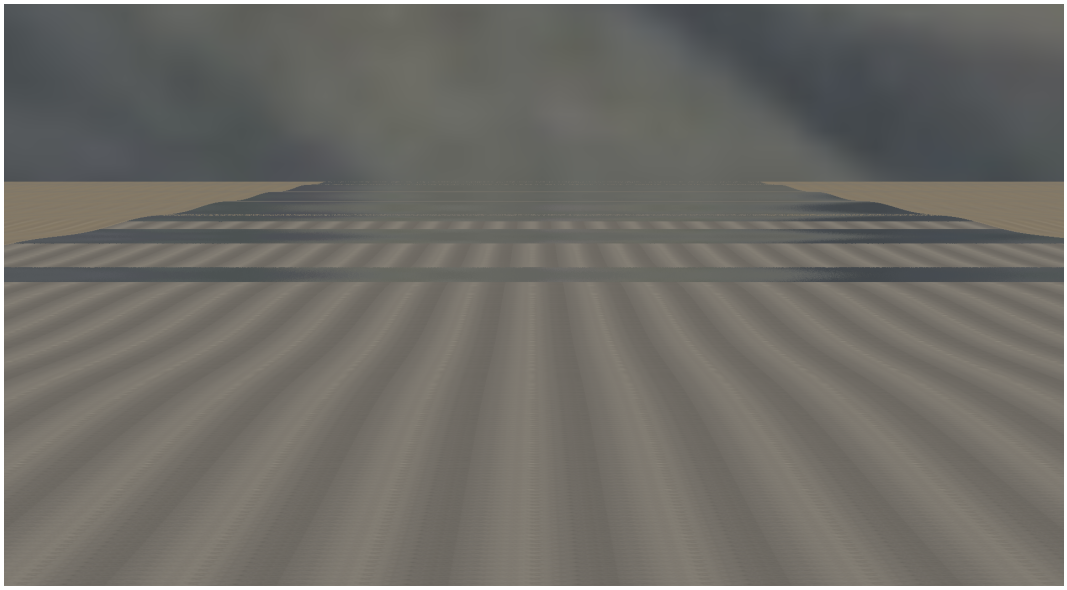
Real-time Dynamic Mirage Rendering

Version of December 3, 2025



Q.B. van Velthoven

Real-time Dynamic Mirage Rendering

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Q.B. van Velthoven
born in Delft, the Netherlands

Computer Graphics and Visualization Group
Department of Intelligent Systems
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

Real-time Dynamic Mirage Rendering

Author: Q.B. van Velthoven
Student id: 4483162

Abstract

Mirages are a visual phenomenon consisting of the appearance of a mirrored image of an object, without the presence of an actual mirror surface, due to light rays that are curved because of continuous refraction in the air, which relates to differences between the surface and ambient temperature. As temperature is defined in space, while a typical standard rasterization pipeline only processes surfaces, these phenomena are difficult to reproduce. Approximating the nonlinear ray path with ray marching becomes taxing due to the long light paths. Current approaches use acceleration structures and have not been implemented in a rasterizer.

We present two methods that make dynamic real-time rendering of mirages possible, which fit well in the rasterization pipeline. The first solution uses a second camera to capture surface temperature and normal information below the view ray, and approximates the nonlinear path of the ray in as few steps as possible. The second method obtains the surface information in screen space instead, making it faster, but potentially less accurate in heterogeneous scenes.

Results show that both methods are capable of rendering mirages dynamically and in real-time when the surface is relatively flat. Therefore, both methods, especially the second, faster method, could be used for the rendering of mirages on relatively flat faces, enabling real-time dynamic rendering of mirages in video games on those types of surfaces.

Thesis Committee:

Chair:	Dr. R. Marroquim, Faculty EEMCS, TU Delft
University supervisor:	Dr. R. Marroquim, Faculty EEMCS, TU Delft
University supervisor:	Prof. Dr. E. Eisemann, Faculty EEMCS, TU Delft
Committee Member:	Dr. J. Urbano, Faculty EEMCS, TU Delft
Committee Member:	Dr. P. Kellnhofer, Faculty EEMCS, TU Delft

Preface

This master project has been carried out by Quint van Velthoven at the Delft University of Technology under the supervision of Prof. Dr. Elmar Eisemann and Dr. Ricardo Marroquim. This project aims to create a method that renders mirages dynamically in real-time with little pre-processing.

I would never have imagined that the process of writing a thesis would end up looking like this when I started working on my thesis in February 2020: the outbreak of a pandemic and being confronted with a severe illness were both absent in my study planning. The project timeline may have changed partially due to these events for the worse, but I am proud of the final result and glad that I can finally share this with the computer graphics community.

I would like to express my gratitude to several people who helped to bring this thesis to an end. First of all, I thank Prof. Dr. Elmar Eisemann for his guidance and feedback during most of this project and the preceding seminar. I also would like to thank Dr. Ricardo Marroquim for guiding me during the last months and helping to bring an end to this all.

Secondly, it is appropriate to express my gratitude to Dr. Julián Urbano and Dr. Petr Kellnhofer for being willing to be part of the thesis committee.

I would also like to thank my friends and family, who definitely made these tough months more bearable. Their presence has been an enormous joy to me after a day of reading papers or writing code alone in my apartment. Special thanks go to my mother, who cared for me throughout the project and even more so during my treatment. She often asked when I would finally hand in my thesis, which must mean that she cannot wait to learn more about the difficulties that come with rendering mirages; I hope that this work meets her expectations. Finally, I would like to thank the medical personnel who helped me during the thesis process, especially those working at the Reinier de Graaf Gasthuis.

Q.B. van Velthoven
Delft, December 2025

Contents

List of figures

Chapter 1

Introduction

A mirage is a visual phenomenon consisting of the appearance of a mirrored image of an object without the presence of an actual mirror surface, due to temperature differences in the air. These deceiving images are commonly spotted on the road in the summer, in the desert, and above the sea. They are an intriguing subject to capture, and Figure 1.1 shows examples. However, they have been mostly absent from real-time video games and applications, with the exception of road mirages being added before to a racing video game (Nelva, 2015).

Heat shimmering (or heat haze) is a closely related phenomenon where light rays and, therefore, images are slightly distorted due to refraction caused by air temperature differences. The effect is noticeable without the presence of a mirage and vice versa, as mirages simply require sufficient refraction in the same direction so that the light ray would become parallel to the surface under or above the light ray. Heat shimmering is often approximated in real-time applications by capturing the scene in a texture and then sampling that texture with varying texture coordinates, which are altered with some noise function or texture

(a) A superior mirage (reflection above object).

(b) An inferior mirage (reflection below object).

Figure 1.1: (a) A superior mirage of a ship above Finnish waters. Source: (Mirages in Finland). (b) An inferior mirage on the road of a car's front and vegetation. The snow shows that high ambient temperatures are not required to have mirages. A part of a frame from the film *Wind River (2017)* (Sheridan, 2017).

Figure 1.2: Heat shimmering in a video game. Obtained from Fire and Haze: Wind Waker Graphics Analysis Series.

(a) Belgian Grand Prix in F1 2021 video game.

(b) F1 2022 Belgian Grand Prix.

Figure 1.3: (a) A zoomed-in frame from the F1 2021 video game that does not contain mirages. (b) Footage from the actual race with plenty of mirages. Both images capture the same corner and are taken from roughly the same position.

to mimic random temperature fluctuations (Fire and Haze: Wind Waker Graphics Analysis Series). This approximation allowed popular twenty-three-year-old video games Super Mario Sunshine (Nintendo GameCube, 2002) and The Legend of Zelda: The Wind Waker (Nintendo GameCube, 2002) to contain many applications of heat shimmering already, as displayed in Figure 1.2. Meanwhile, rendered mirages have been mostly absent. Figure 1.3 shows how a lack of simulated mirages can reduce the realism of video games.

Mirages do not form because of temperature fluctuations, but due to air refraction caused by a stable air temperature gradient instead, as total internal reflection is required to obtain the reflection. Therefore, adding randomness to the texture coordinates will not work well for simulating mirages. Even if that was the case, it is still a crude approximation that forgoes the physics that causes a mirage.

What complicates the simulation of mirages is the ray's nonlinear path, which depends on local atmospheric information. Linear paths are easier to compute since they only change direction when they intersect with an object. Another issue is that some methods do not work for dynamic scenes because geometric structures must be built to approximate the

scene (Cao et al., 2010; Mo et al., 2015). An additional problem is that the current approaches do not achieve interactive framerates (Stam and Languénou, 1996; Seron et al., 2005; Gutierrez et al., 2006; Zhao et al., 2006; Cao et al., 2010; Mo et al., 2015). Finally, there is an issue with the accuracy of the functions that have been used to calculate the index of refraction of air in currently existing mirage rendering approaches, which has major consequences in regards to accuracy (Cao et al., 2010; Mo et al., 2015; Stam and Languénou, 1996; Khular et al., 1977; Zhao et al., 2006).

In this project, we present two closely related methods to enable mirage rendering for dynamic scenes in real-time, which require little pre-processing. In our methods, generally slower approaches such as ray tracing and ray marching are circumvented by capturing geometric data in an image buffer using rasterization and approximating the ray's curved path using ray equations, followed by techniques borrowed from screen-space reflection rendering. Ray equations have been used before to render mirages (Cao et al., 2010; Mo et al., 2015), but these solutions did not rely on scene data captured via a rasterizer. Access to this data has been made possible using temperature textures to store surface temperature information (Zhao et al., 2006), which has not been done for mirage rendering in the rasterization pipeline before. Our first method obtains the scene data with a second camera so that even some of the invisible geometry from the main camera is accounted for, and uses the mean averages of select parts of the surface to approximate the required surface attributes. Our second method works fully in screen-space and is, therefore, faster at the cost of accuracy.

The only preparation required for these methods to work is designing temperature textures for objects that must be able to have mirages above them to allow for access to necessary temperature information; all the other required data is already available through deferred rendering, except for the normal information that the first method obtains with the second camera.

A limitation of the presented approach is the inability to render mirages that require refraction outside the view ray's epipolar slice. The presented methods also seem not to perform as well for surfaces with much height deviation, compared to (relatively) flat surfaces.

This work also includes a thorough comparison of the different functions used before in mirage rendering to calculate the air's refractive index. This comparison is used to improve realism while keeping the run time in check.
In summary, our contributions are:

1. A comparison of functions for computing the air's index of refraction on accuracy and speed;
2. A comparison of different air temperature models on accuracy and speed;
3. A correction of a function for the refractive index of air in an earlier mirage-rendering method.
4. One method to render mirages with scene information captured from a second camera;
5. Another approximating method to render mirages completely in screen-space.

Chapter 2

Background

Rendering mirages accurately requires an understanding of how mirages are formed in real life. This chapter discusses how natural processes such as heat transfer and atmospheric refraction can cause a mirage to appear under the right circumstances.

2.1 Rasterization pipeline

Rasterization is a rendering technique where primitives in a 3D space, often triangles, are transformed to a 2D raster image consisting of pixels. Although the exact steps in this transformation process may differ, Figure 2.1 shows what the different stages typically look like based on graphics APIs such as DirectX and OpenGL. Here, we solely focus on the stages that are most relevant for our work, which are the vertex and fragment shader stages. First a shader, which is a small program that runs on the graphics processing unit (GPU), operates on the primitives' vertices and typically transform their local coordinates to world space coordinates by matrix multiplication. The fragment shader is where operations are performed per fragment, i.e., per part of a primitive that fills a pixel, and where usually the shading of objects is done and where textures are applied to models. Finally, some per-fragment operations, such as a depth test (only parts of faces closest to the camera should be visible) are performed before the output is written to a framebuffer. The framebuffer has 1 frame's worth of pixel data that can be rendered directly to the screen or be used in a different rendering pass.

Deferred rendering The framebuffer can have more than 1 attachment that can be written to simultaneously, enabling fragment shaders to have multiple outputs, e.g., 1 depth buffer and 4 color buffers. This is particularly useful for deferred rendering, a rendering technique where objects' geometry and texture data are first rendered to different color attachments, and the actual shading is only performed in a separate rendering pass that uses those color attachments and one or two screen-filling faces. The color buffers are called a G-buffer ('geometry buffer') and the G-buffer typically consists of depth, position, albedo, normal, and material (e.g., roughness) information. Figure 2.2 shows how the G-buffer may look

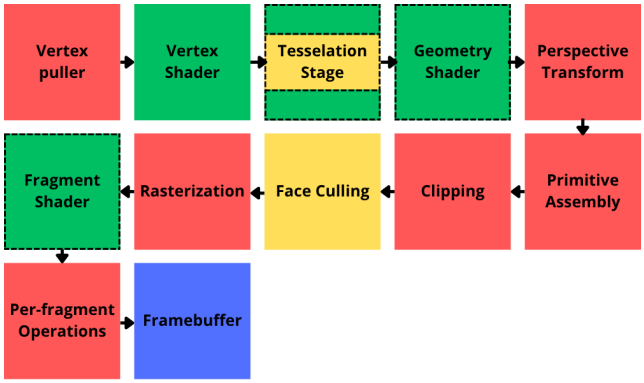


Figure 2.1: A typical rasterization pipeline. Green stages are programmable, yellow stages are configurable, red stages are fixed function stages, and the framebuffer is the output. The stages surrounded by a dashed line are optional, although the fragment shader is still often defined. The arrows indicate the data flow, which starts with the vertex puller stage.

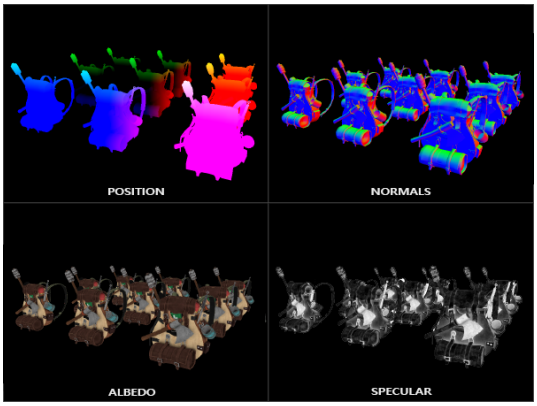


Figure 2.2: A G-buffer consisting of 4 color buffers. Obtained from de Vries (2015).

like. Deferred rendering can have multiple advantages, but the biggest one is that it prevents the waste of shading computations on fragments that end up being obstructed.

2.2 Mirages

A mirage is an optical phenomenon where an inverted reflection appears without the use of a mirror or reflecting object due to the presence of a temperature gradient. Although mirages are a common sight, and research has been conducted on mirages since the 1800s (Young, 2015), there is no definitive answer to the question of what causes them. The consensus is that mirages appear due to refraction: a temperature gradient above objects causes enough atmospheric refraction to allow for total internal reflection to take place and thus not only refract but also reflect light. The reflection that then appears is the mirage. The temperature gradient causes the air temperature above a surface to change in a monotonic fashion until

the air temperature has become equal to the ambient temperature; the ambient temperature is the local air temperature, comparable to the temperature mentioned on the weather forecast. The temperature gradient can be negative (i.e., the air temperature decreases with distance from the object) or positive. The former causes an inferior mirage, which is a mirage with the inverted image below the reflected object; the latter causes a superior mirage, where the inverted reflection is above the original object. Figure 1.1 presents examples of superior and inferior mirages.

2.2.1 Cause of temperature gradient

The temperature gradient is often caused by the sun warming up surface materials. For example, asphalt has a high absorptivity for solar radiation, which means that it absorbs most of the energy from the solar light and, therefore, quickly warms up (Çengel, 2002, p. 589), causing the common road mirages. Oceans have a very low albedo, meaning they absorb most solar radiation as well (Thermodynamic: Albedo). However, both inferior and super mirages can appear above sea as the sea temperature changes slower than the ambient temperature due to the thermal capacity of the large, deep pools of water.

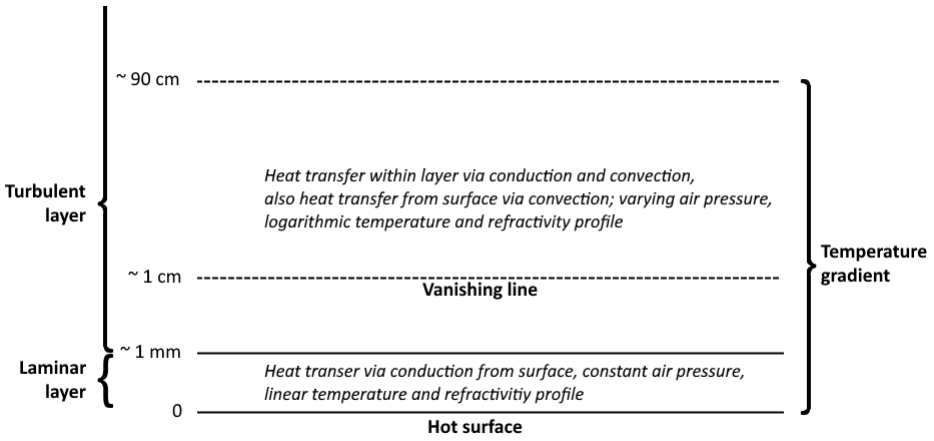


Figure 2.3: Overview of the different boundary layers and their properties above a hot surface in the case of an inferior mirage. The temperature gradient is noticeable up to approximately 90 cm, meaning that the air temperature can be considered equal to the ambient temperature above that.

Now that it has become clear why the surface temperature reaches an adequate level to cause the formation of mirages, we examine the causes of temperature gradients with the use of Figure 2.3. For inferior mirages, the air temperature close to the ground is higher than the ambient temperature due to convection from the surface. The layer of air closest to the surface, which is the laminar layer, is influenced by conduction instead of convection and its height is approximately a few millimeters (Wakid, 2019; Young, 2015). It is presumed that there is no pressure difference between the bottom and top of the laminar layer (Wakid, 2019). The temperature, and therefore the index of refraction, changes linearly in

the laminar layer, as the laminar layer can be compared to a solid dielectric slab heated by convection (Wakid, 2019), and the temperature gradient is linear in such a slab (Çengel, 2002, p. 87). The vanishing line, which is the line where the original image and reflection image intersect as shown in Figure 2.4, is slightly higher at around 1 cm above the surface (Wakid, 2019).

