Glass Refraction Using BSDF for Smooth Dielectrics

Ananda Ferreira Gomes

anaf@itu.dk

Graphics Programming, 2025

KGGRPRG1KU

1. Motivation and Problem Statement

Glass is a material frequently perceived in the daily life. Depending on the glass object's shape and the material's thickness and because of its transparent dielectric properties, its refraction of light creates a striking visual effect, bending and distorting whatever lays behind it. The medium is commonly used in visual arts to create sensory experiences, from analogue photography [6] to digital 3D rendering [5]. While for the latter, the context of arts allows for creative freedom and doesn't require physical accuracy in the representation of glass, in fields such as architecture, a realistic visualisation is crucial and demands sophisticated technology [7].

Physically based rendering (PBR) is an approach to photorealistic rendering through accurate modeling of the physics of light scattering [8]. Bidirectional Scattering Distribution Function (BSDF) is one approach to modeling scattered light. In this project, the goal is to create the effect of refraction through glass. Therefore, BSDF for smooth dielectric materials is implemented, chosing glass as the refracted material. Rasterization is used for rendering, which poses limitations and outputs an simplified representation of refraction. However, in the scope of this project, refraction is used as stylisation and an approximation is sufficient. The implementation follows the theory of the book *Physically Based Rendering* [8], which could be applied by more sophisticated rendering techniques.

2. Settings

The project includes two project specific GUI windows for designing and debugging. *Material Properties* includes sliders to set the *Refraction Index* and the *Roughness*, two properties of the material's surface.

Refraction Index: The refraction index varies for different materials and determines how much the light is bent when entering the material. For glass it's commonly 1.5, the default value in the GUI. The range is from 1.0 to 2.42, the approximate indices of air and diamond respectively. The relevance of this value will be elaborated on in the implementation section.

Roughness: The default value is set to 0.01 which implies a smooth surface. The roughness depends on the finish of the surface and is not the type of material. In physically based rendering, it's the central parameter of the microfacet model, a method used for achieving more realistic light scattering by imitating the roughness of surfaces at micro level in the physical world. In the scope of this project the microfacet model was not

implemented in the BSDF, but was taken over from exercise 8 for the specular reflection of the point light.

The elements in the *Debug* window are mainly used as a debug tool to study the interpolation of reflection and transmission, and inspect either one in isolation.

Tran Green: Tints the transmission in green up to the absolute maximum intensity 1.0.

Refl Red: Tints the reflection in red up to the absolute maximum intensity 1.0.

Tran Intensity: Blends the transmission in and out from zero to hundred percent.

Refl intensity: Blends the reflection in and out from zero to hundred percent.

3. Implementation

The project is based on the course's repository for exercise 8, where a BRDF was implemented following the Lecture on physically based rendering. The project specific code is in the file shaders > bsdf.glsl, an adaptation of the exercise's lambert-ggx.glsl. The *SurfaceData* struct was simplified to only contain the *normal*, *refractionIndex*, and *roughness*, since albedo, metalness and ambient occlusion don't necessarily apply to glass. *SampleEnvironment( )* is kept as well as the geometry and distribution functions for the implementation of the microfacet model for rough dielectrics. Those functions are only applied in *ComputeSpecularReflectionDirect( )* to compute the specular reflection of the direct pointlight, which is also taken over from the original exercise code. *ComputeDirect( )* returns the interpolation of the reflection of the pointlight and no light based on the Fresnel term, which will be elaborated on further below.

This project follows the implementation of specular reflection and transmission as described in [1]. The specifics of the implementation will be elaborated in the following subsections.

3.1 BSDF Render Equation

The BSDF is the computation of the scattered light and combines BRDF and BTDF for reflection and transmission respectively. The rendering equation for light at a specific point *x* perceived from a specific direction *ω* is as shown in equation E(1) by [2].

