Implementation of the hair scattering model

Hair shading extra point

Lorenzi Flavio, mat. 1662963

Computer Graphics

Introduction

Nowdays, there are several methods to model hair and fur in the world of computer graphics, but it remains a very challenging task: in particular we are witnessing a complex geometry which light is scattered making everything very realistic, but at the same time very difficult to realize.
In this work we see how it is possible to generate efficient hairshading by following and studying the work published by Matt Pharr: *"The implementation of a hair scattering model"*, integrating it to our **YoctoGL** library and trying to replicate some results, also providing new original ones.

Geometry hints

Before moving on to the practical implementation, it is important to provide the main theoretical concepts and elements related to the geometry used in the mentioned paper.
First of all we know that given a direction ω at a point on a curve, the angle θ is defined by the angle between ω and the normal plane at the point (thick line). The curve's tangent vector at the point is aligned with the x axis in the BSDF coordinate system. For a direction ω, the angle ϕ is found by projecting the direction into the normal plane and computing its angle with the y axis, which corresponds to the curve's ∂p/∂v in the BSDF coordinate system (Fig. 1).

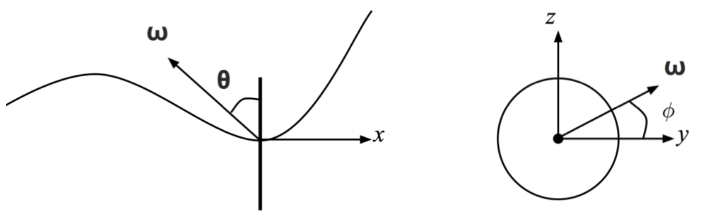


Figure 1

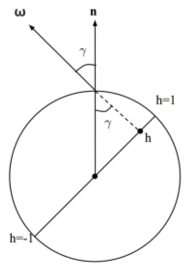


Figure 2

Then given an incident direction ω of a ray that has intersected a projected curve on the normal plane, we can parameterize the length of the curve with h ∈ [-1,1]. Furthermore, given the h for a radius that has intersected the curve, it is possible to calculate the angle between γ and ω, and the normal to the surface of the curve at the point of intersection; with h = 1 we will have that the two angles are identical, sin γ = sin ω, given the unit radius (Fig.2).

Now for the next section (implementation) we need to define some important geometric parameter, implemented into the code as following:

```
static const int pMax = 3;              // the segments of scattered light: checking for evaluation
const float h = 0.0f;                   // [-1,1] offset along the curve width
static const float eta  = 1.75f;        // refraction index for hair interior
vec3f sigma = {2.9, 5.1, 10.2};         // absorption coefficient (that can change the color)
static const float beta_m = 0.3f;       // longitudinal hair roughness
static const float beta_n = 0.3f;       // azimuthal hair roughness
float sin2kAlpha[3], cos2kAlpha[3];     // (two) angles where scales are offset
```

NB: using a larger pMax speeds up the rendering, but some details are lost (so I put the default val).
NB: A difference with the original implementation is that it uses a spectral rendering with Spectrum type while our is RGB (only over 3 channels): so we just need a vec3f type that replace it.

Implementation: scattering from hair

Hair and fur have three main components to consider: the *Cuticle* that is the outer layer (surface) carachterized by a nested series of scales at differents small angles α (Fig. 3); the *Cortex* (next layer) that generally account 90% of the hair volume; the *Medulla* that is the center core at the middle of the object. In the proposed model the following assumptions are made: the Cuticle can be modeled as a rough dielectric cylinder with the α angles; so the inside of the hair always absorbs the light (scattering inside is not directly modelled) and it is also assumed that the scattering can be modeled accurately by the BSDF (same point for incoming and outgoing of the light).