Above the laminar layer lies the turbulent layer, named after the buoyancy and turbulence present there. Buoyancy in the turbulent layer is caused by gravity (Wakid, 2019), while turbulence is the result of the differences in air pressure. The air pressure within the layer differs partially due to heat transfer (conduction and convection) (Zhao et al., 2006). The heat transfer is responsible for the temperature gradient near the surface, where the air temperature is different from the ambient temperature. The length of the temperature gradient is not clearly defined, with claims that the inferior mirage is formed "in the lowest few meters of the atmosphere" (Young, 2015, p. 172) and experiments that show that the temperature gradient stops at around 90 cm from the surface (Wakid, 2019). At this height, the air temperature has become equal to the ambient temperature in the area.

No consensus exists on whether mirage formation takes place in the laminar layer or the turbulent layer: one hypothesis is that the mirage formation depends on the temperature gradient below the vanishing line (Wakid, 2019), while another is that the layer above it with its turbulence, pressure differences and logarithmic refraction index profile causes mirages (Young, 2015). The second hypothesis seems more plausible: a linear profile for the index of refraction cannot produce inverted images (Biot, 1810), and the temperature gradient and, therefore, the refractivity profile in the laminar layer is presumed to be linear as Figure 2.3 indicates. In this view, the vanishing line does not mark the lower boundary of the turbulent layer, like Wakid (2019) claims, but lies within it as depicted in Figure 2.3.

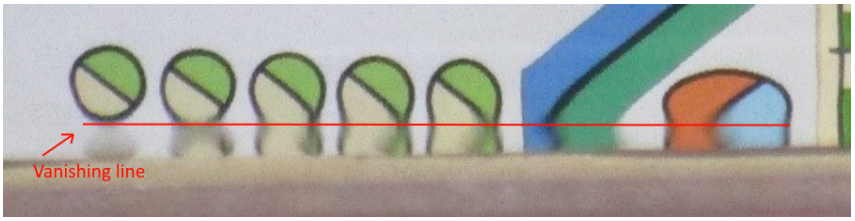


Figure 2.4: Indication of the vanishing line for an inferior mirage. Image originally from Wakid (2019), adapted by ourselves.

2.2.2 Refraction

Although it is clear how temperature gradients are formed, it remains unsettled why mirages take the form of inverted reflections. Therefore, we must examine the relation between refraction and (air) temperature.

The refractive index of air depends on multiple factors: temperature, pressure, vacuum wavelength, humidity and CO_2-content (Edlén, 1966; Ciddor, 1996). The vacuum wavelength of light is constant, and the air humidity and the partition of CO_2 in the air mixture usually do not vary much. The air pressure does differ and generally lowers slowly with

an increase in altitude, as less air above weighs down on the air below. However, the difference in air pressure close to the surface can be neglected when mirages are caused by local hot spots, such as the road mirage (Khular et al., 1977; Seron et al., 2005). The only remaining factor that could impact the refractive index of air is temperature, and the temperature gradient caused by such hot spots changes the air temperature above the surface significantly (Wakid, 2019). Therefore, it is probable that mirages are caused by refraction, and the reflection-like results could then be explained by total internal reflection, an effect that shows up with refraction when the angle between the normal and the incoming ray of light (*the angle of incidence*, θ_i) is larger than the critical angle θ_c.

The critical angle is only defined when the ray goes from a medium of lower propagation speed to one of a higher propagation speed and depends on the ratio of the indices of refraction of the two media. There is always already some partial reflection with every refraction when the angle of incidence is smaller than the critical angle. As Figure 2.5 shows, the angle of refraction θ_r becomes larger when the angle of incidence becomes larger itself, if light travels slower through the first medium than the second medium. This behavior leads to a situation where the angle of refraction becomes 90° with a smaller angle of incidence, which is then equal to the critical angle. When the angle between the incidental ray and the normal then becomes even larger, the ray no longer refracts along the side of the medium. Instead, it only reflects within the first medium: total internal reflection.

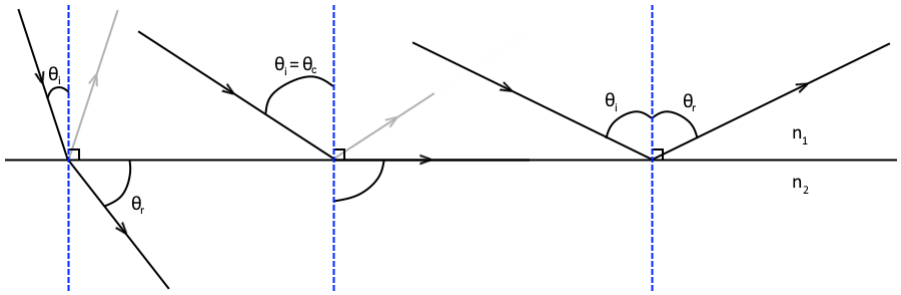


Figure 2.5: Three light rays go from a medium of a lower propagation speed to a medium of a higher propagation speed. The left ray mostly refracts, but there is always at least partial reflection too in this situation. The angle of incidence of the middle ray is equal to the critical angle, resulting in an angle of refraction of 90° and partial reflection. The right ray's angle of incidence is larger than the critical angle, meaning there is no refraction anymore and only total internal reflection.

The refractive index of air is only slightly affected by a significant change in temperature. This, combined with the fact that the temperature gradient is only noticeable close to the surface (in the case of inferior mirages), means that light rays need to be affected by temperature differences for a more extended period to refract enough to achieve total internal reflection, making it generally required for the ray to be close to parallel with the surface. It is also necessary for total internal reflection to occur that the light ray traverses air layers with increasing phase velocities so that the ray refracts away from the surface. The monotonic temperature gradient ensures these increasing phase velocities, since hotter

air has a lower refraction index and, consequently, a higher phase velocity, as can be deduced from Equation (3.10). All in all, refraction due to a temperature gradient seems to be the most logical explanation of mirage formation.

A theory exists that not a temperature gradient but the roughness of the surface, combined with a small viewing angle between the surface and the camera, causes mirages to form. Appendix A explains why the mirages produced by continuous refraction in the air are different from the reflections caused by a rough plane and small viewing angle. Therefore, this work does not consider them further.

In summary, a temperature gradient forms above surfaces due to solar irradiation, causing light rays to refract and even reflect with sufficient refraction. The necessary amount of refraction to achieve total internal reflection is obtained if the angle between the ray and surface is small since the most refraction happens near the surface: the air temperature decays quickly farther from the surface, and a significant difference in air temperature is required to change the index of refraction of air slightly.

Based on this background information, we will not only devise our own methods but also evaluate the physical bases of previous methods. For the latter, we will consider different functions to calculate the index of refraction of air.

Chapter 3

Related Work

This chapter discusses previous methods for the rendering of mirages. However, we also pay attention to different methods to compute the index of refraction of air, as approximations use different parameters and equations. Less accurate values for the air's index of refraction lead to less realistic mirages.

3.1 Refraction index calculation

While there is only one actual index of refraction for air in a given situation, multiple approaches exist to calculate the index: calculating the density of air and, therefore, its refractive index normally involves multiple parameters and can become complex and computationally expensive (Edlén, 1966; Ciddor, 1996), warranting methods that approximate the air's index of refraction.

Height-based function The method created by Khular, Thygarajan & Ghatak generates index values for both inferior and superior mirages using height as the only input parameter (Khular et al., 1977). In their method, other parameters that influence the index of refraction of air, such as those affecting the density and air humidity, are disregarded because the variation of pressure is in the case of mirage formation "negligible" and moisture "can be ignored", making the density and, therefore, the refractive index depend only on the temperature (Khular et al., 1977, p. 90). Based on earlier findings that the temperature profile above hot land was found to be roughly exponential and that the temperature gradient could be regarded as considerable only up to approximately 1 meter above the ground (Fleagle, 1956), the authors came up with two functions where height is the input and the refractive index changes exponentially to a maximum height of 1 meter.

The consequence of making the function based on height is that the method gives the same index of refraction for the air that is a certain height above a surface, regardless of the surface and ambient temperature. Therefore, mirages that should look different due to bigger differences between surface and ambient temperature will look the same with this method.

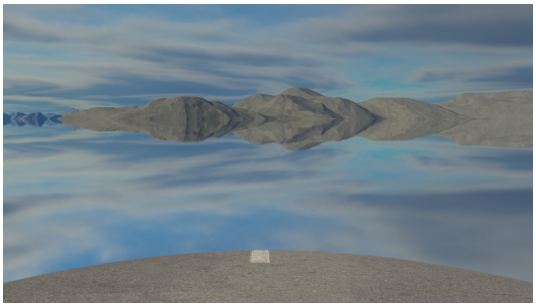


Figure 3.1: A render of an inferior mirage obtained using the function from Khular, Thygarajan & Ghatak. The camera is positioned 1 meter above the surface.

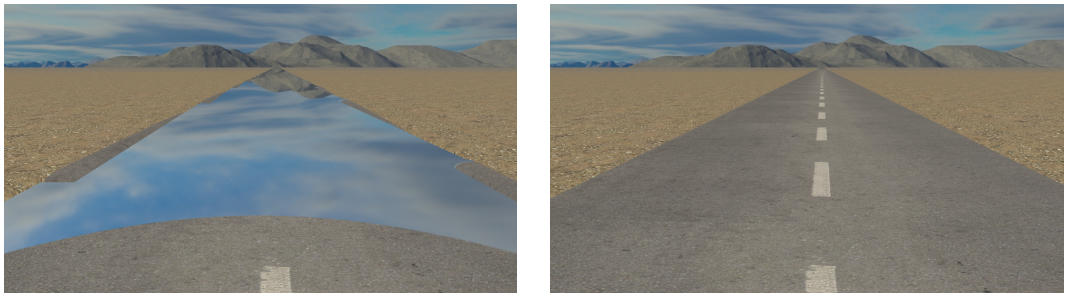
Another problem is that it is not entirely based on empirical data, but partly on the authors' assumption of a 10%-variation in the air's index of refraction between the surface and the end of the temperature gradient. The authors already suggest that the function might allow for too much, unrealistic variation of the refractive index (Khular et al., 1977), and Figure 3.1 shows how mirages will incorrectly appear only a few meters away from the observer if the function is used.

Zhao's and Edlén's functions In a method created by Edlén and later corrected by Birch & Downs, the index of refraction is calculated in two steps (Edlén, 1966; Birch and Downs, 1994). They first calculate the dispersion influence on the index n_s (Equation (3.1)), and then multiply the outcome by the density influence on the refractive index (Equation (3.2)) to get the refractive index of air n. The method accepts air pressure P and the air temperature in degrees Celsius t as input, as well as inputs that (CO_2-concentration, humidity, and wavelength λ). However, these extra inputs can be considered constants in our situation, as we explained in the previous chapter that temperature is presumed to be the most important factor in the formation of mirages. The standard atmospheric pressure (the average atmospheric pressure on Earth at sea level) can be used for the pressure, as the air pressure barely changes within the volume affected by the temperature gradient: $P = 101,325$ Pa (Khular et al., 1977; Seron et al., 2005). σ in Equation (3.1) is the reciprocal of the wavelength in micrometers.

$$(n-1)_s(\sigma) = 10^{-8}(8342.54 + 2406147[130-\sigma^2]^{-1} + 15998[38.9-\sigma^2]^{-1}) \tag{3.1}$$

$$n(t,P) = 1 + \frac{P(n-1)_s}{96095.43} \cdot \frac{1 + 10^{-8}(0.601 - 0.00972t)P}{1 + 0.0036610t} \tag{3.2}$$

Edlén's updated method has an empirical basis, as the values in Equations (3.1) and (3.2) were chosen to minimize the differences between the calculated densities and refrac-



(a) Render using Zhao et al.'s function.

(b) Render using the function by Edlén & Birch.

Figure 3.2: Road mirages rendered from the same position with the refractive index function used by Zhao et al. and the Edlén-Birch function Zhao et al. intended to use. The road mirage appears too close to the observer when Zhao et al.'s function is used.

tive indices outcomes and those found in laboratory experiments (Edlén, 1966) – a clear improvement over the assumed '10%-variation' of the height-based function.

$$n(t,P) = \frac{P}{96095.43} \cdot \frac{1 + 10^{-8}(0.601 - 0.00972t)P}{1 + 0.0036610t} \tag{3.3}$$

Zhao et al. tried to use Edlén's method based on the source they refer to (Zhao et al., 2006), but their implementation (Equation (3.3)) differs from Edlén's; it fails to include n_s and the addition of 1 at the end results in inaccurate outputs for n that are still fairly close to '1', which likely obfuscated the errors. As Figure 3.2 shows, the mistake results in wildly different mirages. Appendix B offers a more comprehensive analysis of Zhao et al.'s derivation and its inaccuracies.

Functions with built-in temperature model A simple method that only requires temperature as input was proposed by Minnaert et al. (Minnaert, 1993):

$$n(T) = 1 + \frac{T_0(n_0 - 1)}{T}, \tag{3.4}$$

where T is the input temperature in Kelvin, T_0 is a temperature of 273 K, and n_0 is the refraction index of air with temperature T_0 ($\sim$1.00023). This function is based on the fact that "the amount by which the refractive index of air exceeds that of a vacuum is directly proportional to the air density, that is, inversely proportional to the absolute temperature" (Minnaert, 1993, p. 68).

Stam & Languénou use 'distance from the temperature source' together with the ambient and surface temperature to incorporate a temperature model in Minnaert's function (Stam and Languénou, 1996). The authors note that the temperature above planar objects usually has an exponential drop-off based on the surface temperature and a drop-off distance d_0. The air temperature at a distance h from the surface with a surface temperature T_s could then be calculated as shown in Equation (3.5), where T_a is the ambient temperature. A proper value for d_0 is unclear, although it is mentioned that a hot asphalt road has

been found to have a drop-off length of approximately 1 cm (Stam and Languénou, 1996; Minnaert, 1993) or 2 cm (van der Werf, 2011).

$$T(T_s,h) = T_a + (T_s - T_a)\exp(-h/d_0) \tag{3.5}$$

Several other functions have incorporated the temperature model created with Equation (3.5)), but are not based on Equation (3.4) to obtain a value for refractive index n. Gutierrez et al. calculate the air density ρ with the ideal gas law (Equation (3.6)), the influence of the wavelength on the refractive index with Cauchy's dispersion formula (Equation (3.7)), and finally the index of refraction with Gladstone-Dale's law (Equation (3.8)) (Gutierrez et al., 2006). In these equations, the pressure P can be considered constant again, as well as the wavelength λ and M (the mean mass of the molecules of a mixed atmosphere in kg/mol^{-1}) under the assumption of monochromatic light and of there being no local changes in atmosphere composition.

$$\rho(h) = \frac{PM}{8.314510T(h)} \tag{3.6}$$

$$n_d(\lambda) = 28.79E-5\cdot(1+\frac{5.67E-5}{\lambda^2})+1 \tag{3.7}$$

$$n(h,\lambda) = \rho(h)\cdot(n_d(\lambda)-1)+1 \tag{3.8}$$

The ideal gas law can also be derived to a different function for the air's refractive index, and this function by Van der Werf is shown in Equation (3.9) (van der Werf, 2011). This function has the disadvantage over the one used by Gutierrez et al. that fewer parameters are accepted, which becomes an issue when the wavelength of the light needs to be altered. Another disadvantage is that Van der Werf has not provided an error bound nor mentioned having compared the function outcomes with actual data. Nonetheless, Van der Werf's function does have a physical basis, and when parameters other than the temperature are considered constant, the two approaches are as computationally intensive as each other.

$$n = 1 + \frac{7.872E-7P}{T} \tag{3.9}$$

Boundary-layer-theory-based temperature model Next to using exponential decay to model the relation between air temperature and height, it is possible to use a function based on a heat transfer model: Section 2.2.1 already mentioned that boundary layer theory could be applied to mirages (Young, 2015). Such a model exists, and it is not too computationally intensive to obtain air temperatures (Young, 2015). The user needs to provide values for the height of the laminar layer y_t, the strength of the convection, and the strength of the temperature decay Θ.

An interesting difference between the two different temperature models is that one uses exponential decay (Equation (3.5)) to obtain air temperatures, while the other uses a logarithmic profile (Young, 2015). The temperature in Equation (3.5) is divided by e every drop-off length d_0 farther from the surface with the former, while Young's approach makes

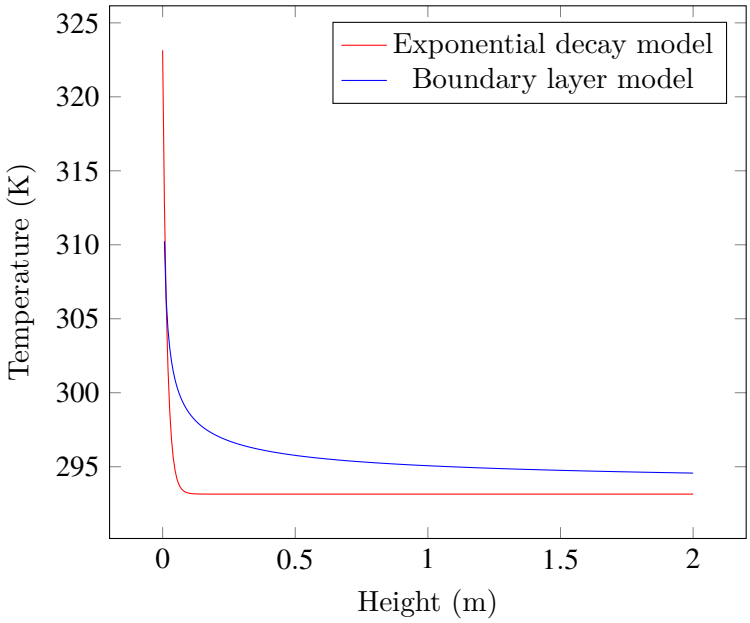


Figure 3.3: An example that shows the difference in the temperature decay that comes with an increase in height for the two different temperature models. The surface temperature was set to 321.5 K and the temperature model parameters were set so that the temperature models resulted in temperatures close to those found in an earlier experiment by Wakid (2019).

the temperature drop Θ degrees Celsius every e farther from the surface, which seems similar but is quite different as Figure 3.3 shows: the logarithmic model decays relatively much slower than the exponential model.

