Implementing Disney principled BRDF for yocto-gl library

Indro Spinelli

Abstract

This project presents an implementation of the ”principled” brdf proposed by [1] for the yocto-gl library. This includes redesigned structs for handling materials, the i/o functions for reading ”.obj” scenes, sampling functions, evaluating functions, weighting functions and the actual BRDF.

I. Disney principled BRDF

This shading model was designed to be art-directable and not necessarily physically correct under the request of Disney's artists. They followed these five principles meanwhile building this brdf:

- Intuitive rather than physical parameters should be used.
- There should be as few parameters as possible.
- Parameters should be zero to one over their plausible range.
- Parameters should be allowed to be pushed beyond their plausible range where it makes sense.
- All combinations of parameters should be as robust and plausible as possible

II. Parameters

Only one colour and ten scalar parameters are used to control this model. The scalar parameters are in a range between 0 and 1, but also higher values can be used. Here follows a brief description of each parameter.

- baseColor - the surface colour, usually supplied by texture maps.
- subsurface - controls diffuse shape using a subsurface approximation.
- metallic - the metallic-ness (0 = dielectric, 1 = metallic). This is a linear blend between two different models. The metallic model has no diffuse component and also has a tinted incident specular, equal to the base colour.
- specular - incident specular amount. This is in lieu of an explicit index-of-refraction.
- specularTint - a concession for artistic control that tints incident specular towards the base colour. Grazing specular is still achromatic.
- roughness - surface roughness, controls both diffuse and specular response.
- anisotropic - degree of anisotropy. This controls the aspect ratio of the specular highlight. (0 = isotropic, 1 = maximally anisotropic).
- sheen - an additional grazing component, primarily intended for cloth.
- sheenTint - amount to tint sheen towards base colour.
- clearcoat - a second, special-purpose specular lobe.
- clearcoatGloss - controls clearcoat glossiness (0 = a satin appearance, 1 = a gloss appearance).

Finding the balance between these parameters has been a very difficult task at first, but after some time it is possible to have a lot of control over the material. Given my very small experience in defining materials, I got reasonable images after a couple of days. One thing that the authors stressed both in the paper and in following notes is that for the lighting (in particular for area lights) they had to develop a new approach with a large learning curve. Actually, within the yocto-gl library, the results seem reasonable without any modification.

