CMSC740 Final Project - Disney BRDF Implementation

Simon Wang
University of Maryland
College Park, MD, USA
scwang00@umd.edu

Abstract

Disney has long been at the forefront of computer graphics innovation, and the scenes create through their work has inspired viewers all over the world. Inspired by this, I decided to implement Disney style shading techniques as a new brdf in the existing Aris Renderer from our course assignments. To demonstrate the results, I render several scenes that showcase the diverse capability of the brdf, and also discuss approaches to improving the usability and features of the brdf.

CCS Concepts

• **Computer Graphics → Advanced BRDFs**.

Keywords

BRDF, Disney, Shading, PyTorch, CMSC740

ACM Reference Format:
Simon Wang. 2024. CMSC740 Final Project - Disney BRDF Implementation. In *Proceedings of Advanced Computer Graphics (CMSC740).* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

In 2012, Walt Disney Animation Studios released a technical report describing the implementation of their "principled" shader BRDFs used to render famous movies like Wreck-it Ralph [1]. There are some existing implementations that can be found online that I will discuss in the following section. However, none of these sources go in-depth about how their implementations relate to the technical report published in 2012. Furthermore, all are written in shader languages intended to be applied only analytically when rendering CGI tasks, as opposed to applied to Monte Carlo path tracing based rendering as we have studied in class. So, wanting to dive deeper into understanding more about how Disney actually creates the visual effects we know and love, I decided to approach converting existing implementations and adding my own modifications into our base PyTorch renderer.

2 Related Works

The overall intuition and general implementation details are described in the 2012 technical report from Disney titled "Physically Based Shading at Disney" [1]. In this report, engineer Brent Burley introduces the motivation for Disney's work in developing a shading system well suited for artists. The report discusses how they used the MERL 100, a set of 100 BRDF visual samples, to analyze various BRDFs in a tool they developed called BRDF explorer [3].

The BRDF explorer tools allows users to load multiple analytic BRDF models written in shader language, and analyze many detailed observations of their features. Using this tool, the report then goes on to show various findings from their investigations into both diffuse and specular obseervations.

Finally, they discuss the development of their new reflectance model, which aimed to be more intuitive and easy to use rather than purely accurate. This was considered at the request of many artists who desired a shading model that was more flexible to direct in an artistic context, and not necessarily completely physically accurate. As a result, the model is called the "principled" model, hinting at the fact that features are sometimes approximated around physical properties but with some liberties taken in favor of usability.

3 Approach

3.1 The Model

The Disney model contains one color parameter and 10 scalar parameters, described as follows from the report [1]:

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

3.2 Existing Base Code

As mentioned in the introduction, I am building upon the Aris renderer I worked on completing as part of the CMSC740 Advanced

CMSC740, 2024, College Park, MD

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/XXXXXXX.XXXXXXX

Graphics course. It is an implementation of Monte Carlo path tracing, including many variations of sampling technique and brdf options to render different scenes. In particular, for this project I built upon the microfacet BRDF implementation as a base, and used the multiple importance sampling path tracer as the rendering algorithm for my scenes. I chose the microfacet BRDF after carefully reading through the Disney technical report, and seeing that the microfacet distribution detailed in the appendix was very similar to the model implemented during our course assignments.

NOTE: add the formulas here?

3.3 Official Disney HLSL BRDF Implementation

Through my research on related and existing works, I found the source code for the aforementioned Disney BRDF Explorer tool on their official GitHub: https://github.com/wdas/brdf. Looking through the source folder, I found a "disney.brdf" file which is an implementation of the Disney BRDF in shader language. So, throughout my approach to implementing the brdf in my own renderer, I referenced this code alongside the technical report to properly understand how the concepts described in text could be converted into my Tensor-based PyTorch code.

3.4 Implementation

Initially, I assumed it would be quite simple to just convert the existing shader code into Python, then just plug in the same rendering approaches used throughout the course of the semester. However, the model is actually quite a bit more complex. I faced many challenges understanding various implementation details, and eventually rewrote many sections from scratch while referencing the technical report instead. In particular, there were many choices made for details and coefficients that either were not mentioned explicitly within the text or were only included in the appendix of the report.