A disadvantage that comes with at least the theoretical model is that it is specifically designed for inferior mirages, while we want to support both inferior and superior mirages. We presume that the logarithmic model can be adapted to deal with the positive temperature gradient that comes with superior mirages, as flipping the temperature gradient direction 180° will result in a superior mirage if the temperature gradient resulted in a inferior mirage before. However, air temperature measurements are required to see if such a flipped temperature gradient can indeed be a good approximation of reality.

Ciddor's function Ciddor's approach to obtaining index values is quite similar to that of Edlén, and accepts the same five input variables (the wavelength of the light, temperature, humidity, CO_2-level, and pressure) (Ciddor, 1996). Ciddor's method even yields the same results as the updated Edlén method for the range in which Edlén's method gives valid values (Ciddor, 1996). The main advantage of this method is that it gives valid results for a wider domain of atmospheric conditions: it works for temperatures between -40 °C and +100 °C, much more than Edlén's +15 °C and +30 °C (Ciddor, 1996; Edlén, 1966). Air temperature values can easily be outside the second domain, as the surface temperatures

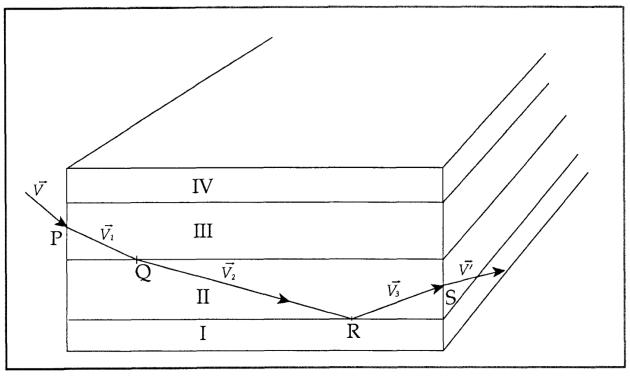


Figure 3.4: The rendering of an inferior mirage with a layer-based approach. A view ray V is refracted at points P, Q, R, S as layers have different indices of refraction. Total internal reflection takes place at R. Image obtained from Berger et al. (1990).

of materials above which inferior mirages sometimes are observable (e.g., desert sand and asphalt) have been measured outside this range (Buxton, 1924; Kallas, 1966). Ciddor's method also has a smaller error bound than Gutierrez et al.'s method (2-5E-10 vs. 1E-3) (Ciddor, 1996; Gutierrez et al., 2006), giving it the smallest error bound from all methods discussed in this work of which the error bound is known. The disadvantage of Ciddor's method is the computational complexity: even when all parameters except for temperature are considered constant, it still takes eight sequential steps to calculate the index of refraction; for reference, Edlén's method takes 1 step under the same conditions.

Ciddor's function, like Edlén's function, depends on an external temperature model to turn distance from the surface and surface temperature into an air temperature. The first option would be to use Equation (3.5) and exponential decay to model the relation between temperature and height. A second option would be to use a function based on a heat transfer model as described in Section 3.1.

To compare the methods for refractive index calculation more effectively based on accuracy and run time, an explicit analysis is performed in Section 4.1.2 to find the function that best fits the goal of our method: the real-time dynamic rendering of mirages.

3.2 Rendering mirages

With a realistic temperature gradient and profile for the index of refraction of air, it is possible to render mirages via a ray marching process, which is a process where the ray path is computed one step at the time at position P_i and where the next position $P_{i+1} = P_i + t \cdot \hat{R}_i$ with $\hat{R}_i$ being the refraction vector at position P_i and t the length of the step. Here, we will investigate previous work.

Layered mesh Berger, Trout & Levit were the first to implement atmospheric effects such as inferior and superior mirages in a ray tracer (Berger et al., 1990). They represented

layers as invisible geometry, where changes in the refractive index occur. The layers are horizontal as they should correspond to air layers with different temperatures. Visualizing the mirage is then achieved by calculating the ray intersections with these different layers while changing the view ray's direction at each layer intersection according to Snell's law (Equation (3.10)), as depicted in Figure 3.4. However, the authors deemed the mirages too clean-looking, which moved them to add noise to the direction of the ray leaving the virtual object and white noise to the final image (Berger et al., 1990).

$$\frac{\sin(\theta_2)}{\sin(\theta_1)} = \frac{v_2}{v_1} = \frac{n_1}{n_2} \tag{3.10}$$

While this is a good starting point, several problems need to be resolved. First, the method does not work well with surfaces that are not perfectly horizontal due to the horizontal layers and the layer's normals being parallel to the vertical world axis (Berger et al., 1990). This constraint is a disadvantage, but not one unique to this approach, as multiple methods assume a constant, vertical temperature gradient when rendering mirages (Seron et al., 2005; Mo et al., 2015). It is a problem nonetheless, as mirages above slanted surfaces can be more precise if a slanted gradient, which has the direction of the surface normal, is used instead.

A second problem is the placement of the layered boxes in the scene. Mirages often appear in larger scenes since they are only visible under a minimal angle between the view ray and the surface. Consequently, a high number of boxes may need to be placed, which increases with more obstacles and more variation in surface height in the scene. Furthermore, updating the boxes automatically due to changes in the scene in real-time seems impossible, considering that a similar mesh consisting of tetrahedra instead of boxes needed to be precomputed (Mo et al., 2015). Consequently, this approach is only suitable for static scenes.

The largest issue is, however, that the method is not compatible with the standard rasterization pipeline, as typically no information is available about geometry after the first intersection, making successfully intersecting a view ray from the camera with multiple layers difficult, if not impossible.

Adaptive mesh The idea of storing heterogeneous media within a mesh has evolved beyond the use of layered boxes. Using a tetrahedral mesh instead of horizontal boxes makes it easier to work with non-flat surfaces (Cao et al., 2010; Mo et al., 2015). Mirages could then again be visualized with a ray tracer, where the rays now intersect tetrahedra instead of boxes. The tetrahedron's vertices can be assigned different refractive indices, leading to a gradient within every tetrahedron along which the index values change (Mo et al., 2015). The ray path within a tetrahedron with such a constant index-gradient has a closed-form solution (Cao et al., 2010), meaning that an analytic ray equation exists to compute the ray path per tetrahedron. Calculating the full ray path would then entail computing the ray path per tetrahedron until it exits the mesh. Unfortunately, the transcendental function in the ray formulation prevents an analytic calculation of the intersection point between the ray and a side of the tetrahedron (Cao et al., 2010). The bisection method can be used for intersection computation, which is slower than an analytic solution and might even require building an

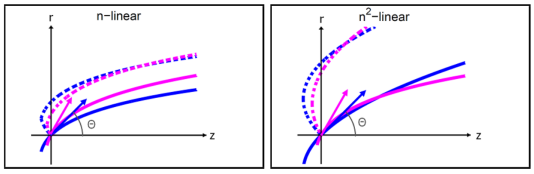


Figure 3.5: Analytic ray curves in n- and n^2linear media profiles. The pink and blue curves have a launch angle of 60° and 45°, respectively. The dashed curves are obtained by mirroring the launch angle of the same-colored curve in the r-axis, which is parallel to the surface in the case of mirage formation. The dashed curves lead to inferior mirages, as z in this image is the direction of the index gradient. Notice how the ray curves differ significantly between both images while the angle between the gradient and the original ray direction are the same. Image obtained by adapting an image from Mo et al. (2015).

acceleration structure to speed it up (Cao et al., 2010). Later, a different ray formulation has been used in the context of mirage rendering, which does have a closed-form solution for the intersection with planar elements when the gradient is constant in the squared refractive index instead of in the refractive index (Mo et al., 2015). The subsequent advantage of faster testing for intersection comes with the disadvantage that the ray curve is going to be wildly different because of this different gradient (Mo et al., 2015), as shown in Figure 3.5.

Another problem, one that prevents real-time dynamic mirage rendering, is that the tetrahedral mesh needs to be precomputed (Cao et al., 2010; Mo et al., 2015). For this and aforementioned reasons, meshes do not seem ideal for the real-time dynamic rendering of mirages. However, Cao et al.'s work also listed that mirages could be considered a particular case of gradient-index optics (Cao et al., 2010). They represented the earth surface with connected quads, and each quad would result in a different frustum as shown in Figure 3.6. The normal of the quad would be used as the gradient direction within the frustum. The ray path can then be calculated by applying the ray equations at every frustum, of which there typically would be fewer than ten (Cao et al., 2010).

Subdividing the view frustum into so few frustums will not be enough to render mirages above surfaces with much local relief, but this approach allows for non-vertical and differing temperature and index-gradients and, therefore, we build upon this method.

Improvements to ray marching The previously discussed mesh-based approaches effectively use ray marching to render mirages or atmospheric refraction in general while utilizing the mesh to limit the number of ray-marching steps and store refractive index data. Alternative approaches reduce the total ray-marching steps without using a mesh by reducing and increasing the step size if there is more and less atmospheric refraction, respectively (Seron et al., 2005; Gutierrez et al., 2006). These approaches require less memory since no mesh no geometric data needs to be loaded and work in dynamic scenes, as no preconstructed mesh is required. The air's refractive index is often obtained with a function that transforms distance from the surface into an index.

The methods work with an estimated error that is calculated every step and a user-set

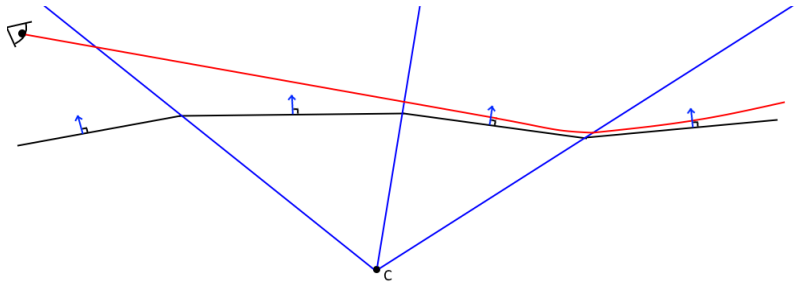


Figure 3.6: A 2D view of how a ray's direction changes once per frustum, as the analytic ray equations are used once every frustum to compute the next part of the ray path. The ray is seemingly not affected by refraction in the first two frustums, as the ray is too far from the surface.

tolerance; if the error is too high compared to the tolerance, more, smaller steps are taken to better approximate the ray path. Generally, the error is estimated by comparing the computed coordinates of the next point on the path and the coordinates of an approximating point. For instance, Seron et al. estimate the error by computing point P_i twice, with step size t and step size $\frac{t}{2}$, and $P_{i,t/2}$ becomes the next position if the estimated error is smaller than a set tolerance (Seron et al., 2005). Accuracy and efficiency can be further improved by switching from ray marching to a more efficient form of numerical integration to solve the ray path equation. Ray marching is essentially an application of the first-order Euler method to the ray path equation (Seron et al., 2005). The Dormand-Prince method is a 4th/5th-order member of the family of Runge-Kutta methods that converges much faster than other methods: "In terms of convergence, the chosen Dormand–Prince method is orders of magnitude faster than simpler numerical resolution methods like Euler" (Gutierrez et al., 2006, p. 1005). Using numeric integration methods over analytic solutions as discussed in Section 3.2 has the advantage that they are general and can be applied across arbitrary atmospheric conditions. In contrast, analytic solutions depend on the chosen index of refraction function and may not even be an option if the refractive index function is too complex to obtain an analytic solution for the ray path.

However, the generality can also be a drawback, as it prevents taking advantage of having more information about a temperature or refractive index model. For instance, Seron et al. use exponential decay (Equation (3.5)) to model air temperatures near the surface, but their general method does not use this information to perform smaller steps closer to the surface. Consequently, the aforementioned layer-based approach, a method designed solely for rendering mirages, proved to be faster and more accurate than the Euler method, which reported run times in the order of minutes (Seron et al., 2005). The Dormand-Prince method did not report real-time run times either: rendering a 400×800 frame containing a mirage took 4 to 5 minutes on the CPU (Gutierrez et al., 2006). Although these times are undoubtedly shorter now due to hardware improvements and may be reduced further by increasing the tolerance, real-time performance remains uncertain. Therefore, analytic approaches seem the better solution for mirage rendering, or at least in the case of (rela-

tively) flat surfaces and under the condition that analytic solutions are indeed available. If there is relatively much height deviation in the scene, calculating the ray path with the analytic function is more difficult without well-defined temperature gradients. In that situation, numerical integration of the ray path equation may give more accurate results at the cost of a potentially higher run time, which could possibly be lowered by using temperature or refractive index model information to alter the step size more efficiently.

Heat visualization Simulating the heat flow in the scene is another means to visualize not only mirages but heat shimmering too (Zhao et al., 2006). Temperature textures can be used to (indirectly) measure the temperature on object surfaces, and a large 3D grid is used to simulate the heat flow in the air due to conduction and convection by updating the air temperature at the grid points. The advantage of actual simulation of thermodynamics is that realistic, dynamic heat shimmering is obtainable, while mesh-based methods have to rely on approximations such as adding random noise to the light ray (Berger et al., 1990). These alternative approaches do not respond dynamically to scene changes and may lack accuracy even in static scenes if noise textures are created without heat flow simulation.

The downside of heat visualization through simulation is the high run time. A grid needs to be updated every timestep, and the grid needs to be large since mirages can appear anywhere in the view frustum. As a result, the simulation alone can prevent interactive framerates, and the rendering (using ray marching) increases the run time even further (Zhao et al., 2006). A voxelization of the scene objects is required (Zhao et al., 2006), which can be obtained relatively fast dynamically (Crassin et al., 2011), but which increases the run time further. As a result, achieving real-time dynamic mirages with heat simulation seems unlikely, prompting us not to build upon this method of mirage rendering in our work.

Nonetheless, the concept of temperature textures introduced together with this method can be used to solve one big problem with mirage rendering in the rasterization pipeline: storing and reading the surface temperature efficiently. Zhao et al. describe two different methods for obtaining surface temperature information: either the temperature is directly stored in a new texture, or the temperature is indirectly obtainable by storing multiple parameters for the sun and object materials; calculating the solar energy on an approximation mesh; and projecting the radiation on the actual object surface to be able to calculate the heat and equilibrium temperature (Zhao et al., 2006). The second approach is depicted in Figure 3.7. The authors prefer the former for objects that radiate heat themselves (e.g., a radiator or oven) and the latter for the remaining objects. The first method seems a better fit for a real-time method that needs to work in a standard rasterization pipeline, as storing and reading the surface temperature of an object becomes as easy as rendering models with temperature textures to an image for use in a later rendering pass. This process is similar to deferred rendering, which involves writing geometry information to textures first, so that they can be accessed in a later rendering pass to apply shading to the scene.

The second approach with object materials has the advantages that object data does not have to be altered to change the temperature in the scene and that using material properties may be more intuitive than choosing realistic temperature values directly. While the approximating mesh should be possible to create in real-time with a dynamically created height map, there seems to be no simple way to measure the projection area later in the ras-

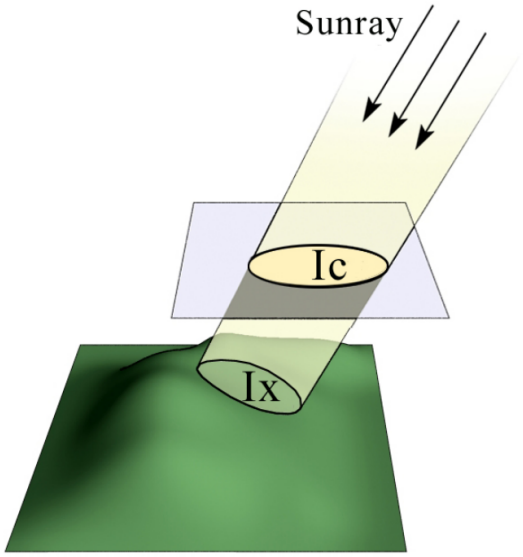


Figure 3.7: The equilibrium surface temperature on the object surface is computed in multiple steps. First, the energy of the solar irradiation I_C on an approximation mesh consisting of stitched planes is computed (here, this mesh consists of only one plane which approximates the actual green object surface). The heat on the surface is subsequently calculated by projection I_C on the surface, giving I_X. Finally, the surface temperature is computed with I_C, the size of the projection of I_X, and surface material attributes. Original figure by Zhao et al. (2006).

terization pipeline. Storing the temperature directly in textures bypasses this issue and also means that one or two rendering passes can be saved since no approximation mesh needs to be constructed. Furthermore, these temperature textures can still be made in an offline environment beforehand, so no artist has to make up temperatures and can instead rely on physical simulation. Therefore, the first approach of storing temperature data directly in the texture seems the better choice in our situation.

Blobs A distinct approach to mirage rendering involves the use of blobs, which are spheres defined by a position, radius, and temperature, to indicate volumes of air with a varying refractive index (Stam and Languénou, 1996). The ray path through these blobs is not computed using ray marching, but with an approximation: all the blobs are assigned a weight based on their respective temperature and distance to the ray origin, after which the ray's end point is approximated using the first two terms of a series that represents the perturbation as shown in Figure 3.8. The first term of this series is the unperturbed ray itself. Blobs change over time by advecting the centers through a wind field and decreasing the temperature over time, and thus the method combines ideas from the mesh-based and physics-based methods. The physical simulation can, of course, be left out to save resources, but that means no turbulence simulation and heat shimmering.

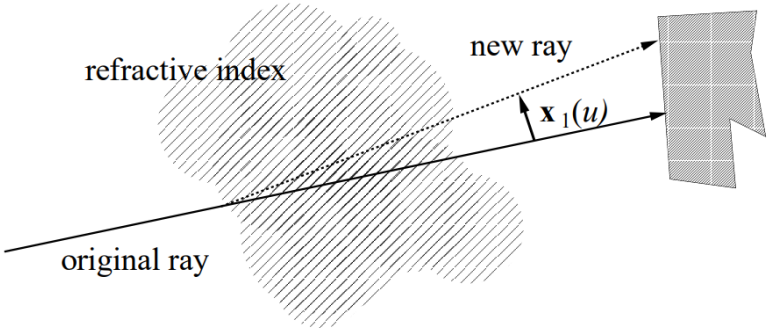


Figure 3.8: The refraction image is, in spite of the presence of many blobs, computed in one step with a linear perturbation. $x_1(u)$ is the first perturbation that partially transforms the original ray into the new, refracted ray. Original figure by Stam and Languénou (1996).

