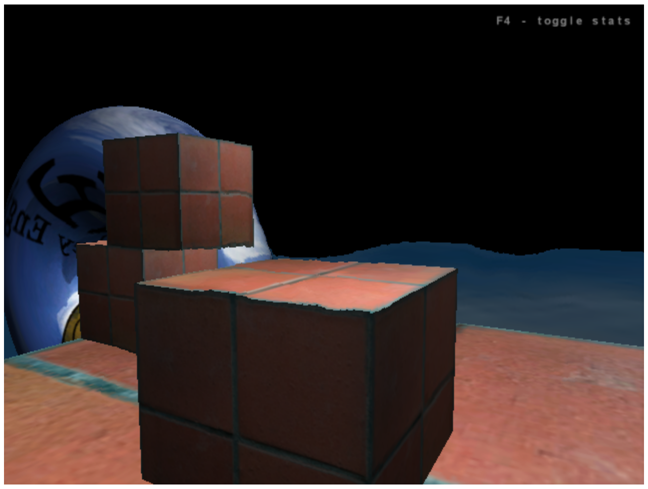
Additionally, we realize that the actual gradient does not matter except in the case where an object is not only inside the heated area, but also inside the area of heat change. Because of this, we can replace the gradient, a three dimensional region, with a single surface showing the gradient's total disturbance.

Thus, our model is very simple. In addition to the scene geometry, we keep track of a set of surfaces representing areas of notable temperature gradient. These areas are textured with the intensity of this gradient around that surface.

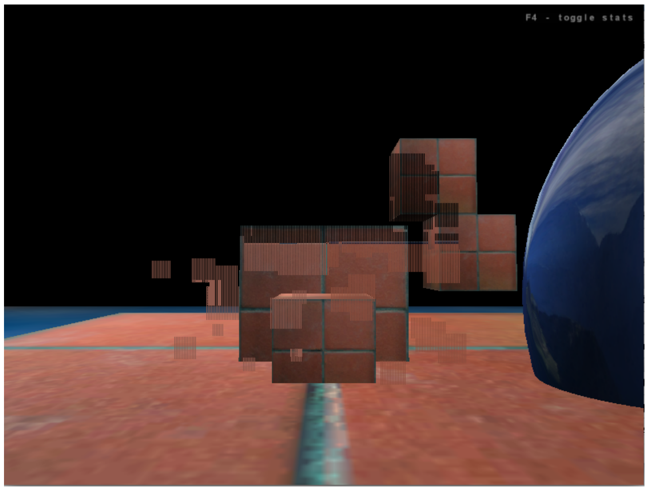
We render in two passes: first we render the regular scene geometry, saving the screen image and zbuffer as textures. Next, we render the surfaces representing temperature gradients. Using a fragment shader, we perform the following operation on each pixel of temperature gradient:

1. Compute the distance from the scene geometry by comparing the pixel's zbuffer with our stored zbuffer data at that pixel from last pass.
2. Compute the shift caused by refraction by multiplying this distance with the slope indicated by the intensity texture of this temperature gradient.
3. Add this shift to the pixel's coordinates to get the screen coordinates of the appropriate pixel.
4. Retrieve the color from our stored screen image.

This approach should work on any set of temperature gradient surfaces, but we primarily tested on particle systems set to display our texture gradient. The movement of the particles is responsible for the iconic shimmer effect associated with a mirage.

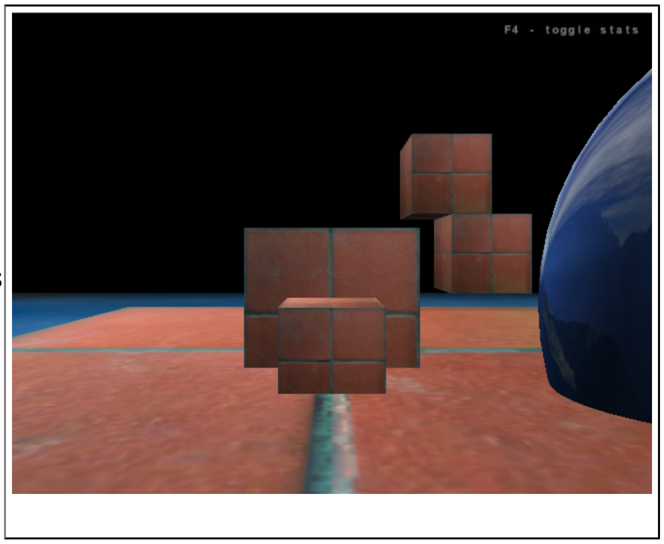
	

Challenges



The scene capture shows up at the center of the screen space and clamps everywhere else.

One challenge that set us back some time was finding the method of turning screen space into texture space. This step is performed in our solution through a vertex shader. For vertex shaders, you must transform the given vertex in model space to screen space which is done by multiplying the vertex by the model view matrix and then the projection matrix. This resulting position is now in screen space. We need to transform this value further to get it into texture space for our fragment shader which perform the refraction from the current scene render.



