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

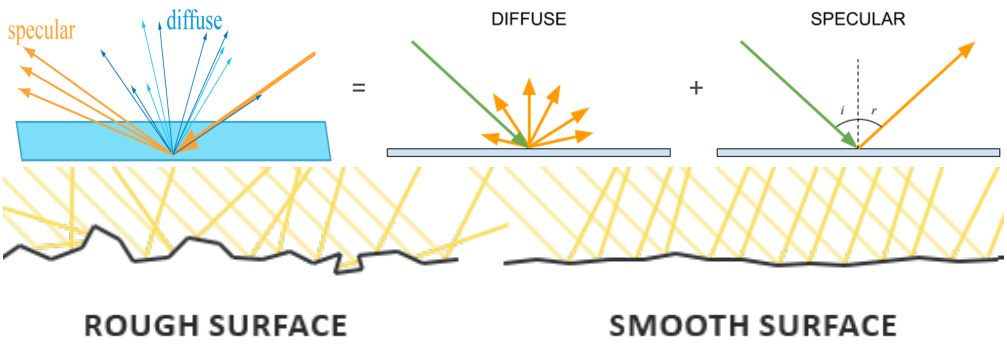
3.3 Cook-Torrance specular BRDF

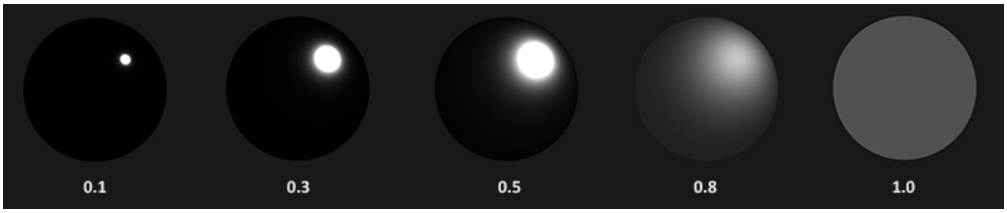
3.3.1 Microfacet Surface Model

