Exercise 7

Computer Graphics - COMP.CE.430

Physically Based Shading

This week we will implement a physically based shading model based on microfacet theory. We will be using the specular BRDF term mentioned in the lectures:

$$f_{spec}(\boldsymbol{l}, \boldsymbol{v}) = \frac{F(\boldsymbol{h}, \boldsymbol{l}) G(\boldsymbol{l}, \boldsymbol{v}, \boldsymbol{h}) D(\boldsymbol{h})}{4|\boldsymbol{n} \cdot \boldsymbol{l}||\boldsymbol{n} \cdot \boldsymbol{v}|}$$

where *F* is the Fresnel term, *G* is the joint shadowing-masking function and *D* is the normal distribution function. We will use Schlick's approximation for the Fresnel term *F*, the Smith shadowing and masking function for *G,* and the Trowbridge-Reitz/GGX normal distribution function for *D*.
There's quite a lot of math involved, so you may want to consider starting from a new project or a new shader pass so you can have a clean slate.

Problem set

1. Implement a physically based shading model using a Lambertian diffuse term for the transmitted/scattered light and the specular BRDF term for reflected light. **(2p)**

   For the specular BRDF, we will have to evaluate the equation

   $$f_{spec}(\boldsymbol{l}, \boldsymbol{v}) = \frac{F(\boldsymbol{h}, \boldsymbol{l}) G(\boldsymbol{l}, \boldsymbol{v}, \boldsymbol{h}) D(\boldsymbol{h})}{4|\boldsymbol{n} \cdot \boldsymbol{l}||\boldsymbol{n} \cdot \boldsymbol{v}|},$$

   where ***l*** is the light direction vector, ***v*** is the view vector, ***h*** is the half vector

   $$\boldsymbol{h} = \frac{\boldsymbol{l} + \boldsymbol{v}}{|\boldsymbol{l} + \boldsymbol{v}|}$$

   and ***n*** is the normal vector. There are multiple variants for each of the functions *F, G* and *D*, but we will use the most common ones: Schlick's approximation of the Fresnel term for *F*, Smith's joint shadowing and masking function for *G,* and the Trowbridge-Reitz/GGX normal distribution function for *D*.

   Schlick's approximation is $F(\boldsymbol{h}, \boldsymbol{l}) \approx F_0 + (1 - F_0)(1 - (\boldsymbol{h} \cdot \boldsymbol{l})^+)^5$, where F_0 is a characteristic specular color for the material, and the plus sign in the exponent means that the result of the dot product is clamped to values between 0 and 1. For dielectrics (non-metals), a hard-coded value of $F_0 = 0.04$ is commonly used. For metals, the color of the metal is typically used as the value for F_0.

   Due to the popularity of the Trowbridge-Reitz/GGX distribution and the Smith masking-shadowing function, there has been a focused effort to optimize the combination of the two. We can therefore use some simplifications for the rest of the

specular BRDF. We will skip the derivations here and use the following simplifications:

$$\frac{G(\boldsymbol{l},\boldsymbol{v},\boldsymbol{h})}{4|\boldsymbol{n}\cdot\boldsymbol{l}||\boldsymbol{n}\cdot\boldsymbol{v}|} = \frac{0.5}{\mu_o\sqrt{\alpha^2+\mu_i(\mu_i-\alpha^2\mu_i)}+\mu_i\sqrt{\alpha^2+\mu_o(\mu_o-\alpha^2\mu_o)}},$$

where α is equal to $roughness^2$. The equation uses the variable replacement $\mu_i = (\boldsymbol{n}\cdot\boldsymbol{l})^+$ and $\mu_o = (\boldsymbol{n}\cdot\boldsymbol{v})^+$ for brevity. Roughness is a value between 0 and 1 that describes the bumpiness of the surface on a microgeometry level.

The GGX distribution is $D(\boldsymbol{h}) = \frac{\alpha^2}{\pi(1+(\boldsymbol{n}\cdot\boldsymbol{h})^2(\alpha^2-1))^2}.$

Multiply these together to compute the specular BRDF term.

For the diffuse term, we will model diffuse scattering with a Lambertian term. In this case the BRDF diffuse term is $f_{diff}(\boldsymbol{l},\boldsymbol{v}) = \frac{\rho}{\pi}$ where ρ is the *albedo*. The albedo is just an RGB value, similar to the diffuse color we used in Phong shading. The division by π is caused by the fact that integrating a cosine factor over the hemisphere yields a value of π.

The Lambertian model does not account for the fact that light reflected at the surface is not available for scattering. A basic way to account for this is to use the Fresnel term to weigh the diffuse term: $f_{diff}(\boldsymbol{l},\boldsymbol{v}) = (1-F(\boldsymbol{h},\boldsymbol{l}))\frac{\rho}{\pi}$.