We were able to get the primary part of the math for this transformation. It didn't give a great result (top to the right).

```
vTexCoord = (gl_Position.xy + vec2(gl_Position.w)) * 0.5;
```

With a long look through internet with google's help we were able to find a blog article that provided that last small bit to the formula. [2]

```
vTexCoord = (gl_Position.xy + vec2(gl_Position.w)) * 0.5;
vTexCoord /= gl_Position.w;
```

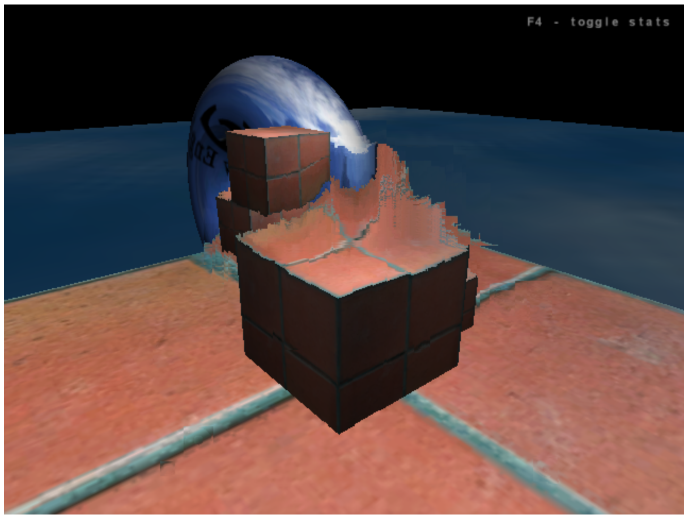
Presenting much better results. At the time the fragment shader simply sampled that position on screen. So the result shows nothing special. It does not change the scene.

The top image to the right also displays another issue we encountered but could not fix for Z's machine where a stripping effect occurred on the area mirage geometry was being rendered.

Limitations

Our approach has a number of limitations, some inherent in the design and others because of errors in our implementation.

2. Wolfgang Engel http://diaryofagraphicsprogrammer.blogspot.com/2008/09/calculating-screen-space-texture.html



The major limitation in our design is that we cannot model cases where light bends behind an object. This is due to the fact that by the time our mirage pass has begun, all geometry behind that immediately visible in the screen has been discarded. With an inferior mirage, this problem can only happen when a heat distortion appears near a busy ceiling, and thus is rather unlikely. However, with superior mirages, the problem occurs near a busy floor. This is much more likely, and our effect is less convincing on these mirages. To the left you can see the errors this causes in a superior mirage: you should be able to see behind the cube, but instead see the cube itself being pulled upwards (and creating numerous artifacts in the process).

Additionally, we cannot model a gradual temperature gradient that intersects an object. We can approximate a gradual temperature gradient using a series of gradient surfaces, but if an object intersects these surfaces the jump will be visible.

Other limitations are due to errors in our implementation. To properly handle a gradual gradient using multiple gradient textures, we need to first sort them in painter's algorithm order, then draw every one. Because of limitations in the JME engine we created our implimentation in, we could not find a way to sort geometry before it goes through our rendering pass. To alleviate this, we only draw the frontmost geometry. However, this is still technically incorrect.

Additionally, an error in our shader exists that causes it to treat all rays hitting the surface as being perpendicular to the surface. We should take the angle to the camera into account in these cases. Thankfully, this specific error is almost imperceptible. It would be visible in cases of overlapping distortions, but because of the above error these cannot be seen anyway.

Further Work

The limitations in our approach caused by errors in our implementation could be fixed. This would allow a large number of gradual gradient effects to be possible that can't be achieved with the implementation's current state.

Limitations due to our actual approach are more difficult to fix. The problem of what to do in the case of an object inside of a gradual gradient could potentially be fixed by intelligently generating temperature gradient textures according to the camera's position. This, however, would require that the simulation know the 3D temperature distribution, which is currently not present.

Our big limitation, that of not being able to see behind objects, would be much more difficult to fix. Because the initial (plain) rendering pass discards the information that would be needed to do this properly, it seems that the only solutions would involve additional passes or some method of storing the last few layers of the zbuffer and scene texture when drawing. It may also be possible to solve this by marking locations where it causes errors and do an extra rendering pass afterwards.

Acknowledgements

The Java Monkey Engine (http://www.jmonkeyengine.com/) - Graphics/Game engine effect was developed on.

nymon, a Java Monkey Engine Forum User - assisted in finding a way to grab the depth buffer.

Michael "Z" Goddard - Performed initial work on code, writing a very simple test for the pass (org.rpi.mirage.TestMiragePass), primary work on the pass (org.rpi.mirage.MiragePass), developing the vertex shader (quad.vert) and an initial simple shift based on depth fragment shader (quad.frag).

Justin Tolmar White - Created the particle system we used for testing, and set the engine up for multiple passes. Adjusted the shaders to take an intensity texture, made the fragment shader (quad.frag) capable of handling multiple shift angles, fixed errors in the shaders, and set up the system to be able to simultaneously handle both hot and cold temperature gradients.

Bibliography

Ye Zhao, Yiping Han, Zhe Fan, Feng Qiu, Yu-Chuan Kuo, Arie E. Kaufman, and
Klaus Mueller. Visual Simulation of Heat Simmering and Mirages. http://portal.acm.org/
citation.cfm?id=1234989.1234995&coll=Portal&dl=GUIDE&CFID=26772015&CFTOKEN=9

Marc Berger, Terry Trout, Nancy Levit. Ray Tracing Mirages. http://www2.computer.org/portal/web/csdl/doi/10.1109/38.55151

Chris Wyman. An Approximate Image-Space Approach for Interactive Refraction. http://www.cs.uiowa.edu/~cwyman/publications/files/approxISRefract/approxISRefr.pdf

Scott T Davis, Chris Wyman. Interactive Refractions with Total Internal Reflection. http://www.cs.uiowa.edu/~cwyman/publications/files/refractionWTIR/refrWithTIR.pdf