$$L_s(\boldsymbol{x}, \boldsymbol{\omega}) = \int_{\mathbb{S}^2} L_i(\boldsymbol{x}, \boldsymbol{\omega}_i)\, f_s(\boldsymbol{x}, \boldsymbol{\omega}_i \rightarrow \boldsymbol{\omega})\, |\langle \boldsymbol{n}_{\boldsymbol{x}}, \boldsymbol{\omega}_i \rangle|\, \mathrm{d}\boldsymbol{\omega}_i \qquad \text{E(1)}$$

where n_x is the point's normal, ω_i is the incident light direction. The equation returns an aggregation of light in space S^2, for scattered light it's a sphere centered around *x*. The

highlighted f_s is the BxDF, eg. BRDF, BTDF or BSDF term, which describes how the incoming light radiance L_i is scattered through reflection and transmission. According to [2], for BSDF, f_s is defined as:

$$f_s(\boldsymbol{x}, \boldsymbol{\omega}_i \rightarrow \boldsymbol{\omega}) = F f_r^{\text{specular}}(\boldsymbol{x}, \boldsymbol{\omega}_i \rightarrow \boldsymbol{\omega}) + (1 - F) f_t^{\text{specular}}(\boldsymbol{x}, \boldsymbol{\omega}_i \rightarrow \boldsymbol{\omega}) \quad \text{E(2)}$$

the sum of specular reflection $f_r^{specular}$ and specular transmission $f_t^{specular}$. Both terms are scaled in respect to the Fresnel value *F*, which has two purposes, 1) it takes into account the law of energy conservation by which energy in a closed system in constant, meaning the energy of scattered light cannot exceed the energy of the incoming light, 2) its value determines if the light should be reflected or refracted based on the view angle and ratio of refraction indices of the two materials interfaced at the given point *x*.
The scattered light is computed in *ComputeScatteredLighting( )* (Code Block 1). Here the transmissioned and reflected light are interpolated by glsl's *mix( )* function using the Fresnel value returned by *FresnelBSDF( )*, which will be explained later. But first we need to understand the relevance of the refraction indices.

```
vec3 ComputeScatteredLighting(vec3 reflection, vec3 transmission, SurfaceData data,
vec3 viewDir)
{
   vec3 inDir = viewDir;
   float cosThetaIn = dot(inDir, data.normal);
   float fresnel = FresnelBSDF(cosThetaIn, 1.0f, data.refractionIndex);
   vec3 lighting = mix(transmission, reflection, fresnel);

   return lighting;
}
```

Code Block 1

3.2 Snell's Law

The computation of transmitted light relies on Snell's law E(3), which is based on the refraction indices of the incident material η_i and transmitted material η_t. The refraction index determines the velocity of light passing trough the material, perceived as bending when the light enters from a material with a different index.

$$\eta_\text{i} \ \sin\theta_\text{i} = \eta_\text{t} \ \sin\theta_\text{t}\,. \quad \text{E(3)}$$

The angles from incident direction to normal θ_i and from out direction to normal θ_o are equal (Figure 0). With the given out direction ω_o and the surface normal in point *x*, the cosine of the incident angle $cos\theta_i$ can be calculated by taking the dot product of the two vectors. With the Pythagorean identity, we can solve for $sin\theta_i$ then by Snell's Law derive $sin\theta_t$ to finally get $cos\theta_t$.This is applied in *GetCosThetaTr( )* (CodeBlock 2).

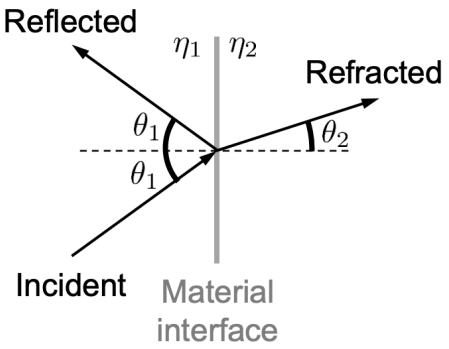

Figure 0: 1 = i; 2 = t. Reproduced from [2].

