Physically based Rendering

Fabian Meister

Matrikelnummer: 298 5828
E-Mail: meisterfa68426@th-nuernberg.de
Technische Hochschule Nürnberg Georg Simon Ohm

1 Einleitung

Der Begriff Physically Based Rendering/Shading (kurz PBR) beschreibt einen Überbegriff, welcher verschiedene Rendering Methoden und Techniken umfasst. Diese basieren auf physikalischen Theorien und Prinzipen, welche darauf ausgerichtet sind, die Wechselwirkung zwischen Licht und Materie so korrekt wie möglich zu modellieren. Zu diesen physikalischen Gegebenheiten zählt beispielweise die Energieerhaltung innerhalb eines Systems. (vgl. S.133 David Wolf und DeFries)

Das PBR ist dennoch keine physikalisch korrekte Simulation des Lichtes, da es Approximations-funktion verwendet, um den Arbeitsaufwand und somit die gesamt Berechnungsdauer zu verringern. Aus diesem Grund wird es Physically Based (zu dt. physikalisch basierendes) Rendering genannt und nicht Physically Rendering. (DeFries)

2 Physikalische Grundlagen der Radiometrie

3 Reflectance Equation

3.1 Bidirectional Reflective Distribution Function

3.2 Lambert Diffuse BRDF

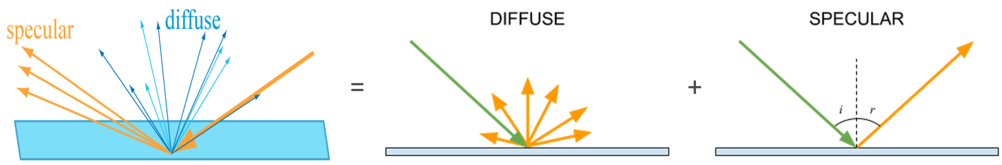
3.3 Cook-Torrance specular BRDF

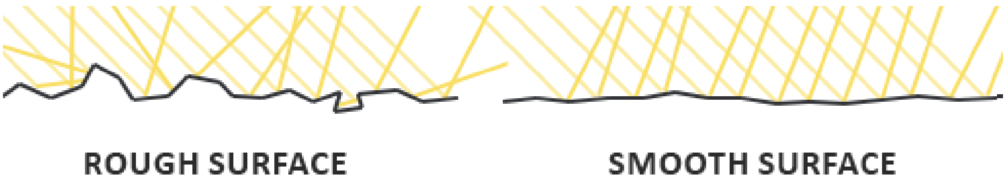
3.3.1 Microfacet Surface Model

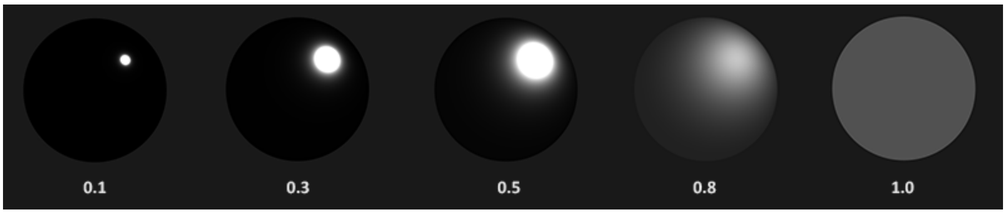
3.3.2 Normal distribution function

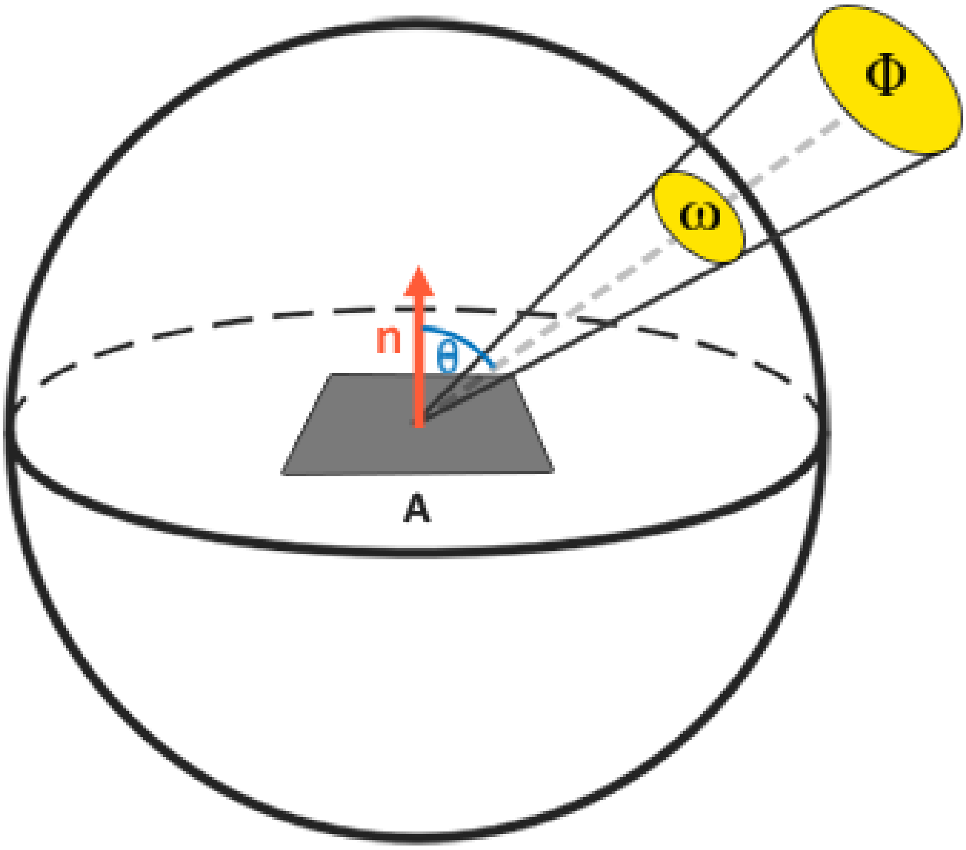
