IM420 Advanced Computer Graphics – WS 2020
Assignment 5 - PBR Material

Egle Sabaliauskaite, Sahabaj Barbhuiya

22nd January, 2021

1.1 Task

This assignment report and code snippets aims to explain the implementation of *Physically Based Rendering(PBR)* materials. This assignment is based on the lectures and project template of *pbr_solution*. In this assignment report we will discuss about the concept of Physically Based Rendering(PBR), where we try to imitate the light in a more realistic way, rather than using the conventional Blinn-Phong lighting models.

To recreate the PBR materials, we have used a plastic panel object with texture and a wooden object and implemented the newly created PBR materials on a sphere and a cube in our application.

1.2 Physically based rendering(PBR)

PBR or physically based rendering aims to imitate light in a more realistic way. Physically based rendering looks more realistic as compared to the conventional Blinn-Phong lighting models. PBR is also based on the principles of physics. To implement the PBR materials, we need to use *Bidirectional Reflectance Distribution Function (BRDF)*, which s a 4D function that describes the radiance of incoming and outgoing right ray vectors v and l. To implement the PBR material we use *diffuse BRDF* and *specular BRDF*. The diffuse BRDF can be given as the following equation:

$$f_d = \frac{c}{\pi}$$

where, c is the diffuse albedo or surface color(diffuse surface texture).

The specular BRDF is more complex than the diffuse BRDF, which can be given as the following equation:

$$f_s(l, v) = \frac{D(h)F(v,h)G(l,v,h)}{4(n \cdot l)(n \cdot v)}$$

where,

- D is the normal distribution function, which approximates the amount the surface's microfacets are aligned to the halfway vector $\boldsymbol{h}$, influenced by the roughness of the surface

- F is the Fresnel equation, which describes the ratio of surface reflection at different surface angles
- G is the geometry function, which describes the self-shadowing property of the microfacets

1.3 Approach to the solution

To implement the PBR materials, we first took some pictures of a plastic panel and a wooden object with some texture. Figure. 1.1 shows the two materials which is used in our PBR pipeline.

Figure 1.1: Plastic and wooden material used to recreate PBR material

Each of the surface parameters such as albedo, normal map, metallic, roughness, and ambient occulsion can be modeled as textures. All the material parameters are defined in the fragment shader as uniforms.

```
1 // material parameters from Textures
2 uniform float useTextures;
3 uniform sampler2D albedoMap;
4 uniform sampler2D normalMap;
5 uniform sampler2D metallicMap;
6 uniform sampler2D roughnessMap;
7 uniform sampler2D aoMap;
```

To recreate object's appearance as our PBR material, we used and application *Materialize* which can generate different textures from an image. Materialize application generated albedo, normal map, metallic, roughness, and ambient occulsion textures. An example of different textures for our plastic material is shown in the Figure. 1.2 with different textures which will be used in our PBR pipeline.

Figure 1.2: Different textures to be used in PBR Pipeline as textures

All the computations related to diffuse and specular BRDF takes place in the fragment shader to be used in our PBR pipeline. Although, we have to delegate the textures from the main application to our fragment shader so that the textures can be used in the PBR pipeline. The following code snippet from the main application shows the usage of assets such as models and textures to be used.

```
AssetManager assets({
    {   "plastic", //group
    {   { "model"   , "../resources/simple/sphere.obj" },
        { "transformation"   , glm::scale(glm::mat4(1.0f), glm::vec3(1.0))  },
        //{ TEX_FLIP, true }, // if true, causes textures to be flipped in y
        { "albedo" , "../resources/objects/plastic/plastic_diffuse.bmp" },
        { "normal"  , "../resources/objects/plastic/plastic_normal.bmp"},
        { "metallness"  , "../resources/objects/plastic/plastic_metallic.bmp"},
        { "roughness"  , "../resources/objects/plastic/plastic_height.bmp"},
        { "ao"  , "../resources/objects/plastic/plastic_ao.bmp"}
        }
        },

    {   "wood", //group
    {   { "model"   , "../resources/simple/cube.obj" },
        { "transformation"   , glm::scale(glm::mat4(1.0f), glm::vec3(1.0))  },
        //{ TEX_FLIP, true }, // if true, causes textures to be flipped in y
        { "albedo" , "../resources/objects/wood/wood_diffuseOriginal.png" },
        { "normal"  , "../resources/objects/wood/wood_normal.png"},
        { "metallness"  , ""},
        { "roughness"  , "../resources/objects/wood/wood_height.png"},
        { "ao"  , "../resources/objects/wood/wood_ao.png"}
        }
        } });
```