In the implementation, *etaI* is the index of air and a constant 1.0, while *etaT*, the index of the glass material, can be adjusted as described in Settings. One can also interpret Snell's Law as a definition of the ratio between the indices, which is used in the code as *etaIOverT*. $Cos\theta_t$ is needed to compute not only the transmission direction but also the Fresnel term.

```
float GetCosThetaTr(float cosThetaIn, float etaIOverT)
{
   float sinThetaIn = sqrt(max(0.0f, 1.0f - cosThetaIn * cosThetaIn )); // (sin
theta)^2 + (cos theta)^2 = 1
   float sinThetaTr = sinThetaIn * etaIOverT; // Snell's Law
   float cosThetaTr = sqrt(max(0.0f, 1.0f - sinThetaTr * sinThetaTr ));
   return cosThetaTr;
}
```

Code Block 2

3.3 Fresnel Reflectance

With the simplified assumption that the incoming light is unpolarized, the Fresnel reflectance at the interface between two dielectric materials, in this case air and water, can be computed by taking the reflectance for parallel $r\,/\!/$ and perpendicular $r\perp$ polarized light into account [1]:

$$F_{\mathrm{r}} = \frac{1}{2}(r_{\parallel}^2 + r_{\perp}^2). \qquad \text{E(4)}$$

The equation is implemented in *FresnelBSDF( )* (Code Block 3). With its parameters, it first computes *cosThetaTr* as described in the previous subsection. If *cosThetaTr* is nonpositive, *total internal reflection* (TIR) needs to be handled. This is done by returning the maximum Fresnel reflectance of 1.0 since no transmission is possible. TIR actually occurs when the sine of the transmission angle is zero or bigger, in which case *GetCosTheaTr( )* would return 0 and the value of *cosThetaTr* is checked instead.

F0 and *f90* represent $r_{\parallel}$ and $r_{\perp}$ respectively and are dependent on the refraction indices and the cosines of the incident and transmitted angles. They determine how much light will be reflected if the view direction is paralled or perpendicular to the normal. Removing the Transmission Intensity in the GUI and shifting the Refraction Index value will show its impact on the the Fresnel Reflectance.

```
float FresnelBSDF(float cosThetaIn, float etaIn, float etaTr)
{
   float etaI = etaIn, etaT = etaTr;
   cosThetaIn = clamp(cosThetaIn, -1, 1);

   // // Relevant for ray tracing only … ->

   float cosThetaTr = GetCosThetaTr(cosThetaIn, etaI/etaT);
   if(cosThetaTr <= 0.0f) return 1.0f; // total internal reflection

   float f0 = ((etaT * cosThetaIn) - (etaI * cosThetaTr)) /
              ((etaT * cosThetaIn) + (etaI * cosThetaTr));
   float f90 = ((etaI * cosThetaIn) - (etaT * cosThetaTr)) /
              ((etaI * cosThetaIn) + (etaT * cosThetaTr));
   return (f0 * f0 + f90 * f90) / 2.0f ;
}
```

Code Block 3

3.4 Reflection

For the smooth specular reflection *ComputeSpecularReflection( )* (Code Block 4) simply retrieves the reflection direction *reflectionDir* using glsl's *reflect( )* function, which derives the vector with simple vector geometry using the normal and incident direction, which here is the inverted view direction *-viewDir*, now pointing from the camera to the point *x*. With the reflection direction, the light or color from the environment map is sampled and returned as the reflected light.

```
vec3 ComputeSpecularReflection(SurfaceData data, vec3 viewDir)
{
   vec3 inDir = -viewDir;

   vec3 reflectionDir = reflect(inDir, data.normal);
   vec3 specularReflection = SampleEnvironment(reflectionDir, 0.0f);

   return (specularReflection +
       vec3(data.reflectionRed,0,0)) * data.reflectionIntensity;
}
```

Code Block 4

3.5 Transmission

To get the smooth transmitted light, the environment is sampled, this time using the transmission direction. *TransmissionDir* is computed with the cosine of the transmitted angle retrieved with *GetCosThetaTr( )*, which as a reminder uses Snell's Law. Again the case of TIR is handled, returning no transmission as a vector 3 of zeros. The formula for the transmission direction is based on the fact that a vector ω is the sum of two vectors, one perpendicular $\omega\perp$ and one parallel $\omega\,/\!/$ to the surface normal, with the condition that they all lie in a plane (Figure 1).

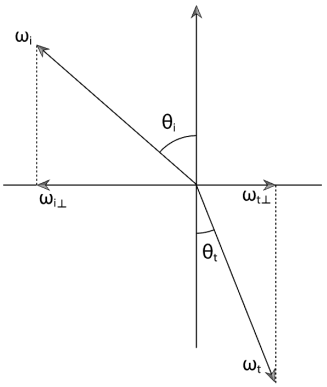

Figure 1: *ω_i* is the incident vector, *ω_t* the transmitted vector. Reproduced from [1].