Both the diffuse and specular terms should be weighed by the cosine term $\cos\theta = (\boldsymbol{n}\cdot\boldsymbol{l})^+$ (Lambert's cosine law).

Remember that the diffuse term only applies to dielectric materials (non-metals).

2. Read the subsurface albedo value from a texture map and use that instead of a constant albedo value in your shading equations. **(1p)**

   You can go to https://ambientcg.com/ to download tons of high-quality texture maps for free. The downloadable files also come with other textures such as metallic, roughness, and normal maps, but for this exercise, we will only need the base color map.

   Download a texture of your choice and load it in SHADERed. You can do this by right-clicking in the *Objects* tab and selecting *Create Texture*. After creating a texture, you can bind it to a Shader Pass by right-clicking it and selecting *Bind*. You need to declare a uniform in the shader to read the texture, such as

```
uniform sampler2D albedo_texture;
```

   Once you have bound the texture and declared the uniform, you can sample the texture by using the texture function:

```
vec3 albedo = texture(albedo_texture, uv).rgb;
```

The variable *uv* is a *vec2* containing the texture coordinates. There's one caveat here; albedo / diffuse textures are usually authored in the sRGB color space. We need values to be in *linear* space when we compute our shading equations. You'll need to inverse the sRGB transformation to get the texture back in linear space. Using a gamma value of 2.2, we can approximate it with

```
albedo = pow(albedo, vec3(2.2));
```

If you don't do this, your texture will look way too bright and washed out.

Bonus

1. Read the roughness and metalness values from a texture and use those in your shading equations **(1p)**

   The textures available on https://ambientcg.com/ have metallic and roughness maps as well, so just repeat the previous step for those textures as well. Note that these maps are NOT meant to be in sRGB colorspace. If your material isn't metallic, it probably won't have a metalness map, so you can just use a float constant instead.

2. Use normal maps instead of geometric normals in your shading equations **(1p)**

   Normal maps are defined in tangent space, so you'll need to create a tangent space basis and either transform the normal vectors to world space or the light and view vectors to tangent space. Either approach is valid. We can create the transformation matrix from the *tangent*, *bitangent* and *normal* vectors. This matrix is often called a *TBN* matrix. We can use the cross product to compute the bitangent vector from the normal and tangent vectors. We already have the normal vector. If your model has tangent vectors (all the SHADERed built-in geometry do), you can add the tangent vector to the *Input layout* for the shader pass and read that in the vertex shader. The tangent input variable is a *vec3* in SHADERed.

   You can compute the bitangent using the cross product:

   ```
   vec3 bitangent = normalize(cross(normal, tangent));
   ```

   You can create a TBN matrix directly from these vectors:

   ```
   mat3 tbn = mat3(tangent, bitangent, normal);
   ```

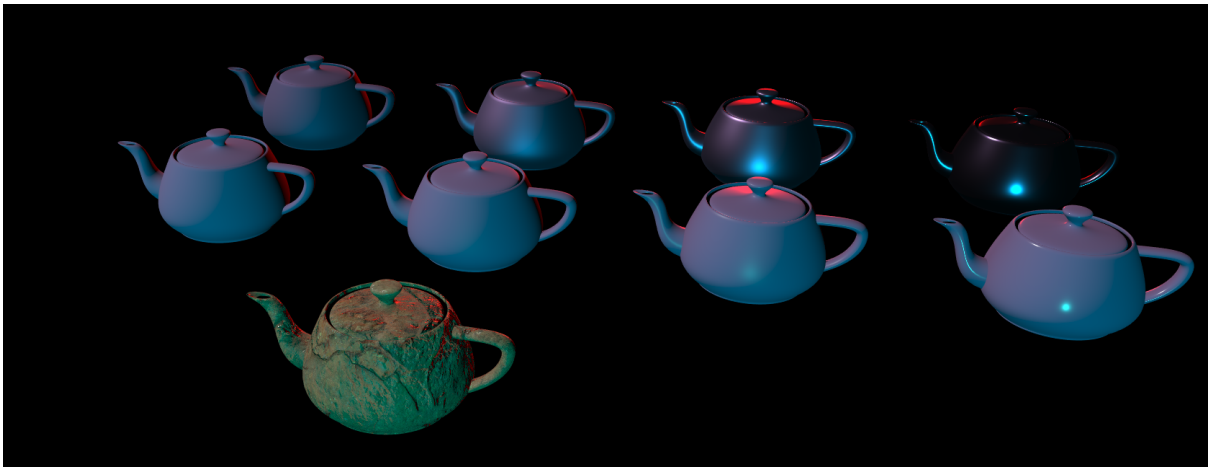
   This matrix will transform points and vectors from tangent space to world space. If you want to transform from world space to tangent space, you need the inverse of this matrix. The inverse in this case is the same as the transpose, since the TBN matrix is a pure rotation matrix.