In the main() loop of the fragment shader, we use the delegated textures coming from the main application, as all the materials parameters from the textures are already defined in the fragment shader. The following code snippet shows the usage of textures coming from the main application which will demonstrate different textures in the models.

```
void main()
{
    vec3 N = normalize(Normal);
    vec3 V = normalize(camPos - WorldPos);
    vec3 albedo = Albedo;
    float metallic = Metallic;
    float roughness = Roughness;
    float ao = AO;

    if (useTextures > 0)
    {
        albedo = pow(texture(albedoMap, TexCoords).rgb, vec3(2.2));
        metallic = texture(metallicMap, TexCoords).r;
        roughness = texture(roughnessMap, TexCoords).r;
        ao = texture(aoMap, TexCoords).r;

        N = getNormalFromMap();
    }
```

1.4 Summary and Results

Figure. 1.3 shows the sphere model after applying the plastic material and Figure. 1.4 shows the cube after applying the wood material respectively in the PBR pipeline.

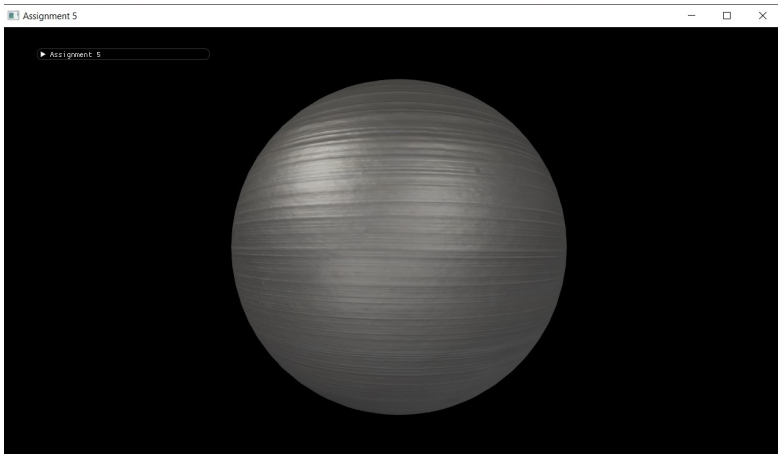

Figure 1.3: Plastic material applied to a sphere object

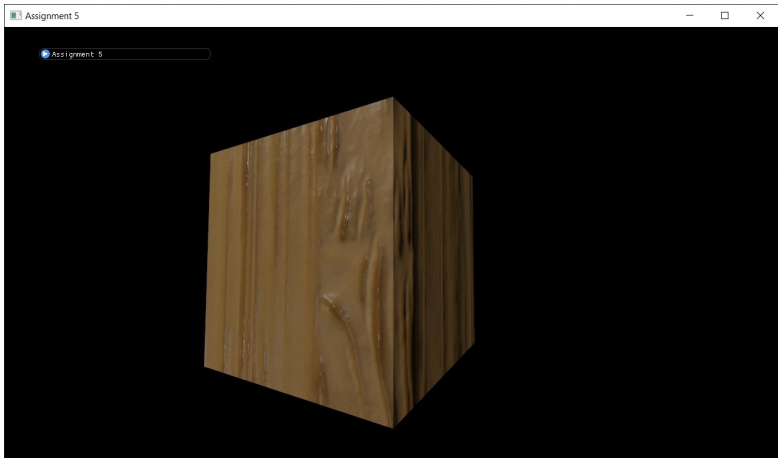

Figure 1.4: Plastic material applied to a sphere object

This assignment was based on the artistic approach of authoring PBR materials and applying the PBR materials in the PBR pipeline. Using the *Materialize* application helped us to set and tweak the physically based input values to generate the different textures to be used in PBR pipeline based on the physical surface properties of real-world materials. The major advantage of using PBR render pipeline is, the physical properties of a surface remains same even if the environment lighting is changed, which will helps the developers to get the more realistic results. The PBR render pipeline has be used in game engines (*Unreal Engine 4*) to recreate PBR materials to make the objects looks more realistic.