



JOSIP JURAJ STROSSMAYER UNIVERSITY OF OSIJEK
SCHOOL OF APPLIED MATHEMATICS AND INFORMATICS

Undergraduate Program in Mathematics and Computer Science

Physically Based Rendering of the Human Eye

BACHELOR THESIS

Supervisor:

doc. dr. sc. Domagoj Ševerdija

Author:

Josip Cestar

Osijek, 2025

Contents

1	Introduction	1
2	OpenGL	3
2.1	Understanding OpenGL	3
2.2	Setup	4
2.2.1	GLFW	4
2.2.2	GLAD	4
2.2.3	Assimp	4
2.2.4	STB	4
3	Pipeline	5
4	Core Techniques of Realistic Rendering	7
4.1	Blending	7
4.2	Physically Based Rendering	10
4.3	Image-Based Lighting	13
5	Shader Implementation	15
5.1	Normal Mapping	15
5.2	GGX Normal Distribution Function	16
5.3	Schlick-GGX Geometry Function	17
5.4	Smith Geometry Function	18
5.5	Fresnel-Schlick Approximation	19
5.6	Subsurface Scattering	20
5.7	Cook-Torrance BRDF	20
5.8	Ambient Lighting and Image-Based Lighting	21
5.8.1	Diffuse IBL	21
5.9	Final Results	21
	Literature	23
	Summary	25

1 | Introduction

3D visualization is a technical and artistic multistep process used in various industries, including manufacturing, architecture, and multimedia, to create 3D designs of objects that exist or are planned for the real world. By representing graphics with both shape and depth, 3D visualization brings projects to life for engineers and artists alike. Its greatest advantage is its ability to parallel reality. One of its key benefits is that it reduces costs and minimizes reliance on expensive physical prototypes. In a world where time is money, it enables faster time-to-market and improves sales potential through high-quality renders.

A crucial part of this process is rendering. Rendering is the transformation of a three-dimensional scene into a two-dimensional image. All aspects of computer graphics, from animation to geometric modeling and texturing, must pass through rendering to become visible. Achieving realism in 3D visualization involves techniques such as photorealistic rendering, global illumination, and physically based rendering (PBR). These methods are essential for creating lifelike images that enhance user experience and engagement. Physically based rendering (PBR) simulates the physical behavior of light and its interaction with materials to achieve realistic visuals. It applies algorithms based on accurate physical formulas to replicate real-world materials, resulting in photorealistic environments and scenes.

Rendering software is easy to use but difficult to master. Such software typically provides tools for modeling, animation, UV mapping, and more. Rendering has become pervasive, opening new frontiers for creative expression. Some popular software packages include 3ds Max, Maya, Cinema4D, and Blender.

The main goal of this thesis is to build a PBR shader from scratch. The most effective way to test its potential is to apply it to the human eye. The complexity of the eye makes it an ideal subject because of its transparency, its ability to refract light, and its intricate internal structure.

This paper is divided into three sections. In the first section, we discuss OpenGL and its main functionality, as well as the libraries and functions used in our program to load the model, apply textures, and render them. In the second section, we present our rendering pipeline. In the final section, we provide an overview of the shader implementation and discuss how we achieved a realistic result.

2 | OpenGL

2.1 Understanding OpenGL

As an Application Programming Interface (API), OpenGL provides developers with an extensive set of functions to manage graphics and images. While the specification defines how each function should perform and what its output should be, it does not dictate the exact implementation.

Developers must ensure their implementations produce compliant results. Because OpenGL is an open-ended specification, different versions deliver consistent results to users, but their underlying mechanics may differ.

The development of OpenGL libraries is handled by graphics card manufacturers. Each graphics card has its own version of OpenGL, specifically tailored for that card or series. When OpenGL exhibits unusual behavior, the issue often lies with the graphics card manufacturer or the company responsible for maintaining the library.

OpenGL is a large state machine: a collection of variables that define how it operates. The state of OpenGL is commonly referred to as the OpenGL context. By setting options, manipulating buffers, and rendering with the current context, developers can modify OpenGL's state.

