Universidad
Rey Juan Carlos

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA DE SOFTWARE

Curso académico 2023/2024
Gráficos por Computador
Práctica 1: Iluminación

Grupo 30:
Stefano Tomasini Hoefner
Rémi Cazoulat

Índice

Índice de figuras

1. Objetivos de la práctica

El objetivo principal de esta práctica es implementar y manipular diferentes tipos de fuentes de luz, que iluminan unos cubos mediante diferentes cómputos dentro del shader de fragmentos. También se exige la capacidad de desplazar la cámara para que se pueda visualizar la escena y observar cómo se comportan las distintas luces según a dónde apunta la cámara, o su lejanía. Concretamente, se implementan estos tipos de iluminación: ambiental, difusa, especular, focal, direccional, y puntual.

2. Cámara

Para definir la cámara, se hace uso de la matriz `lookat` de la biblioteca `gl-matrix`. Para crear esta matriz 4x4, hacen falta tres vectores tridimensionales: el vector `pos` que comprende la posición de la cámara, el vector `front`, que es a donde la cámara mira, y el ultima, el vector `up`, que es a donde apunta la parte superior de la cámara.

2.1. Translación

Para transladar la cámara, se debe alterar el vector `cameraPos`. Al vector `cameraPos` se le suma el vector `cameraUp`, que determina el movimiento vertical, y/o el vector `cameraFront` si se adelanta o retrocede, y/o el vector resultante del producto vectorial entre el vector `cameraFront` y el vector `cameraUp` si se produce un movimiento lateral.

2.2. Rotación

Para cambiar a donde apunta la camara, hay que actualizar el vector `front` según dos ángulos: `yaw` y `pitch`. `yaw` controla la rotación lateral, mientras que `pitch` determina la rotación vertical dentro de lo que sería un plano vertical paralelo a la dirección de la cámara.

2.3. Controles

- W / Z / Flecha arriba : Adelanta la cámara
- S / Flecha abajo : Retrocede la cámara
- A / Q / Flecha izquierda : Desplaza la cámara a la izquierda
- D / Flecha derecha : Desplaza la cámara a la derecha
- Espacio: ascender
- Shift + espacio: descender
- C : Reinicia la posición y la rotación de la cámara
- Cursor : Cambia la direccion de la camara

3. Luces

3.1. Luz ambiental

Este es el tipo de iluminación más simple que existe, ya que no tiene en cuenta la dirección de nada, se aplica indististamente sobre todos los fragmentos.

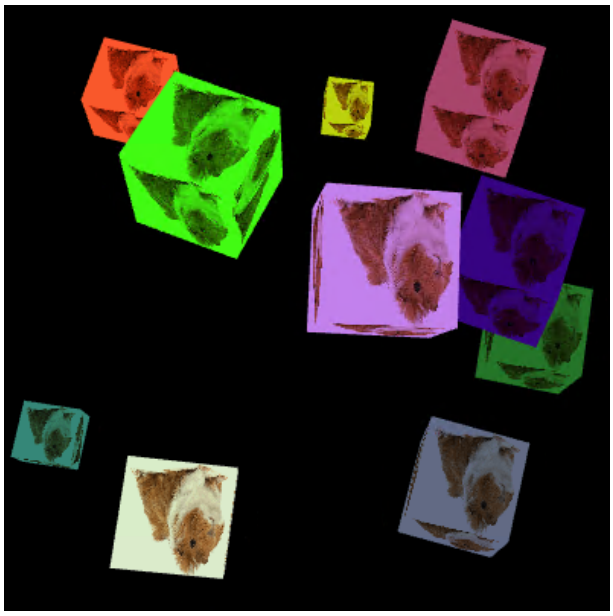

Figure 1: Luz ambiental

Nota: ahora hay más texturas entre los cubos, las imágenes quedaron desactualizadas

