Advanced Computer Graphics

Lab 2: Physically Based Rendering

12/11/2020

Abel Pérez
abel.perez01@estudiant.upd.edu
NIA: 205254

Sixto Pineda
sixtosarwed.pineda01@estudiant.upf.edu
NIA: 205614

I. Información general sobre el Framework e ImGui

Para poder visualizar los resultados obtenidos con el Phisically Based Rendering model, desde el Framework cargamos 3 objetos distintos: un casco, un candelabro y una esfera. Como el PBR model tiene distintas iluminaciones a tener en cuenta (luz directa, luz indirecta, luz emitida...) desde el ImGui podremos cambiar entre los distintos objetos y los distintos tipo de iluminación.
Para poder navegar entre estas distintas configuraciones, hemos definido las siguientes variables:

- **figure**: esta variable unicamente se utiliza en el ImGui del Framework para indicar qué objeto vamos a utilizar y así poder enviar los parametros adecuados al shader.
- **objectSelect**: dado que la manera en que obtenemos los mapas de la roughness es distina entre el casco y los otros dos objetos, esta variable nos permite indicarle al shader cómo cargarla.
- **numShader**: gracias a esta variable, dentro del shader a través de un if-else podemos seleccionar el tipo de iluminación que deseamos visualizar (iluminación directa, iluminacion indirecta o ambas). Además, en el caso del casco, independientemente del tipo que utilicemos también obtendremos la iluminación emitida por el propio objeto.

Las distintas modificaciones que hicimos en el framework, se encuentran en ”material.cpp”, ”material.h” y ”application.cpp”:

- **Material.h**: primero de todo, añadimos nuevas texturas a la clase material: albedo, metalness, normal, roughness y emissive, que posteriormente utilizaremos en el shader para aplicarlas sobre las figuras.
  A continuación creamos una subclase de StandardMaterial, ”PBR_material”, con un parámetro de luz, 6 texturas distintas que hacen referencia a la HDRE original y sus cinco niveles, el mapa LUT, para poder computar el término especular del IBL, y dos parámetros que hacen referencia a los comentados anteriormente: shadSelect_(numShader) y objectSelect_(objectSelect).
  Y por último el constructor, al que le pasamos todos y cada uno de los parámetros mencionados, y su respectivo ”setUniforms” para pasar los parámetros al shader.
- **Material.cpp**: en esta clase aplicamos el constructor de la clase creada PBR_material. Con los parámetros pasados, inicializamos el shader, las texturas de las figuras (albedo, metalness,etc...), las texturas referentes al entorno HDRE, el mapa LUT, la luz, y los parámetros referentes a escoger el tipo de modalidad de iluminación y figura.
  Una vez realizado el constructor completamos el ”setUniforms” de la subclase creada. Pasamos los parámetros de igual forma que lo hicimos en la práctica anterior, pero cabe destacar que a la hora de cargar todas las texturas añadimos un entero al final: ”setUniform(”u_texture_albedo”, txt_albedo, 0)”, con el fin de enviar todo de forma secuencial al shader y se cargue de forma ordenada.
- **Application.cpp**: Aquí nosotros únicamente modificamos el constructor ”Application”, el método ”update” y por último el ”renderInMenu”.
  - **Application**: cargamos las figuras sin pasarle la *mesh* al nodo puesto que lo haremos en ”update” con los condicionales según las opciones escogidas por el usuario en el menú ImGui. De igual forma cargamos el skybox, pero en este caso sí podemos cargar directamente la *mesh*, puesto que es una figura que no modificamos desde el menú, y le aplicamos el material correspondiente al HDRE.
    Cargamos todos y cada uno de los niveles del HDRE como texturas que posteriormente pasaremos al constructor al crear un ”PBR_material”.
    Y por último generamos la luz, la textura emisiva, y cargamos el shader junto con las texturas de LUT y las respectivas a las figuras generadas previamente.
  - **update**: aquí es donde nosotros decidimos si se mostrará por pantalla la esfera, el casco o el candelabro y con qué tipo de iluminación.
    Para ello, primero de todo comprobamos qué tipo de figura mostrar: casco (figure = 0), candelabro (figure = 1) o esfera (figure = 2). Hecho esto cargamos la figura en la *mesh* y generamos el material PBR que le pasaremos

al nodo.
Cabe decir que hemos usado un condicional con los parámetros: lantern, sphere y helmet; que usamos para mostrar las figuras con una escala apropiada puesto que vimos que el candelabro tenía un tamaño demasiado grande.