The OpenGL libraries are written in C, but bindings exist for many other languages. OpenGL was built with abstractions that allow C programs to be translated into higher-level languages. Objects play a vital role in this translation, encapsulating structure and data for various imaging tasks. These objects include buffers, textures, and shaders. Each controls a specific part of the rendering pipeline. By structuring state information on objects, OpenGL makes translation processing more efficient and convenient.

2.2 Setup

Creating the project is the first step in rendering images, but it must first be set up. The project uses GLFW for window management, GLAD for loading OpenGL functions, STB for loading images, and Assimp for loading 3D models. These libraries streamline the workflow, making model rendering more efficient.

2.2.1 GLFW

Input and window creation are handled by GLFW, an OpenGL context management library. GLFW simplifies the setup of a rendering environment, allowing developers to focus on graphics rather than platform-specific details. A window with an attached OpenGL context can be created on a variety of platforms using GLFW.

2.2.2 GLAD

GLAD (OpenGL Loader Generator) handles the loading of OpenGL function pointers. Since modern OpenGL versions expose functions dynamically rather than through static linking, GLAD ensures these functions are integrated into the application and accessible. By handling OpenGL's versioning and extension system, GLAD guarantees compatibility with the system's OpenGL implementation.

2.2.3 Assimp

Assimp, or the Open Asset Import Library, is a powerful tool for importing and exporting 3D models. Its primary function is to load model data into its generalized data structures, enabling developers to work with multiple file formats without implementing support for each individually. This makes it versatile and easy to integrate into projects.

2.2.4 STB

As part of the STB libraries, `stb_image.h` is a popular single-header C library that simplifies the loading of various image formats. It converts raw pixel data from PNG, JPEG, BMP, GIF, and TGA files into usable OpenGL textures. It also provides options for loading in RGB or RGBA formats and includes functionality for image scaling and flipping.

3 | Pipeline

After setting up OpenGL, GLFW, and GLAD, we enable key OpenGL states such as depth testing, ensuring that rendering handles depth correctly. All shaders in the scene are compiled and linked, including those for Physically Based Rendering (PBR), background rendering, and utility shaders responsible for tasks such as Bidirectional Reflectance Distribution Function (BRDF) computation and environment map conversion. To render the eye model, textures and models must be loaded, including albedo, normal, specular, and roughness textures.

To enable Image-Based Lighting (IBL), we convert a High Dynamic Range (HDR) image into cubemap format. This conversion is handled by the equirectangular-to-cubemap shader and stored in a framebuffer for efficient access. An irradiance map is then computed to simulate diffuse global illumination, ensuring that the model receives realistic ambient light.

To simulate how rough materials blur their reflections, we perform prefiltering of the environment map. This step provides reflection data at multiple levels of detail, enabling more accurate rendering of reflections on both smooth and rough surfaces. Additionally, a BRDF lookup table is generated to simplify the shading calculations required for blending direct and indirect lighting correctly.



Figure 3.1: Final render of the model.

To achieve physically accurate rendering of the model, the precomputed IBL data is combined with the PBR shader. By using the BRDF lookup table, we achieve physically correct lighting and reflections. For correct blending, the lens model must be rendered with transparency, which requires temporarily disabling depth testing.

The background skybox is rendered using the environment cubemap. This creates the illusion of an immersive scene while providing both reflections and ambient light. Dynamic lighting is updated in real time with each frame based on the position and color of the light sources, ensuring that objects respond correctly to both direct and indirect illumination.

This rendering pipeline produces a visually rich and physically accurate result, with realistic reflections, ambient light, and material shading.

4 | Core Techniques of Realistic Rendering

With the rendering pipeline established, we now turn to the core techniques that make realism possible. In this chapter, we discuss three key components: blending, Physically Based Rendering (PBR), and Image-Based Lighting (IBL). These techniques form the foundation of realistic rendering by enhancing visual fidelity and immersion. By examining their function and interaction, we gain a clear understanding of how they support advanced rendering in our program.

4.1 Blending

Blending in OpenGL is the technique used to render transparency. Transparency ensures that objects, or parts of them, are not drawn as solid colors but as a combination of their own colors and those of the objects behind them.