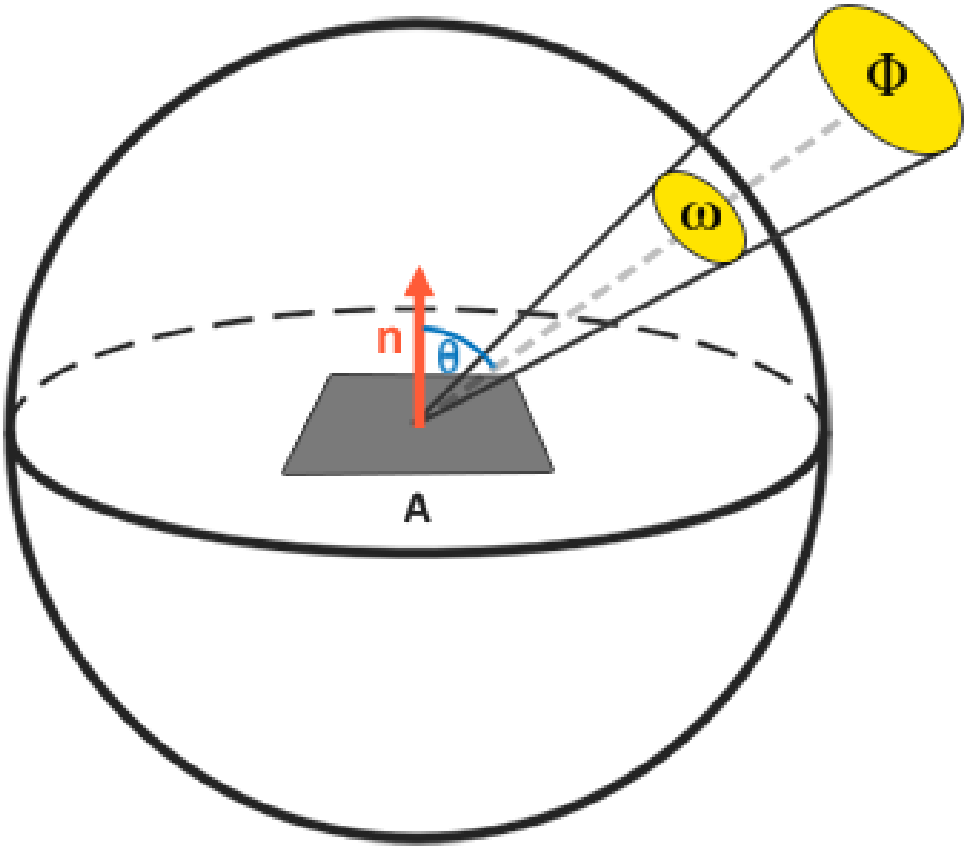
3.3.2 Normal distribution function

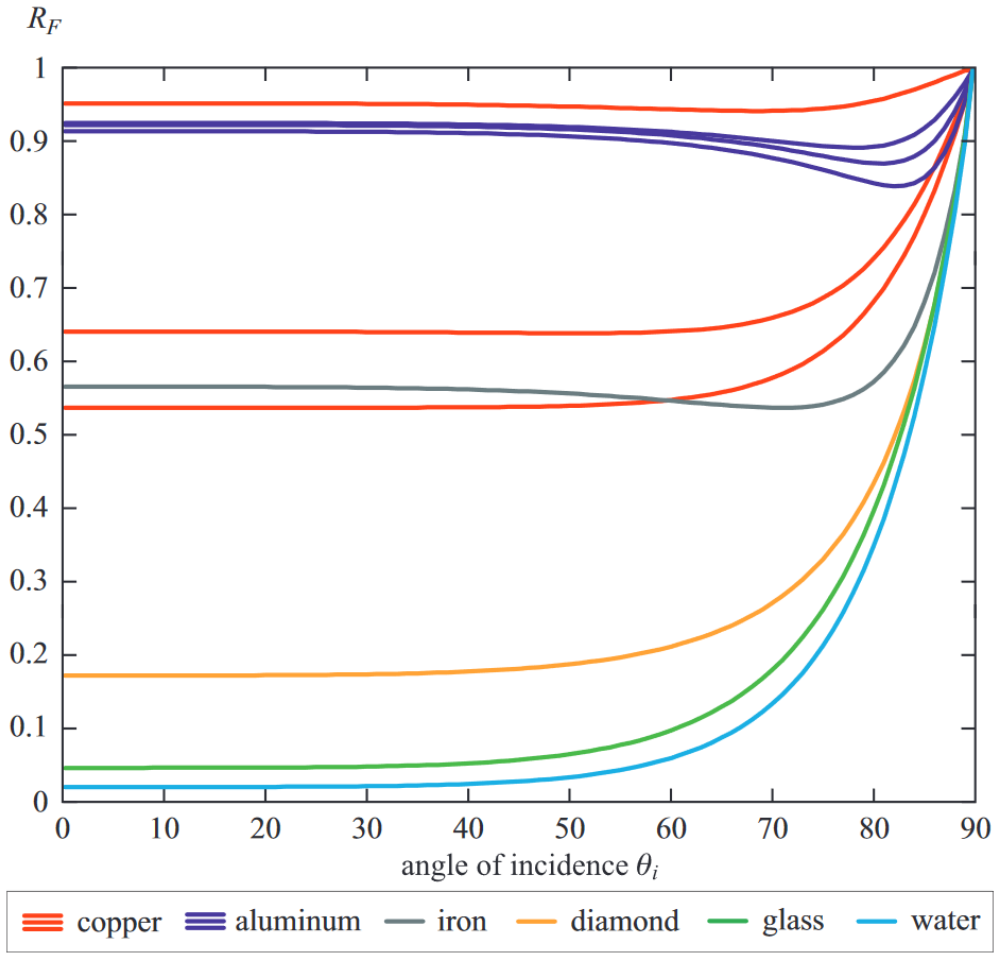
3.3.3 Geometry function

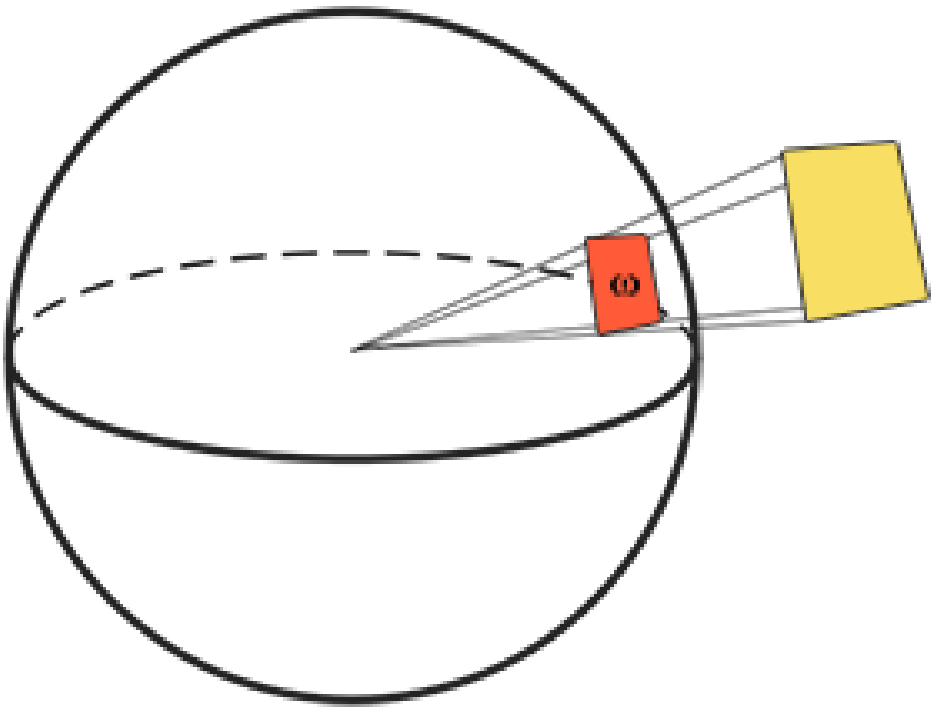
3.3.4 Fresnel equation