4 Implementation Details

In general, the implementation of the Disney BRDF system is one large blended model, with various effects, both diffuse and specular, weighted and combined together to form one singular output value. Staying mostly consistent with the existing course renderer implementation, I created a new Python class called DisneyBrdf, along with a utility function file. To make the code more readable, I broke down separate evaluation functions for diffuse, specular, subsurface, and clearcoat, then combined them all with the proper weighting approach in the eval function. I will now proceed to cover all of these parts at a high level because my implementation largely follows the official paper and code, although I will highlight any modifications I made or points of confusion I was able to overcome.

4.1 Diffuse Model

The diffuse model is based on the Schlick Fresnel approximation

$$F(\theta) = F_0 + (1 - F_0)(1 - cos\theta)^5$$

In typical diffuse models, the diffuse Fresnel factor is often given by $(1 - F(\theta_l))(1 - F(\theta_V))$ where the factor is a combination of the fresnel terms for the lighting and viewing directions. However, in the disney models, the term is instead calculated by modifying the retroreflection response based on the roughness parameter as follows

$$f_d = \frac{baseColor}{\pi}(1+(F_{D90}-1)(1-cos_{\theta_l})^5)(1+(F_{D90}-1)(1-cos_{\theta_v})^5)$$

$$F_{D90} = 0.5 + 2 * roughness * cos^2\theta_d$$

4.2 Specular Model

The specular model is made up of several components, including a D, F, and G term. Furthermore, other parameters such as sheen and anisotropic also influence the specular behavior of the model. So, this section was the most challenging for me to implement.

Specular D: Searching through the paper, the specular D section actually doesn't describe much of the actual implementation, besides the fact that they found using roughness squared yielded more linear mapping. However, in part B.2, we find the GTR2 anisotropic formula derivation, which if we look at the brdf code from BRDF explorer is what the specular D term actually uses. In short, the D term is the base specularity, and is controlled by the roughness parameter as well as various vector directions (half vector, tangent, bitangent).

Specular F: The F term is effectively a simple linear interpolation between the base specularity Fresnel term and the Schlick approximation of the angle between the incident and halfway vectors. However, in my implementation, I realized that the initial Fresnel term was a bit more complex, and after closer inspection understood that it was actually influenced by the tint color of metallic materials as well. As a result, the initial specularity was calculated by a linear interpolation between the tint color and surface color, weighted by the metallic parameter. The actual tint color also had to be calculated by converting the color to grayscale, then dividing the color by this to adjust the tint.

```
# approximating luminance
to_grayscale = torch.tensor([0.3,0.6,0.1])
luminance = to_grayscale*color

# normalizing the color
tint = torch.where(luminance > 0,
    color/luminance, torch.ones_like(color))
tint_color = torch.lerp(torch.ones_like(tint),
    tint, specularTint)
F0 = torch.lerp(specular * 0.08 * tint_color,
    color, metallic)
```

The final F term is then calculated almost identically to the Fresnel term in the diffuse model section, except using this new specular F0 value.

Specular G: The G term is based on both the Smith shadowing term as well as the GGX term derived by Walter [4]. These terms are combined from both the incident and outgoing directions, and again adjusted by the roughness parameter. It is important to not here, that in the section from the paper they only briefly mention using a gain value of $(0.5 + roughness/2)^2$, but only elaborate on the actual aspect factor calculation in the appendix. So, the full implementation involved calculating the aspect coefficient which was

factored by 0.9 to limit ratios to 10:1, as well as the x,y coefficients as follows:

```
aspect = math.sqrt(1.0 - anisotropic * 0.9)
roughness = 0.5 + roughness/2
# max against 0.001 to prevent divide by zero
ax = max(0.001, roughness**2 / aspect)
ay = max(0.001, roughness**2 * aspect)
...
NdotL = dot(n,wi)å
NdotV = dot(n,wo)
G = smith_GGX_aniso(NdotL, dot(wi,X), dot(wi,Y), ax, ay)
* smith_GGX_aniso(NdotV, dot(wo,X), dot(wo,Y), ax, ay)
```

4.3 Sheen

The sheen color component is calculated using the same tint described from the specular F, linearly interpolating between 1 and the tint weighted by the sheenTint parameter. The final sheen contribution is then the earlier Schlick Fresnel term multiplied by the sheen amount parameter and the sheen color.