For example, imagine two squares: a red opaque square and a green semi-transparent square. If the green square is drawn over the red one, the final color is determined by blending the source (green square) and the destination (red square) colors. The red square is first stored in the color buffer as the destination color, while the green square is then drawn as the source.

During blending, the color values of the incoming fragment (source) are combined with the color values of the pixel already in the buffer (destination). This is done in two steps: first, source and destination factors are applied to the respective color vectors; second, the resulting vectors are summed. If the green square contributes 60% to the final color, the red square contributes the remaining 40%. The blending formula is:

$$\vec{C}_{\text{result}} = \vec{C}_{\text{source}} \times F_{\text{source}} + \vec{C}_{\text{destination}} \times F_{\text{destination}} \quad (4.1)$$

Where:

- $\vec{C}_{\text{source}}$: the color vector output by the fragment shader.
- $\vec{C}_{\text{destination}}$: the color vector stored in the buffer.
- F_{source} : source factor, based on the source alpha.
- $F_{\text{destination}}$: destination factor, based on $1 - \alpha$.

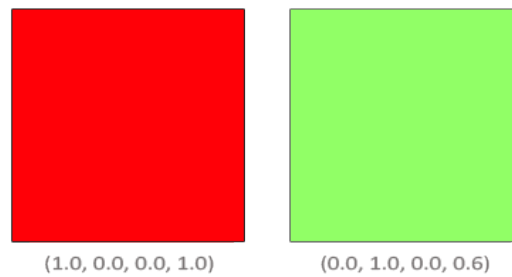


Figure 4.1: Two squares with RGBA values. Source: [5]

$$\vec{C}_{\text{result}} = \begin{pmatrix} 0.0 \\ 1.0 \\ 0.0 \\ 0.6 \end{pmatrix} \cdot 0.6 + \begin{pmatrix} 1.0 \\ 0.0 \\ 0.0 \\ 1.0 \end{pmatrix} \cdot (1 - 0.6)$$

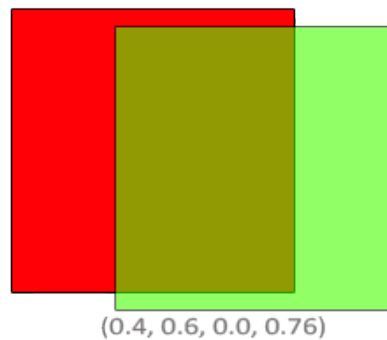


Figure 4.2: Result of blending two squares. Source: [5]

This resulting color is stored in the buffer.

In OpenGL, blending is enabled with the command `glEnable(GL_BLEND)`. The most common configuration is to use the source alpha as the source factor and $1 - \alpha$ as the destination factor:

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

```
1 glDisable(GL_DEPTH_TEST);
2 glEnable(GL_BLEND);
3 glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
4
5 // Render the lens model
6 lensShader.use();
7 lensShader.setMat4("view", view);
8 lensShader.setMat4("projection", projection);
9 lensShader.setVec3("camPos", camera.Position);
```

```
10 lensShader.setMat4("model", model);
11
12 bindTexture({ 0, lensNormal, 0, 0, lensAOMap, 0, 0 });
13 lensModel.Draw(lensShader);
14
15 // Re-enable depth testing and disable blending after rendering
   transparent objects
16 glEnable(GL_DEPTH_TEST);
17 glDisable(GL_BLEND);
18
19 // Render the main model
20 pbrShader.use();
21 pbrShader.setMat4("view", view);
22 pbrShader.setMat4("projection", projection);
23 pbrShader.setVec3("camPos", camera.Position);
24 pbrShader.setMat4("model", model);
25 pbrShader.setMat3("normalMatrix", glm::transpose(glm::inverse(glm::
   mat3(model))));
26
27 bindTexture({ faceAlbedo, faceNormal, 0, faceRoughness, faceAOMap,
   faceSpecular, faceDisplacement });
28 mainModel.Draw(pbrShader);
29
30 // Render the eyes
31 bindTexture({ eyesAlbedo, eyesNormal, 0, eyesRoughness, eyesAOMap,
   eyesSpecular, 0 });
32 eyeModel.Draw(pbrShader);
```