By definition $\omega_i\perp$ equals $sin\theta_i$. Applying Snell's Law, $sin\theta_t = \eta_i / \eta_t * sin\theta_i$, and flipping the incidence vector , we have $\omega_{t\perp} = \eta_i / \eta_t * (-\omega_{i\perp})$. By definiton of ω, we can solve for ω_t to get the final formula:

$$\omega_{\mathrm{t}} = \frac{\eta_{\mathrm{i}}}{\eta_{\mathrm{t}}}(-\omega_{\mathrm{i}}) + \left[\frac{\eta_{\mathrm{i}}}{\eta_{\mathrm{t}}}(\omega_{\mathrm{i}} \cdot \mathbf{n}) - \cos\theta_{\mathrm{t}}\right]\mathbf{n}. \qquad \text{E(5)}$$

The complete implementation of the transmitted light can be seen in Code Block 5.

```
vec3 ComputeSpecularTransmission(SurfaceData data, vec3 viewDir)
{
   float cosThetaIn = clamp(dot(data.normal, viewDir), -1, 1);
   float etaIOverT =  1.0f / data.refractionIndex ;

   // Relevant for ray tracing only … ->

   float cosThetaTr = GetCosThetaTr(cosThetaIn, etaIOverT);
   if(cosThetaTr <= 0.0f) return vec3(0.f); // total internal reflection

   vec3 inDir = -viewDir;
   // etaIOverT = 1 / etaIOverT; // avoid mirroring
   vec3 transmissionDir = etaIOverT * inDir +
       (etaIOverT * cosThetaIn - cosThetaTr) * data.normal;

   vec3 specularTransmission = SampleEnvironment(transmissionDir, 0.f);

   return (specularTransmission +
       vec3(0.0f, data.refractionGreen ,0)) * data.refractionIntensity;
}
```

Code Block 5

3.6 Light Composition

In *lighting.glsl*, the total light composition is computed in *ComputeLighting( )*. The function returns *ComputeBSDFLight( )* unless it receives *true* as the *direct* attribute, in which case the reflection of the direct point light is added through *ComputeBSDFDirect( )*. The latter is an adaptation of the exercise's *ComputeLight( )*, which is why it won't be further described in the scope of this project.

4. Result

As visible in Figure 3, the outcome is a decent refraction of the environment. The transmission is mirrored on the convex surfaces, e.g. the teapot and the outside surfaces of the cups. The reasoning for this is probably that the refracted light bends toward the inward facing surface normal as illustrated in Figure 2 [3]. In reality, the tea pot is expected to be hollow, through rasterized rendering the object is interpreted as solid until hitting the environment map. If the tea pot was interpreted as hollow with just a thin glass interface, the light would be bent a second time away from the inward normal, avoiding the mirroring. A high refraction index of the transmitted material intensifies the bending and is likely

causing the mirrored look. Lowering the refraction index in the GUI removes the mirroring as seen in image D of Figure 3. One could chose to flip the relative refraction index when computing the transmission direction, to avoid the mirroring. That would be an artistic choice and not accurate according to E(5).

While the outcome gives a credible impression of glass, the transmission and reflection are only a rough approximation. Looking back to the render equation E(1), it is in fact an integral over the omega space S^2, centered around x with a positive half oriented towards the point's normal and a negative hemisphere in the opposite direction. The scattered light is computed from an aggregation of a distribution of incident lights within that omega space. In this project, however, the only incident directions sampled are from the view direction and therefore always only in the positive space. This is due to the use of rasterization as the rendering technique.