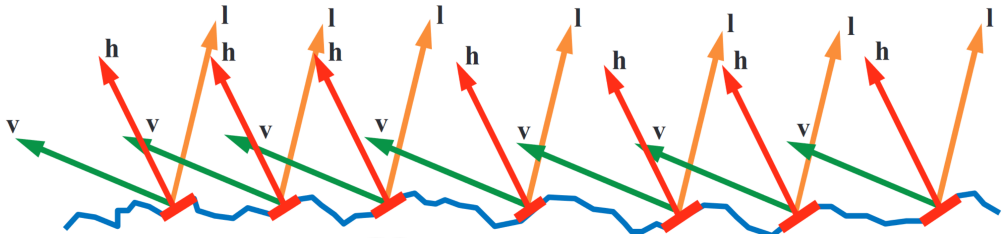


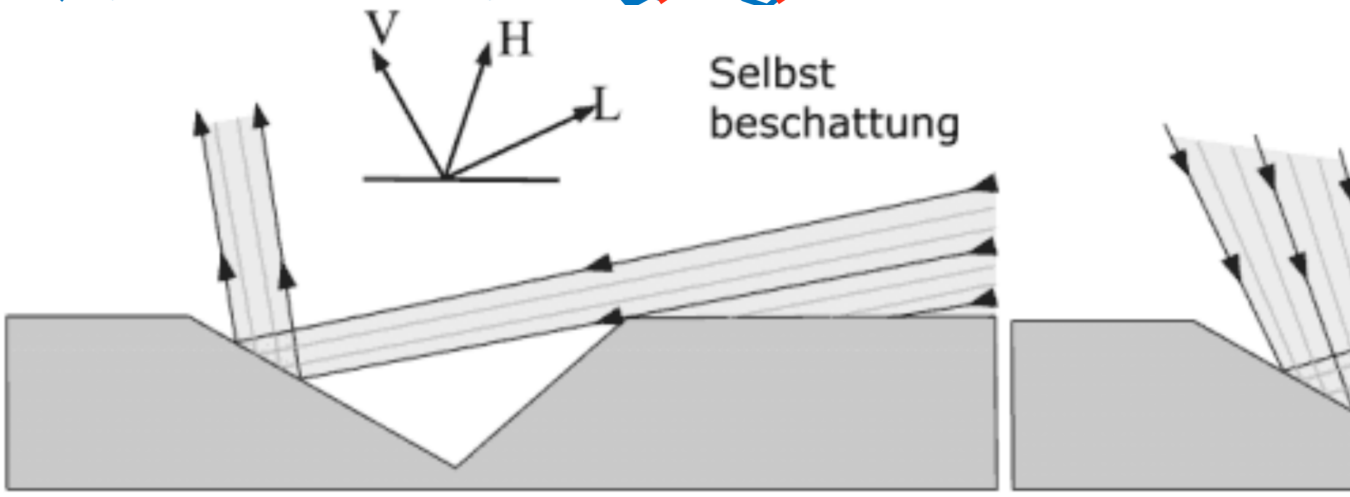

0.1
0.3
0.5
0.8
1.0

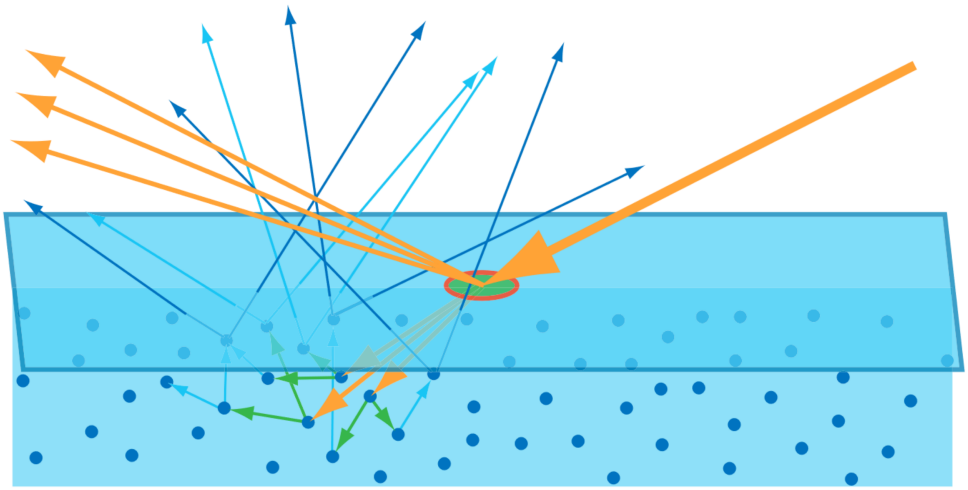

Φ
ω
n
θ
A


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));
}
```

Literatur