In our code, depth testing (`GL_DEPTH_TEST`) is disabled before rendering the transparent object (the lens) and re-enabled afterwards (`GL_DEPTH_TEST`). This is done to ensure proper transparency handling. With transparent objects, depth testing does not work properly, since closer objects usually obscure those farther away. If depth testing is enabled while rendering transparent objects, any pixel that is drawn will have its depth recorded. As a result, further fragments from behind the object would not be blended, even though they should be visible through transparency. By disabling depth testing, we allow OpenGL to blend the colors of the transparent object with those of the objects behind it. This creates the correct visual result, rendering the transparent lenses first then the eyes. Upon rendering the transparent object, depth testing is re-enabled to render opaque objects so we can render correctly without unwanted transparency.

4.2 Physically Based Rendering

Physically Based Rendering (PBR) is a collection of rendering techniques designed to simulate light with physical accuracy, producing realistic images. By applying physics-based models, PBR ensures that light-surface interactions behave consistently under a variety of lighting conditions. Surfaces are described through physical parameters such as metallicity and roughness, which allow materials to maintain a natural appearance regardless of the environment. PBR typically makes use of multiple textures, including albedo, normal, metallic, roughness, and ambient occlusion maps, to describe the properties of a surface in detail.

A central concept in PBR is the *microfacet model*, which represents surfaces as collections of tiny facets too small to be resolved individually. Although invisible to the eye, these microfacets strongly influence the way light scatters. Roughness controls the statistical distribution of these facets: smooth surfaces concentrate reflections, while rougher surfaces scatter them more broadly.

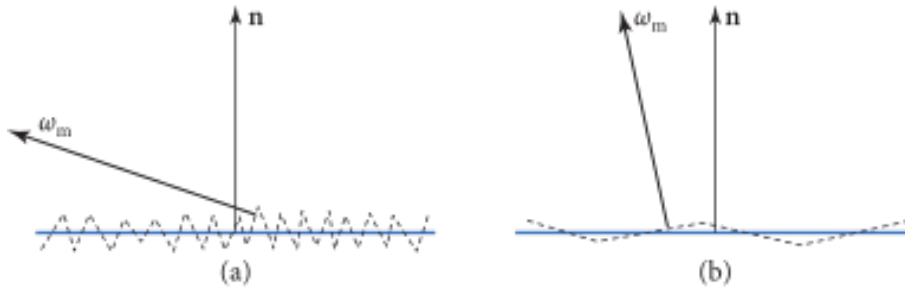


Figure 4.3: Microfacet surface models. Source: [4]

The orientation of microfacets relative to the halfway vector $\mathbf{h}$ determines the sharpness of specular reflections. The halfway vector is defined as:

$$\mathbf{h} = \frac{\mathbf{l} + \mathbf{v}}{\|\mathbf{l} + \mathbf{v}\|}$$

where $\mathbf{l}$ is the light direction and $\mathbf{v}$ the view direction. When many microfacets align with $\mathbf{h}$, reflections appear sharp and bright. Increasing roughness introduces more variation, softening reflections.

Energy conservation is an essential principle in PBR: the total reflected light can never exceed the incoming light (excluding emissive surfaces). As roughness increases, the specular highlight becomes larger but dimmer, preserving energy. Smooth surfaces concentrate energy into sharp highlights, while rough surfaces distribute it more diffusely.

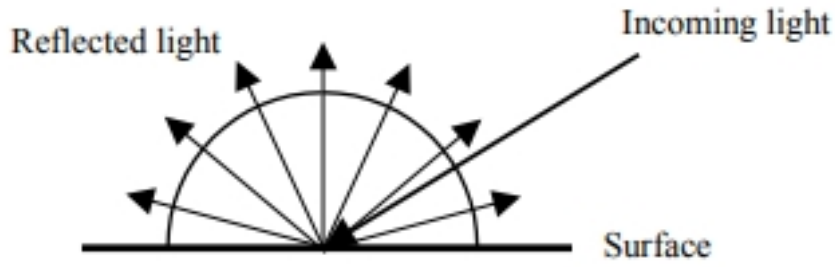


Figure 4.4: Energy conservation in specular reflection. Source: [2]

To satisfy energy conservation, light is split into two components:

- **Specular reflection:** light that bounces off the surface without entering it.
- **Diffuse reflection:** light that penetrates the surface, scatters internally, and exits in random directions.