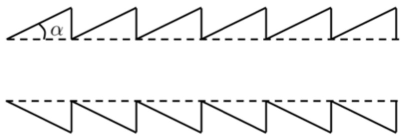


Figure 3

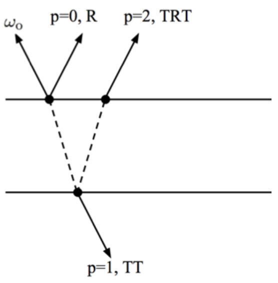


Figure 4

So considering the geometry there are a few quantities related to the directions ω_o and ω_i that are needed for evaluating the hair scattering model. Specifically, the sine and cosine of the angle θ that each direction makes with the plane perpendicular to the curve, and the angle ϕ in the azimuthal coordinate system. Incident light arriving at a hair may be scattered one more time before leaving the hair. They are used to denote the number of path segments it follows inside the hair before being scattered back out to air. For instance p = 0 corresponds to R (reflection), p = 1 is TT, for two transmissions p = 2 is TRT, p = 3 is TRRT and so on.

The hair BSDF is written as a sum over the terms p as following: $f(\omega o, \omega i) = (p=0,\infty)\sum fp(\omega o, \omega i)$.

To make the scattering model implementation and sampling easier, many hair scattering models factor f into terms where one depends only on the angles ϑ and another on φ, the difference between φ_o and φ_i. This semi-separable model is given by:

$$fp(\omega_o, \omega_i) = Mp(\vartheta_o, \vartheta_i)Ap(\omega_o)Np(\varphi)/|cos\vartheta|$$

where **Mp** is the Longitudinal scattering function, **Ap** is the attenuation function coefficent, **Np** is Azimuthal scattering function. Note that the whole project is characterized by these different ways to consider the scattering implementation and obtain different hair looks (in the section Results I put every hair/fur look I achieved).

Therefore we will need to define the following parameters at the beginning of the code, which are important for the two types of scattering used:

```
float gammaO = asin(h);                          // angle between surface normal and direction
float v[pMax + 1];                               // roughness variation trick (for Longitudinal Scattering)
static const float SqrtPiOver8 = 0.626657069f;   // constant used for mapping (for Azimuthal Scattering)
float s = SqrtPiOver8 * (0.265f * beta_n + 1.194f * beta_n * beta_n +
          5.372f * pow(beta_n, 20));             // logistic scale factor from beta (for Azimuthal Scattering))
```

The angles *ϑo* and *ϑi* in the formula can be instead easy computed with the atan operations, done with the function `static vec3f f(const vec3f& wo, const vec3f& wi)` that is characterized by several steps:

1) Compute hair coordinate system terms related to *ϑo* and *ϑi;*
2) Compute the other angles (sinTheta, cosTheta, Gamma) for the refracted ray;
3) Compute the transmittance T of a single path through the cylinder and evaluate the hair BSDF;
4) Compute contribution of remaining terms after pMax.

Now let's see in detail the main functions implemented in the code, enriching the *"volumetric_path rendering"* already implemented in YoctoGL, in particular by modifying the scene evaluation part with many methods taken from the original work, the shaders and the methods of *"sample_brdfcos"* and *"sample_brdfcos_pdf"*. In particular I tried to readjust these last two methods according to the Hair BRDF implementation and according to material I found online about it.

Longitudinal scattering function

The model implemented in the paper was developed by *d'Eon et al. (2011)*, with a very complex geometry and with the specific goal to derive a scattering function that is normalized and sampled directly; although it turned out that this model isn't numerically stable for low roughness variance *v*, which is parametric controlled in that case.
So Mp is the function that defines the component of scattering related to the angles θ. Longitudinal scattering is responsible for the specular lobe along the length of hair and the longitudinal roughness β_m controls the size of this highlight (for example "shadows and definition of details").

```
static float Mp(float cosThetaI, float cosThetaO, float sinThetaI,float sinThetaO, float v) {
  float a  = cosThetaI * cosThetaO / v;
  float b  = sinThetaI * sinThetaO / v;
  float mp = (v <= 0.1f)
            ? (exp(LogI0(a) - b - 1 / v + 0.6931f + log(1 / (2 * v))))
            : (exp(-b) * I0(a)) / (sinh(1 / v) * 2 * v);
  return mp;
}
```

The *v <= 0.1* test in the implementation selects between the two possible formulations.
Note that I0() and LogI0() functions are very important here, because they compute the values of the modified Bessel function of the first kind its logarithm, respectively.