The spheres-based approach could be integrated in the standard rasterization pipeline should be possible without any changes to the method by passing the blob attributes to the GPU as array uniforms, avoiding the need to render them explicitly. The alternative, capturing blob data by rendering blobs to an image, is inadequate, as data of all blobs is required to compute the approximation ray, and blobs could then be occluded or culled. If the method can be adapted so quickly to work with a rasterizer, then it seems strange that no real-time applications have used it to render mirages. One explanation may be that it is more challenging to work with blobs than with, for instance, temperature textures: it seems more natural and less labor-intensive to assign a temperature to a road surface than to place many blobs above the road. Blobs seem better for dealing with objects with no clear boundaries and which are confined to a small area, such as flames or gases. Another reason for the blob's absence, closely related to the previous one, is that blobs are more difficult to place in the scene since they have an index-gradient that starts at the blob center, while mirages are caused by temperature gradients starting at the surface. Finally, the method might be too taxing to include in real-time applications, as the original authors mention that the scene took twice as long to render with refraction compared to the scene without refraction with a run time that does not allow an interactive framerate (Stam and Languénou, 1996). For all these reasons, the blobs-based approach does not meet our requirements.

DriveClub One known example of mirages being rendered in video games is in the video game DriveClub (PS4, 2014). The video game's developers never published how they managed to render mirages in real-time, making it impossible to know the mirages' physical accuracy and how they were obtained. However, based on a private conversation with one of the graphics developers, we can make certain assumptions on how the mirages were achieved; the actual implementation remains unknown due to non-disclosure agreements.

According to the developer, rendering the reflection of mirages can be similar to rendering screen-space reflections caused by specular surfaces, which is why they advised to use this technique to render mirages if they need to be rendered in real-time and dynamically.

While using a screen-space reflections method, the difficulty of rendering mirages lies in knowing where mirages should appear in the scene and what the direction of the 'reflection ray' should be, which is the ray direction vector after reflecting. Since mirages are caused by refraction and total internal reflection, obtaining the reflection ray by reflecting the view ray at the point of intersection between the view ray and the surface leads to an incorrect value for the reflection ray. However, DriveClub likely still ignored refraction under the assumption that the difference is unnoticeable at the large angles that come to play with mirages. This approach avoids a possibly costly computation of the nonlinear ray path, resulting in a faster approximation that also simplifies intersection testing. The developers likely used heuristics such as surface type, surface albedo, surface normal, view direction, and distance from the camera instead of using the aforementioned temperature textures to deduce the existence of a strong enough temperature gradient that can result in mirages. For instance, mirages could be made to appear on roads but not on grass, or only on darker surfaces, based on the notion that darker objects get hotter. General information about the scene state such as the time of day, ambient temperature, and the amount of overcast caused by clouds, are then to be combined with the previously mentioned object heuristics to modulate the amount of mirage formation in the scene. When a mirage is ought to spawn according to all this data, the G-buffer material values can be changed to those of a specular material, so that a reflection will appear that resembles a mirage. The developer recommended for a new implementation to start simple ("Maybe just enable it for dark surfaces [..] with near vertical normals, in a good distance window from the camera, at very shallow view angles [..]") and to iterate further from there to improve the look and make it robust.

The implementation of mirages as just has been described may lead to inaccuracies: the approach of changing several parameters until the mirages seem plausible can easily result in inaccurate mirage formations, based on the (likely limited) knowledge of the graphics programmer on the subject. Figure 3.9 shows one frame of the video game containing an inaccurate mirage already, as the mirage appears to be slightly transparent. Mirages are never transparent, as a light ray reaches the eye through total internal reflection or it does not. There are two different explanations for this inaccuracy. First, the difference in color between the object and the mirage could be caused by the changing of the G-buffer values to make the road material reflective to show mirages. A reflective surface directly impacts the reflection (e.g., the reflection on a metal surface has a different color than the reflection of the same object on a mirror), but this should not happen with mirages as total internal reflection causes the reflection and not the surface. The consequence is that the color of the road may here falsely change the color of the mirage in DriveClub. Another possible explanation for the transparent mirages could be linear interpolation to smooth the borders between the mirage and surface. Surface reflections are not as binary as mirages (which fully appear or not at all), as the reflection's sharpness depends on the fraction of light rays that reflect towards the camera. Therefore, linearly interpolating between the surface and reflection can make sense when the reflection is caused by the surface, but not when it is caused by total internal reflection. The DriveClub developer's advice to avoid harsh edges by "lerping modified G-buffer values between original and desired values with relation to various parameters" may be exactly what causes the issue here.

In short, it is hard to judge whether the mirages obtained with the DriveClub's approach

are accurate, since we only have in-game footage to go by. However, some issues already seem to be caused by treating total internal reflection as a more usual specular or diffuse reflection. Nonetheless, using a screen-space reflection method to render the image of the mirages is an idea that can be incorporated in our methods. However, to prevent the aforementioned issues that may have come with ignoring refraction, we require some changes to the standard approach to screen-space reflections in order to make them fit for total internal reflections.

Figure 3.9: A frame from the video game DriveClub displaying a mirage. Source: (Nelva, 2015).

To the best of our knowledge, the methods discussed in this chapter constitute the most important research to date in the field of mirage rendering and are most relevant to the methods presented in this work.

Chapter 4

Index of refraction computation

In this chapter, we evaluate the different temperature models and functions for the refractive index of air, so that we choose the function that gives the best trade-off in accuracy and speed for our method.

4.1 Index of refraction computation

Section 3.1 evaluated different refractive index functions based on their scientific basis, meaning this section focuses on the accuracy and run time of the different refractive index functions. Since air temperature is the most important parameter in these equations to determine mirage formation, we begin with a numeric evaluation of the two different air temperature models discussed in Section 3.1.

4.1.1 Air temperature models

The first temperature model uses distance from the surface h, surface temperature T_s and drop-off length d_0 to have the air temperature decay exponentially, as depicted in Equation (3.5). The second temperature model is based on boundary-layer theory and introduces a logarithmic temperature decay. The specific model Young utilizes to apply boundary-layer theory has as input parameters the thickness of the laminar layer y_t; the change in temperature per increase of e in height above the laminar layer Θ; and the Obukhov length L, which is proportional to the thickness of the surface layer.

The quality of the two models can be assessed by comparing the air temperature at different distances from the surface with empirical data (Wakid, 2019). Since both functions model a nonlinear relation between height and air temperature, we used nonlinear least squares to fit both models to the data. We fit both unconstrained and constrained, as the former produced questionable values, such as near zero values for L that ought to be be between -10 and -100 instead (Young, 2015). Finally, we also compared the models' mean squared errors (MSEs) using parameter values provided by different authors (Stam and Languénou, 1996; van der Werf, 2011; Young, 2015). The results are presented in Tables 4.1 and 4.2.

	$T(0.00)$	$T(0.01)$	$T(0.03)$	$T(0.13)$	$T(0.55)$	$T(0.89)$	MSE	MSE (to 0.13 m)	Parameters
T_s = 63 °C	33.5	31.31	30.45	29.15	29.5	29.5	.28802	1.932	d_0 = 0.02
	33.5	30.69	29.91	29.5	29.5	29.5	2.9477	3.4216	d_0 = 0.01
	33.5	31.28	30.13	29.71	29.5	29.5	.45182	.17774	d_0 = 0.017 (fitted)
	33.5	31.43	30.03	29.76	29.5	29.5	.07922	.61883	d_0 = 0.030, a = 5.546 (fitted)
	33.5	31.66	29.44	29.63	29.5	29.5	.1721	.75815	d_0 = 0.016 (T_s = 48 °C fitted)
	33.5	31.38	29.8	29.51	29.5	29.5	1.3671	1.5506	d_0 = 0.012 (T_s = 32 °C fitted)
T_s = 48 °C	32.5	31.33	29.98	29.92	29.5	29.5	.38259	.94889	d_0 = 0.02
	32.5	30.51	29.44	29.5	29.5	29.5	.93639	1.2796	d_0 = 0.01
	32.5	30.37	29.44	29.58	29.5	29.5	.12111	.55667	d_0 = 0.016 (fitted)
	32.5	30.78	29.54	29.6	29.5	29.5	.9735	.33525	d_0 = 0.037, a = 9.522 (fitted)
	32.5	30.99	29.45	29.63	29.5	29.5	.21932	.70399	d_0 = 0.017 (T_s = 63 °C fitted)
	32.5	30.19	29.48	29.51	29.5	29.5	.40561	.48341	d_0 = 0.012 (T_s = 32 °C fitted)
T_s = 32 °C	30.5	30.28	29.28	29.68	29.5	29.5	.72882	.59323	d_0 = 0.02
	30.5	29.65	29.47	29.5	29.5	29.5	.25772	.88658	d_0 = 0.01
	30.5	29.7	29.77	29.5	29.5	29.5	.47721	.71582	d_0 = 0.014, a = 3.484
	30.5	29.65	29.35	29.5	29.5	29.5	.49408	.74113	d_0 = 0.012 (fitted)
	30.5	29.14	29.05	29.56	29.5	29.5	.8766	.31489	d_0 = 0.017 (T_s = 63 °C fitted)
	30.5	29.73	29.9	29.54	29.5	29.5	.96091	.44136	d_0 = 0.016 (T_s = 48 °C fitted)

Table 4.1: The mean squared errors obtained by comparing the temperatures observed by Wakid (2019) with the temperatures produced with the exponential decay temperature model, which is defined by Equation (3.5). a potentially replaces e as base in Equation (3.5). The computed air temperatures $T(h)$ are in degrees Celsius with h being the distance from the surface in meters.

Exponential decay model Table 4.1 shows that the drop-off lengths that give lower MSEs increase nonlinearly with $T_s - T_a$, with d_0 here being 0.012 m (T_s = 32 °C), 0.016 m (T_s = 48 °C) , and 0.017 m (T_s = 63 °C). These values align with the estimates found in literature (0.01 m and 0.02 m) (Stam and Languénou, 1996; van der Werf, 2011), validating those as reasonable estimates. Fitting the exponential base in Equation (3.5) led to fairly different bases, increased ranges of drop-off length, and barely any reduction of the MSE, justifying choosing for a constant base e to limit the number of variables.

boundary-layer-theory model Table 4.2 shows lower MSEs from the the boundary-layer-theory model than the exponential decay model for the 63 °and 48 °C data sets if L is unconstrained. Constraining L to the predefined range (-100 ¡ L ¡ -10) (Young, 2015), the MSEs yielded are higher than those found with the exponential decay model with optimal configurations. The unusual small $|L|$, which indicates very strong convection (Young, 2015), might be caused by the laboratory setting of the experiment and the different atmospheric conditions experienced there and not outside. We also expected L to be constant, since L is proportional to the thickness of the surface layer, which is essentially the same for varying temperatures (Wakid, 2019). Attempts to find a constant L that resulted in low MSEs for all three surface temperatures failed, though swapping the fitted L values for the 48 °C and 63 °C sets still resulted in lower MSEs than those found with the first method for the same surface temperatures.

We did not expect Θ to be constant and to change with varying temperature differences, as we stated before that "Θ is the change in potential temperature for each factor of e change

	$T(0.00)$	$T(0.01)$	$T(0.03)$	$T(0.13)$	$T(0.55)$	$T(0.89)$	Parameters	MSE	MSE (to 0.13 m)
T_s = 63 °C	63.0	40.374	32.049	24.671	20.768	19.954	L = -0.185, Θ = 10.875 (fitted)	1.167743	1.73763
	63.0	45.952	37.883	27.396	18.073	15.401	L = -10.0, Θ = 7.420 (fitted. L ¡= -10.0)	18.86982	20.8752
	63.0	47.735	42.405	37.857	35.505	35.018	L = -0.136. Θ = 7.562 (T_s = 48 °C fitted parameters)	131.3353	88.01917
	63.0	55.196	53.111	51.532	50.76	50.603	L = -0.032. Θ = 4.973 (T_s = 32 °C fitted parameters)	539.0668	353.0499
	63.0	47.267	41.478	36.347	33.633	33.068	L = -0.185, Θ = 7.562	103.1009	72.06005
	63.0	52.654	48.846	45.472	43.688	43.316	L = -0.185, Θ = 4.973	324.3606	221.9456
	63.0	41.047	33.382	26.842	23.459	22.758	L = -0.136, Θ = 10.875	4.898724	3.934774
	63.0	45.935	41.374	37.921	36.235	35.891	L = -0.032, Θ = 10.875	132.7389	77.95109
	63.0	41.409	34.064	27.9	24.739	24.086	L = -0.118, Θ = 10.875 (mean L)	9.325149	6.318844
T_s = 48 °C	48.0	32.735	27.405	22.857	20.505	20.018	L = -0.136, Θ = 7.562 (fitted)	0.042508	0.063675
	48.0	36.829	31.542	24.67	18.561	16.811	L = -10, Θ = 4.862 (fitted. L ¡= -10.0)	8.668225	9.519976
	48.0	40.196	38.111	36.532	35.76	35.603	L = -0.032, Θ = 4.973 (T_s = 32 °C fitted parameters)	139.1122	89.58794
	48.0	25.374	17.049	9.671	5.768	4.954	L = -0.185, Θ = 10.875 (T_s = 63 °C fitted parameters)	129.7091	83.71013
	48.0	36.134	32.962	30.561	29.388	29.149	L = -0.032, Θ = 7.562	44.20614	25.63403
	48.0	32.267	26.478	21.347	18.633	18.068	L = -0.185, Θ = 7.562	1.793416	0.885545
	48.0	32.987	27.879	23.593	21.395	20.941	L = -0.118, Θ = 7.562 (mean L)	0.468494	0.281115
T_s = 32 °C	32.0	24.196	22.111	20.532	19.76	19.603	L = -0.032, Θ = 4.973 (fitted)	0.349495	0.47044
	32.0	26.955	24.566	21.463	18.704	17.913	L = -10.0, Θ = 2.196 (fitted. L ¡= -10.0)	3.916823	4.366438
	32.0	16.735	11.405	6.857	4.505	4.018	L = -0.136, Θ = 7.562 (T_s = 48 °C fitted parameters)	140.3375	86.62642
	32.0	9.374	1.049	-6.329	-10.232	-11.046	L = -0.185, Θ = 10.875 (T_s = 63 °C fitted parameters)	539.9729	340.5024
	32.0	21.961	18.456	15.465	13.918	13.598	L = -0.136, Θ = 4.973	19.83917	10.26467
	32.0	21.654	17.846	14.472	12.688	12.316	L = -0.185, Θ = 4.973	28.33157	14.37005
	32.0	22.127	18.768	15.949	14.504	14.205	L = -0.118, Θ = 4.973 (mean L)	16.28927	8.486888

Table 4.2: The mean squared errors obtained by comparing the temperatures observed by Wakid (2019) with the temperatures produced with the boundary-layer-theory model, as implemented by Young (2015). The parameters are obtained by adjusting the L and Θ parameters to minimize the mean squared error per data set. These ideal parameters are used for different data sets as well to see if the error remains within an acceptable range. The computed air temperatures $T(h)$ are in degrees Celsius with h being the distance from the surface in meters.

in height above y_t" (Young, 2015, p. 172). Finding the optimal Θ seems complicated due to the wide range it can seemingly span (2.916 to 10.875 for the surface temperatures used in Table 4.2). Determining optimal values for Θ and L requires access to temperature data that is often not available to achieve the lowest MSEs. Therefore, we expect the simpler exponential decay model, which only has 1 parameter with a seemingly more limited range, to outperform the complex boundary-layer-theory model in practice, making it our choice for the temperature model.

4.1.2 Numeric evaluation of refractive index functions

Speed We measured the speed of the different refractive index functions by computing the mean run time of 1,000,000 function calls. The computations were performed on the CPU in a custom C++ application with compiler optimizations disabled; the function would in the end be called on the GPU instead, but timing just the function is more difficult on the GPU, and we expect all tested functions to react similarly to the change in environment. The results are visible in Table 4.3.

Refraction index function	Equation(s)	Temperature model	Total run time (s)
Height-based function (Khular et al. (1977))	Khular et al. (1977)	-	0.193975
Zhao et al. (2006)	(3.3)	Exponential decay	0.202452
Zhao et al. (2006)	(3.3)	boundary-layer theory	0.281047
Minnaert (1993)	(3.4)	Exponential decay	0.192596
van der Werf (2011)	(3.9)	Exponential decay	0.202554
Edlén (1966), Birch and Downs (1994)	(3.1), (3.2)	Exponential decay	0.203804
Edlén (1966), Birch and Downs (1994)	(3.1), (3.2)	boundary-layer theory	0.271231
Gutierrez et al. (2006)	(3.6), (3.7), (3.8)	Exponential decay	0.255265
Ciddor (1996)	Ciddor (1996)	Exponential decay	0.435236
Ciddor (1996)	Ciddor (1996)	boundary-layer theory	0.526027

Table 4.3: The total time it takes to run 1 million iterations of various combinations of refraction index functions and temperature model functions.

Table 4.3 shows that all configurations can easily be run in real-time on the CPU, as the mean run time of each tested configuration is below a millionth of a second. Nonetheless, some configurations seem much faster than others. For instance, the two functions that are tested with both temperature models are, on average, 35.9% faster when using the exponential decay temperature model. Furthermore, Ciddor's function takes approximately twice as long to run as Edlén's function, and Gutierrez's function has a longer run time of 0.255 s per million iterations compared to the other functions using the same temperature model ($\tilde{0}$.20 s/million calls).

Accuracy The metrics we use to measure the accuracy of the different refraction index methods are the smallest angle between the surface normal and viewer at which mirages appear, and the mean squared error when compared to a baseline function.