In most non-metallic surfaces, both diffuse and specular reflection occur. Metallic surfaces, however, absorb transmitted light, producing only specular reflection without a diffuse component.

The rendering equation formalizes these principles. It describes the outgoing radiance from a surface point as the sum of emitted light and reflected light:

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{\Omega} f_r(p, \omega_i, \omega_o) L_i(p, \omega_i) (\omega_i \cdot n) d\omega_i$$

where:

- $L_o(p, \omega_o)$: outgoing radiance at point p in direction ω_o .
- $L_e(p, \omega_o)$: emitted radiance (zero for non-emissive materials).
- $L_i(p, \omega_i)$: incoming radiance from direction ω_i .
- f_r : bidirectional reflectance distribution function (BRDF).
- n : surface normal.

The BRDF defines how much light arriving from ω_i is reflected toward ω_o , depending on the material properties.

The BRDF is typically decomposed into several components:

- **Diffuse term:** often modeled by Lambertian reflection, assuming equal scattering in all directions.
- **Specular term:** modeled using microfacet theory, producing glossy reflections.

- **Normal Distribution Function (NDF):** describes the statistical distribution of facet orientations.
- **Geometry function:** accounts for self-shadowing and masking between facets.
- **Fresnel term:** determines reflectance depending on the angle of incidence and refractive index.
- **Roughness parameter:** controls the spread of specular highlights.

Together, these elements provide a flexible but physically accurate model of light–surface interaction, which is the foundation of PBR.

4.3 Image-Based Lighting

Image-Based Lighting (IBL) enhances PBR by incorporating real-world environmental data into the rendering process. Instead of relying solely on manually placed light sources, IBL uses environment maps, usually captured as high dynamic range (HDR) images, to simulate ambient and reflective lighting. This allows objects to appear naturally integrated into their surroundings.

IBL is particularly important because it captures both diffuse and specular contributions from the environment. This makes materials appear more plausible, as their lighting reflects the scene in which they are placed.

Environment maps store radiance in all directions, often represented as cube-maps. A cubemap consists of six square textures (front, back, left, right, top, bottom) that form a cube surrounding the object.



Figure 4.5: Example of an HDR image used for IBL. Source: [7]

To compute diffuse lighting, an irradiance map is generated by integrating incoming light from the environment:

$$L_{\text{diffuse}}(N) = \int_{\Omega} L_i(\omega_i) \cdot \max(0, \omega_i \cdot N) d\omega_i$$

This produces smooth, non-directional illumination that captures the general color of the environment.

For specular reflection, prefiltered environment maps are used. These maps are mipmapped so that each level corresponds to a different roughness value. Smooth surfaces use higher-resolution reflections, while rough surfaces use blurred reflections. The reflection direction R is calculated as:

$$R = 2(N \cdot V)N - V$$

where N is the surface normal and V the view vector.

The specular BRDF is combined with the environment data:

$$f_{\text{spec}}(L, V, N) = \frac{D(H, N) \cdot G(L, V, N) \cdot F(V, H)}{4(N \cdot L)(N \cdot V)}$$

The total reflected light is then given by:

$$L = k_d \cdot L_{\text{diffuse}} + k_s \cdot L_{\text{specular}}$$

where:

- k_d : diffuse reflection coefficient (related to albedo).
- k_s : specular reflection coefficient (related to metallicity and roughness).
- L_{diffuse} : diffuse irradiance.
- L_{specular} : prefiltered specular reflection.

Through this combination of diffuse and specular contributions, IBL provides a realistic simulation of how objects interact with their environment.

5 | Shader Implementation

In this chapter the shader implementation for Physically Based Rendering (PBR) of the human eye is described. Each component of the rendering pipeline models a physical phenomenon of light–surface interaction. For the eye, these models are especially important because its appearance is dominated by fine optical details: the glossy transparent cornea, the soft subsurface scattering of the sclera, and the complex texture and microstructure of the iris. The following sections present the most important shader functions, their mathematical formulation, and their specific contribution to eye rendering.