Para implementar una luz ambiental, tienen que entrar dos variables uniformes pertenientes a esta luz al shader de fragmentos: su intensidad definida (un `float` de 0.0 a 1.0), y su color (`vec3`). Al multiplicar estos por el color que sería "innato" al fragmento y asignar el resultado a fragColor (`out vec4`), se obtiene una iluminación ambiental para todos los fragmentos de la escena.

```
vec3 ambient = ambientStrength * ambientColor;
fragColor = vec4(ambient * objectColor * texture(uTexture, vTexCoord).rgb, 1.0);
```

3.2. Luz difusa

Para implementar una iluminación difusa, hay que tener lo siguiente en cuenta: la dirección de la luz hacia el fragmento (en nuestro caso la luz proviene de la cámara, así que su vector de dirección sería: `normalize(fragPos-cameraPos)`), y la normal de la cara en la que está el fragmento (que proviene del shader de vértices). Al multiplicar escalarmente ambos vectores normalizados mediante `dot()`, si son perfectamente paralelos, se devolvería un 1.0. Por el contrario, si son perpendiculares se devolvería un 0.0. Este valor entre 0.0 y 1.0 determina la intensidad de la iluminación difusa para el fragmento.

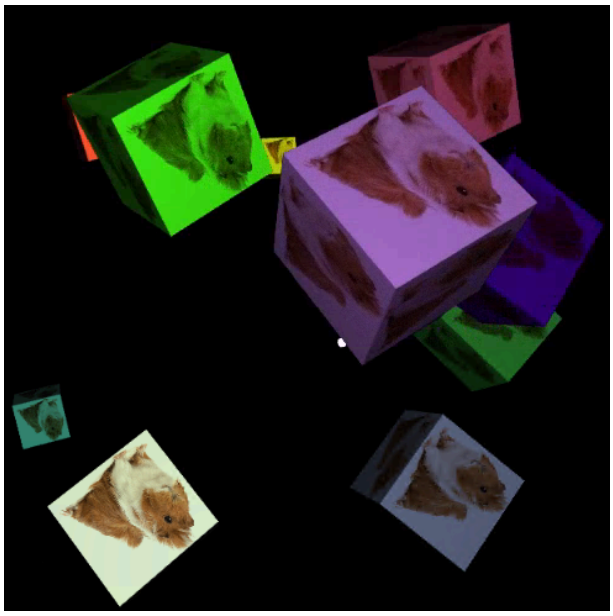

Figure 2: Luz difusa (exclusivamente)

Cómo se puede observar, las caras que apuntan directamente a la cámara están plenamente iluminadas, mientras que las que están desviadas reciben menos iluminación.

En nuestro código se computa así, siendo `diffuseStrength` un `float` uniforme que se utiliza para atenuar la luz arbitrariamente:

```
diffuse =  max(dot(norm, lightDir), 0.0) * diffuseColor * diffuseStrength;
```

Y se suma al resto de luces:

```
fragColor = vec4((ambient+diffuse)*objectColor*texture(uTexture, vTexCoord).rgb,1.0);
```

3.3. Luz especular

La iluminación especular se produce cuando la reflección de una luz sobre una cara resulta ser paralela al vector dirección desde la la cámara hasta el fragmento.

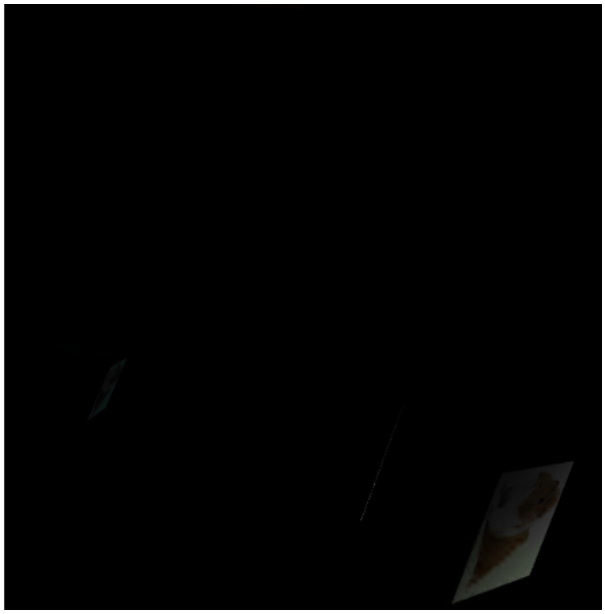

Figure 3: Luz especular (exclusivamente)