Different roughness values *v* are used for different values of *p*. For *p* = 1, roughness is reduced by an empirical factor that models the focusing of light due to refraction through the circular boundary of the hair. In sample_brdfcos method I put the Mp sampling to compute θ (longitudinal variance), as shown in the original paper:

```
v[0] = sqrt(0.726f * beta_m + 0.812f * beta_m * beta_m + 3.7f * pow(beta_m, 20))
v[1]        = .25 * v[0];
v[2]        = 4 * v[0];
v[3]   = v[2]              //for p = 3 and v[p]=v[2]
```

Absorption in fibres: attenuation function

The Ap term describes how much of the incident light is affected by each of the scattering modes p. It incorporates two effects: *Fresnel reflection and transmission* at the hair–air boundary and the *absorption of light* passing through the hair (for p> 0).

In short, it is possible to model "how much" light the surface absorbs and consequently the color.

Its implementation is characterized by several parts in the code; for instance the function f already described above used to compute distances between longitudinal and azimuthal projections.

Now the main function to consider is surely Ap (), that returns an array with the values of Ap up to pMax and a final value that accounts for the sums of attenuations for all of the higher-order scattering terms:

```
static std::array<vec3f, pMax + 1> Ap(float cosThetaO, float eta, float h, const vec3f& T) {
        std::array<vec3f, pMax + 1> ap;
        // Compute p=0 attenuation at initial cylinder intersection
        float cosGammaO = sqrt(1 - h * h);
        float cosTheta  = cosThetaO * cosGammaO;
        float f = FrDielectric(cosTheta, 1.f, eta);
        ap[0] = {f, f, f};
        // Compute p=1 attenuation term
        ap[1] = (1 - f) * (1 - f) * T;
        // Compute attenuation terms up to p = pMax
        for (int p = 2; p < pMax; ++p) ap[p] = ap[p - 1] * T * f;
        // Compute attenuation term accounting for remaining orders of scattering
        ap[pMax] = ap[pMax - 1] * f * T / (vec3f{1.f, 1.f, 1.f} - T * f);
        return ap;
}
```

Azimuthal scattering function

In this section we will model the component of scattering dependent on the angle ϕ; so first we need to know how an incident ray is deflected by specular reflection and trasmission in the normal plane. This is done by the function:

```
inline float Phi(int p, float gammaO, float gammaT) {
  return 2 * p * gammaT - 2 * gammaO + p * pi;
}
```

Then we can use a logistic function `inline float Logistic(float x, float s)` to represent surface roughness, so that a range of directions centered around the specular direction can contribute to scattering.

In particular it was useful to consider a particular Trimmed Logistic function, normalized and defined usually on the interval [−π, π], but for flexibility it takes values over the range [a, b].
Now we have the pieces to be able to implement the azimuthal scattering distribution, done with the Np() function that computes the Np term, computing the angular difference between ϕ and Φ(p, h) and evaluating the azimuthal distribution with that angle, as following:

```
inline float Np(float phi, int p, float s, float gammaO, float gammaT) {
        float dphi = phi - Phi(p, gammaO, gammaT);
        // Remapping dphi to [-\pi,\pi]
        while (dphi > pi) dphi -= 2 * pi;
        while (dphi < -pi) dphi += 2 * pi;
        return TrimmedLogistic(dphi, s, -pi, pi);
}
```

