1 Introduction

Materials reflect the way light interact with objects, and are thus of great importance to the realism of a scene. In previous practicals we used the simplified Phong model to shade our objects. However this model is limited and do not take into account some important aspects of real life materials. As introduced during the lecture, material surfaces will be represented as micro-facets with varying orientations and organization. Moreover, we introduced the concept of roughness and self occlusions of the micro-facets. Additionally we also introduced a new lighting model that follows the reflectance equation:

\[ L_o(v) = \int_\Omega f_r(l,v)L_i(l)n \cdot ld\omega_i \text{ where} \]

In this model, the amount of reflected, refracted and diffused light is described by a BRDF function. One major aspect of the BRDF function is its energy conservative property, imposing that the amount of light reflected \( k_s \) must follow the \( k_s = 1 - k_d \) constraint, \( k_d \) being the amount of light diffused. Several BRDF models have been proposed during the past decades, in this practical we will focus on the Cook-Torrance model. This model use the Lambertian distribution function as the diffuse part and propose a specular distribution function based on three component functions: the normal Distribution function, the Fresnel equation and the Geometry function. The final BRDF function equation is:

\[ f_r = k_d f_{\text{lambert}} + k_s f_{\text{cook-torrance}} \]

with

\[ f_{\text{lambert}} = \frac{c}{\pi} \]

\[ f_{\text{cook-torrance}} = \frac{DFG}{4(\omega_o \cdot n)(\omega_i \cdot n)} \]

Let’s dig a little bit deeper into the three component of the specular distribution. The normal distribution function (NDF) gives an estimation of the amount of micro-facets facing the half-vector \( H = V + L \), the more micro-facets face the half-vector the less spread the specularities on the surface are. In this practical we will use the GGX distribution function as the NDF:

\[ \text{NDF}_{\text{GGXTR}}(n, h, \alpha) = \frac{\alpha^2}{\pi((n \cdot h)^2(\alpha^2 - 1) + 1)^2} \]

The geometry function, here represented by the Schlick function, gives the amount of occluded micro-facets with respect to a given direction. Note that we final distribution takes into account occluded faces with respect to the view vector and with respect to the light vector, taking into consideration both the obstruction and shadowing phenomena.

\[ G_{\text{SchlickGGX}}(n, v, k) = \frac{n \cdot v}{(n \cdot v)(1 - k) + k} \text{ with } k = \frac{(\alpha + 1)^2}{8} \]

\[ G(n, v, l, k) = G_{\text{sub}}(n, v, k)G_{\text{sub}}(n, l, k) \]
Finally the Fresnel function model the Fresnel effect which states that more reflection occurs at
gazing angles, a common phenomenon that can be observed when looking at puddles for instance.

\[ F_{\text{Schlick}}(h,v,F_0) = F_0 + (1 - F_0)(1 - (h \cdot v))^5 \]  

We are now ready to start implementing the PBR shading into our current framework. First,
we will add micro-facet properties to the HitSurface structure, adding roughness, ambient occlusion
and metallic properties.

```c
struct HitSurface {
    vec3 hit_point;
    vec3 normal;
    vec3 color;
    float roughness;
    float ao;
    float metallic;
};
```

Next, in the same way we implemented the Phong illumination function, we will implement the
PBR direct illumination function which will be called inside the directIllumination function.

```c
def PBR(in HitSurface hit, in Light l, in vec3 l_dir)
{
    vec3 ambient = vec3(0.03) * hit.color * hit.ao;
    vec3 F0 = vec3(0.04);
    F0 = mix(F0, hit.color, hit.metallic);
    vec3 N = ...
    vec3 Ve = ...
    vec3 H = normalize(Ve + l_dir);
    float attenuation = ...
    vec3 light_color = ....
    return ambient + computeReflectance(N, Ve, F0, hit.color, l_dir, H, light_color, attenuation, hit.metallic, hit.roughness);
}
```

Below, we provide you all the necessary functions in order to compute the outgoing reflectance:

```c
float DistributionGGX(vec3 N, vec3 H, float roughness)
{
    float a = roughness*roughness;
    float a2 = a*a;
    float NdotH = max(dot(N, H), 0.0);
    float NdotH2 = NdotH*NdotH;
    float nom = a2;
    float denom = (NdotH2 * (a2 - 1.0) + 1.0);
    return nom / denom;
}

float GeometrySchlickGGX(float NdotV, float roughness)
{
    float r = (roughness + 1.0);
    float k = (r*r) / 8.0;
    float nom = NdotV;
    float denom = NdotV * (1.0 - k) + k;
    return nom / denom;
}

float GeometrySmith(vec3 N, vec3 V, vec3 L, float roughness)
{
    float NdotV = max(dot(N, V), 0.0);
    float NdotL = max(dot(N, L), 0.0);
    float ggx2 = GeometrySchlickGGX(NdotV, roughness);
    float ggx1 = GeometrySchlickGGX(NdotL, roughness);
    return ggx1 * ggx2;
}
vec3 fresnelSchlickRoughness(float cosTheta, vec3 F0, float roughness)
{
return F0 + (max(vec3(1.0 - roughness), F0) - F0) * pow(((1.0 + 0.0000001/avoid 0 power undefined behaviors/) - cosTheta, 5.0);
}

vec3 computeReflectance(vec3 N, vec3 Ve, vec3 F0, vec3 albedo, vec3 L, vec3 H, vec3 diffuse, float attenuation, float metallic, float roughness)
{
    vec3 radiance = diffuse * attenuation;
    // cook-torrance brdf
    float NDF = DistributionGGX(N, H, roughness);
    float G = GeometrySmith(N, Ve, L, roughness);
    vec3 F = fresnelSchlickRoughness(max(dot(H, Ve), 0.0), F0, roughness);
    vec3 kS = F;
    vec3 kD = vec3(1.0) - kS;
    // metallic materials do not diffuse, only reflect
    kD *= 1.0 - metallic;
    vec3 nominator = NDF * G * F;
    float denominator = 4 * max(dot(N, Ve), 0.0) * max(dot(N, L), 0.0) + 0.001;
    vec3 specular = nominator / denominator;

    // add to outgoing radiance Lo
    float NdotL = max(dot(N, L), 0.0);
    return (kD * (albedo) / PI + specular) * radiance * NdotL;
}

Finally, two last changes have to be taken into consideration to complete our model: first shadows have to be modeled in a more realistic manner, thus we will take the PBR ambient term into account for shadowed surfaces. Additionally, the reflection attenuation factor we used in our previous model was completely arbitrary, whereas in this new PBR context it can be computed accurately using the Fresnel function.

vec3 directIllumination(in HitSurface hit, inout float refl)
{
    vec3 color = vec3(0);
    for(int i = 0 ; i < light_nbr ; i++)
    {
        Ray l_ray;
        float shadow_fact = 0.3;
        if(lighted)
        {
            color += PBR(hit, lights[i], l_ray.rd);
        }
        else
        {
            color += PBR_{ambient}*shadow_fact;
        }

        // Reflection factor
        vec3 Ve = ...
        vec3 H = ...
        refl = fresnelSchlickRoughness(max(dot(H, Ve), 0.0), mix(vec3(0.04), vec3(1.0f), hit.metallic), hit.roughness).y*hit.ao;
    }
    return color;
}

Below are some outputs of PBR raytraced spheres and planes with different material properties.
(a) Rough Di-electric Materials  
(b) Smooth Metal Materials  
(c) Rough Metal Materials  
(d) Semi Electric Materials  
(e) Random Materials