Hair BSDF based on PBRT implementation

Eduardo Rinaldi 1797800

1 Introduction

The shading of hair in *Yocto/GL* was done by considering these as a matte material; this extra credit aims to shade hair in a more realistic way by implementing an **hair bsdf**.
The implementation is just a porting of the pbrt implementation (*pbrt v.3*) and obviously it follows their paper.

2 Code organization

Yocto/GL and *PBRT* follow two **different code styles** with **different programming paradigms** (Yocto is data oriented and PBRT follows OOP); the main difficulty was therefore to understand what the inherited classes in the PBRT code actually do, and how to write the code in *"yocto style"*.

In the following sections, the organisation of the materials and the management of the scattering and evaluation functions are explained.

2.1 Hair material

2.1.1 Data

A new member has been added to:

```
struct material_data
{
      ...
      hair_data hair; // <--
};
```

It is another `struct` that stores the following elements:

- `sigma_a`: absorption coefficient of the hair interior
- `color`: hair color
- `eumelanin` and `pheomelanin`:

  - Concentration of **eumelanin** is the factor that causes the difference between black, brown, and blonde hair. (black hair has the most eumelanin and blonde hair has the least; white hair has none).
  - **Pheomelanin**, causes hair to be orange or red.

- `method`: method for picking `sigma_a` (from stored `sigma_a`, from `color` or from `eumelanin` and `pheomelanin` concentration)
- `eta`: index of refraction inside hair
- `beta_m`: longitudinal roughness
- `beta_n`: azimuthal roughness
- `alpha`: scale angle
- `pmax`: number of "internal bounces" in hair surface

This struct can be found in `yocto_scene.h:118`

2.1.2 Data evaluated on intersection point

There are also other data used by hair bsdf that are evaluated on the intersection point (`eval_material(..)` in `yocto_pathtrace.cpp:80`):

- `rv[pmax + 1]`: roughness value for each value of `p`
- `h`: parameterize the circle's diameter, h = ± 1 corresponds to the ray grazing the edge of the circle, and h = 0 corresponds to hitting it edge-on. This value is parametrized using `uv.y` (`v` value)
- `gamma_o`: γ_o angle between surface normal and outgoing direction
- `s`: azimuthal logistic scale factor
- `sin_2k_alpha` and `cos_2k_alpha`: $sin(2^k\alpha)$ and $cos(2^k\alpha)$ are alpha terms for hair scales
- `w2bsdf`: world to bsdf frame, since pbrt do all computation in a coordinate system with:
  - x-axis aligned with the curve tangent (in our case we model hair as lines not curves)
  - z-axis aligned with the shading normal

  (More info about bsdf coordinate system in chapter 9 of pbrt)

All these data are stored in `struct hair_point` which is stored in

```
struct material_point
{
     ...
     hair_point hair; // <--
};
```

This struct can be found in `yocto_scene.h:275`.

2.2 Hair BSDF functions

For bsdf functions I decided to follow the same structure already used in yocto, i.e. using 3 functions:

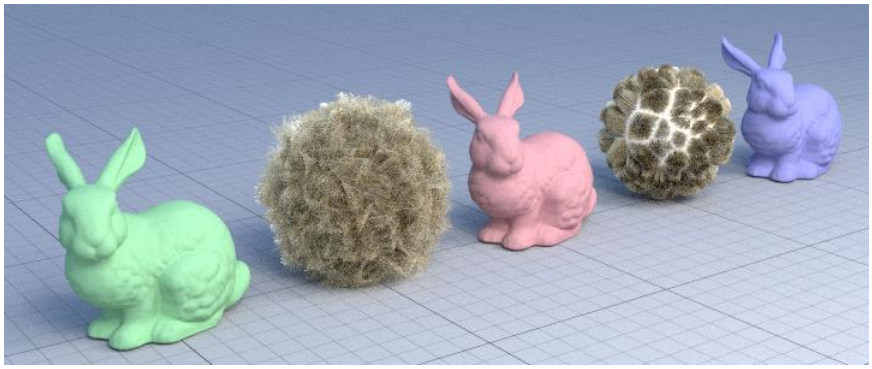
1. `eval_hairbsdf()`: for evaluating the hair bsdf, called in `eval_bsdfcos(...)`
2. `sample_hair()`: for sampling a direction ω_i (incoming), called in `sample_bsdfcos(...)`
3. `sample_hair_pdf()`: for obtaining the pdf of the sampling function, called in `sample_bsdfcos_pdf(...)`