Scattering model evaluation

We now have almost all of the pieces we need to be able to evaluate the model. This is the last step before making the whole system working together. We have to consider the tipical small angles along the surface of the fiber, of value α, in the scattering process. For the R terms, the presence of the scale angle α can be modelled by adding the value 2α to the previous one. For the term TT, p=1, the angle is rotated in the opposite direction by α, to compensate the double transmittance effect. Finally for TRT a rotation by -4α works well for the whole effect.
In particular I moved to the brdfcos_pdf() and I modified it with an adapted version of the “evaluate hair BSDF” section in the original work, as following:

```
// Compute PDF sum for hair scattering events
        float phi = phiI - phiO;
        float pdf = 0;
        for (int p = 0; p < pMax; ++p) {
                // Compute sinThetaO and cosThetaO terms accounting for scales
                 float sinThetaOp, cosThetaOp;
                 if (p == 0) {
                         sinThetaOp = sinThetaO * cos2kAlpha[1] - cosThetaO * sin2kAlpha[1];
                         cosThetaOp = cosThetaO * cos2kAlpha[1] + sinThetaO * sin2kAlpha[1];
                }
                // Handle remainder of p values for hair scale tilt
                else if (p == 1) {
                        sinThetaOp = sinThetaO * cos2kAlpha[0] + cosThetaO * sin2kAlpha[0];
                        cosThetaOp = cosThetaO * cos2kAlpha[0] - sinThetaO * sin2kAlpha[0];
                        } else if (p == 2) {
                                sinThetaOp = sinThetaO * cos2kAlpha[2] + cosThetaO * sin2kAlpha[2];
                                cosThetaOp = cosThetaO * cos2kAlpha[2] - sinThetaO * sin2kAlpha[2];
                        } else {
                                sinThetaOp = sinThetaO;
                                cosThetaOp = cosThetaO;
                        }
                        // Handle out-of-range cosThetaO from scale adjustment
                        cosThetaOp = abs(cosThetaOp);
                        pdf += Mp(cosThetaI, cosThetaOp, sinThetaI, sinThetaOp, v[p]) * apPdf[p] *
                                        Np(phi, p, s, gammaO, gammaT);
                        }
```

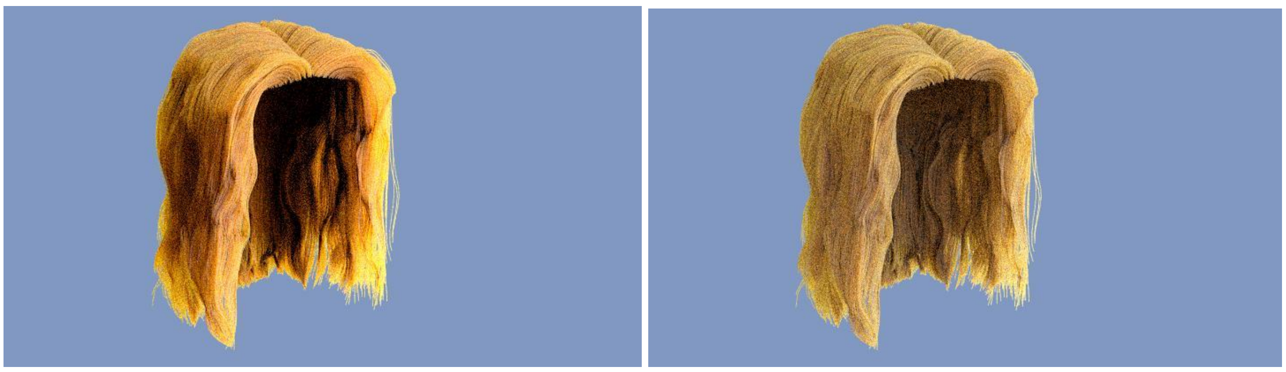
Results

For the experiments I mainly used two types of "hairy" models: 1) a modified (own version called "fur") of the YoctoGL *10_hair* scene, obtained in the first homework; 2) a simple hair model found online (I called "hair").

In this section we will also focus on three important parameters: *sigma*, *beta_n* and *beta_m*. In fact, by changing these it is possible to see how the model changes its look.
All images are rendered with the default resolution 720x720, with 2048, with max rendering time about 15-30 minutes.

By default I implemented a light brown absorption with sigma = {0.09, 0.3, 2.6}, beta_m = 0.3 and beta_n = 0.3 (as suggested in the paper).

Longitudinal scattering function



beta_m = 0.1 beta_m = 0.6

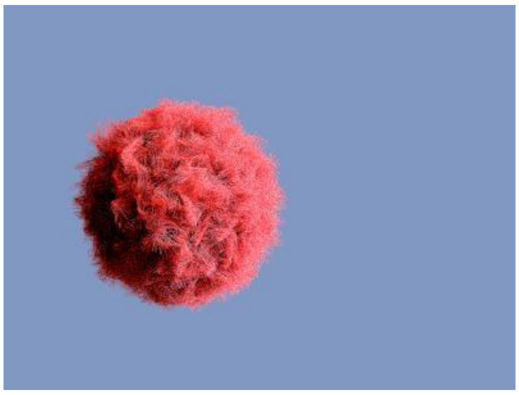
We can see in the left image a better definition of shadows and attention to details: indeed Longitudinal scattering is responsible for the highlight along the length of hair and the roughness controls the sharpness of this highlight.

