# Hair shading

## Implementation:

* To be able to load the pbrt hair scenes I’ve made changes to yocto\_modellio and yocto\_sceneio. In particular:
  + I’ve added support for the pbrt curve shape by converting every Bezier curve into four lines (using interpolate\_bezier from yocto\_geometry to compute the vertices of the lines)
    - To significantly improve performance I merged together inside a single shape the lines that came from the same model
  + I’ve added support for the hair material properties (sigma\_a, eumelanin, pheomelanin, eta…)
* I’ve also added support for the hair material properties in the functions for loading json scenes.
* In yocto\_pathtrace I’ve added to the pathtrace\_material struct the hair material properties and I’ve defined the functions to set them that are then called in ypathtrace and yipathtraces.
* I’ve created a new library, called yocto\_hair, that implements the hair shading functions eval\_hair, sample\_hair and sample\_hair\_pdf.
* Finally inside yocto\_pathtrace I’ve added to the struct pathtrace\_brdf the parameters used by the hair shading functions that are computed from the hair material properties inside the eval\_brdf function and then passed to eval\_hair, sample\_hair and sample\_hair\_pdf inside eval\_brdfcos, sample\_brdfcos and sample\_brdfcos\_pdf respectively.

## Tests:

* I’ve tested the hair material using the pbrt scenes *Hair Curls*, *Curly Hair* and *Straight Hair* downloaded from <https://benedikt-bitterli.me/resources/>.
* I’ve also made my own test scenes:
  + *hair\_sigma\_a* to test defining the color of the hair by directly specifying the absorption coefficient *sigma\_a*.
  + *hair\_melanin* to test defining the color of the hair by specifying the amount of *eumelanin* and *pheomelanin*.
  + *hair\_color* to test defining the color directly as an RGB vector.
  + *hair\_beta\_m* to test the effect of having different longitudinal roughness *beta\_m* values.
  + *hair\_beta\_n* to test the effect of having different azimuthal roughness *beta\_n* values.
  + *hair\_alpha* to test the effect of having different scale angle *alpha* values.

All test images were rendered at a resolution of 720 and with 256 samples.

# Texture Synthesis

## Implementation:

* I’ve created a new library, called yocto\_texture, that implements randomized texture tiling following the Disney paper. It contains two main functions:
  + gaussianize\_texture: that does the preprocessing of the texture as described in the paper.
  + lookup\_randomized\_texture: that returns the color given the texture and the uv coordinates.
* I’ve modified yocto\_sceneio to add a new material property randomize to the json scene format to tell the render whether or not the color texture should be randomized.
* I’ve added the same flag randomize to the struct pathtrace\_texture inside yocto\_pathtrace together with an array called inv\_lut that will contain the inverse lut.
* I’ve also changed accordingly the function set\_color defined in yocto\_pathtrace and called by ypathtrace and yipathtraces. If the texture should be randomized it will call gaussianize\_texture that will preprocess the texture and set in the texture struct the flag randomize to true and the array inv\_lut to the calculated inverse lut.
* Finally inside yocto\_pathtrace I’ve changed the function eval\_texture to call lookup\_randomized\_texture if randomize is equal to true.

## Tests:

To test the randomized texture tiling I’ve made two test scenes, *random\_tex* and *no\_random\_tex*, that contain a floor shape with a texture applied on top. The two scenes are nearly identical with the only difference being that *random\_tex* has *randomize* set to *true* and *no\_random\_tex* to false.

All test images were rendered at a resolution of 720 and with 256 samples.