4.4 Subsurface

The subsurface effect is approximated by the Hanrahan-Krueger model [2]. There isn't much detail about this on the official report, but modeling after the BRDF Explorer code as well as looking into the original Hanrahan-Krueger paper, it is my understanding that the approach here is to flatten the retroreflection depending on the roughness parameter. The overall effect is produced by several Fresnel terms, which are then combined in a linear interpolation and returned using the following formula: 1.25 * (fss * (1 / (NdotL + NdotV) - 0.5) + 0.5), where fss is the interpolated Fresnel term, and the dot products are between the surface normal and the incident and outgoing rays respectively.

4.5 Clearcoat

Finally, the clearcoat component is the second of the two specular lobes, and uses the GT model as well. However, in contrast to the base specular components, there are some analytically determined constants used for calculating these terms. For example, the F term is interpolated between the values of 0.04 and 1.0, and the GGX smith shadowing term is scaled by 0.25.

5 Experiments

To test out my newly implemented BRDF model, I tried out rendering many of the same scenes from our assignments, in addition to adding the bunny object to a scene from the provided data folder. Below are several example results using a wide range of parameter values.

5.1 Sample Results

5.2 Discussion of Results

Overall, with 10 parameters the possibilities for rendering effects are endless, but through my experimentation I was able to test a wide range of appearances. As described in the implementation details, some of the parameters are very closely related, such as the specular and metallic properties, as well as the anisotropic parameter that changes the specular highlight. I found the subsurface parameter to be very interesting, as you can see in the 4th figure. Combined with a high sheen the hard-surfaced statue was able to be rendered

Figure 1: A metallic statue

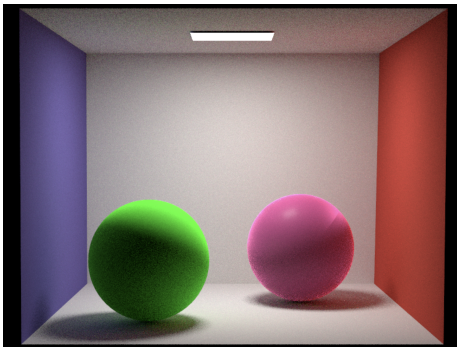

Figure 2: Pink is clearcoated with gloss, green is rough but with some subsurface scattering?

Figure 3: A very shiny and high sheen bunny...

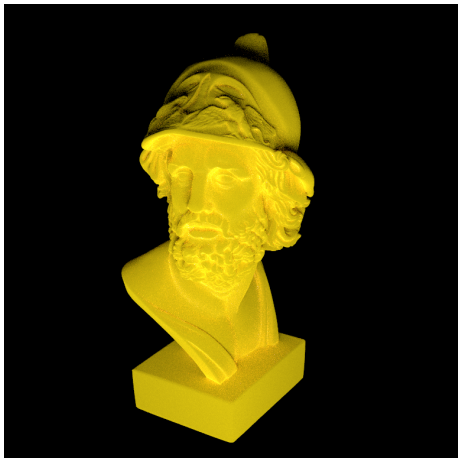

Figure 4: As much subsurface and sheen as possible!

to look somewhat soft and almost felt-like. On the other end of the spectrum, the high metallic parameter for the bunny emphasized many of its small edges from its mesh despite the darkness of the scene. Similarly, the first statue rendering looks drastically different from the yellow one, with much of its features being smooth and metallic. With the spheres I used a higher value of clearcoat, and you can see on the pink ball this results in a slightly different effect from the metallic bunny or statue, as if there is a glossy finish on the surface.

6 Prompt-Based Parameter Tuning with LLMs

As an additional experiment and feature to the renderer, I decided to try using an LLM (in this case GPT4o through my own OpenAI API account) to enable more intuitive prompt-based control of the parameters. Although the Disney report mentioned focusing on ease of use for artists, I still felt that it was challenging to tune all 10 parameters every time I wanted a different scene. While this might be important for a professional artist who needs finer control, if I just want an interesting rendering that looks a certain way as a typical user, I might not know exactly what ratio of anisotropicness or sheen to use just get "a blue shiny ball". So, using some simple Python UI and LLM prompting, I create a script that can generate yaml config format and modify the parameters to match your prompt!