Real-life experiments show an average critical angle of 89.5° (Wakid, 2019), so refractive index functions should lead to similar angles at which mirages first appear. We used the bisection method to find the smallest angles at which mirages were visible for different combinations of refraction index functions and temperature models. Since all functions together have ten different parameters, we kept parameters with supposedly little influence on the mirage formation constant (e.g., air pressure, light wavelength), as well as those which authors considered constant (e.g., thermal roughness for the boundary-layer theory-based temperature model). The only parameters that we changed are the surface temperature T_s, the ambient temperature T_a, the angle θ, and the temperature parameters L, Θ, and d_0. With the results from Section 4.1.1, choosing realistic parameters for the temperature models is straightforward. The results are in Table 4.4.

Refraction index formula	Equation(s)	Temperature model	Intersection distance without refraction (m)	Angle θ (°)	T_s (°C)	y_F (m)	$n(y_0)$	$n(y_F)$	Temperature model parameters
Height-based function (Khular et al. (1977))	Khular et al. (1977)	-	2.3	66.501	-	3.21E-06	1.09068	1.00023	-
Zhao et al. (2006)	(3.3)	Exponential decay	3.45596	73.862	32	7.48E-06	0.982309	0.943601	d_0 = 0.012
Zhao et al. (2006)	(3.3)	Exponential decay	2.23236	65.870	48	7.29E-06	0.982309	0.896473	d_0 = 0.016
Zhao et al. (2006)	(3.3)	Exponential decay	1.77917	60.661	63	1.22E-08	0.982309	0.856318	d_0 = 0.017
Zhao et al. (2006)	(3.3)	boundary-layer theory	3.39722	73.598	32	1.01E-03	0.983755	0.94367	Θ = 4.973, L = -0.032
Zhao et al. (2006)	(3.3)	boundary-layer theory	2.229	65.837	48	1.00E-03	0.982592	0.896486	Θ = 7.562, L = -0.136
Zhao et al. (2006)	(3.3)	boundary-layer theory	1.77582	60.615	63	1.01E-03	0.983034	0.856542	Θ = 10.875, L = -0.185
Minnaert (1993)	(3.4)	Exponential decay	201.55	89.716	32	1.69E-03	1.00022	1.00021	d_0 = 1.304
Minnaert (1993)	(3.4)	Exponential decay	134.964	89.575	48	5.06E-04	1.00022	1.0002	d_0 = 0.579
Minnaert (1993)	(3.4)	Exponential decay	107.6	89.468	63	8.81E-05	1.00023	1.00019	d_0 = 0.067
Minnaert (1993)	(3.4)	Exponential decay	244.343	89.766	32	4.17E-05	1.00021	1.00021	d_0 = 0.012
Minnaert (1993)	(3.4)	Exponential decay	163.838	89.650	48	2.40E-05	1.00021	1.0002	d_0 = 0.016
Minnaert (1993)	(3.4)	Exponential decay	135.26	89.576	63	3.28E-05	1.00021	1.00019	d_0 = 0.017
van der Werf (2011)	(3.9)	Exponential decay	216.78	89.736	32	4.25E-05	1.00027	1.00026	d_0 = 0.012
van der Werf (2011)	(3.9)	Exponential decay	145.451	89.606	48	4.83E-05	1.00027	1.00025	d_0 = 0.016
van der Werf (2011)	(3.9)	Exponential decay	120.001	89.523	63	3.45E-05	1.00027	1.00024	d_0 = 0.017
Edlén (1966), Birch and Downs (1994)	(3.1), (3.2)	Exponential decay	216.78	89.736	32	1.03E-04	1.00027	1.00026	d_0 = 0.012
Edlén (1966), Birch and Downs (1994)	(3.1), (3.2)	Exponential decay	145.451	89.606	48	7.82E-05	1.00027	1.00025	d_0 = 0.016
Edlén (1966), Birch and Downs (1994)	(3.1), (3.2)	Exponential decay	119.795	89.522	63	2.26E-05	1.00027	1.00024	d_0 = 0.017
Edlén (1966), Birch and Downs (1994)	(3.1), (3.2)	boundary-layer theory	212.083	89.730	32	1.02E-03	1.00027	1.00026	Θ = 4.973, L = -0.032
Edlén (1966), Birch and Downs (1994)	(3.1), (3.2)	boundary-layer theory	145.086	89.605	48	1.02E-03	1.00027	1.00025	Θ = 7.562, L = -0.136
Edlén (1966), Birch and Downs (1994)	(3.1), (3.2)	boundary-layer theory	119.453	89.520	63	1.00E-03	1.00027	1.00024	Θ = 10.875, L = -0.185
Gutierrez et al. (2006)	(3.6), (3.7), (3.8)	Exponential decay	189.944	89.698	32	4.83E-05	1.00035	1.00034	d_0 = 0.012
Gutierrez et al. (2006)	(3.6), (3.7), (3.8)	Exponential decay	127.685	89.551	48	4.22E-05	1.00035	1.00032	d_0 = 0.016
Gutierrez et al. (2006)	(3.6), (3.7), (3.8)	Exponential decay	105.439	89.457	63	3.93E-05	1.00035	1.00031	d_0 = 0.017
Ciddor (1996)	Ciddor (1996)	Exponential decay	-	-	32	-	1.00027	-	d_0 = 0.012
Ciddor (1996)	Ciddor (1996)	Exponential decay	-	-	48	-	1.00027	-	d_0 = 0.016
Ciddor (1996)	Ciddor (1996)	Exponential decay	-	-	63	-	1.00027	-	d_0 = 0.017
Ciddor (1996)	Ciddor (1996)	boundary-layer theory	209.375	89.726	32	1.00E-03	1.00028	1.00026	Θ = 4.973, L = -0.032
Ciddor (1996)	Ciddor (1996)	boundary-layer theory	142.257	89.597	48	1.03E-03	1.003	1.00025	Θ = 7.562, L = -0.136
Ciddor (1996)	Ciddor (1996)	boundary-layer theory	116.406	89.508	63	1.01E-03	1.00032	1.00023	Θ = 10.875, L = -0.185

Table 4.4: The smallest angles at which mirages are visible for different formulae for the index of refraction of air, different temperature models and different surface temperatures T_s. The angles have been determined with the bisection method. The folding point height y_F is obtained with Equation (5.3), or by refracting iteratively in the case of the refractive index formula of Ciddor (1996). y_0 = 1.0 m for all configurations, and y_t = 0.001 m for every configuration that uses the boundary-layer-theory temperature model. All other temperature model parameters were obtained by fitting the models to temperature data. T_a = 20 °C.

From Table 4.4 it becomes clear that all functions except those by Khular et al. and Zhao et al. lead to mirages at angles close to or bigger than the aforementioned critical angle of 89.5°and to indices approximately equal to 1.0002 or 1.0003. Khular et al.'s functions, with a critical angle of 66.5°, would even lead to mirages visible 2.3 meters away when observed at a height of 1 meters. The refractive index function of Zhao et al. (Equation (3.3)) results in false positives as well, likely due to known errors, as discussed in Appendix B.

A difference caused by the different temperature models is the folding point height y_f possibly being much lower with the exponential decay temperature model than with the boundary-layer theory-based model. This difference is by design: the logarithmic temperature decay should only happen above the thermal roughness y_t (here equal to 0.001 m), which is not used in Equation (3.5), making it possible for y_F to be smaller than y_t.

We compared the results with those of a baseline function to further judge the accuracy of the different methods. Ciddor's method is supposed to be the most accurate method due to using the most parameters and giving accurate index values for a larger range of parameters than the method of Edlén and Birch & Downs (Ciddor, 1996), making it our choice for the baseline function, given that Ciddor's method also is slower and less functional because of the many more computation steps it has (7 steps more than Edlén's method if temperature is the only variable). We measured the mean squared error for all functions in Table 4.4 except for the height-based function, as that function does not have temperature as input. From the results in Table 4.4, Edlén's function (Equations (3.1) and (3.2)) seems to work particularly well (MSE = 8.207 with the boundary-layer-theory model, MSE = 25.507 with the exponential decay model). The same holds for Van der Werf's function (Equation (3.9)) (MSE = 25.987, exponential decal model). Since the baseline function used the boundary-layer-theory temperature model, the bigger error is partially explained by having used a different model. Gutierrez's function (Equations (3.6), (3.7) and (3.8)) with the exponential decay temperature model resulted in a high MSE of 236.727, which is unexpected as the function is quite similar to that of Van der Werf, and both are based on the gas law.

Unexpectedly, Ciddor's function for the refractive index combined with the exponential decay model does not produces mirages, which is peculiar as both Ciddor's function and the exponential decay temperature model lead to mirages in different configurations.

Overall, Edlén's function (Equations (3.1) and (3.2)) and Van der Werf's function (Equation (3.9)) (both with with exponential decay model) seem to be the best functions to obtain the refractive index of air within a mirage rendering context, as they have similar run times and accuracy, meaning that there is no favorite between the two. Opting for a faster function (the height-based function or Stam's function) comes with a significant drop in accuracy, while choosing a more accurate function (Edlén's function with boundary-layer-theory model) comes with a significant increase in run time and a difficult-to-configure temperature model. We chose to use Van der Werf's function due to its simplicity, which made computing y_F also easier.

Chapter 5

Method

In this chapter, we explain our method and how it can render mirages in real-time dynamically.

5.1 Overview

So far, we have identified three major hurdles for real-time mirage rendering:

1. The rasterization pipeline does not offer general scene access without a performance hit;

2. nonlinear ray traversal;

3. Incorrect index of refraction calculation.

Our main observation is that these hurdles can be overcome by combining ideas from different offline mirage rendering methods: temperature textures make access to temperature data in the rasterization pipeline more feasible, nonlinear ray traversal can be sped up by using analytic ray equations, and functions for the air's index of refraction are available that are more accurate. The main idea is then to use the ray equations with the index of refraction function as input to compute the ray path above a surface in as few steps as possible. The temperature textures are used to obtain surface temperature information, and after an ambient temperature is provided, air temperatures and indices of refraction can be determined using the temperature model and refractive index function. If enough refraction takes place for a mirage to form, then the mirage can be rendered similarly to how specular reflections are rendered in the rasterization pipeline.

While using analytic ray equations over ray marching can lead to big performance gains (Cao et al., 2010), these equations assume media with a constant refractive index-gradient, i.e., planar surfaces with a constant temperature. We take inspiration from Cao et al., who statically created a small number of frustums by extending vertices from the ground up (Cao et al., 2010), subdividing the surface into smaller planes, but instead we require an approach for dynamic scenes that can also dynamically vary the number of planes.

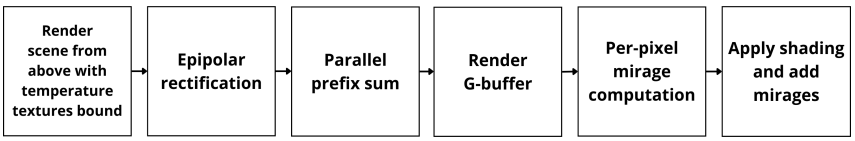


Figure 5.1: A general overview of the different steps in the main method.

If we limit refraction to the plane that holds the view ray and the world up-vector, the ray will stay in the same plane while refracting, and all rays that go over the same part of the surface will remain in the same plane, enabling us to parallelize the processing of surface data and to only compute surface data per plane as opposed to per ray. The surface data that needs to be processed to know the surface's impact on the view ray in that plane would be the surface temperature and the surface normal; the surface's position can be obtained later in the fragment shader through the G-buffer. The mean averages of these two attributes are sufficient to compute plane surfaces below the view ray that approximate the actual surface. The performance advantage lies in that these averages can be obtained for any part of the surface below the view ray efficiently in the fragment shader by computing the parallel prefix sum before on the surface data that has been rectified to have the surface data per view ray in separate rows of an image. After, when we try to find mirages per pixel, we first can estimate the area where the ray may refract due to the temperature gradient, and subdivide that area into exponentially smaller areas until we have found a mirage or given up. As shown in Figure 5.1, this leads to an algorithm with the following steps: computing rectified images with surface data (Section 5.2.1), parallel prefix sum computation (Section 5.2.2), and per-pixel mirage calculation (Section 5.2.3).

In the following section, we explain the core of our method in more detail.

5.2 Algorithmic details

5.2.1 Obtain surface information

Mirages can appear hundreds of meters away from the viewer, meaning that surface temperature information of a very large area and possibly many objects needs to be readily available. Capturing the surface temperatures in a texture means that a camera angle, position and perspective need to be decided upon that can capture all the relevant surfaces (some could obscure others) with enough precision. For our main method we decided to capture surface information in a texture rendered from aerial viewpoint L (Figure 5.2) with a camera view vector perpendicular to the world plane and an orthographic perspective to ensure enough precision for surface information far from the user camera C and since mirages most often are visible on surfaces parallel to the earth surface.

If no mirage is found at layer i, our main method will attempt to find a mirage at layer $i+1$ that has twice as many partitions. Each partition on layer i is divided into two partitions of equal length on layer $i+1$, and the average of the surface data of the two new partitions equals the surface data of the non-divided partition on the previous layer. Computing the

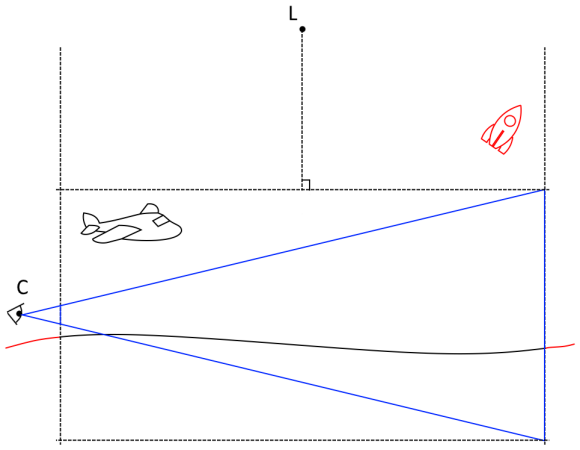


Figure 5.2: A side view of surface information being captured in a texture rendered from aerial viewpoint L with our main method. All objects that are within the box defined by the user camera C's frustum will be captured, other objects (here in red) are not visible by the user and will not be considered.

surface data of all partitions at layer $i+1$ would then be inefficient, as the surface data of one of the 'child' partitions can then be computed with the surface data of the other child and the father partition, as displayed in Figure 5.6.

Normal projection The ray equations used by Cao et al. are restricted to computing ray paths within a plane, which is a problem since the view ray and (mean) normal, which is used for the temperature gradient direction, do not have to lie in the same plane. This issue can be circumvented by projecting the normal onto the plane that holds both the view ray and world-up vector, and using the normalized version of this projection, which we call N'. This decision changes the ray path, but since mirages often appear on relatively horizontal surfaces (Young, 2015), we do not expect large differences for mirages in most cases between those rendered with the original normal versus N'. The potential speedup compared to the alternative for using the ray equations, ray marching, makes this choice worthwhile, based on earlier results (Cao et al., 2010).

Epipolar rectification As the mean temperature and normal of surface attributes below the view ray needs to be computed multiple times, and as we let view rays only refract in one plane, epipolar rectification can be applied to the temperature and normal textures to allow for optimizations such as the computation of the parallel prefix sum. With epipolar rectification, an epipolar slice projects to a row of the rectified texture; thus, the surface data that needs to be accumulated is stored in order and in a cache-friendly manner. Figure 5.3 shows the rectification process in more detail.

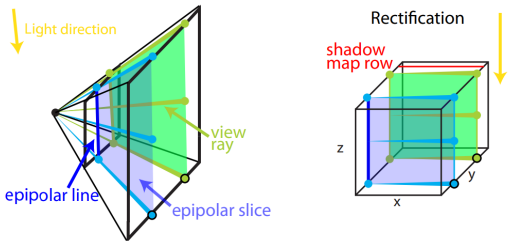


Figure 5.3: The camera frustum before and after rectification. View rays in the same epipolar slice are projected to the same epipolar line. The epipolar lines are parallel after rectification and go in the x-direction. Image originally from Klehm et al. (2014).

5.2.2 Prefix sum calculation

The inclusive prefix sum of a sequence of numbers x_i is a second sequence of numbers y_i where

$$y_i = \sum_{j=0}^{i} x_j \tag{5.1}$$

The arithmetic mean of $\bar{x}_{a,b}$ of any sequence starting at element a and ending at element b can be obtained with Equation (5.2), where y_i is the earlier defined inclusive prefix sum up to element i.

$$\bar{x}_{a,b} = \sum_{j=a}^{b} x_j/(b-a) = (y_b - y_a)/(b-a) \tag{5.2}$$

As Equation (5.2) shows, the mean surface values of any part of the surface can be computed with two texture accesses if the parallel prefix sums have been constructed. We, therefore, compute the parallel prefix sums of the surface data stored in the rectified images to speed up the computation of the different means of the surface data below part of the view ray. Since many view rays go over the same part of the surface, computing the parallel prefix sums could be worthwhile to accelerate the calculation of the surface means. Constructing the parallel prefix sums requires reading all values at least once, making it only worthwhile if computing the mean on the spot for all view rays, the simplest solution, would require more texture accesses. The results in Table 5.1 show that the number of texture accesses dropped significantly in scenes with both relatively little and much mirage formation after using the prefix sum to compute averages. The results also show that the number of texture accesses varies much less when using the prefix sum, which may contribute to a more consistent run time of the method.

5.2.3 Nonlinear path calculation

To be able to find a mirage, we first need to determine the existence of a folding point in the ray path, i.e., the point on the ray where total internal reflection takes place. The path itself also needs to be determined to see where the ray intersects with the scene, in order to find the image of the mirage.

Model	FOV	Total texture accesses (without prefix sum)	Texture accesses to compute prefix sum	Prefix sum texture accesses	Total texture accesses (with prefix sum)	Reduction in total texture accesses with prefix sum
Plane	12	105473856	8388608	3512320	11900928	-88.7%
Plane	90	22820658	8388608	3665920	12054528	-47.1%

Table 5.1: The number of texture accesses with and without using the parallel prefix sum to compute arithmetic means for a scene with relatively few mirages (Plane, FOV 90) and many mirages (Plane, FOV 12) at a resolution of 1280×720. The number of texture accesses necessary to compute the prefix sum is based on the intuitive, serial approach.