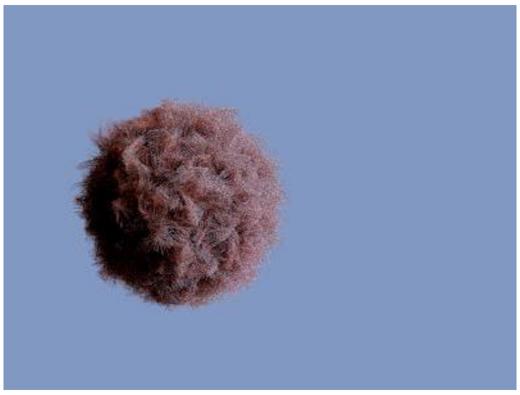
This result is in line with what was shown in the original paper, so it makes us understand that the experiment went ok and the code was well implemented.

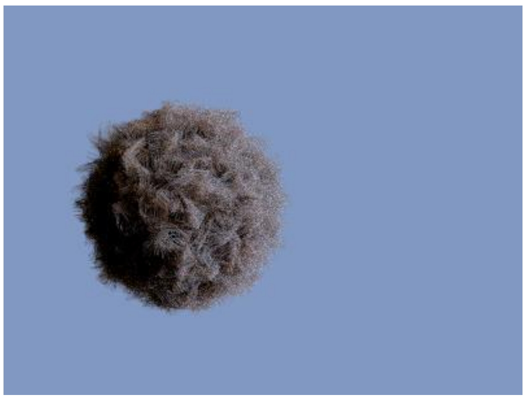
NB: This is done with beta_n (azimuthal roughness) = 0.5 to obtain a little more brightness.

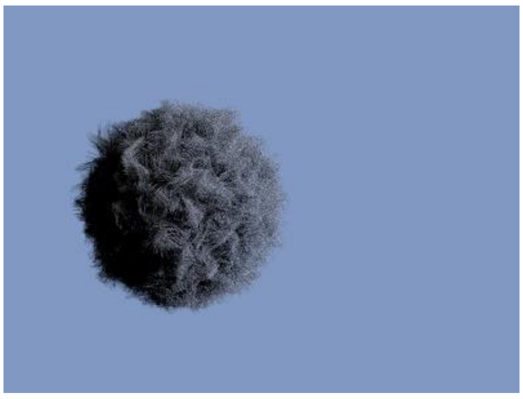
Absorption in fibres: attenuation function

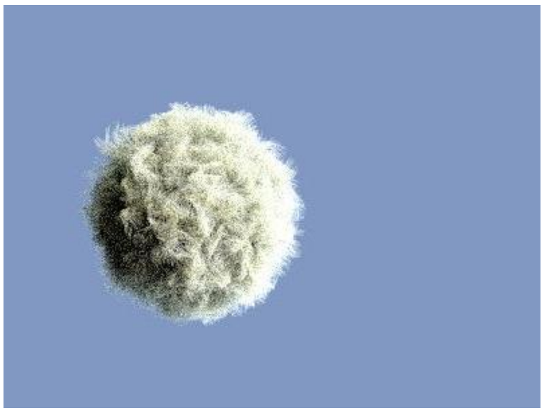
Playing with absorption coefficients sigma = {R,G,B} it is possible to obtain different pigment of the hair with very realistic look.
For this experiment I used only the “fur” model, obtaining the following results:

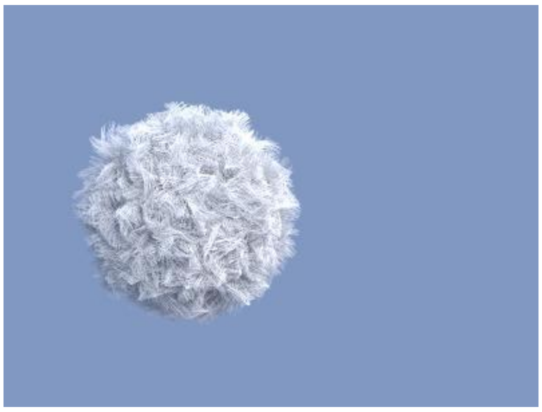

brown/red: {0.09,1.3,2.6}


light brown: {0.6,1.3,2.0}


dark brown: {0.9,1.28,2.66}


black: {2.9,5.1,10.2}

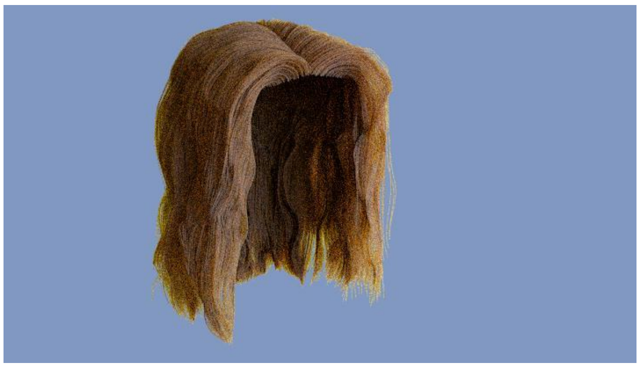

white: {0.08,0.09,0.18}

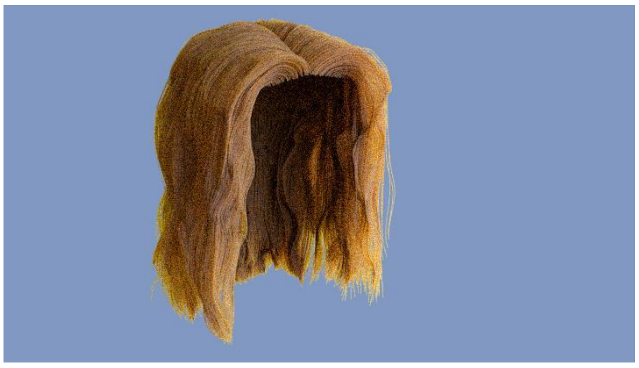

original volumetric_path shader, RGB = (0.7, 0.7, 0.7)

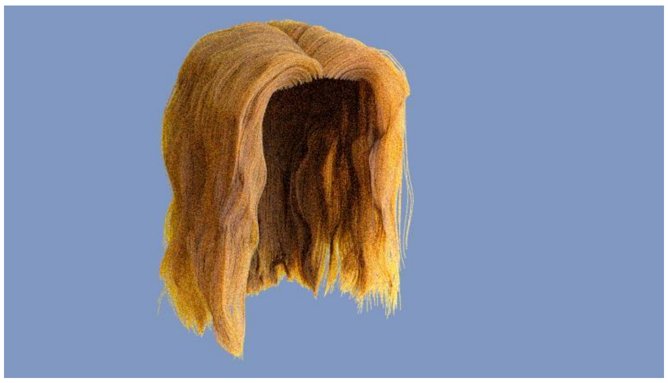
Moreover we can see from this comparison, that in this type of rendering it is a bit more realistic, when compared with the original YoctoGL (which seems more synthetic with a sheer color).
Note that with hair this difference willl be more evident.

Azimuthal scattering function

In this section we can see as the beta_n (azimuthal roughness) factor increases, the brightness of the image increases, obviously due to the azimuthal scattering function.