NOTE: I initially planned on integrating the custom prompting feature into the render command itself to alter the configuration file live, but found that the LLM output was still somewhat inconsistent at times and produced invalid yaml code. So, to demonstrate the potential of the feature I am currently just outputting a file then manually adjusting it to make sure it is correct.

6.1 Prompting

To set up my API call, I first grab a brdf configuration file to use as reference through the command line arguments. This will be the brdf we are asking the model to modify. Then, I send the contents of the file to the model with the following prompt:

```
completion = client.chat.completions.create(
    model="gpt-4o",
    messages=[
     {"role": "system", "content": "You are an expert at
     computer graphics techniques, specifically setting
      parameters in BRDFs to create specific rendering
       effects."},
       {"role": "user",
           "content": [
           {
           "type": "text",
       "text": f"Based on the contents and format of the
           following .yaml brdf config file:
       \n{original_content}\n While absolutely adhering
       to exactly the same format of the original config
           file provided earlier, modify the
        parameter values in the config section to match
         the prompt described in the following text: "
           },
           {
           "type": "text",
           "text": text_input
           }
           ]}
    ]
)

# getting the model response
new_content = completion.choices[0].message.content
```

I then simply write the output to a yaml file, and the results were quite effective. The model seems to be able to understand how tuning various parameters could match the visual appearance described in the user's prompt.

6.2 Results

User prompt: "rough plastic surface" **Changes made:**

```
- name: disney
  config:
    color: [0.2, 0.2, 0.4]
    roughness: 0.8
    metallic: 0
    subsurface: 0
    specular: 0.5
    specularTint: 0
    anisotropic: 0
    sheen: 0
    sheenTint: 0
    clearcoat: 0.7
    clearcoatGloss: 0.8
- name: diffuse
  config:
    color: [0, 0, 0]
```

The model was able to recognize that while the prompt wanted roughness, it also wanted plastic which meant some specularity and also increased the clearcoat to give it a glossy plastic effect.

User prompt: "two copper spheres" **Changes made:**

```
- name: disney
```

Figure 5: plastic ajax

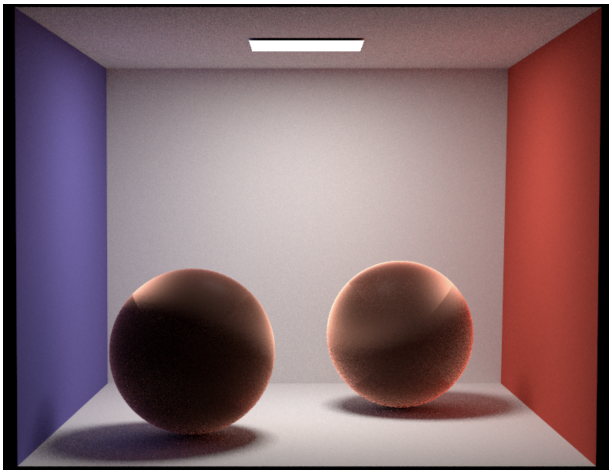

Figure 6: two copper spheres

```
config:
  color: [0.955, 0.637, 0.538]
  roughness: 0.2
  metallic: 1
  subsurface: 0
  specular: 0.5
  specularTint: 0
  anisotropic: 0
  sheen: 0
  sheenTint: 0
  clearcoat: 0.1
  clearcoatGloss: 0.5
```

```
- name: disney
  config:
    color: [0.955, 0.637, 0.538]
    roughness: 0.25
    metallic: 1
    subsurface: 0
    specular: 0.5
    specularTint: 0
    anisotropic: 0
    sheen: 0
    sheenTint: 0
    clearcoat: 0.1
    clearcoatGloss: 0.5
```

In this result even though no specific textures or effects were stated, the model recognized the approximate color of copper as well as the metallic nature of the material and adjusted the brdf effectively to look quite convincing as copper.