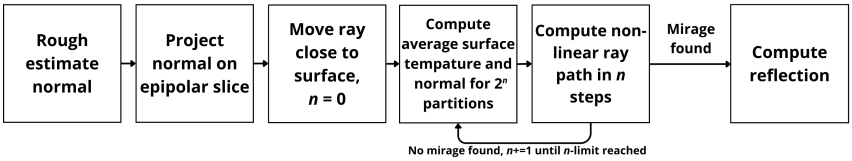


Figure 5.4: A general overview of the different steps necessary to compute a possible mirage per pixel.

Before the nonlinear path of the ray can be computed, the ray's starting position and the surface area which we presume to affect the ray due to a temperature gradient must be determined; otherwise, it is unknown what values to consider for the average surface normal(s) and temperature(s). As the overview of the mirage-computation process in Figure 5.4 shows, a first estimate of the surface normal is used to determine how long the ray can travel towards the surface without being affected by refraction. Because the aforementioned ray equations do not require the ray to be refracting immediately, this estimate can be very safe and just be used to determine a shorter, more precise surface area. Once the ray has been moved closer to the surface (position A in Figure 5.5), then the surface attributes of the surface area that we estimate to affect the ray (the surface below AB in Figure 5.5) can be used to compute the non-linear ray path in one or multiple steps, until a mirage is found or a limit is reached: a process that favors finding mirages. A difficulty with finding the surface values that need to be considered is that B, the position in which the ray is no longer affected by refraction, is only possible to know when the folding point F's position is known, after which the temperature model can be used to determine the distance from the surface where the nonlinear path stops. Therefore, an estimate needs to be made of the distance AB; we empirically found $1.3 \cdot$ the horizontal distance from A to the intersection point of the unperturbed ray with the surface to be a reasonable estimate.

Now we moved the ray position from C to closer to the surface, We assume here that the ray starts at the A in Figure 5.5. Cao et al. (2010) arguably give the easiest approach to determine the existence of a folding point with Equation (5.3):

$$n(y_F) = n(y_0)\cos\phi_0 \tag{5.3}$$

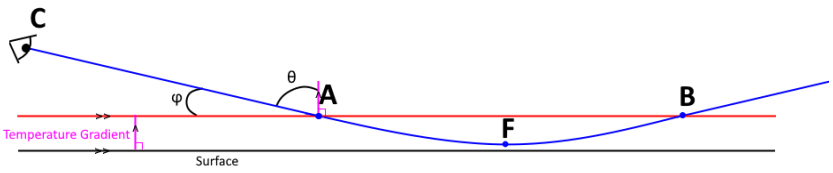


Figure 5.5: An example of inferior mirage formation.

$$x = |n(0)\cos\phi_0 \int_{y_0}^{y} \frac{1}{\sqrt{n(y_0)^2 - (n(0)\cos\phi_0)^2}}\, dy| \tag{5.4}$$

This Equation gives a positive folding point height y_F if a folding point exists, and only requires the refractive index function n (and corresponding parameters such as surface normal N and ambient and surface temperatures T_a and T_S), starting height y_0 and angle between the surface and view ray ϕ_0. Alternatively, refracting through ray marching can result in total internal reflection and, therefore, a folding point at some point, but that is not as fast as the constant solution Equation (5.3) offers. The nonlinear ray path could then be computed indirectly using Equation (5.4) given by Cao et al., where x is the horizontal distance between the points on the ray with heights y_0 and y. A caveat is that the Equation needs to be applied twice if the points are on the opposite sides of a folding point; in the situation depicted in Figure 5.5, $y_E = y_P$ and Equation (5.4) gives $x = 0$. The simplest alternative for finding the ray path would again be ray marching, which is definitely faster per step if the integral in Equation (5.4) cannot be computed analytically, as is the case here for the refractive index function we used (Van der Werf's function using the exponential decay temperature model). We expect the function to outperform ray marching based on previous results (Cao et al., 2010). Any local space position on the ray obtained with Equation (5.4) can be transformed into world space by applying the translation rotated into the world surface direction to the starting point in world space.

Intersection computation If from Equation (5.3) can be derived that there is no folding point (i.e., a negative or undefined value for y_F), then no mirage will appear based on the input and the ray will instead intersect with the surface. Equation (5.3) cannot be used to determine the intersection point, as n is undefined for negative height inputs. The intersection point with the surface plane (which has $y = 0$) can be calculated with Equation (5.4) if no mirage appears, though, making it a viable approach to still include the impact of refraction on the view ray in the final result. The point of intersection between the nonlinear ray and a plane perpendicular to the surface at a horizontal distance from the ray starting point, which we use to divide the ray path into steps, can be computed indirectly with the bisection method since no analytic solution for this intersection computation exists (Cao et al., 2010). An alternative is to use another root-finding method such as the one by Ridders, which converges quicker than the linearly-converging bisection method but still

guarantees to converge, unlike the faster-converging root-finding method by Newton and Raphson Press (2007). We opt for the slower-converging bisection method, because Ridders' method would require three calls to compute x with Equation (5.4) per step instead of one, and computing x is expensive as it requires numeric integration. Similar to Cao et al., we do use the bisection method to find the intersection point between a refracted ray and other scene objects that are not part of the surface, and we do this before and after the end of the nonlinear ray path. Once the ray is straight again after refracting, finding the ray-scene intersection point can be done more efficiently by looking for any intersection when the ray intersects with centerlines in screen space to avoid sampling the same pixel from the G-buffer's depth texture. A cheaper alternative that may work well if there are not too many scene objects is to not look for intersections after refraction and use the ray vector to sample the cubemap: mirages often appear far away and only fill a small section of the screen, making less accurate reflections less obvious.

Adapting the step count The number of steps in which the nonlinear ray path is computed can be varied to trade off speed for a possible increase in accuracy. We can choose to approximate the surface over which the nonlinear path of the ray goes as if it were one planar, homogeneous surface, or subdivide this surface area as if consisted of multiple connected planes and consequently use each plane's normal and surface temperature to compute part of the nonlinear path. Reducing the step count as long as the negative effect on the accuracy of the mirage is less than the increase in framerate is arguably a fair trade-off. Our approach following this paradigm is to use the minimum number of steps that result in a mirage. If no mirage has been found, the area(s) can be split into two and twice as many steps with half the step size can be run to try to find a mirage, until a maximum step count has been reached or a mirage has been found. Stopping once a mirage is visible assumes that the presence of the reflection is more important than the quality of the reflection. This assumption has been commonly made before in many real-time applications such as video games that use cheaper, less accurate reflection techniques such as cubemap reflections to still include a reflection in the scene. The alternative could be to keep on increasing the step count as long as the difference between the obtained mirage and the mirage obtained by doubling the step count by halving the step size is larger than a set tolerance, something that has been done before to dynamically change the ray marching step size (Seron et al., 2005). We opt against this alternative approach as it can lead to calculating thrice the number of steps, while calculating the step is, in our case, already expensive due to the texture accesses, numeric integration and intersection tests.

Since obtaining the surface data involves multiple expensive texture reading operations, avoiding this unnecessary computation can have a significant positive impact on the run time, especially if the maximum number of layers at which a mirage may be found grows. To be precise, the fraction of the times the surface data needs to be computed with this optimization compared to the maximum number of surface data computations can be computed as follows:

$$f = \frac{2^m}{2^{m+1} - 2^s} \tag{5.5}$$

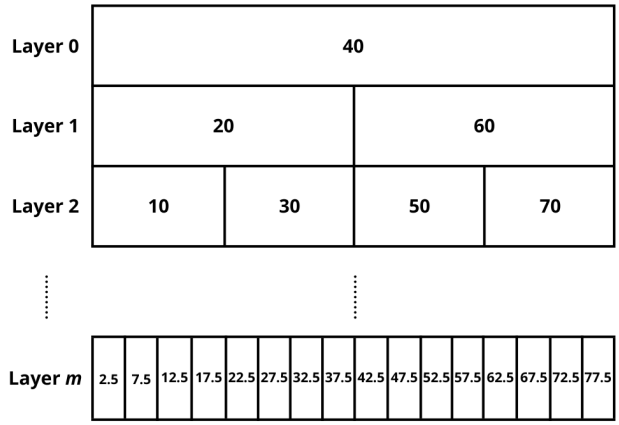


Figure 5.6: The average temperature over an area if computed in 1, 2, 4, .., 2^m steps, where m is the index of the layer. The surface data value of the parent partition P_{AB} is the average of its children P_A and P_B: $P_{AB} = \frac{P_A+P_B}{2}$. Therefore, $P_B = 2P_{AB} - P_A$.

In Equation (5.5), f is the aforementioned fraction, m is the maximum layer index (assuming the first possible layer has index 0) and s is the index of the starting layer ($s = 0$ if no layers are skipped). This equation can be understood if one remembers that the number of computations is always equal to the number of partitions in the last layer (= 2^m) as this approach only computes texture data when no data is available. Implementing the optimization entails storing all the surface data (the normal and surface temperature) in the format of a binary tree, which may be done in an array. When the second child node, i.e., the second subpartition, needs to be computed, the data of the first child node and the parent node is used to quickly compute the normal and surface temperature for the second subpartition as done in Figure 5.6. The main disadvantage is having to store all computed surface data in memory, but this increase is negligible compared to the memory to store all the different loaded textures.

5.3 Extensions

The method described in Sections 5.1 and 5.2 enables the rendering of mirages, but effects that accompany mirages such as heat haze and jitter are neglected. We show how these effects can be incorporated in the mirage rendering. Further, we present a second method that is similar to the one just described, but uses approximations to render mirages at a lower run-time.

5.3.1 Approximation method

We can approximate the presented method with fewer steps by only using surface information obtained while rendering the G-buffer. To be precise, we remove the steps where

the scene is rendered from a top view and the epipolar rectification takes place, as well as the parallel prefix sums computation steps. The surface normals and surface temperatures are captured with the generation of the G-buffer instead, requiring the rendering of an image with temperature information next to the usual images of which the G-buffer consists. Drawing this extra image is only a small cost in terms of memory and run time, especially compared to rendering steps only present in the main method, making the second method faster and less memory-consuming.

The main disadvantage of the second method is that it is based on the assumption that the surface values at the point of intersection between the non-refracted view ray and the surface are a good representation of the surface values close to the view ray. The fact that mirages appear mostly on flat surfaces makes the potential negative impact of this assumption smaller. Also, in scenes with more height variation where mirages appear, the main method that can take more steps can be used instead.

5.3.2 Precomputing integrals

The function that computes the horizontal distance to any point on the nonlinear ray (Equation (5.4)) is computationally intensive, as there is no analytical solution for the integral for our choice of refractive index function and temperature model. Equations (5.4) and (3.5) shows that the integral only requires three input values (angle ϕ, temperature difference T_s - T_a, and distance from the surface y) to be computed, and all three parameters have a limited range in practice (0.5–0.0 degrees, 30.0 °C–0.0 °C, 20–0 cm), allowing us to precompute the integral's values beforehand. Later, the values can be read from a 3D-texture in the fragment shader, and missing values can consequently be obtained with trilinear interpolation.

Table 5.2 lists the average difference between the trilinearly interpolated values and those computed on the spot. The results show that the error grows rapidly the smaller ϕ gets, and that even the biggest values for ϕ result in a $\cos(\phi)$ that is very close to 1. We used ϕ instead of its cosine to index the table with precomputed values to avoid larger angles having higher precision than smaller ones. The negative error can be attributed to underestimating the integral at the smallest angles: the more small angles were encountered, the larger the average error became. The underestimation is likely caused by too few precomputed values for the smallest angles, and might be increased by single-precision floating point's only $\tilde{7}$ decimals of precision.

Table 5.3 lists the mean run time when the integral is computed, compared to when all integral values are calculated in the fragment shader. Precomputing seems to become effective when the method uses more layers, as that increases the total of calls to compute the integral. How many mirages there are on the screen does not appear to have a big impact on the run time difference with precomputation. Overall, precomputing is an extension that, at a small cost in terms of accuracy, significantly lowers the run time of the main method when a high number of steps are taken to compute the mirages.

Lowest $\cos(\phi)$	**Average absolute error (m)**
0.99995	-0.034
0.99999	-0.099
0.999999	-0.945
0.9999995	-2.691
0.9999996	-3.820
0.9999997	-8.139
0.9999998	-12.736
0.9999999	-14.474

Table 5.2: The average absolute error measured between trilinearly interpolated precomputed values for the integral and the actual value. For each row, the left column indicates the cosine of the biggest angle included in the random distribution, with the smallest angle ϕ being 0°. The table of precomputed values has the dimensions $400 \times 800 \times 50$ (y_0, ϕ, T_s, respectively). Each average error was obtained after comparing 100,000 interpolated and actual values.

Model	FOV	Layers	Run time (without precomputation)	Run time (with precomputation)	Change in run time with precomputation
Road	2	1	5.123	5.324	3.9%
Road	2	2	5.362	5.439	1.4%
Road	2	3	6.355	6.255	-1.6%
Road	2	4	8.175	7.626	-6.7%
Road	2	5	16.299	12.255	-24.8%
Road	2	6	38.721	25.189	-34.9%
Road	90	1	4.792	4.881	1.9%
Road	90	2	4.801	4.999	4.1%
Road	90	3	5.540	5.581	0.7%
Road	90	4	6.966	6.798	-2.4%
Road	90	5	13.014	9.420	-27.6%
Road	90	6	23.611	14.778	-37.4%

Table 5.3: The run times for the road scene when relatively little (FOV 90) and much mirage formation (FOV 2) is visible and when the maximum number of steps to compute mirages is increased.

5.3.3 Jitter and heat shimmering

Mirage formation and heat shimmering are both caused by a temperature gradient, but the latter is mainly caused by temperature fluctuations over the ray path. Extending the presented methods to include these fluctuations is nontrivial, as the advantages of our approach rely on the assumption that the ray only refracts in one plane (limiting the refraction caused by the fluctuations to one direction) and on the idea that we compute the mirage in as few steps as possible (making it difficult to add some random offset to the ray direction with each step). This addition of small random offsets to mimic the small, natural variations in the air density that cause rays of light to travel a slightly different path, has been used in the past in mirage rendering at the last moment to make image looking less 'clean' and more realistic (Berger et al., 1990), but doing this introduces more of a 'jitter' effect than heat shimmering: the heat haze effect is a result of turbulence, and unlike jitter, turbulence is not completely random. Zhao et al. (2006) apply heat flow simulation and ray marching to render both mirages and heat shimmering, which is not an option in our situation (see Section 3.1).

The way we would extend our methods is with a screen-space approach that either uses distortion textures that are either precomputed or dynamically generated, and takes available range and temperature information into account. Repasi and Weiss (2011) use range information, an image without turbulence, and real-time turbulence simulation based on empirical data to generate an image sequence with turbulence. In our situation, the G-buffer can be used for the range and image without turbulence, but temperature information is not accounted for in their model, which could cause unrealistic or even unwanted heat shimmering in scenes. Another problem is that applying the heat shimmering as a post-processing effect is that the refraction causing mirages can be different from the refraction causing the heat shimmering, leading to incorrect situations where a ray could both cause a mirage and refract due to turbulence in a way that should never allow a mirage. Therefore, the method by Repasi & Weiss seems a good base approach that must be adapted first to make use of all the information our method has available. Due to time constraints, we only implemented jitter in a similar way to Berger et al. (1990) and left the addition of heat shimmering as future work.

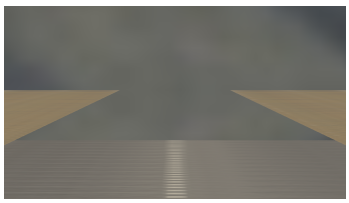
Chapter 6

Implementation

In this chapter, we provide all the information necessary for a successful implementation of the methods.

6.1 Anisotropic filtering

For the second, approximating method, we recommend using anisotropic filtering for the temperature textures if mipmaps are used. As Figure 6.1 shows, using mipmaps for the temperature textures can cause the temperature values for surfaces farther away in the scene to change and average out over the surface, falsely shrinking the mirage. This issue can become prevalent with mirages as they are more prevalent at small angles between the view ray and surface, similar to the artifacts visible with trilinear filtering without anisotropic filtering. Anisotropic filtering increases the quality of the filtered texture value by using texels that approximate the shape of the filter box, while isotropic filtering assumes a square filter box. The effective filter box and the area from which texture samples need to be used can differ significantly if the filter box is oblong-shaped, which happens at these small viewing angles. An alternative to enabling anisotropic filtering is to avoid mipmaps and opt for nearest-neighbor or bilinear filtering.



(a) Mirage rendered with anisotropic filtering enabled.

(b) Mirage rendered with anisotropic filtering disabled.

Figure 6.1: The road mirage rendered with and without anisotropic filtering enabled.

6.2 Epipolar rectification

Both linear and nonlinear transformation methods can be used for epipolar rectification. The linear rectification method is faster as it does not require a second rendering pass to resample the image, but the linear approach produces an image where distances are not preserved, unlike the non-linear rectification method (Baran et al., 2010; Klehm et al., 2014). The non-linear rectification method seems easier to employ here, as the preservation of world space distances along the x-axis means that after rectification, the prefix sum and arithmetic mean can be calculated straightforwardly. This preservation of distances does not hold for linear rectification, as calculating the prefix sum and arithmetic mean in the same manner thereafter would result in a weighted average with higher weights for geometry closer to the camera's near plane. Even if different texture coordinates are used during the prefix sum computation to get an unweighted average, there is still the problem that the perspective projection of the linear rectification results in a large part of the texture being reserved for surfaces closer to the camera. Consequently, values obtained from the rectified texture are likely not as precise as when they are obtained with a non-linear rectification, which involves an orthographic projection. For this reason, we used nonlinear epipolar rectification.