- **renderInMenu**: por último tenemos un menú con tres secciones: Light, Figures y Illumination.
  Tal y como hemos comentado anteriormente, aquí es donde hacemos uso de las variables "figure" y "numShader" para poder cambiar el valor de éstos según la opción escogida por el usuario en el menú y posteriormente en "update" generar una figura y una iluminación en concreto.

II. Pasos previos a calcular la iluminación

En esta sección, presentaremos algunos pasos que realizamos antes de calcular la iluminación de los objetos y que son comunes tanto para la iluminación directa como para la indirecta. De ahora en adelante toda la implementación está realizada en el shader:

1) **Cargar texturas y mapas de color**: obtenemos los valores de las texturas, mapa de roughness... Dado que el albedo y la luz emitida por el casco son texturas RGB, les aplicamos la Gamma correction.
2) **Calculamos vectores que necesitaremos**: lo hacemos siguiendo el esquema que encontramos en (Fig. 1) [1].
3) **Aplicamos mapa de normales**: para obtener una textura con más detalle, utilizamos la función "perturbNormal" que nos permite aplicar un mapa de texturas al vector N. Esta perturbación nos ofrece unos vectores normales más imperfectos y por lo tanto más realistas al interactuar con la luz (los objetos no son lisos sino que tienen imperfecciones).
4) **Corrección de color**: este es el último paso que realizamos. Una vez hemos hecho todos los cálculos para obtener la iluminación, hacemos una corrección de color a través del Tone map y la Gamma correction. En el caso del casco, antes de aplicar la Gamma correction le sumamos la luz del casco tal y como se indica en [2].

III. Iluminación directa

Para calcular la iluminación directa, hemos utilizado las fórmulas que encontramos en [3], [4], pero adaptadas a la programación con el soporte encontrado en [5], [6].
A continuación, vemos los pasos seguidos en el shader para calcular la iluminación:

1) **Iluminación directa**: para calcular el componente difuso y especular utilizaremos una interpolación lineal que dependerá de la metalness del objeto, en función de cómo de dieléctrico o conductor es [1]:

```
c_diff = metalness * 0 + (1 - metalness) * albedo;
F0 = metalness * albedo + (1 - metalness) * 0.04;
```

Una vez calculadas estas variables, procedemos a calcular la iluminación:

- **Iluminación difusa** (Fig. 2a): para calcular el componente difuso utilizamos la siguiente fórmula:

$$f_{diff} = \frac{c_{diff}}{\pi} \tag{1}$$

- **Iluminación especular** (Fig. 2b): para calcular el componente especular utilizamos la siguiente fórmula:

$$f_{spec} = \frac{FGD}{4(N \cdot L)(N \cdot V)} \tag{2}$$

donde

$$F = F_0 + (1 - F_0)(1 - (L \cdot N))^5 \tag{3}$$

$$G = G'(L)G'(V),\ G'(X) = \frac{N \cdot X}{(N \cdot X)(1-k)+k},\ k = \frac{(roughness+1)^2}{8} \tag{4}$$

$$D = \frac{\alpha^2}{\pi((N \cdot H)(\alpha^2 - 1) + 1)^2},\ \alpha = roughness^2 \tag{5}$$

- **Iluminación directa total** (Fig. 2c): sumamos ambas iluminaciones, $f = f_{diff} + f_{spec}$.

2) **Luz incidente**: calculamos la luz incidente (L_i), que es un vector de color con una intensidad.
3) **Iluminación total** (Fig. 2d): $L_0 = fL_i(N \cdot L)$

IV. Iluminación indirecta

Para la iluminación indirecta, principalmente nos hemos guiado por [1], y también utilizamos la interpolación lineal mencionada anteriormente. Los pasos seguidos son los siguientes:

1) **Iluminación indirecta**:
   - **Iluminación difusa** (Fig. 3b): para la iluminación difusa, nuestro objetivo es obtener el color del objeto que podemos ver debido a la iluminación del entorno independientemente de la posición de la cámara. Para ello utilizamos la función *getReflectionColor* con el vector *N* como argumento, de manera que para cada nivel del SkyBox obtenemos el color reflejado en el objeto (Fig. 3a). El color obtenido lo mezclamos con el propio color difuso del objeto.
   - **Iluminación especular** (Fig. 3e): en este caso, nuestro objetivo es obtener el color que percibimos del objeto gracias al reflejo obtenido debido a la iluminación del Skybox. Los calculos están divididos en dos partes:
     - **Specular sample** (Fig. 3c): volvemos a utilizar la función *getReflectionColor* pero esta vez le pasamos como argumento el vector reflejado desde la cámara hacia la superficie del pixel ($-V \cdot N$). De esta manera, obtenemos el color del SkyBox reflejado en el objeto desde nuestro punto de vista.
     - **Specular BRDF** (Fig. 3d): en este caso utilizamos el LUT, que nos ofrece la respuesta de la textura del objeto en función de su roughness y la posición desde la que lo estemos observando. Lo que hacemos es multiplicar el componente especular del objeto por el ángulo de visión que tenemos del píxel ($N \cdot V$) y añadirle la roughness del punto correspondiente.
2) **Iluminación total** (Fig. 3f): una vez hemos calculado las dos partes, las sumamos: $IBL = IBL_{diff} + IBL_{spec}$.

V. Iluminación directa + indirecta

Una vez hemos calculado la iluminación directa y la indirecta, podemos calcular la iluminación total del objeto. Para esto simplemente tenemos que sumar ambas iluminaciones. Además, como hemos explicado en la sección 2, una vez las sumamos realizamos la corrección de color con el Tone map, sumamos la iluminación del objeto (solamente en el caso del casco) y realizamos la Gamma correction. En la (Fig. 4) podemos ver la iluminación global obtenida para los 3 objetos.

VI. Skybox

Para el Skybox creamos un ”textureCube” que nos mostrará información dependiendo de la dirección desde la cámara hacia el punto que estemos mirando. Dado que la información de las imágenes HDR viene en espacio lineal, no tendremos que realizar ningún ”degamma”, pero sí que tendremos que corregir el color con el Tone map y la Gamma correction para así poner el espacio de color acorde con los objetos [1].

VII. Anexo

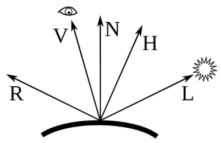
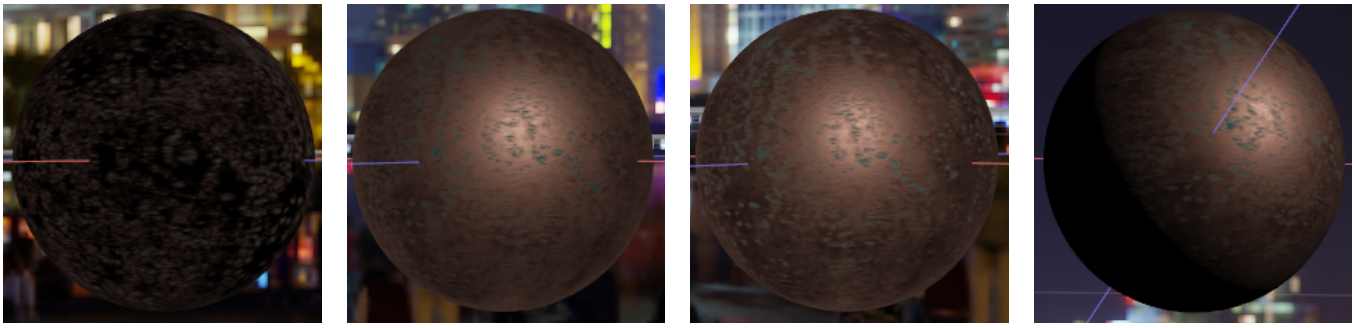