3.3.3 Geometry function

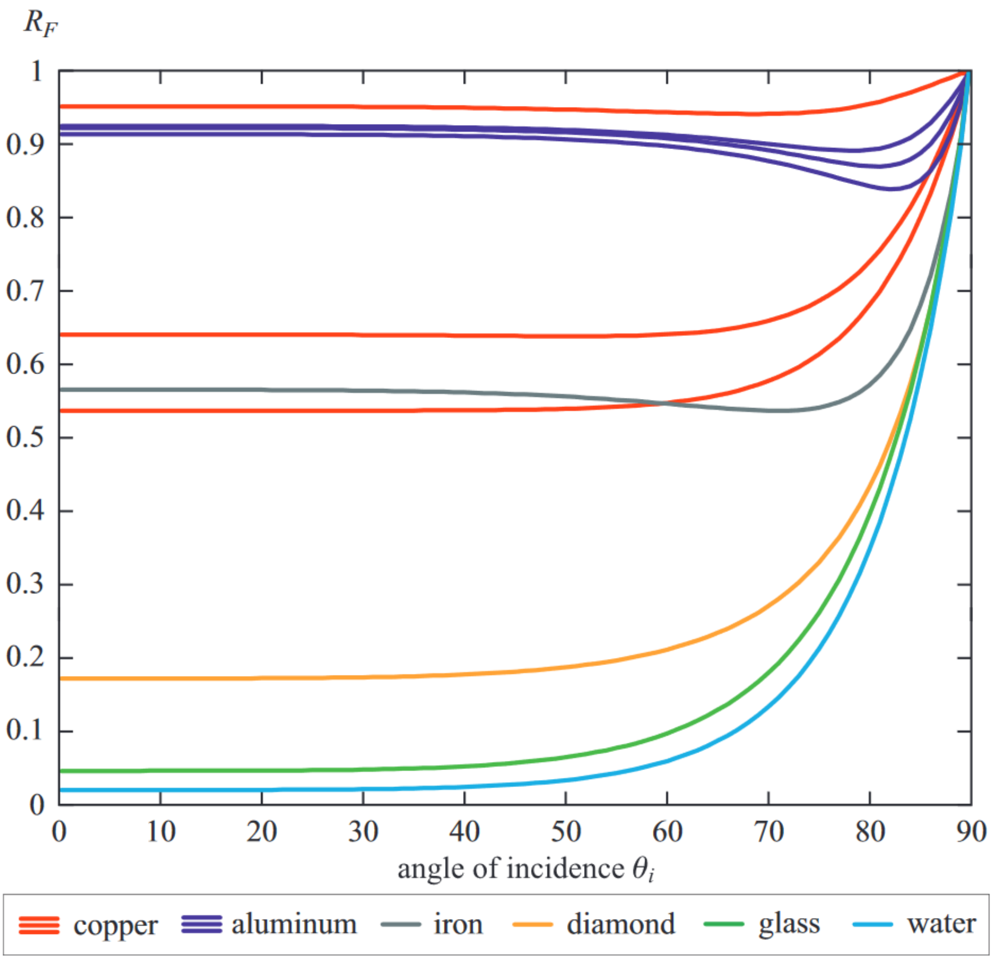
3.3.4 Fresnel equation

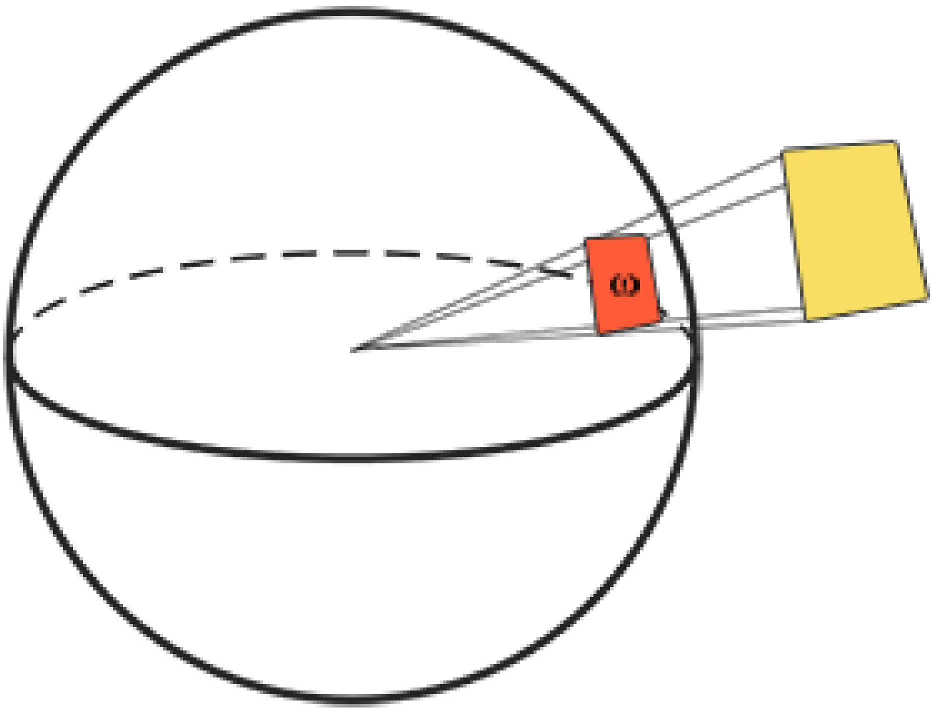


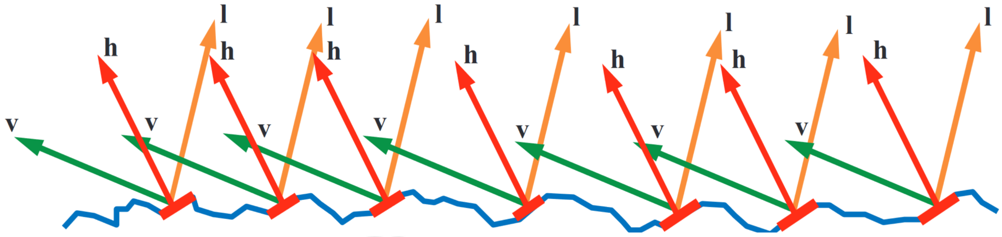


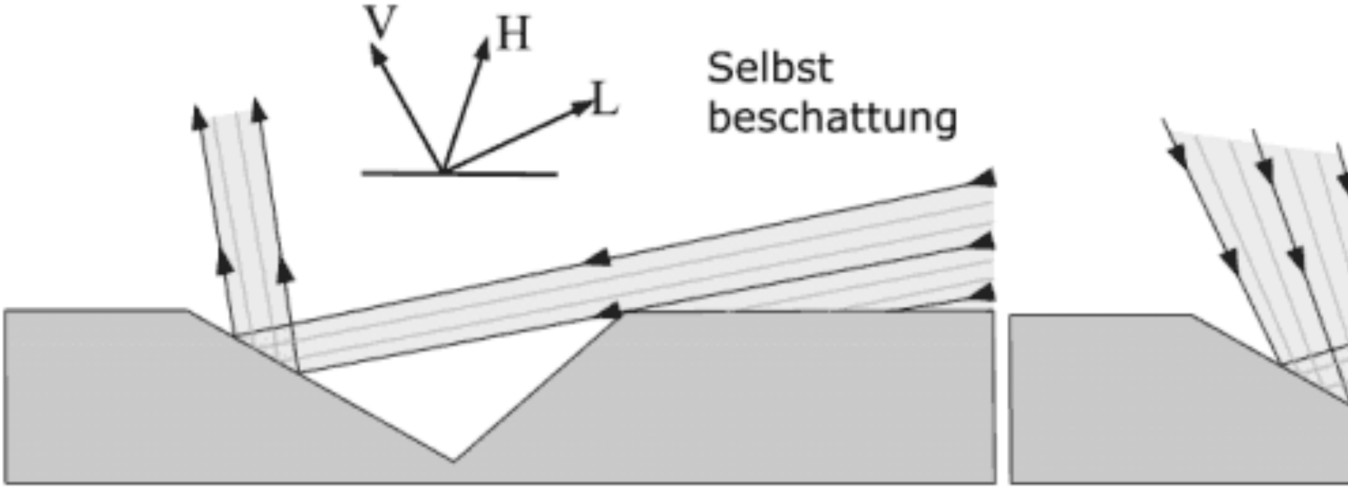

0.1
0.3
0.5
0.8
1.0

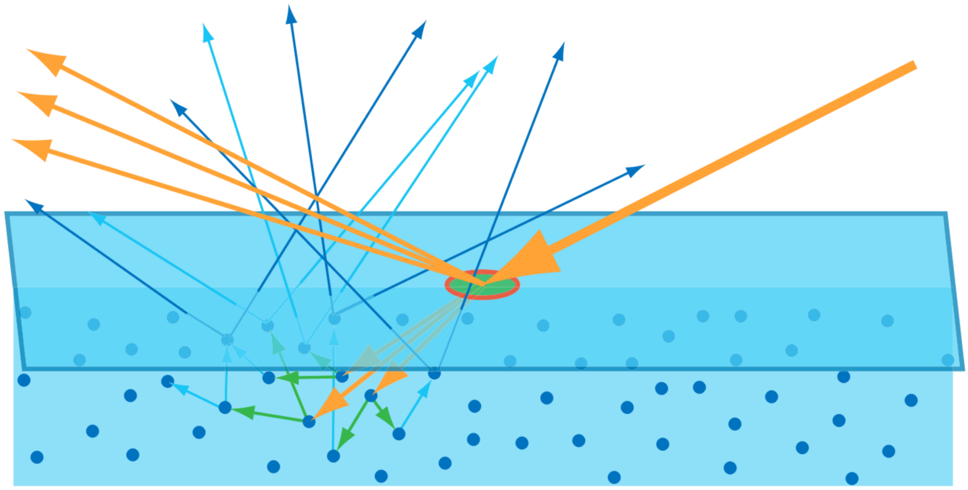

Φ
ω
n
θ
A


R_F
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
10
20
30
40
50
60
70
80
90
angle of incidence θ_i
copper
aluminum
iron
diamond
glass
water