6.3 Parallel prefix sum

Our method for computing the parallel prefix sum was implemented with synchronization between steps and shared memory to store data between steps is based on an earlier implementation (Sellers et al., 2015). With this implementation, additions are performed partly in parallel. For instance, the prefix sum $y_{0..3}$ of the numbers $x_{0..3}$ would be calculated in two steps as follows:

$$\begin{aligned} y_0 &= x_0 \\ y_{0..1} &= x_0 + x_1 \\ y_2 &= x_2 \\ y_3 &= x_2 + x_3 \end{aligned} \tag{6.1}$$

And for the second step:

$$\begin{aligned} y_{0..2} &= y_2 + y_{0..1} \\ y_{0..3} &= y_3 + y_{0..1} \end{aligned} \tag{6.2}$$

Any prefix sum $y_{0..n}$ with n being in the order of 2 can be calculated in $\log_2(n)$ steps this way. This reduction in step count should result in a faster solution than a fully sequential solution (Sellers et al., 2015), which is depicted in Equation (6.3), in spite of the higher number of expensive texture read-operations that comes with the implementation we use; from Equation (6.1) it becomes clear that 50% more texture reads are performed. It is for that reason that Klehm et al. argue that it might be faster to calculate the prefix-sum with "the simplest implementation", referring to an approach that follows Equation (6.3) (Klehm et al., 2014, p. 75).

$$\begin{aligned} y_i &= y_{i-1} + x_i \\ y_0 &= x_0 \end{aligned} \tag{6.3}$$

	Resolution						
	32^2	64^2	128^2	256^2	512^2	1024^2	2048^2
Simple method	1.152	1.165	1.189	1.389	1.623	2.743	7.176
Parallel method	1.142	1.155	1.166	1.197	1.319	1.822	3.840

Table 6.1: The run time (in ms) of the first mirage rendering method for the non-rotated plane model while employing the simple method and parallel method to compute the parallel prefix sum for textures of different resolutions.

We compared the run time of the main method for the homogeneous non-rotated plane model with an alternative version that uses the sequential approach to computing the prefix sum, and we did this comparison for different texture sizes. Since the prefix sum needs to be calculated for different textures, 'the simplest version' could in our case either mean a version where the different prefix sums are calculated in parallel (fewer updates to texture coordinates) or one after the other (might improve cache coherency). We found no difference in run time between these two simplest versions and opted to use the latter for the comparison. The results are available in Table 6.1.

The results show that the parallel implementation leads to lower run times. The computation of the prefix-sum does not depend on the used model or the content of the input textures. Therefore, a similar absolute difference in run time could be expected with different scenes. Interestingly, this approach, which uses synchronization and shared memory, is faster than the intuitive approach, while similar approaches were found to be slower previously (Klehm et al., 2014). A possible explanation could be slight differences in implementation and technology over the years.

Our current implementation comes with the one notable disadvantage that the local thread count could be a bottleneck: the minimum value of gl_MaxComputeWorkGroupSize for OpenGL 4.3 and later versions is 1024 (Kessenich et al., 2019), meaning that the current implementation is not able to deal with textures with a resolution higher than 2048×2048. This issue can be circumvented by using multiple work groups, but this could potentially negatively impact the run time, making it necessary to compare the adapted implementation against the simple implementation again.

Chapter 7

Results

We tested the presented methods against other approaches for the visualization of mirages to verify whether the presented methods meet the requirements we have set. The different methods compete on both accuracy and run time; therefore methods are compared against each other on these attributes.

7.1 Experiment setup

All methods have been tested in a custom OpenGL application on a laptop running an Intel i7-9750H CPU and an NVIDIA RTX 2060 GPU. The results were computed at various often-used screen resolutions (1920×1080, 2560×1440, 3840×2160).

No method seems to exist against which we can fairly compare the two presented methods in terms of performance. While real-time mirage rendering has been achieved before in the video game DriveClub, it likely did not use refraction to obtain the mirages, and the exact method and parameters are unknown (Section 3.2). Other previous approaches to mirage rendering used offline rendering methods and did not achieve interactive framerates. Furthermore, sometimes acceleration structures were used, which makes dynamic scenes very challenging.

Nonetheless, we have chosen to create a ray-marching shader to still have a framework against which we can compare our methods. To improve the run time and accuracy of the ray-marching shader, we have implemented enhancements that the typical ray-marching approach does not have (e.g., ray marching the nonlinear path only when the ray is close to a surface). Similar enhancements are incorporated in the two methods. The step size of the ray-marching method is adapted to match either the quality or speed of the first, main method. It is important to note that none of the tested approaches is the ground truth, as the presented methods limit refraction to the epipolar slice and none of the tested methods have access to the distance to the closest surface in all the scenes. Nonetheless, all three approaches should lead to correct results for the homogeneous plane model without slope due to its geometric simplicity and uniform temperature.

Different models of varying complexity are used to test the different methods. Table 7.1 lists the models, the number of triangles they consist of, and the surface temperatures used.

Model	Triangle count	Surface temperature (°C)
Plane	2	48
Rotated plane	2	48
Desert road (flat)	200,800	48
Desert road (bumpy)	650,250	48
Desert	838,860	48
Ocean	2,928,200	32

Table 7.1: The different models used to demonstrate the presented methods.

The surface temperatures are selected from the three different temperature data sets that we used before.

7.2 Main experimental findings

Table 7.2 shows the average rendering times for the different models for both methods and the ray-marching approach, as well as the rendering time for the scene without mirages. The field of view (FOV) used for each configuration is given too, as lowering the FOV results in a 'zoom in'-effect on the mirage and potentially higher rendering times. Figure 7.1 shows a comparison between the mirages' visuals for the different approaches.

Based on both the results shown in Figure 7.1 and Table 7.2, our main method generally performed better in terms of quality and speed than ray marching for the simplest scenes (plane, rotated plane), and similar in quality but better in speed for the desert road models, when the resolution was the highest or the FOV the lowest, the situations which contained the most pixels with mirages. In the plane scenes, the surface is flat and homogeneous, which are situations that benefit a method that tries to compute the nonlinear path in as few steps as possible and uses mean surface data below the view ray to compute the path. The reason ray marching does achieve equal results with lower run times for the 1920×1080 (with high FOV) and 2560×1440 resolutions for these models may likely be the cost of the epipolar rectification and prefix sum calculation, which does pay off when mirages fill a larger part of the screen. In the case of the both desert road models, the main method renders more mirages than the second method and the ray-marching method likely indicating false positives caused by using average surface values. For the desert model, the main method performed considerably worse, missing most (parts of) mirages that are visible with ray marching with a similar speed, although the method does render some mirages along the slopes of the small hills, which is the specific scenario where we thought the averaging of the surface could help and beat the second method. However, in the case of the ocean scene, the main method gives a mirage effect that seems more realistic than the one provided by the ray marching, especially when the results in Figure 7.1 are compared to the real-life mirage in Figure 7.2; the ray-marching method shows more pixels with mirages, especially around the top of the waves, but no 'omega sun' can be spotted.

Model	Resolution	FOV (degree)	Run time					Notes
			No mirages	Method 1	Method 2	Ray marching (similar quality)	Ray marching (similar speed)	
Plane	1920×1080	12	0.922	3.593	1.353	4.238	3.535	$t_q = 1.0$, $t_s = 1.8$
	1920×1080	90	0.923	3.420	1.224	2.664	3.479	$t_q = 1.0$, $t_s = 0.7$
	2560×1440	90	1.599	4.245	2.050	3.992	4.466	$t_q = 1.0$, $t_s = 0.9$
	3840×2160	90	3.668	6.946	4.808	8.112	6.863	$t_q = 1.6$, $t_s = 2.5$
Rotated plane	1920×1080	8	0.906	3.751	1.440	3.111	3.828	$t_q = 1.0$, $t_s = 0.7$
	1920×1080	90	0.922	3.553	1.233	2.579	3.461	$t_q = 1.0$, $t_s = 0.5$
	2560×1440	90	1.563	4.350	2.051	3.746	4.285	$t_q = 1.0$, $t_s = 0.7$
	3840×2160	90	3.591	7.072	4.765	8.202	7.034	$t_q = 1.0$, $t_s = 2.0$
Desert road (flat)	1920×1080	2	1.571	7.954	4.411	12.021	7.911	$t_q = 1.0$, $t_s = 2.5$
	1920×1080	90	1.306	5.444	1.793	4.221	5.465	$t_q = 1.0$, $t_s = 0.4$
	2560×1440	90	1.978	6.378	2.745	5.361	6.227	$t_q = 1.0$, $t_s = 0.6$
	3840×2160	90	4.061	9.766	5.580	8.886	10.144	$t_q = 1.0$, $t_s = 0.9$
Desert road (bumpy)	1920×1080	4	2.011	7.578	3.328	9.444	7.539	$t_q = 1.0$, $t_s = 1.7$
	1920×1080	90	1.820	5.334	2.415	3.798	5.354	$t_q = 1.0$, $t_s = 0.73$
	2560×1440	90	2.428	7.959	3.517	5.107	7.880	$t_q = 1.0$, $t_s = 0.4$
	3840×2160	90	4.302	9.676	6.696	8.486	9.716	$t_q = 1.0$, $t_s = 0.72$
Desert	1920×1080	20	2.020	5.734	2.414	17.151	5.897	$t_q = 0.1$, $t_s = 0.8$
	1920×1080	90	2.031	5.701	2.380	12.501	5.683	$t_q = 0.1$, $t_s = 0.7$
	2560×1440	90	2.596	6.618	3.129	16.712	6.519	$t_q = 0.1$, $t_s = 1.0$
	3840×2160	90	4.401	10.328	5.666	33.252	10.257	$t_q = 0.1$, $t_s = 2.2$
Ocean	1920×1080	18	5.264	10.733	5.721	29.274	10.807	$t_q = 0.1$, $t_s = 5.2$
	1920×1080	90	5.098	10.527	5.449	19.720	10.564	$t_q = 0.1$, $t_s = 1.2$
	2560×1440	90	5.908	11.546	6.467	26.803	11.555	$t_q = 0.1$, $t_s = 2.0$
	3840×2160	90	8.433	15.047	9.772	44.358	15.033	$t_q = 0.1$, $t_s = 20.0$

Table 7.2: The run time (in ms) of our methods for different scenes, resolutions and FOVs. The run time of each scene while rendering no mirages and rendering mirages with a ray marching approach are included as well to allow for a comparison. The step size of the ray marching method is adapted to aim for either similar results as the first (and second) method or similar run times as the first method. These step sizes are included as t_q and t_s, respectively. The layer count for the main method is also adapted to give the best results for lowest run time and is, therefore, included as well.

Overall, fewer mirages appeared in the complex scenes with the main method compared to ray marching.

The second, approximating method outperforms the main method significantly in terms of run time, and the visual results between the two methods are practically identical for the aforementioned simpler scenes (plane, rotated plane, flat desert road) except for the bumpy desert road scene, where the main method had quite different visual output. The second method has also been shown to be faster than the ray-marching approach in all scenarios, with the caveat that the method misses (parts of) mirages similarly to the main method in the more complex scenes. The approximating method also misses more mirages than the main method in the desert and ocean scenes, even if the main method uses only one step to compute the mirages in those scenes, another indication that the averaging of the surface data below the ray can increase the chance of (false) mirages appearing. Increasing the maximum step count even further shows even bigger differences in the desert scene between the two methods, with more mirages appearing, especially along short slopes.

Increasing the layer count, allowing for a few extra steps, was necessary for the main

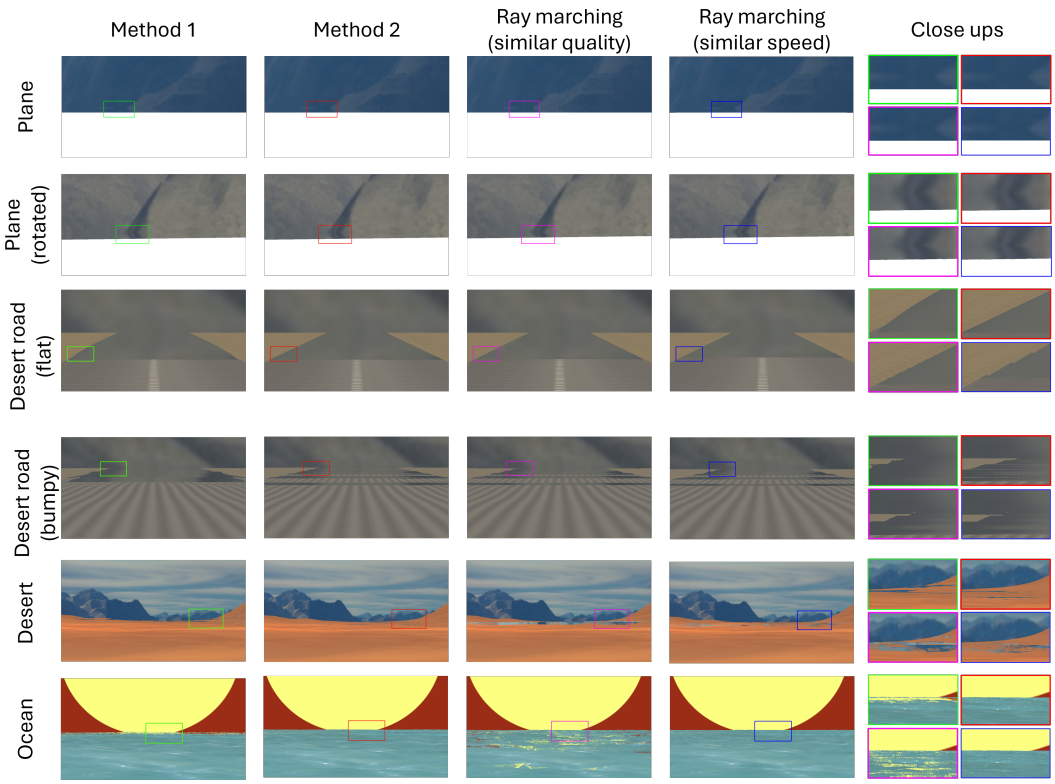


Figure 7.1: Mirages rendered in the various test scenes with the main method, the approximation method, and ray marching method aiming for similar quality and for similar speed as our methods. All mirages are rendered at a resolution of 1920×1080 with a low FOV to increase the size of the mirages.

Figure 7.2: An inferior mirage appearing above the ocean. The sun and mirage intersect, an image that is referred to as an 'omega sun' based to the image resemblance to the Greek character. A frame from the film The Blue Lagoon (1980) Kleiser (1980).

Scene	FOV	Layers	Run time without trick to reduce texture accesses (ms)	Run time with trick to reduce texture accesses (ms)	Change in run time with trick
Plane	12	1	2.782	2.782	0%
Plane	12	2	3.029	3.019	-0.3%
Plane	12	3	3.306	3.328	0.7%
Plane	12	4	4.280	4.275	-0.1%
Plane	12	5	8.127	8.086	-0.5%
Plane	12	6	23.490	22.988	-2.1%
Plane	12	7	79.835	77.141	-3.4%

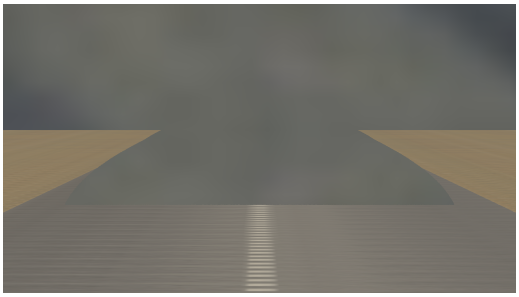
Table 7.3: The change in run time after using the trick to reduce texture accesses in our main method by computing the average surface attributes of a child node with the data of its parent node and the other child node.

method for the desert road scenes and the desert scene, but even in those situations, increasing the value above 2 would result in barely more mirages. For this reason, we ran the tests without precomputing integral values, as the results in Table 5.3 indicated that precomputing would only have a positive impact on run time with a layer count above 3. If the layer count is 1, the trick to reduce texture accesses by using information from previous layers obviously does not work, and with higher layer counts there seems to be barely any reduction in run time as Table 7.3 shows. The impact of texture accesses on the run time seems to be low and only become an issue when too many steps are taken.

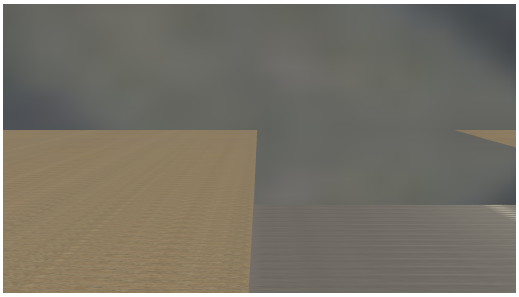
7.3 Averaging surface data per epipolar slice

Figure 7.3 shows incorrect behavior that comes with our choice to average the surface temperature and normal per epipolar slice: when the epipolar slice is perfectly aligned with the hotter side of the road (Figure 7.3c), no mirages appear on the sand and no mirages are missing on the road. However, when the camera is either in the middle of the road or next to the road, then parts of the sides of the road do not have a mirage while they should have one, because for those pixels, the epipolar slice goes over the sand, which drops the average surface temperature enough to not be able to render mirages.

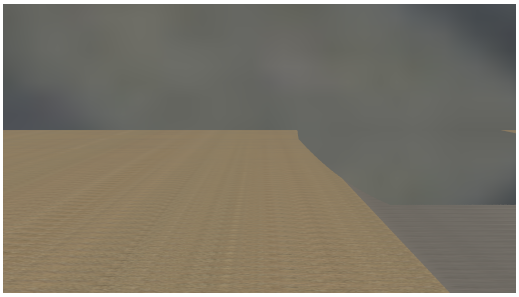
Another interesting result is that doubling the number of steps resulted in more mirages being drawn on the sand, as we see when comparing the results for the desert road model in Figures 7.3 and 7.1. A possible explanation is that the the epipolar slices and the surface area division fall just ‘right’ so that the average temperature of some of the pixels which first had sand in Figure 7.3 is raised by relatively more road surface.



(a) Camera in the middle of the road.



(b) Camera on the edge of the road.



(c) Camera on the left of the road.

Figure 7.3: Parts of the road mirage are falsely not being rendered with the first method in images (a) and (c), because the surface area used for is not aligned with the road in those situations. The problem is exacerbated in these images by doing the ray path computation in 1 step.

Chapter 8

Discussion

Based on the results, it has become clear that both presented methods are able to render mirages in various realistic situations. The first method renders more mirages than the second method, but also takes much longer to complete.

This is in line with our initial expectations: the main method uses scene information obtained by rendering from a separate view orthogonal to the surface, and by computing the average surface normal over a part of the surface that includes the backside of bumps, the angle between the observer and the average surface normal may become bigger, big enough to cause a mirage. Moreover, increasing the number of steps exponentially until a mirage is found or a limit is reached may result in additional mirages, as using the average surface values of a larger area can lead to false positives that would not appear when smaller steps are taken; a deliberate design choice that favors speed over accuracy.