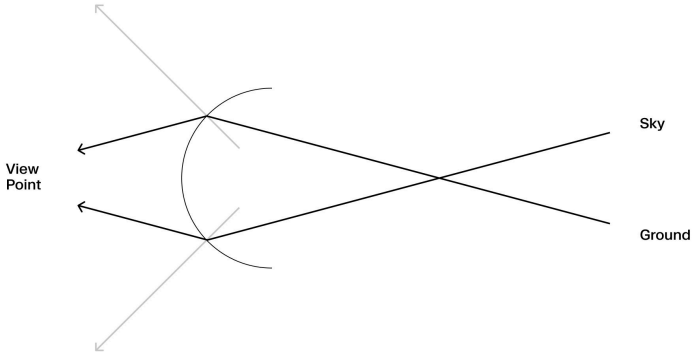

Figure 2: Rough illustration of refraction through a convex surface. Grey: Normals of the surface points. Black: Refracted rays. Adapted from [3].

5. Future improvements

While the project outcome is overall satisfactory, there are two main points of improvement that would elevate the scattered lighting significantly.

5.1 Microfacet Model for Rough Dielectric BSDF

Due to lack of time, the microfacet model for rough surfaces wasn't implemented. It is currently only applied to the direct pointlight reflection, which was taken over from the course material. The proposed BSDF of this project is only for perfectly smooth surfaces where one can assume perfect reflection and transmission. This doesn't reflect the physical world, where the light is scattered through the surface's roughness at micro level and only a fraction is reflected and refracted, due to shadowing and masking. For BSDF not only the

reflection but also the transmission takes the roughness into account. The next step of this project would be to implement the Microfacet Model following [4].