ω







```
float geomSmith( float dotProd ) {
  float k = (Material.Rough + 1.0) * (Material.Rough + 1.0) / 8.0;
  float denom = dotProd * (1 - k) + k;
  return 1.0 / denom;
}
```

```
vec3 F = schlickFresnel(hDotV);
vec3 kD = mix((vec3(1.0) - F), vec3(0.0), Material.Metal);
(kD * diffuseBrdf /  PI  + specBrdf) * lightIntensity * nDotL
```

```
if (meshNumber == 1){
  model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f,
      1.0f, 0.0f));
  setMatrices();
  mesh1->render();
```

```
}
else if (meshNumber == 2){
  ...
}
else if (meshNumber == 3){
  ...
}
```

```
if (glfwGetInputMode(window, GLFW_CURSOR) == GLFW_CURSOR_DISABLED)
```

```
while( ! glfwWindowShouldClose(window) && !glfwGetKey(window,
    GLFW_KEY_ESCAPE) ) {
  keypress = "";
  [...]
  processKeypress(window, keypress);
  scene->update(float(glfwGetTime()), keypress);
  [...]
}
```

```
front.x = sin(glm::radians(Yaw)) * cos(glm::radians(Pitch));
front.y = sin(glm::radians(Pitch));
front.z = ((-1) * cos(glm::radians(Yaw))) * cos(glm::radians(Pitch))
    ;
```

```
void processKeypress(GLFWwindow* window, std::string& keypress)
{
    if (glfwGetKey(window, GLFW_KEY_1) == GLFW_PRESS)
        keypress = "gold";
    else if (glfwGetKey(window, GLFW_KEY_2) == GLFW_PRESS)
        keypress = "copper";
    [...]
    else if (glfwGetKey(window, GLFW_KEY_C) == GLFW_PRESS)
        animate = true;
}
```

```
void ScenePbr::processKeyboardInput(std::string& keypress, float
    deltaT)
{
  if (keypress == "forward")
    camera.ProcessKeyboard(FORWARD, deltaT);
  else if (keypress == "backward")
    camera.ProcessKeyboard(BACKWARD, deltaT);
  [...]
}
```

```
glfwGetCursorPos(window, &xpos, &ypos);
scene->updateMouseMovement(xpos, ypos);
```

```
if (objMaterial == "gold")
{
  // Gold
  drawSpot(glm::vec3(0.0f, 0.0f, 1.5f), metalRough, 1, glm::vec3(1,
      0.71f, 0.29f));
}
[...]
else if (objMaterial == "copper")
{
  // Copper
```

```
    drawSpot(glm::vec3(0.0f, 0.0f, 1.5f), metalRough, 1, glm::vec3
        (0.95f, 0.64f, 0.54f));
}
```

$\Phi = \frac{\Delta Q}{\Delta t}$ $\omega = \frac{A}{r^2}$ $L = \frac{\Delta\Phi\cdot}{\Delta\Omega\cdot\Delta A\cdot\cos\varepsilon}$ $E = \frac{\Delta E}{\Delta A}$ $E = \int_\Omega L\cdot\cos\varepsilon\ d\omega$ $L_o\left(p,\omega_o\right) = \int_\Omega f_r\left(p,w_i,W_o\right)L_i\left(p,w_i\right)n\cdot w_i\,dw_i$ $f_r\left(p,w_i,w_o\right) = \frac{\Delta L_o\left(p,\omega_o\right)}{\Delta E_i\left(p,w_i\right)}$ $f_{Lambert} = \frac{c}{\pi}$ $f_{CookTorrance} = \frac{D\cdot G\cdot F}{4\cdot\left(w_0\cdot n\right)\left(w_i\cdot n\right)}$ $D_{GGXTR}\left(n,h,\alpha\right) = \frac{\alpha^2}{\pi\cdot\left(\left(n\cdot h\right)^2\cdot\left(\alpha^2-1\right)+1\right)^2}$ $G_{SchlickGGX}\left(n,v,k\right) = \frac{n\cdot v}{\left(n\cdot v\right)\cdot\left(1-k\right)+k}$ $k_{direct} = \frac{\left(\alpha+1\right)^2}{8}$ $G_{Smith}\left(n,v,l,k\right) = G_{sub}\left(n,v,k\right)\cdot G_{sub}\left(n,l,k\right)$ $F_{Schlick}\left(h,v,F_0\right) = F_0+\left(1-F_0\right)\cdot\left(1-\left(h\cdot v\right)\right)^5$

$$L_0\left(p,\omega_0\right) = \int_\Omega\left(k_d\cdot\frac{c}{\pi}+k_s\frac{D\cdot G\cdot F}{4\cdot\left(w_i\cdot n\right)\left(\omega_0\cdot n\right)}\right)\cdot L_i\left(p,w_i\right)\cdot n\cdot\omega_i\ d\omega_i$$