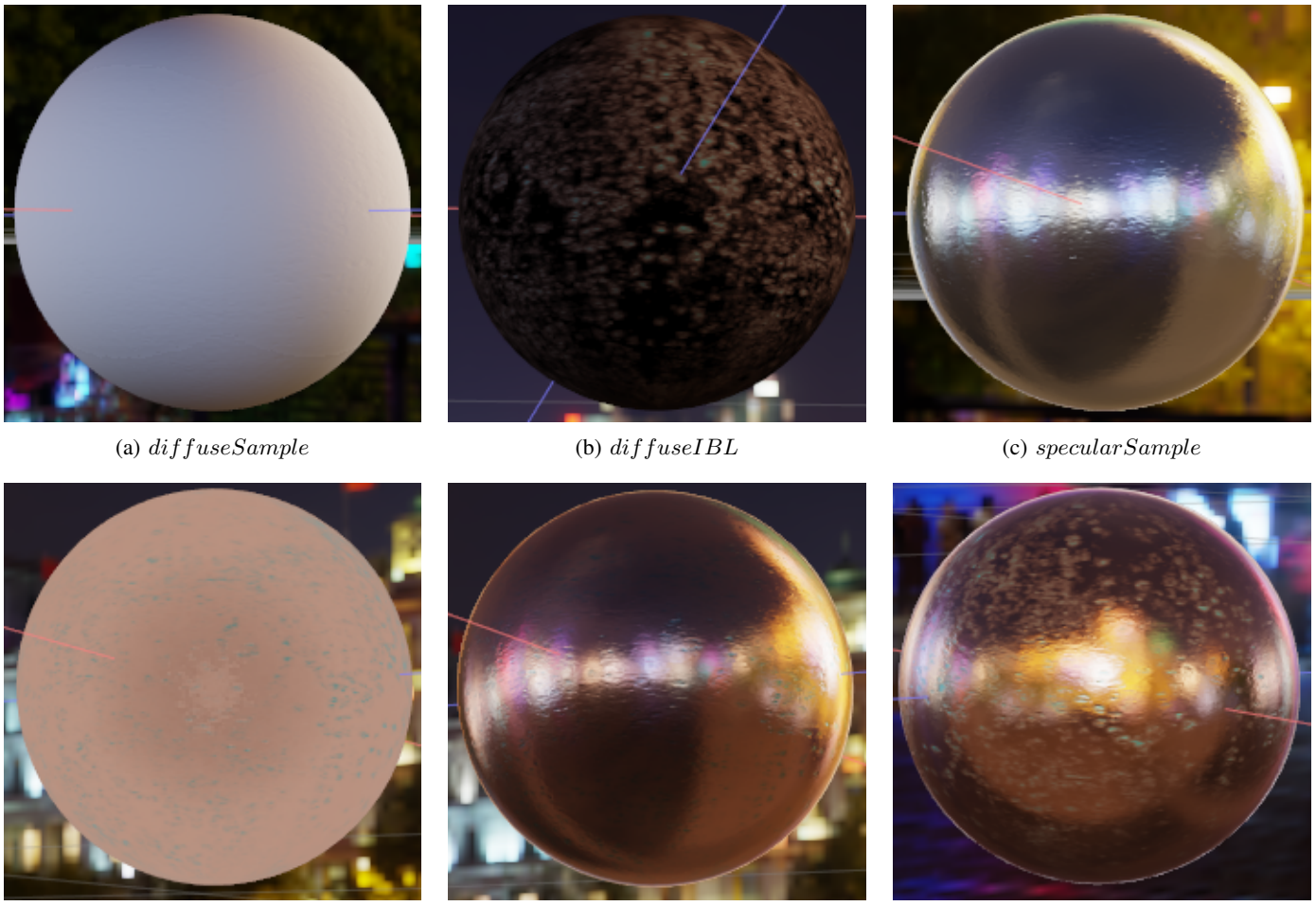
Fig. 1: Light equation vectors

VIII. Resultados obtenidos



(a) f_{diff} (b) f_{spec} (c) $f_{diff} + f_{spec}$ (d) $L_0 = fL_i(N \cdot L)$

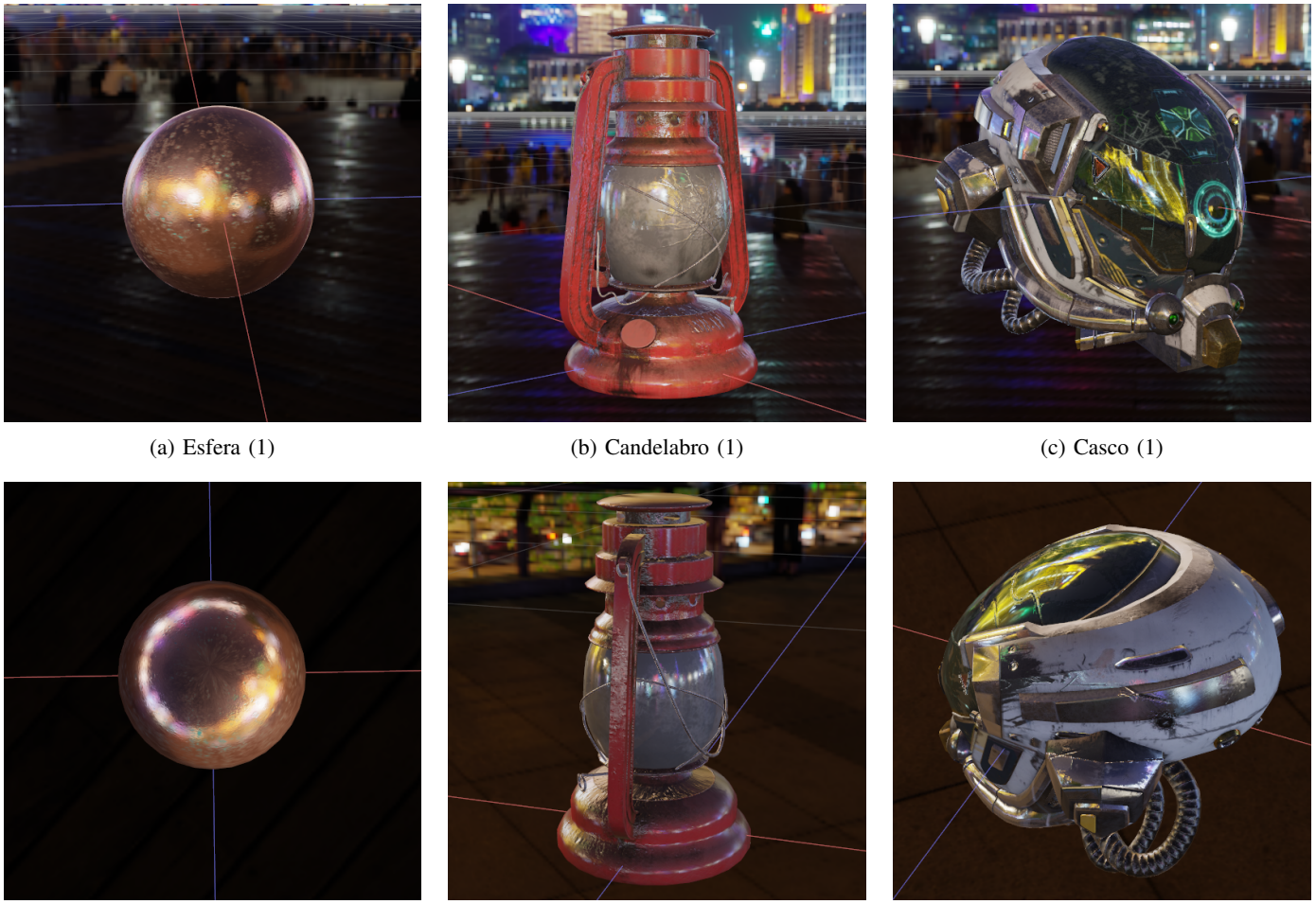
Fig. 2: Iluminación directa en la esfera



(a) $diffuseSample$ (b) $diffuseIBL$ (c) $specularSample$

(d) $specularBRDF$ (e) $specularIBL$ (f) IBL

Fig. 3: Iluminación indirecta en la esfera



(a) Esfera (1) (b) Candelabro (1) (c) Casco (1)

(d) Esfera (2) (e) Candelabro (2) (f) Casco (2)

Fig. 4: Iluminación total para distintos objetos

REFERENCES

[1] "Phhysically based rendering lab 2," *slides de Advanced Computer Graphics*, 2020.

[2] "pbr.txt," *Guía para aplicar el shader proporcionada por Discord.*

[3] "Photo-realistic rendering," *slides de Advanced Computer Graphics*, 2020.

[4] "Photo-realistic rendering 2," *slides de Advanced Computer Graphics*, 2020.

[5] LearnOpenGL, "Theory," Recuperado de: https://learnopengl.com/PBR/Theory (07/11/2020).

[6] ——, "Lighting," Recuperado de: https://learnopengl.com/PBR/Lighting (08/11/2020).