a) The diffuse BRDF is view independent and therefore only returns a simple constant:

$$f_r(\omega_o, x, \omega_i) = \frac{k_d}{\pi}$$

We divide by π here to allow setting the diffuse coefficient k_d (also known as `albedo`) to values between 0 and 1, yet remain energy-conserving.

Implement the diffuse BRDF:

```
Color Material::Diffuse::eval(float cosThetaI) const
```

`return` $f_r(\omega_o, x, \omega_i)$ on the upper hemisphere and 0 on the lower hemisphere.

b) Both conductor and dielectric materials need to reflect the incoming ray at the surface to compute the direction of the reflected ray. Please derive and implement the mirror direction. We are in a tangent coordinate frame, therefore the normal is always $(0, 0, 1)^T$.

```
Vector3D Material::reflect(Vector3D v)
```

c) In the first task you already researched how to refract rays at the interface of media with different indices of refraction. For dielectric materials such as glass, we need to compute the direction of the refracted rays given the indices of refraction for the two media. Please note: If the ray is coming from underneath the surface, this means that η_1 and η_2 must be swapped, as we are now leaving the material with η_2. We then can pretend that the ray actually came from above by flipping the sign of the Z-component of the direction.

With that in mind, implement

```
Vector3D Material::Dielectric::refract(Vector3D omegaO) const
```

Note that there are some input configurations with total internal reflection. In that case, simply `return` $(0, 0, 0)^T$, since there is no refraction direction.

d) For refractions, we also need to know the Fresnel term. For dielectrics, it boils down to the following equations:

$$R_\perp = \left(\frac{\eta_o \cdot \cos\theta_o - \eta_t \cdot \cos\theta_t}{\eta_o \cdot \cos\theta_o + \eta_t \cdot \cos\theta_t} \right)^2$$
$$R_\parallel = \left(\frac{\eta_t \cdot \cos\theta_o - \eta_o \cdot \cos\theta_t}{\eta_t \cdot \cos\theta_o + \eta_o \cdot \cos\theta_t} \right)^2$$
$$R = (R_\perp + R_\parallel)/2$$

Here, η_o (`etaO`) is the index of refraction of the outgoing ray and η_t (`etaT`) is the index of refraction of the transmitted (refracted) ray. As before, one has to be careful to consider rays from above and below the surface. The given code already picks the correct indices of refraction for you.

Implement the Fresnel term for dielectrics given $\cos\theta_o$ and $\cos\theta_t$ in

```
float Material::Dielectric::fresnel(float cosThetaO, float cosThetaT) const
```

e) For meshes with smooth surfaces, modeling their curved appearance through explicit geometry alone is infeasible. Instead, we can interpolate between surface normals defined for each vertex.

Please compute the interpolated surface normal given the barycentric coordinates $\lambda_1, \lambda_2, \lambda_3$ in

```
Normal3D ShadingIntersection::computeShadingNormal() const
```

You don't have to normalize here, since we're transforming the normal from the mesh's coordinate system to the world space afterwards and normalize then.

The given implementation returns the geometric normal, so you can already see something – just remove it. For testing, we recommend using the `bunny.obj` mesh and setting the ray tracer to the "Normal" mode.

f) Finally, our scene needs some physically plausible light. For illumination by point lights, the `class PointLight` is already implemented. You can query its incident radiance $L_i(x)$ (assuming no shadow). To check if there is a shadow, simply check the scene for an intersection with its `shadowRay`. When there is no intersection, add up the product of incident radiance $L_i(x, \omega_i)$, BRDF $f_r(\omega_i, x, \omega_o)$ and cosine $\cos\theta_i$. Return the sum over all lights in the scene in the function

```
Color RayTracer::computeDirectLight(const Scene& scene, const
   ShadingIntersection& its)
```

Use the `shadingFrame` to convert your ray direction to the local space for evaluating the BRDF and cosine terms. You can assume the material to be diffuse here.

We will learn how to deal with non-diffuse materials later.