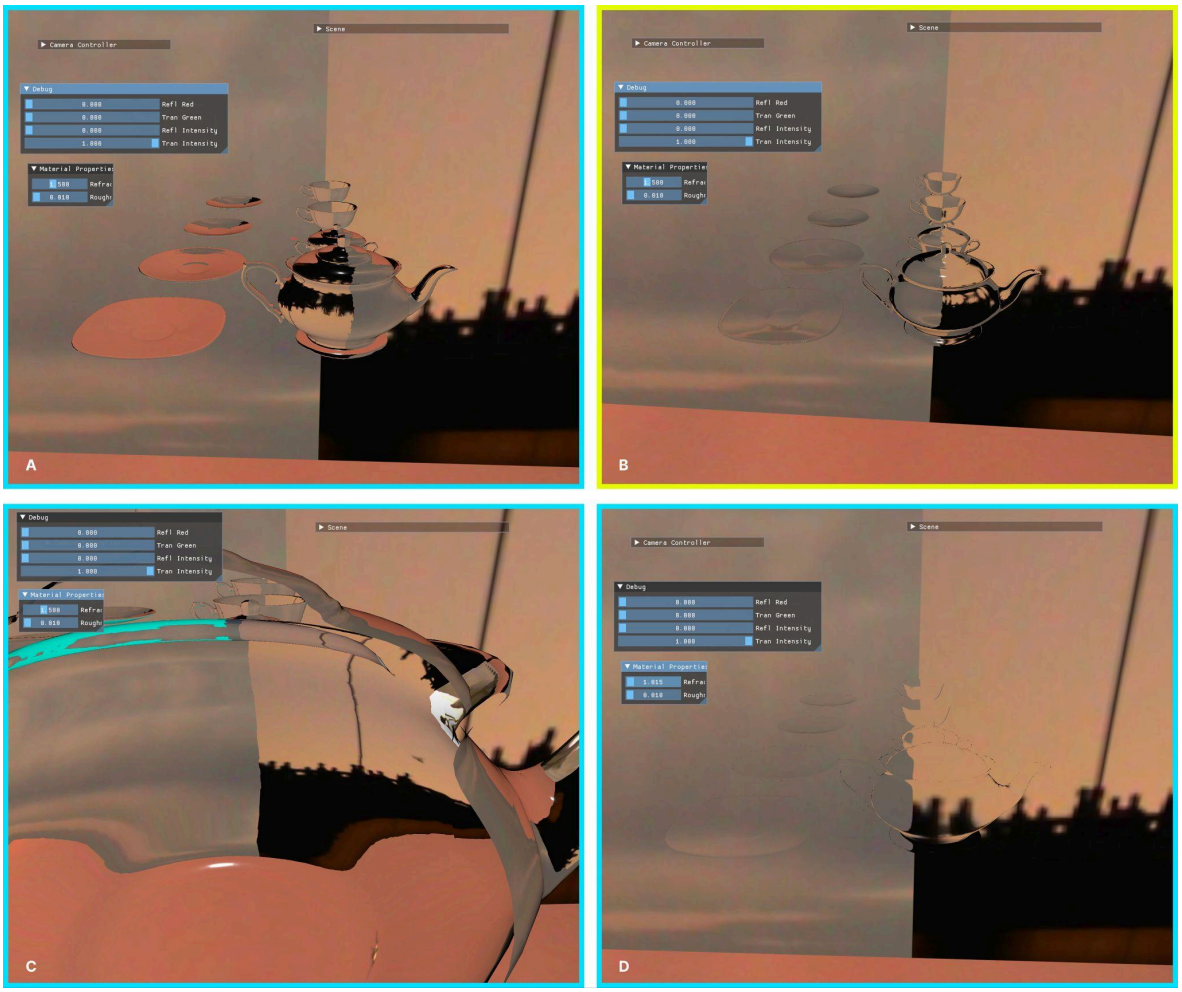

Figure 3: Screenshots of the glass objects' transmission. Cyan framed: ratio of indices is η_i / η_t; Yellow framed: ratio of indices is flipped. C: View from inside the object where the surface is concave. D: refraction index of 1.015.

5.2 Ray Tracing

Instead of rasterization, a more sophisticated rendering approach would be ray tracing. When transmitting a material, it usually has a finite thickness framed by a front and a back face. With rasterization only the front face is taken into account, as it is the closest to the camera and data behind it is discarded. Ray tracing would enable refraction into the glass through the front face and out of the glass through the back face, allowing for a more accurate refraction direction and light sampling. This would solve the inaccurate mirrored transmission on the hollow teapot, assuming its mesh is constructed as hollow.

The previous also explains why the back edges of the glass objects or other objects behind the closest glass surface are not visible, but only the environment map where the light is sampled from. Through ray tracing, light can be refracted through multiple transmittable surfaces and eventually reflected back through transmitted materials. The transmitted and reflected directions are considerate of all objects in the environment and redirected in a physically accurate way.

The rendering technique is inclusive of the negative omega space. In rasterization, the incident light is entering the glass material at point *x* coming from the view point, while with ray tracing it can also be exiting the glass material, depending on its origin and direction. In that case, when implementing BSDF, it is necessary to determin if the incident direction is entering the glass material. If not, the refraction indices are exchanged since the glass is now the incident and the air the transmitted material. While in theory this is key in achieving the correct light condition when using BSDF, In *FresnelBSDF( )* (Code Block 3) and *ComputeSpecularTransmission( )* (Code Block 5) the code handling this logic is commented out since it has no impact with rasterization in this specific implementation.

5. References

[1] Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2018. *Physically based rendering: From theory to implementation* (3rd ed.). Chapter: Specular Reflection and Transmission. Retrieved from https://www.pbr-book.org/3ed-2018/Reflection_Models/Specular_Reflection_and_Transmission

[2] Shuang Zhao. 2017. *Refraction & BSDFs*. Lecture slides, CS295: Advanced Computer Graphics, University of California, Irvine. Retrieved from https://ics.uci.edu/~shz/courses/cs295/slides/14_bsdf.pdf

[3] Juno Kim and Phillip J. Marlow. 2016. Turning the World Upside Down to Understand Perceived Transparency. I-Perception, 7(5). https://doi.org/10.1177/2041669516671566 (Original work published 2016)

[4] Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. 2007. Microfacet models for refraction through rough surfaces. In Proceedings of the 18th Eurographics conference on Rendering Techniques (EGSR'07). Eurographics Association, Goslar, DEU, 195–206.

[5] Caroline Vang. 2024. *CPH:DOX*. Retrieved from https://carolinevang.com/cphdox/

[6] Mishael Fapohunda. 2024. *Untitled*. Retrieved from https://mishaelfapohunda.com/9o46jolzhulfkpqewj14d51khao54c

[7] Dezeen. 2024. Lumion 3D rendering software designed for unparalleled ease. *Dezeen* (May 30, 2024). Retrieved from https://www.dezeen.com/2024/05/30/lumion-3d-rendering-software-design-unparalleled-ease/

[8] Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2018. *Physically based rendering: From theory to implementation* (3rd ed.). Retrieved from https://www.pbr-book.org/