5.1 Normal Mapping

Normal mapping allows the simulation of fine surface details without modifying the underlying mesh geometry. For the eye, this technique is critical for reproducing structures such as the radial furrows of the iris or the subtle veins visible in the sclera. Without normal mapping, the eye would appear unnaturally smooth and plastic-like.

Since the normals stored in a normal map are defined in *tangent space*, they must be transformed into *world space* where all lighting calculations are performed. The following function performs this transformation:

```
1 // Tangent-space normal to world-space conversion
2 vec3 getNormalFromMap()
3 {
4     vec3 tangentNormal = texture(normalMap, TexCoords).xyz * 2.0 -
5     1.0;
6
7     vec3 Q1 = dFdx(WorldPos);
8     vec3 Q2 = dFdy(WorldPos);
9     vec2 st1 = dFdx(TexCoords);
10    vec2 st2 = dFdy(TexCoords);
11
12    vec3 N = normalize(Normal);
13    vec3 T = normalize(Q1*st2.t - Q2*st1.t);
14    vec3 B = -normalize(cross(N, T));
15    mat3 TBN = mat3(T, B, N);
16
17    return normalize(TBN * tangentNormal);
18 }
```

Mathematically this can be expressed as:

$$N_{\text{world}} = \text{TBN} \cdot N_{\text{tangent}}$$

Where:

- N_{tangent} : The normal sampled from the normal map.
- TBN : The matrix formed by tangent T , bitangent B , and geometric normal N .

In the context of the human eye:

- **Iris:** The iris contains complex radial grooves. A normal map can represent these fine variations, giving the iris its characteristic striated appearance.
- **Sclera:** Subtle vein structures can be encoded into the normal map, breaking up the otherwise smooth surface and contributing to realism.
- **Cornea:** The cornea is smooth and typically does not require normal perturbation. For this part of the eye, the unmodified geometric normal is usually sufficient.

5.2 GGX Normal Distribution Function

The microfacet model assumes a surface is made of countless microscopic facets. The *Normal Distribution Function* (NDF) describes the statistical distribution of these facet orientations. The GGX distribution is widely used in PBR because it produces realistic highlights that fall off gradually.

```

1 float DistributionGGX(vec3 N, vec3 H, float roughness)
2 {
3     float a = roughness * roughness;
4     float a2 = a * a;
5     float NdotH = max(dot(N, H), 0.0);
6     float NdotH2 = NdotH * NdotH;
7
8     float nom = a2;
9     float denom = (NdotH2 * (a2 - 1.0) + 1.0);
10    denom = PI * denom * denom;
11
12    return nom / denom;
13 }
```

Mathematically:

$$D_{\text{GGX}}(N, H, \alpha) = \frac{\alpha^2}{\pi((N \cdot H)^2(\alpha^2 - 1) + 1)^2}$$

Where:

- N : Surface normal.
- H : Halfway vector between view and light direction.
- α : Squared roughness value.

Eye-specific interpretation:

- **Cornea:** The cornea is very smooth, corresponding to a low roughness. GGX therefore concentrates most microfacets around the normal, producing sharp, well-defined specular highlights.
- **Sclera:** The sclera has a slightly higher roughness, leading to softer, wider highlights. This difference distinguishes it from the highly polished cornea.
- **Iris:** The iris is somewhat rough and irregular, meaning its highlights are more scattered. This prevents it from appearing mirror-like, which would be unrealistic.

5.3 Schlick-GGX Geometry Function

The geometry function describes shadowing and masking between microfacets: some facets are hidden from the light or from the viewer. The Schlick-GGX function provides an efficient approximation for this effect.

```

1 float GeometrySchlickGGX(float NdotV, float roughness)
2 {
3     float r = (roughness + 1.0);
4     float k = (r * r) / 8.0;
5
6     float nom = NdotV;
7     float denom = NdotV * (1.0 - k) + k;
8
9     return nom / denom;
10 }
```

Mathematically:

$$G_{\text{SchlickGGX}}(N \cdot V, \alpha) = \frac{N \cdot V}{(N \cdot V)(1 - k) + k}$$

with

$$k = \frac{(\text{roughness} + 1)^2}{8}.$$

For the eye:

- This function ensures that as the viewing angle becomes more grazing, fewer microfacets contribute to reflection, softening the specular response.
- It prevents overly strong highlights at glancing angles on the sclera and iris, while keeping the corneal highlight strong but realistic.