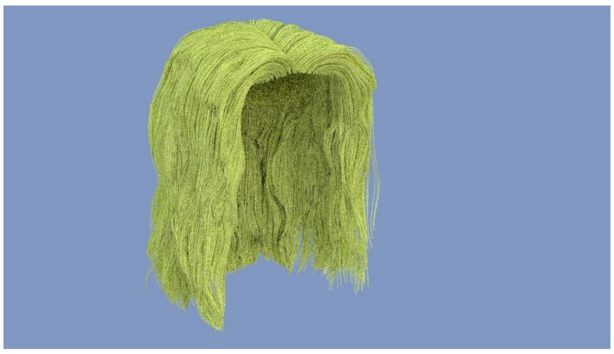

beta_n = 0.3

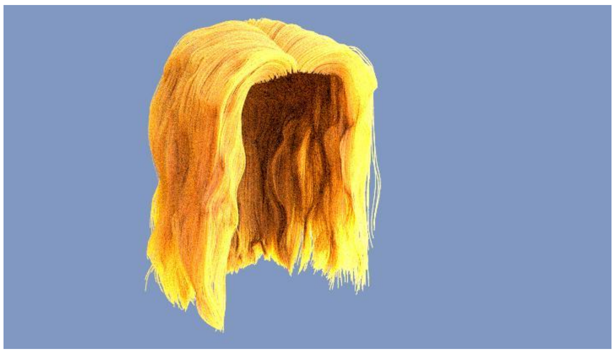

beta_n = 0.35


beta_n = 0.4

NB: in my work, the model is very sensitive to the beta_n changes, also for small variations.
So an extreme variation of beta_n even could change the color too, maximizing the shine of the model, but remaining very realistic despite everything.

Compare with original yoctoGL implementation


RGB = {1,1,0}, classic color with volpath


Light-brown sigma with (very high) beta_n = 0.9

I made this further comparation, considering the model rendered both with YoctoGL (left side) and HairShading (right side), to see how much the model can change, becoming in some ways a little more realistic with the new implementation.
NB1: the beta_n is very high in the right model, so it is very shining.

NB3: it would be very interesting here try to compare more complex scenes, to see a deeper difference, in terms of realism: so this will be added as future implementation for the project.

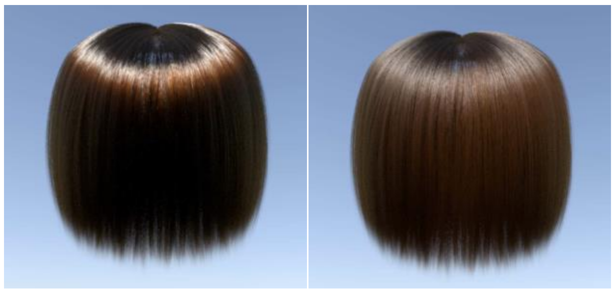
Further development and conclusion

In the future it could be very interesting try to replicate the "white furnace" test case proposed in the original paper: in this work we have seen a possible implementation of a hair scattering model wherein if hair doesn't absorb any of the light passing through it, then all of the incident light should be reflected. If such a hair is illuminated with uniform incident radiance, the reflected radiance should be exactly the same as the incident radiance. The white furnace test checks this, making sure that reflected radiance is one given unit incident radiance. The implementation proposed in the paper tests a variety of azimuthal and longitudinal roughnesses.
Following we can see a possible implementation of it, in pseudo code:

```
⟨Hair Tests⟩ ≡
  TEST(Hair, WhiteFurnace) {
      RNG rng;
      Vector3f wo = UniformSampleSphere({rng.UniformFloat(),
                                         rng.UniformFloat()});
      for (Float beta_m = .1; beta_m < 1; beta_m += .2) {
          for (Float beta_n = .1; beta_n < 1; beta_n += .2) {
              ⟨Estimate reflected uniform incident radiance from hair⟩
          }
      }
  }
```

Note that for each roughness, we should compute a Monte Carlo estimate of the spherical-directional reflectance; each sample is evaluated by first sampling a random offset along the hair h and then computing the fraction of reflected radiance for a random incident direction.

Today we have seen a particular type of Hair Shading, on which it is mainly based on: -the implementation of a single-base fiber scattering model that allows efficient Monte Carlo rendering of multiple fiber scattering traced by the path; - a reparameterization of the absorption coefficient and roughness parameters that is more intuitive and allows for an efficient workflow, while remaining physically consistent. I have obtained good results that are close to those achieved in the original paper; I also tested this implementation playing a bit with parameters, assigning new values mainly to the beta and sigma values, achieving new results too.
However as further future work and with more time available, it could be very interesting try to replicate the original models used in the paper (as the one showed below) and see differences.


variation of longitudinal roughness in the original model

References

https://www.pbrt.org/hair.pdf

https://github.com/mmp/pbrt-v3/blob/master/src/materials/hair.cpp

https://benedikt-bitterli.me/resources/

Chiang, M. J.-Y., B. Bitterli, C. Tappan, and B. Burley. 2016. A practical and controllable hair and fur model for production path tracing. Computer Graphics Forum (Proceedings of Eurographics 2016)

https://github.com/xelatihy/yocto-gl