   Values in the normal map are between 0 and 1, but normal vectors are between -1 and 1, so you need to rescale the value by multiplying it by 2 and subtracting one before using it in shading equations.

```
vec3 normal = mapped_normal * 2.0 - 1.0;
```

Note: You may find two different kinds of normal maps; ones labeled “DX” and ones labeled “GL”. The difference is that DirectX generally uses a left-handed coordinate convention, so the bitangent vector points in the opposite direction (the result of the cross product defines the bitangent direction, and in a left-handed coordinate system the result of the cross product is flipped compared to a right-handed coordinate system). The only difference between these normal maps is that the green channel is flipped. Normal maps are usually also NOT in sRGB colorspace.

Example scene

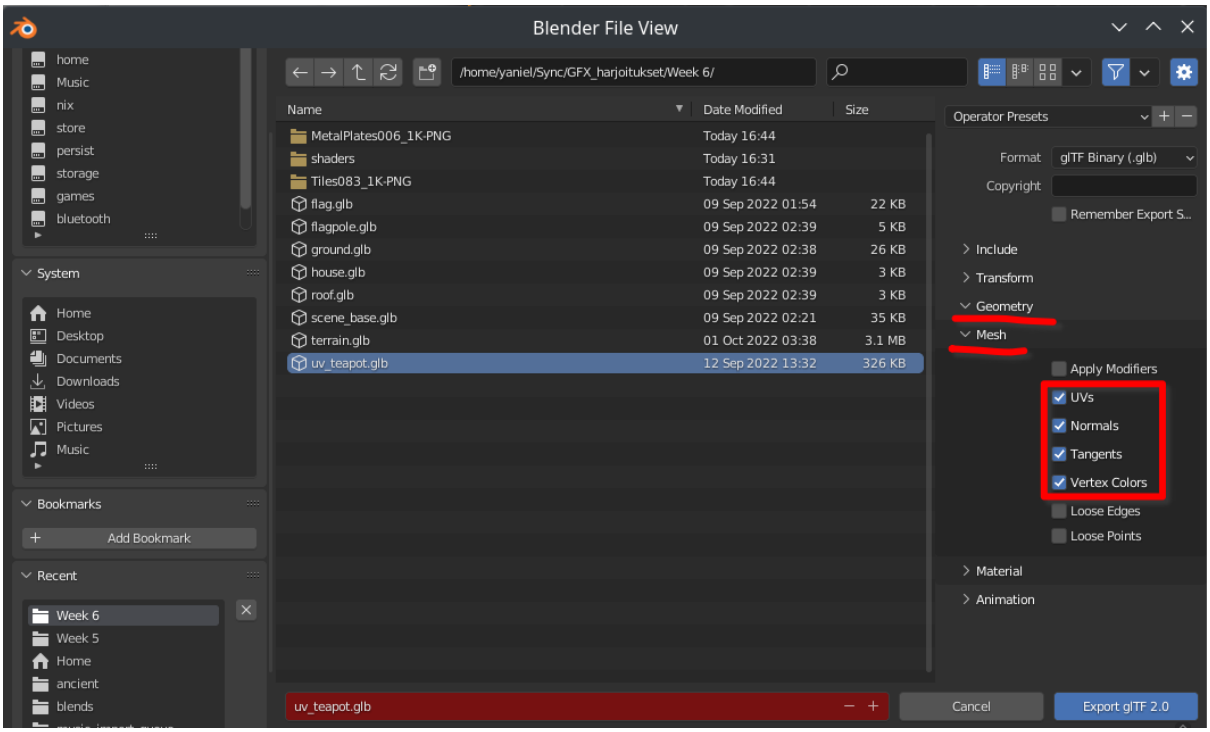


The scene shows metallic and dielectric teapots with roughness increasing from right to left. The back-row teapots are metallic.

Appendix A. Preparing your own 3D models for normal mapping

Tangents don’t by default get exported with your 3D models from 3D art tools such as Blender. It is possible to compute them based on texture coordinates and vertex positions, but this is surprisingly complicated to get right. It is much easier to export the tangents with the 3D mesh. In Blender, you can export the tangents by selecting the option in the *Export*

menu:



Blender exports tangents as *vec4*, where the *w* component denotes the *handedness*. The handedness is a value between -1 and 1 that denotes the direction of the bitangent vector. The bitangent should be multiplied by the handedness value after computing the cross product to get the correct direction. SHADERed however only exposes the tangent variable as a vec3, so you don't need to worry about this in this exercise.