2.3 Utility functions

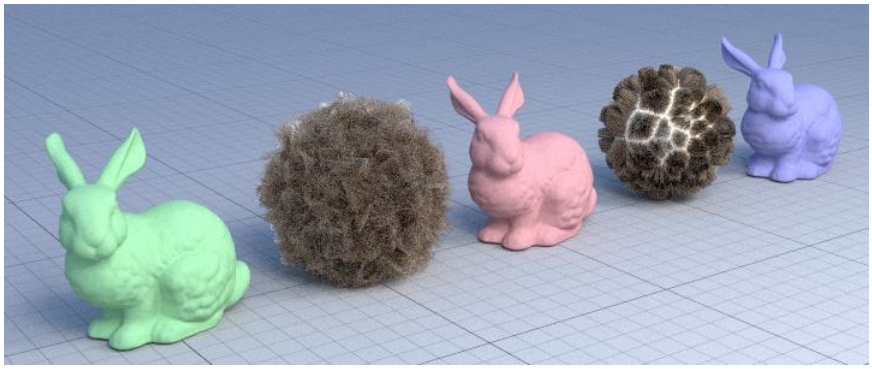
Other utility functions used for this extra credit in the previous functions can be found in `yocto_pathtrace.cpp` from **line 125**

3 Results

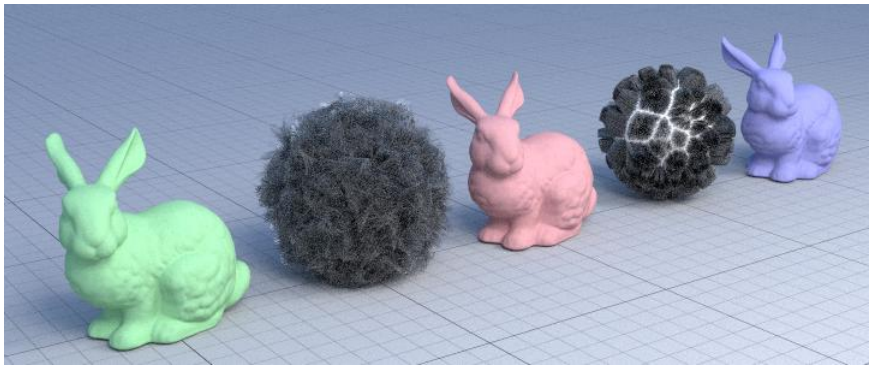
3.1 Hair balls



(a) eumelanin=0, pheomelanin=0.41



(b) eumelanin=1.2, pheomelanin=0



(c) eumelanin=8, pheomelanin=0

Figure 1: Hair balls, color from concentration

3.2 Straight hair

(a) $\beta_m = 0.1$, $\beta_n = 0.3$ (b) $\beta_m = 0.3$, $\beta_n = 0.3$ (c) $\beta_m = 0.9$, $\beta_n = 0.3$

Figure 2: Straight hair, with different β_m values

3.3 Curly hair

(a) $\beta_m = 0.3$, $\beta_n = 0.08$ (b) $\beta_m = 0.3$, $\beta_n = 0.3$ (c) $\beta_m = 0.3$, $\beta_n = 0.92$

Figure 3: Straight hair, with different β_n values

These results are stored in `"out/extra/"` directory

4 How to test the code

4.1 CLI only

I didn't implement a full list of customizable parameters and also I didn't change the json parser for parsing additional material parameters, I just only put an extra argument `--hairbsdf` that enable for hair the usage of this bsdf instead of the matte one.

4.2 Interactive

I **strongly** suggest to test this implementation in the interactive mode.

To use this bsdf for hair instead of the matte one you must enable `"use pbrt hair bsdf"` check.

At this point the renderer should use the pbrt bsdf for hair and it will compute `sigma_a` from the material color specified in `<scene>.json`.

If you want to customize hair parameters you can enable `"use gui hair material values"` check, from there you can customize:

- Hair color (based on σ_a, color or eumelanin and pheomelanin concentration)
- β_m and β_n
- p_{max}

Figure 4: Interactive menu

Results are also visible in `shade_eyelight` for a faster preview.

5 Refractive extra credit

For this extra credit the only thing to do for adding refraction was to call:

- `eval_refraction()` in `eval_bsdfcos()`
- `sample_refraction()` in `sample_bsdfcos()`
- `sample_refraction_pdf()` in `sample_bsdfcos_pdf()`

Fun fact: I implemented this extra credit because otherwise some images of the homework won't match (there are refractive materials in test scenes).

(a) Without refraction (b) With refraction

Figure 5: Coffee machine

(a) Without refraction (b) With refraction

Figure 6: Bedroom

(a) Without refraction (b) With refraction

Figure 7: Kitchen