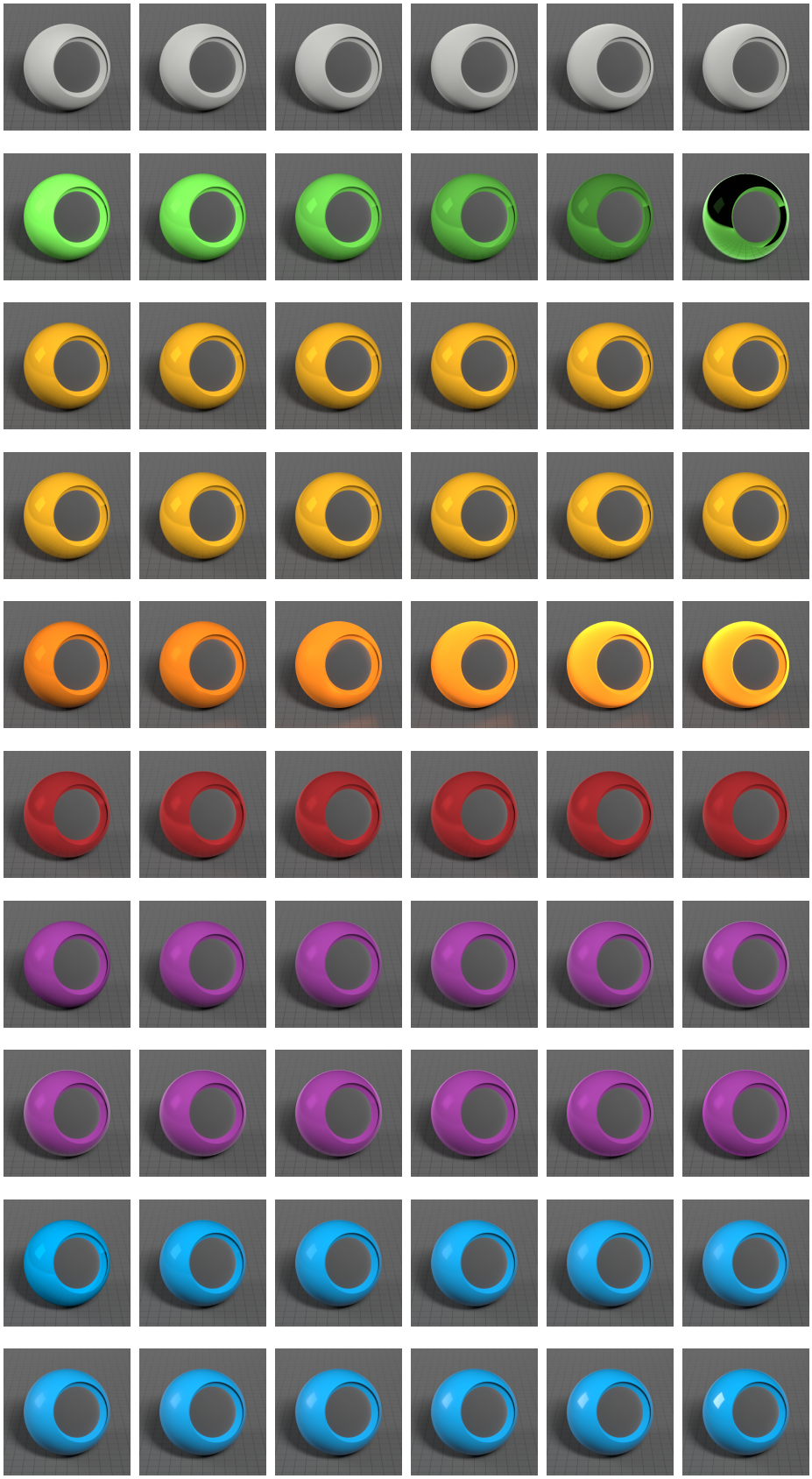

Figure 1: Examples of the effect of Disney's BRDF parameters. Each parameter is varied across the row from zero to one with the other parameters held constant.

III. Architecture

The mode is composed of two main components, the diffuse and the specular. The diffuse term is composed by:

- base diffuse
- Hanrahan-Krueger subsurface

The specular term instead:

- primary specular (anisotropic GTR2)
- clearcoat (GTR1)
- sheen

The sheen, mainly used for clothes, is relatively small and not considered for importance sampling. Thus 3 lobes are computed:

- weighted cosine hemisphere for the diffuse
- GTR1 normal distribution for the clearcoat
- GTR2 normal distribution for the primary specular lobe

For sampling the BRDF, I used the method of [6]. Thus the ratio between diffuse and speular sample is given by a russian roulette where the ratio is given by:

$$Ratio_{diffuse} = (1 - metallic)/2 \tag{1}$$

The specular samples are divided into sampling the GTR2 and GTR1 distribution with the following ratio:

$$Ratio_{clearcoat} = 1/(1 - clearcoat) \tag{2}$$

IV. Git-tex

Modified functions in yocto_gl.h:

- struct material
- struct environment
- inline void visit
- struct obj_material
- struct gl_lights
- void set_stdsurface_material
- void set_stdsurface_constmaterial

Modified functions in yocto_gl.cpp:

- material:: material
- environment::environment
- scene* obj_to_scene
- obj_scene* scene_to_obj
- struct trace_point
- vec3f eval_emission
- vec3f eval_point_brdfcos
- vec3f eval_brdfcos
- float weight_ggx_brdfcos
- float weight_point_brdfcos
- std::tuple<vec3f, bool> sample_point_brdfcos
- trace_point eval_point (one for env and one for shape points)
- float weight_light
- static auto trace_shaders
- void trace_sample
- trace_lights make_trace_lights

- std::vector<obj_material*> load_mtl
- void save_mtl
- gl_lights make_gl_lights
- void set_stdsurface_lights

New functions in yocto_gl.cpp:

- float SchlickFresnel
- float GTR1
- float GTR2
- float GTR2_aniso
- float smithG_GGX
- float smithG_GGX_aniso
- vec3f mon2lin
- vec3f sample_gtr2_aniso
- vec3f sample_gtr1
- vec3f eval_disney
- float weight_disney
- std::tuple<vec3f, bool> sample_disney

Thanks to the implementation on Disney's repository [2] and pbrt [3] I did not had to start fom scratch. The posts of [4], [5] and [6] were also full of insights.

REFERENCES

[1] burley, "Physically based shading at disney," http://blog.selfshadow.com/publications/s2012-shading-course/course$_content$.

[2] wdas, "Walt disney animation studios," http://simon-kallweit.me/rendercompo2015/report/.

[3] pharr, "pbrt-v3," https://github.com/mmp/pbrt-v3/blob/master/src/materials/disney.cppL320.

[4] shihchinw, "Implementing disney principled brdf in arnold," http://shihchinw.github.io/2015/07/implementing-disney-principled-brdf-in-arnold.html.

[5] pestana, "Implementation of the disney principled brdf," http://www.alexandre-pestana.com/disney-principled-brdf-implementation/.

[6] kallweit, "Final project report," http://simon-kallweit.me/rendercompo2015/report/.