[3, 10, 14, 6] [5] [6, 7] [13, 16, 2, 6] [15] [13][13][8, 17, 1] [6] [12] [6][19, 6][7, 18] [7] [4, 6, 18, 7] [7, 14, 6, 9, 4] [7, 14, 6, 9, 4] [4] [4, 7, 6][4, 6][4][6] [6, 11] [6, 11, 4] [6, 11, 4] [6] [6] [11][11] [6] [6] [4, 11][4, 11, 6][6] [6]

Literatur

[1] Online-tutorium computergrafik. http://www.bungenstock.de/studienarbeit/lambertapplet/lambert.html. [Online; Accessed: 2020-12-03].

[2] Tros laserstrahlung teil - allgemeines. http://regelwerke.vbg.de/vbg_trosla/tros_la0/tros_la0_0_.html. [Online; Accessed: 2020-12-03].

[3] Adopting a physically based shading model. https://seblagarde.wordpress.com/2011/08/17/hello-world/, 2011. [Online; Accessed: 2020-12-03].

[4] Tomas Akenine-Mller, Eric Haines, and Naty Hoffman. *Real-Time Rendering, Fourth Edition*. A. K. Peters, Ltd., USA, 4th edition, 2018.

[5] Jürgen Beyerer, Fernando Puente León, and Christian Frese. *Radiometrie*, pages 179–201. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[6] Joey de Vries. *Learn OpenGL: Learn modern OpenGL graphics programming in a step-by-step fashion*. Kendall & Welling, June 2020.

[7] Nikolaus Gebhardt. Einige brdf modelle. [Online; Accessed: 2020-12-03].

[8] J.J. Gray. Johann heinrich lambert, mathematician and scientist, 1728 – 1777. *Historia Mathematica*, 5(1):13 – 41, 1978.

[9] Romain Guy and Mathias Agopian. Physically based rendering in filament. https://google.github.io/filament/Filament.html#overview/physicallybasedrendering. [Online; Accessed: 2020-12-03].

[10] Koray Hagen. Physically-based rendering: Theory and practice. https://de.slideshare.net/korayhagen/physically-based-rendering, April 2015. [Online; Accessed: 2020-12-03].

[11] Naty Hoffman. Background: Physically-based shading. SIGGRAPH '12, 2010.

[12] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, August 1986.

[13] Siegfried Kokoschka. Grundlagen der lichttechnik. `http://www.lti.kit.edu/rd_download/Grundlagen_der_Lichttechnik(1).pdf`. [Online; Accessed: 2020-12-03].

[14] Wes McDermott. The pbr guide - part 1. `https://academy.substance3d.com/courses/the-pbr-guide-part-1`, 2018. [Online; Accessed: 2020-12-03].

[15] Gordon Müller. Beschleunigung strahlbasierter rendering-algorithmen. Master's thesis, Rheinische Friedrich-Wilhelms-Universitat Bonn, 1997.

[16] Jürgen Nolting and Günter Dittmar. How does it work? – teil 13: Radiometrische grundbegriffe. `https://www.hs-aalen.de/uploads/publication/file/9371/1005_Nolting.pdf`. [Online; Accessed: 2020-12-03].

[17] Alex Ryer, Ultraviolet Light, and Visible Light. Light measurement handbook, 1997.

[18] Ulrich Weidenbacher. *Beleuchtungsmodelle.*

[19] David Wolff. *OpenGL 4 Shading Language Cookbook: Build High-Quality, Real-Time 3D Graphics with OpenGL 4.6, GLSL 4.6 and C++17, 3rd Edition.* Packt Publishing, 2018.