Para implementarla, se debe calcular el vec3 viewDir que es la direccion que va desde la posición de la camara hasta la del fragmento, y el vec3 reflectDir que es la reflección del vector de la direccion de la luz sobre el vector normal de la cara a la cual pertenece el fragmento:

```
viewDir = normalize(viewPos - fragPos);
vec3 reflectDir = reflect(lightDir, norm);
vec3 specular = specularStrength * pow(max(dot(viewDir, reflectDir), 0.0),
specularRadius) * specularColor;
```

3.4. Luz puntual

Una luz puntual consiste en una fuente de luz que irradia haces de luces hacia todas las direcciones partiendo de una posición concreta.

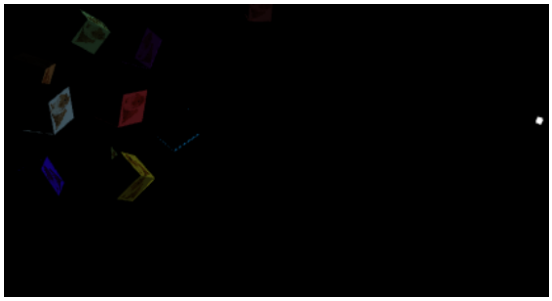

Figure 4: Luz puntual

Figure 5: Luz puntual

A la derecha se puede ver la luz puntual (el pequeño cubo blanco) y del otro lado están los cubos iluminados por esta. En la imagen de la derecha, la luz puntual está por atrás de los cubos, por lo que de frente están poco iluminados. Nuestra luz puntual orbita dentro de un plano horizontal, en torno al punto de origen de la escena. Para calcular la intensidad de esta luz para un determinado fragmento, se debe calcular el vector dirección de la luz hacia el fragmento, substrayendo sus posiciones y normalizando:

```
pointLightDir = normalize(pointLightPos - fragPos);
```

Después, podemos multiplicar escalarmente el vector dirección normalizado y el vector normal del fragmento con `dot()`. Para que no pueda resultar un float negativo, se extrae el máximo entre el producto escalar y cero. Este valor se multiplica con el color establecido de la luz, y el factor arbitrario de fuerza que se establece desde el lado de JavaScript, para obtener la intensidad final. `vec3 pointLight = max(dot(norm, pointLightDir), 0.0) * pointLightColor * pointLightStrength;`

3.5. Luz focal

Una fuente de iluminación focal comprende una luz que origina desde cierto punto pero que solo se irradia en forma de un cono que se define por un ángulo que define hasta dónde hay iluminación de intensidad máxima, y un ángulo de cutoff, dónde la intensidad bajo a cero. De esta forma, se consigue un gradiente de iluminación entre estos límites del foco.