The result is that more mirages were visible in scenes with smaller bumps (bumpy desert road, ocean) using the first method than when rendered with the second method or ray marching. However, as ray marching should be more accurate in these situations, given its small steps and the height deviation in the scene, we can only conclude that the first method indeed rendered false positives in certain situations.

The lower accuracy indicated by these false positives, which were not as noticeable with the other approximating method, was not always obtained at run times lower than the run times of the second and ray-marching methods.

The results also show that the second method was much faster than both ray marching and the main method in every situation, while it only missed many mirages in the desert and ocean scenes. We expect that in these scenes with a lot of height deviation in all directions, the sampled surface normal is performing worse as an indicator for mirages, and more expensive methods, such as the first method or ray marching, could be used instead.

When enough mirages appear on the screen or when a small step size is required to render many mirages, the main method is shown to be faster than ray marching, making it the preferred choice in those situations.

If neither of the presented methods is a viable solution, ray marching is also shown to perform in real time while delivering similar results in most situations, which was unexpected based on earlier work. Using temperature textures and only ray marching close to the surface has helped to decrease the run time.

8.1 Limitations

The presented methods show several limitations. The restriction that refraction can only be applied in one plane means that mirages may look slightly different from their real-life counterpart and other approaches. Other, more minor limitations of the methods are not accounting for the earth's curvature while calculating the surface height and not using different indices of refraction for different wavelengths of light, which becomes an issue when mirages of objects very far away or other forms of atmospheric refraction need to be rendered, respectively. The biggest limitation that comes with both presented methods is that they are pure approximations, and they have then also shown to possibly render completely different results than a method such as ray marching when the scenes look differently from what we assumed a mirage-causing scene to look like (relatively flat, fairly homogeneous).

Finally, the rendering of heat shimmering is not supported by the presented methods in their current form, which may seem strange as both effects are caused by the same temperature gradient.

Chapter 9

Conclusion

In this work, we presented two methods to enable real-time rendering of mirages in dynamic scenes. To achieve more realistic-looking mirages, we evaluated the accuracy and speed of different air temperature models and functions for the air's refractive index. During this investigation, we also identified an error in the refractive-index function used in an earlier mirage-rendering method.

Both methods successfully use analytic ray equations to accelerate the nonlinear ray path computation. The use of temperature textures enables the approximation of the temperature gradient above the surface.

The main method's choice to use epipolar rectification to process surface temperature and normal values from an image to obtain the average surface values under the ray for the temperature gradient is expensive. Nevertheless, it is worthwhile if the mirages occupy a large enough portion of the screen and there are small bumps or slopes that would cause mirage formation, which a cheaper solution would miss.

The second method, which uses the surface temperature and normal values from the G-buffer for the temperature gradient, is much faster than the main method and ray marching in all situations. The quality of the mirages for both methods is better than that of ray marching for plane surfaces and relatively flat surfaces. In other situations with surfaces with significant height deviation, ray marching, combined with optimizations that we used to reduce the step count, can be used to compute the nonlinear path for higher-quality mirages.

Utilizing the advised temperature models and refractive index functions also contributes to higher accuracy, lower run times and easier use when rendering real-time mirages. Therefore, this work leads to a large jump in the field of real-time mirage rendering in dynamic scenes.

9.1 Future work

The methods' limitations and ideas left unexplored due to scope- or time constraints may inspire future work. For instance, the derivation of ray equations for rays where the index gradient and initial ray direction do not lie in the same plane makes the projection of the

surface normal to the epipolar slice unnecessary and should therefore result in more accurate ray paths.

A second suggestion for future research is using 1D height fields for the rectified view rays to divide the ray in steps more effectively and potentially prevent the false positives we experienced. All steps being equal size, as done in our first method, requires many steps when a mirage is caused by one small part of the surface (e.g., a small bump) or might be missed, while the height field can be used to find adequate step sizes of varying lengths. Previous work already shows that such height fields can be computed efficiently (Chen et al., 2011).

Lastly, if the presented methods are extended so that they can render both mirages and heat shimmering based on the temperature data, then both effects are finally rendered simultaneously and based on the same temperature information, instead of heat shimmering being simulated in real-time applications by sampling a texture with varying texture coordinates. We already explained how such an extension could look like in Section 5.3.3, but that potential approach still has issues that need to be resolved first.

Bibliography

Ilya Baran, Jiawen Chen, Jonathan Ragan-Kelley, Frédo Durand, and Jaakko Lehtinen. A hierarchical volumetric shadow algorithm for single scattering. *ACM Transactions on Graphics (TOG)*, 29(6):1–10, 2010.

Marc Berger, Terry Trout, and Nancy Levit. Ray tracing mirages. *IEEE Computer Graphics and Applications*, 10(3):36–41, 1990.

Jean-Baptiste Biot. *Recherches sur les réfractions extraordinaires qui ont lieu près de l'horizon*. Garnery, 1810.

KP Birch and MJ Downs. Correction to the updated edlén equation for the refractive index of air. *Metrologia*, 31(4):315, 1994.

PA Buxton. The temperature of the surface of deserts. *The Journal of Ecology*, pages 127–134, 1924.

Chen Cao, Zhong Ren, Baining Guo, and Kun Zhou. Interactive rendering of non-constant, refractive media using the ray equations of gradient-index optics. In *Computer Graphics Forum*, volume 29, pages 1375–1382. Wiley Online Library, 2010.

Yunus A Çengel. *Heat Transfer: A practical approach*. McGraw-Hill, 2002.

Jiawen Chen, Ilya Baran, Frédo Durand, and Wojciech Jarosz. Real-time volumetric shadows using 1d min-max mipmaps. In *Symposium on Interactive 3D Graphics and Games*, pages 39–46, 2011.

Philip E Ciddor. Refractive index of air: new equations for the visible and near infrared. *Applied optics*, 35(9):1566–1573, 1996.

Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, and Elmar Eisemann. Interactive indirect illumination using voxel cone tracing. In *Computer Graphics Forum*, volume 30, pages 1921–1930. Wiley Online Library, 2011.

Joey de Vries. Deferred shading. `https://learnopengl.com/Advanced-Lighting/Deferred-Shading`, 2015. Accessed: 2025-11-13.

Bengt Edlén. The refractive index of air. *Metrologia*, 2(2):71, 1966.

Fire and Haze: Wind Waker Graphics Analysis Series. https://medium.com/@gordonnl/fire-and-haze-b4561743b17, 2017. Accessed: 2022-08-11.

Robert G Fleagle. The temperature distribution near a cold surface. *Journal of Meteorology*, 13(2):160–165, 1956.

Diego Gutierrez, Francisco J Seron, Adolfo Munoz, and Oscar Anson. Simulation of atmospheric phenomena. *Computers & Graphics*, 30(6):994–1010, 2006.

Francis A. Jenkins and Harvey E. White. *Fundamentals of Optics*. McGraw-Hill, 2001.

BF Kallas. Asphalt pavement temperatures. *Highway Research Record*, 150, 1966.

John Kessenich, Dave Baldwin, and Randi Rost. *The OpenGL® Shading Language, Version 4.60.7*. 2019.

E Khular, K Thyagarajan, and AK Ghatak. A note on mirage formation. *American Journal of Physics*, 45(1):90–92, 1977.

Oliver Klehm, Hans-Peter Seidel, and Elmar Eisemann. Prefiltered single scattering. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 71–78, 2014.

Randal Kleiser. The Blue Lagoon [Film]. 1980.

David R Lide. *CRC handbook of chemistry and physics*, volume 84. CRC press, 2003.

Marcel Minnaert. *Light and Color in the Outdoors*. Springer-Verlag New York, 1993.

Mirages in Finland. https://finland.fi/life-society/mirages-in-finland/, 2001. Accessed: 2022-05-10.

Qi Mo, Hengchin Yeh, and Dinesh Manocha. Tracing analytic ray curves for light and sound propagation in non-linear media. *IEEE transactions on visualization and computer graphics*, 22(11):2493–2506, 2015.

Giuseppe Nelva. https://www.dualshockers.com/heres-what-driveclubs-mirages-look-like-on-ps4-just-when-you-thought-it-couldnt-look-any- 2015. Accessed: 2023-09-13.

William H Press. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

Endre Repasi and Robert Weiss. Computer simulation of image degradations by atmospheric turbulence for horizontal views. In *Infrared Imaging Systems: Design, Analysis, Modeling, and Testing XXII*, volume 8014, pages 279–287. SPIE, 2011.

Graham Sellers, Richard S Wright, and Nicholas Haemel. Opengl superbible: Comprehensive tutorial and reference, 2015.

Francisco J Seron, Diego Gutierrez, Guillermo Gutiérrez, and Eva Cerezo. Implementation of a method of curved ray tracing for inhomogeneous atmospheres. *Computers & Graphics*, 29(1):95–108, 2005.

Taylor Sheridan. Wind River [Film], 2017.

Jos Stam and Eric Languénou. Ray tracing in non-constant media. In *Eurographics Workshop on Rendering Techniques*, pages 225–234. Springer, 1996.

M Taghi Tavassoly, Soghra Osanloo, and Ali Salehpour. Mirage is an image in a flat ground surface. *JOSA A*, 32(4):599–603, 2015.

Thermodynamic: Albedo. `https://web.archive.org/web/20160820011858/https://nsidc.org/cryosphere/seaice/processes/albedo.html`, 2016. Accessed: 2016-08-20.

Siebren Y van der Werf. Noninverted images in inferior mirages. *Applied Optics*, 50(28): F12–F15, 2011.

Nabil W Wakid. Temperature profile and double images in the inferior mirage. *arXiv preprint arXiv:1911.03507*, 2019.

Andrew T Young. Inferior mirages: an improved model. *Applied optics*, 54(4):B170–B176, 2015.

Ye Zhao, Yiping Han, Zhe Fan, Feng Qiu, Yu-Chuan Kuo, Arie E Kaufman, and Klaus Mueller. Visual simulation of heat shimmering and mirage. *IEEE transactions on visualization and computer graphics*, 13(1):179–189, 2006.

HuaiChun Zhou, ZhiFeng Huang, Qiang Cheng, Wei Lü, Kui Qiu, Chen Chen, and Pei-feng Hsu. Road surface mirage: a bunch of hot air? *Chinese science bulletin*, 56(10):962–968, 2011.

Appendix A

Image coherency

Despite thermodynamics supporting the idea that a difference in temperature causes the formation of mirages and a large amount of research endorsing this theory, there is still research that opposes this idea (Zhou et al., 2011; Tavassoly et al., 2015). One hypothesis is that mirages are not caused by a temperature gradient but instead by increase in image-forming light rays at higher incident angles (Tavassoly et al., 2015); when an observer looks at a rough plane at such an angle, then no reflections are visible due to the rough plane imposing random phases on the rays' wavefronts which destroys constructive and destructive interference, while both are necessary for image formation (Tavassoly et al., 2015). An increase in the incident angle leads to more constructive interference in the specular direction for the diffracted rays and, consequently, a clearer image. This improvement in image quality is also due to rough objects reflecting more light at higher angles of incidence, as described by the Fresnel equations (Jenkins and White, 2001, p. 524). The result is that specular reflections such as the one depicted in Figure A.1 appear at grazing angles of incidence on large rough surfaces such as desert planes, even if there is no temperature gradient.

While these particular reflections share similarities with the mirages presented in this work, that does not mean that these reflections should be called mirages too, or that an increase in image coherency is the true cause of mirage formation. Not only is there proof that a temperature gradient can cause mirages (see Section 2.2), there are also clear differences between mirages and these specular reflections (Wakid, 2019). For example, the angle between the observer and the surface normal required to have mirages is significantly bigger than the smallest reported angle of the aforementioned reflections, 89.5° vs. 75° (Wakid, 2019; Tavassoly et al., 2015). Another critical difference is that mirages, caused by total internal reflection, cannot be directly affected by the roughness of the surface as the light ray does not reflect of the surface, unlike this other type of reflections. In short, both refraction and improved image coherency can create unexpected reflections at vast angles, but they are different types of reflections and should, therefore, be treated differently. We defined the ones caused by a temperature gradient as mirages because the consensus is that mirages are caused by a temperature gradient (Young, 2015). The reflections caused by an improved image coherency are, therefore, not further discussed in this work.

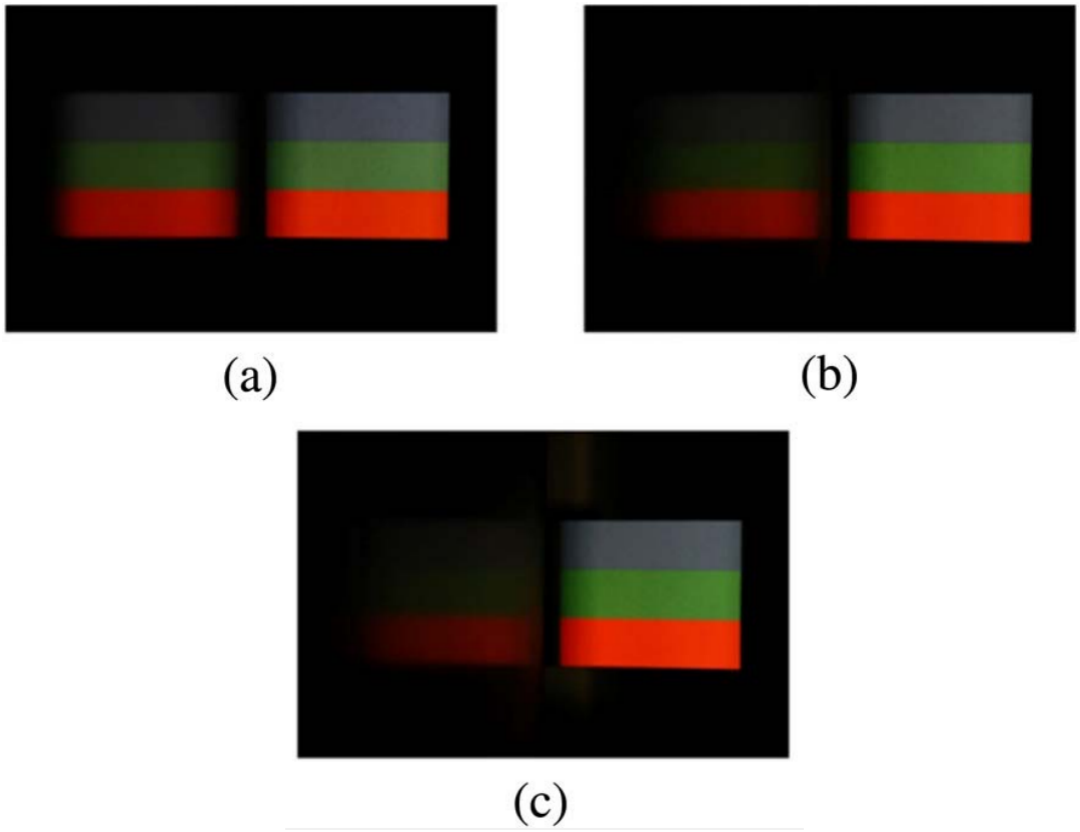


Figure A.1: An object (right side of each photograph) and its reflections (left side) captured at increasingly larger angles between the camera and and the rough plane (4, 9, and 15°). The smaller the angle, the better the image quality of the reflection due to the increase in image coherency. Images originally from Tavassoly et al. (2015), adapted by ourselves.

Appendix B

Correction in refractive index function of Zhao et al.

Zhao et al. attempted to use the method created and updated by Edlén and Birch & Downs, respectively, in their mirage-rendering method (Zhao et al., 2006). We believe it is likely that two mistakes have been made in their implementation, resulting in very different results that may seem plausible at first glance.

$$n(T,P) = \frac{P}{96095.43} \cdot \frac{1+10^{-8}(0.601-0.00972T)P}{1+0.0036610T} \tag{B.1}$$

$$(n-1)_s(\sigma) = 10^{-8}(8342.54+2406147[130-\sigma^2]^{-1}+15998[38.9-\sigma^2]^{-1}) \tag{B.2}$$

$$n(T,P) = 1+\frac{P(n-1)_s}{96095.43} \cdot \frac{1+10^{-8}(0.601-0.00972T)P}{1+0.0036610T} \tag{B.3}$$

When Edlén's method (Equations (B.2) and (B.3)) and Zhao et al.'s method (Equation (B.1)) are compared to each other, two differences are noticeable. The first mistake is that Zhao's version misses the addition of '1' at the end, which would result in any produced refractive index having a value of only slightly more than zero if it had been the only mistake. Since it is common knowledge that the index of refraction of air is approximately 1, it would have been hard to believe that this mistake has found its way back in the actual implementation of Zhao et al.: it would then be more likely that Zhao et al. forgot to write the additional term, especially when the handbook they used to obtain the Edlén equations from provide equations for $(n-1)$ instead of n (Lide, 2003)[p. 10-224]. However, we think this addition is most likely absent from Zhao et al.'s implementation too since the dispersion term $(n-1)_s$ in Equation (B.1) is not shown either, which would be the second mistake. The handbook might be the cause of the issue here again, as it only mentions in text that the dispersion term must be multiplied with the multiplier in Equation (B.3) to obtain $(n-1)$ (Lide, 2003)[p. 10-224]. Unfortunately, one mistake partially nullifies the other, which may very well have been the reason why they have gone unnoticed. For example, with a wavelength of 500 nm, standard atmospheric pressure and a temperature of 30 °C, Zhao

et al.'s version (Equation (3.3)) gives $n = 0.95037181407$, while Edlén's version gives $n = 1.00026512846$. If Zhao et al. only forgot to add 1 at the end (mistake 1), then they would get $n = 0.00026512846$; if they only forgot to multiply with the dispersion term (mistake 2), then $n = 1.95037181407$. Both errors together ensure that Equation (B.1) gives values close to 1 by accident, making them less noticeable. Therefore, we presume that both errors have slipped in the implementation of the index of refraction function in the work of Zhao et al., making Equation (3.3) an inferior function to obtain the air's refractive index with that should not be used.