5.4 Smith Geometry Function

The Smith geometry function combines shadowing/masking for both light and view directions.

```

1 float GeometrySmith(vec3 N, vec3 V, vec3 L, float roughness)
2 {
3     float NdotV = max(dot(N, V), 0.0);
4     float NdotL = max(dot(N, L), 0.0);
5     float ggx2 = GeometrySchlickGGX(NdotV, roughness);
6     float ggx1 = GeometrySchlickGGX(NdotL, roughness);
7
8     return ggx1 * ggx2;
9 }
```

Mathematically:

$$G_{\text{Smith}}(N, V, L, \alpha) = G_{\text{SchlickGGX}}(N \cdot V, \alpha) \times G_{\text{SchlickGGX}}(N \cdot L, \alpha).$$

This ensures light occlusion is considered consistently for both incoming and outgoing directions, stabilizing the appearance of specular highlights across the eye surface.

5.5 Fresnel-Schlick Approximation

The Fresnel effect describes how the proportion of reflected light increases as the viewing angle becomes more grazing. This is particularly noticeable in the eye, where the cornea reflects the environment strongly at shallow angles.

```
1 vec3 fresnelSchlick(float cosTheta, vec3 F0)
2 {
3     return F0 + (1.0 - F0) * pow(clamp(1.0 - cosTheta, 0.0, 1.0),
4     5.0);
5 }
```

Mathematically:

$$F(\cos \theta) = F_0 + (1 - F_0)(1 - \cos \theta)^5$$

Where:

- F_0 : Base reflectance at normal incidence, derived from the material's index of refraction.
- $\cos \theta$: Angle between the view vector and the halfway vector.

For the eye:

- **Cornea:** Has a relatively high index of refraction (about 1.376), giving a strong Fresnel effect. This produces bright reflections of the environment at grazing angles, a key feature of realistic eye rendering.
- **Sclera/Iris:** Lower reflectivity and rougher surfaces, so the Fresnel effect is subtler.

5.6 Subsurface Scattering

Light penetrates into translucent materials, scatters, and re-emerges, producing soft shading. The sclera and iris both exhibit subsurface scattering due to their tissue composition and pigmentation. This makes them appear soft and organic rather than opaque.

```
1 vec3 sssDiffuse = sssColor * (1.0 - exp(-thickness * sssScale *
    NdotL));
```

Mathematically:

$$\text{sssDiffuse} = \text{sssColor} \cdot \left(1 - e^{-\text{thickness} \cdot \text{sssScale} \cdot (N \cdot L)}\right)$$

- sssColor: The scattering color, typically reddish for tissue.
- thickness: The perceived thickness of the surface.
- sssScale: Controls the intensity of scattering.

In the eye:

- The sclera's whiteness comes from scattering of light inside its fibrous tissue.
- The iris' soft transitions of color are influenced by scattering within its layered structure.
- Without SSS, the eye would appear too hard and artificial.

5.7 Cook-Torrance BRDF

The Cook-Torrance BRDF combines the three microfacet terms (distribution, geometry, Fresnel) into the final specular reflection model.

```
1 vec3 numerator    = NDF * G * F;
2 float denominator = 4.0 * max(dot(N, V), 0.0) * max(dot(N, L), 0.0)
    + 0.0001;
3 vec3 specular     = numerator / denominator;
```

Mathematically:

$$f_r(N, V, L) = \frac{D(N, H) G(N, V, L) F(H, V)}{4 (N \cdot V) (N \cdot L)}$$

For the eye:

- This model is essential to capture the glossy, lifelike highlights on the cornea.
- It also handles the more diffuse reflections of the sclera and iris by varying roughness and F_0 .

5.8 Ambient Lighting and Image-Based Lighting

Finally, realistic rendering of the eye requires accounting for the surrounding environment. Image-based lighting (IBL) allows the eye to pick up colors and light from its surroundings.