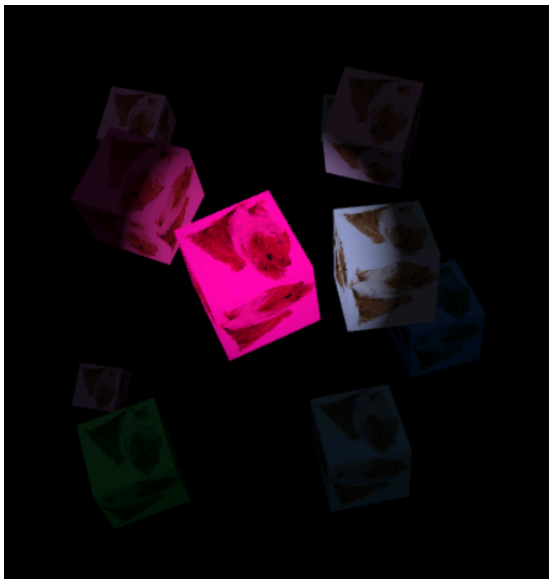

Figure 6: Luz focal

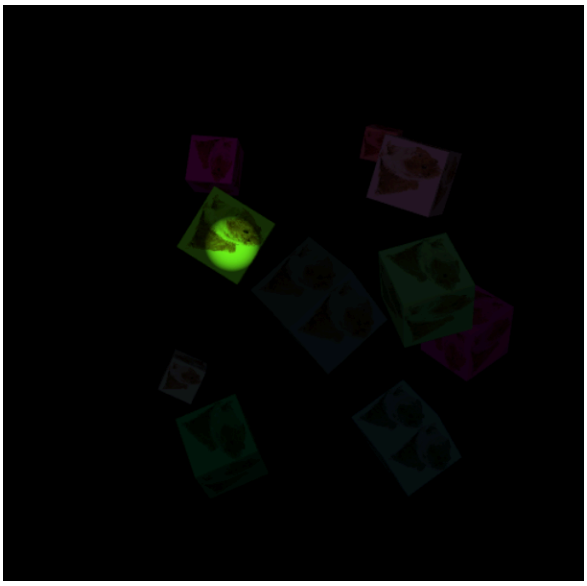

Figure 7: Luz focal con una pequeña apertura

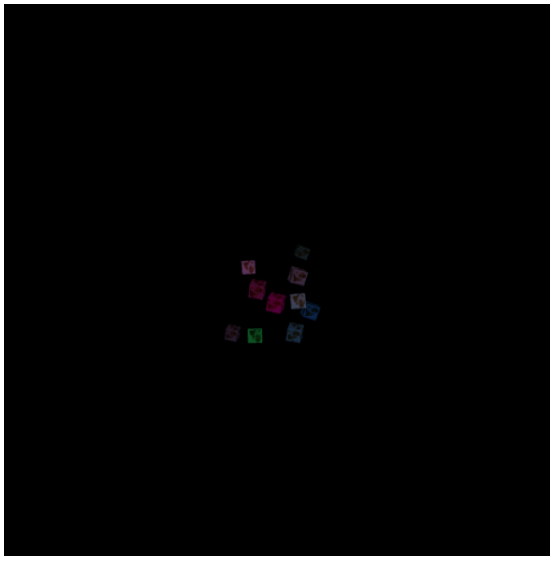

Figure 8: Luz focal atenuada con la distancia

La luz focal está acoplada a la cámara, y se puede cambiar su apertura. Cuando la cámara está lejos de los cubos, se puede ver que la fuerza de la luz focal se atenúa:

```
float distance = length(cameraPosition - fragPos) / spotLightReach;
float fwin = pow(max(1.0 - pow(distance / spotLightReach, 4.0), 0.0), 2.0);
float spotLight_fdistance = fwin * pow(spotLightStrength, 2.0) / (pow(distance, 2.0)
+ 1.0);
```

Para determinar si un fragmento es iluminado por la luz focal, el coseno del ángulo entre la cámara y el fragmento tiene que ser mayor que el coseno del ángulo de apertura del foco. Si es el caso, se atenúa la intensidad de la luz si el fragmento está en el rango "gradiente" de la luz focal:

```
float cosAngleVecs = dot(-L, spotLightDirection)/ (length(-L) *
                                                   length(spotLightDirection));
float cosShadowAngle = cos(spotLightShadowAngle);
float cosCutOffAngle = cos(spotLightCutOffAngle);
if (cosCutOffAngle < cosAngleVecs)
    if (cosShadowAngle < cosAngleVecs) spotLight_fdir = 1.0;
    else {
```

```
        float t = pow((cosAngleVecs-cosCutOffAngle) / (cosShadowAngle-cosCutOffAngle),
    0.5);
        spotLight_fdir = pow(t, 2.0) * (3.0 - 2.0 * t);
    }
else spotLight_fdir = 0.0;
```

Para implementar la atenuación por distancia:

```
float distance = length(cameraPosition - fragPos) / spotLightReach;
float fwin = pow(max(1.0 - pow(distance / spotLightReach, 4.0), 0.0), 2.0);
float spotLight_fdistance = fwin * pow(spotLightStrength, 2.0) / (pow(distance, 2.0) +
1.0);

vec3 spotLight = spotLightColor*spotLight_fdir*spotLight_fdistance;
```

Finalmente, se suma al resto de luces:

```
fragColor = vec4((ambient + specular + diffuse + spotLight) * objectColor *
texture(uTexture, vTexCoord).rgb, 1.0);
```

3.6. Luz direccional

La luz direccional consiste en haces de luces que provienen del infinito, con una cierta dirección.