7 Conclusion

In conclusion, throughout the implementation of the Disney BRDF model into the course PyTorch renderer, I learned a lot about the nuances and details behind Disney's implementation. It was very fascinating to see how the engineers approached developing the model, using classic graphics algorithms as their foundation while using analytical observations to simplify certain aspects for artists. I was challenged in understanding the at-times lack of specific detail in the documentation, but was able to gain deeper understanding for myself because of that fact as well.

7.1 Notes About My Code

For the code included with this final report, I have included the entire aris codebase with my new files included. However, for this project the files I have created can simply be plugged into the existing implementation, and rendered with the commands that can be found in the appendix. The *disney.py* primary BRDF code goes in the aris/brdf/ folder, the $disney_utils$ goes in the aris/utils/folder, and the several new yaml config files are just examples used for my renderings, but the "disney" brdf is in the registry and can be added to any object! I have included these specific files outside of the aris folder as well for ease of viewing.

Acknowledgments

To Robert, for the bagels and explaining CMYK and color spaces.

References

[1] Brent Burley. 2012. *Physically Based Shading at Disney.* Technical Report. Walt Disney Animation Studios.
[2] Pat Hanrahan and Wolfgang Krueger. 1993. Reflection from Layered Surfaces due to Subsurface Scattering.
[3] Walt Disney Animation Studios. 2012. BRDF Explorer. https://github.com/wdas/brdf/blob/main/src/brdfs/disney.brdf
[4] Bruce Walter. 2007. Microfacet Models for Refraction through Rough Surfaces. *Eurographics Symposium on Rendering* (2007).

A Sample Rendering Configurations

Here are a few full configuration files used for my sample renderings. Low samples per pixel values were used to save time.

A.1 Sample Result Configs and Commands

Rendering Command:

```
python3 render.py scene=sceneName
    integrator=path_mis spp=50
```

Custom Prompt Command:

```
python3 aris/utils/custom_prompt.py
    --brdf_path=./config/scene/brdf/cbox_disney.yaml
```

"**Metallic Statue**":

```
- name: disney
  config:
    color: [0.2, 0.2, 0.4]
    roughness: 0
    metallic: 1
    subsurface: 0
    specular: 0
    specularTint: 0
    anisotropic: 0
    sheen: 0
    sheenTint: 0
    clearcoat: 0
    clearcoatGloss: 0
- name: diffuse
  config:
    color: [0, 0, 0]
```

"**Pink and Green Balls**":

```
- name: diffuse
  config:
    color: [0.725, 0.71, 0.68]
- name: diffuse
  config:
    color: [0.161, 0.133, 0.427]
- name: diffuse
  config:
    color: [0.630, 0.065, 0.05]
- name: disney
  config:
    color: [1.0, 0.4, 0.7]
    roughness: 0.2
    metallic: 0.6
    subsurface: 0
    specular: 0.5
    specularTint: 0
    anisotropic: 0
    sheen: 0
    sheenTint: 0
    clearcoat: 1
    clearcoatGloss: 0.5
- name: disney
  config:
    color: [0.2, 1.0, 0.1]
    roughness: 1
    metallic: 0
    subsurface: 1
    specular: 0.5
    specularTint: 0
    anisotropic: 0
    sheen: 0
    sheenTint: 0
    clearcoat: 0
    clearcoatGloss: 0.5
- name: diffuse
  config:
    color: [0.8, 0.8, 0.8]
```

"**Shiny Bunny**":

```
- name: disney
  config:
    color: [1.0, 0.4, 0.7]
    roughness: 0.2
    metallic: 1
    subsurface: 0
    specular: 0.7
    specularTint: 0
    anisotropic: 0.8
    sheen: 0.8
    sheenTint: 0.9
    clearcoat: 0.8
    clearcoatGloss: 0.5
- name: diffuse
  config:
    color: [0.8, 0.8, 0.8]
```

"**Subsurface Ajax**":

```
- name: disney
  config:
    color: [0.9, 0.8, 0.1]
    roughness: 1
    metallic: 0
    subsurface: 1
    specular: 0
    specularTint: 0
    anisotropic: 0
    sheen: 1
    sheenTint: 1
    clearcoat: 0
    clearcoatGloss: 0
- name: diffuse
  config:
    color: [0.8, 0.8, 0.8]
```

A.2 Additional Renderings

Some extra images I rendered.