5.8.1 Diffuse IBL

Diffuse ambient lighting is sampled from an irradiance map and multiplied by the surface albedo:

```
1 vec3 irradiance = texture(irradianceMap, N).rgb;  
2 vec3 diffuse    = irradiance * albedo;
```

Mathematically:

$$\text{diffuse}_{\text{IBL}} = \text{irradiance} \times \text{albedo}$$

For the eye:

- The sclera and iris are strongly influenced by diffuse ambient lighting, ensuring they do not appear flat or underexposed.
- The cornea reflects the environment via specular IBL, producing the familiar reflection of light sources in the eye.

5.9 Final Results

To evaluate the effectiveness of the implemented Physically Based Rendering (PBR) shader, the eye model was rendered under different lighting conditions. The results demonstrate how the combination of microfacet shading, Fresnel effects, subsurface scattering, and Image-Based Lighting (IBL) contributes to a realistic visual outcome.

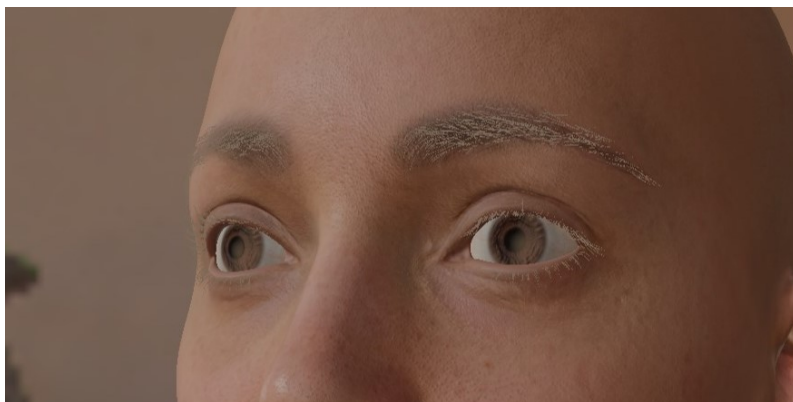


Figure 5.1: Rendered eye model.



Figure 5.2: Rendered eye model under different environmental lighting.

The complete source code, shader implementation, and additional rendering examples are available in the project's *GitHub repository*.

Literatura

- [1] B. BURLEY, *Physically Based Shading at Disney* , Walt Disney Animation Studios, 2012.
- [2] C. WYNN, *An Introduction to BRDF-Based Lighting* , NVIDIA Corporation, 2000.
- [3] N. HOFFMAN, *Background: Physics and Math of Shading*, 2003.
- [4] PHYSICALLY BASED RENDERING: FROM THEORY TO IMPLEMENTATION, , available at <https://pbr-book.org/4ed/contents>.
- [5] LEARNOPENGL, *Physically Based Rendering*, available at <https://learnopengl.com>.
- [6] CODINGLABS, *Physically Based Rendering: Cook-Torrance*, available at <http://www.codinglabs.net/>.
- [7] NVIDIA, *GPU Gems*, available at <https://developer.nvidia.com/gpugems/gpugems/contributors>.

Physically Based Rendering of the Human Eye

Summary

This thesis has presented the implementation of Physically Based Rendering (PBR) techniques for the realistic visualization of the human eye. By combining Bidirectional Reflectance Distribution Functions (BRDFs) with Image-Based Lighting (IBL), we developed a shading model that accurately simulates light-surface interactions under diverse lighting conditions.

The integration of BRDFs captures the detailed behavior of surface reflections and scattering based on material properties such as roughness, metallicity, and albedo. At the same time, IBL incorporates environmental illumination into the model, enabling dynamic and context-aware lighting that enhances realism. Together, these methods provide a physically grounded approach that reproduces the eye's characteristic features.

The results demonstrate that PBR, supported by IBL, is highly effective for rendering complex organic structures like the human eye. Future extensions could incorporate more advanced models for refraction through the cornea or anisotropy in the iris, expanding the realism and potential applications in fields such as medical visualization, computer animation, and virtual reality.

Keywords

Physically Based Rendering, bidirectional reflectance distribution function, image-based lighting, real-time rendering, material reflectance, light scattering, environmental reflection, surface properties, computer graphics, lighting models, rendering techniques.