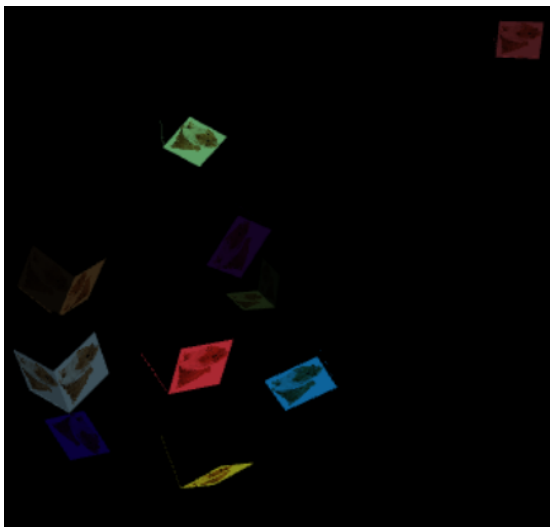

Figure 9: Luz direccional (orbitando dentro de un plano vertical)

Para calcular la intensidad de iluminación para un fragmento, dentro del shader de fragmentos hace falta un `vec3` uniforme arbitrario que determine la dirección de la luz, y un vector tridimensional que comprende la normal de la cara en la que esté el fragmento.

Para calcular la intensidad para un cierto fragmento, se multiplica escalarmente el opuesto de la dirección de la luz por la normal de la cara en la que esté el fragmento. En nuestro caso se computa así, siendo dirLightStrength un atenuador arbitrario de la luz (float entre 0 y 1 asignado en JS):

```
vec3 dirLight = dot(norm, normalize(-dirLightDirection)) * dirLightColor
* dirLightStrength;
```

Y se suma al resto de luces:

```
fragColor =
vec4((ambient+specular+diffuse+spotLight+dirLight)*objectColor*texture(uTexture,
vTexCoord).rgb, 1.0);
```

En nuestra implementación, el vector dirLightDirection cambia de valor en cada llamada a `renderFunc`, ya que sus componentes i y j son dependientes del tiempo:

```
const orbitingSpeed = 0.1;
dirLightDirection[0] = Math.cos(time*orbitingSpeed);
dirLightDirection[1] = Math.sin(time*orbitingSpeed);
```

De esta forma, la luz direccional asemeja la iluminación de que haya un sol en la escena (aunque sería un sol invisible).

3.7. Controles

A cada luz le corresponde un selector de color cuyo canal alfa sirve para atenuar o maximizar la fuerza de la luz, lo cual es útil para hacer *debugging*, o ver cómo se comporta aisladamente cada luz.

Luz focal :

- P: Aumenta la apertura angular definitoria del cono exterior del foco
- M: Hace lo contrario
- shift + P: Aumenta la apertura angular definitoria del cono interior de máxima iluminación del foco
- shift + M: Hace lo contrario

4. Cubos

Para generar los múltiples cubos, se debe pintar cada uno con una nueva matriz modelo en una posición única. Así se producen todos los cubos:

```
for(let i = 0; i < cubePositions.length; i ++)
{
    const angle = (i + 1.0) / 3.0 * time * 50.0;
    let model = mat4.create();
    mat4.translate(model, model, cubePositions[i]);
    mat4.rotate(model, model, Math.radians(angle), [1.0, 0.5, 1.0]);
    shaderProgram.setUniformMat4("model", model);

    const cubeColor = new Float32Array([randomColors[0][i], randomColors[1][i],
randomColors[2][i]]);
    shaderProgram.setUniform3f("objectColor", ...cubeColor);
    gl.activeTexture( gl.TEXTURE0 + i%texturesIds.length);
    shaderProgram.setUniform1i( "uTexture", i%texturesIds.length );
    gl.bindTexture( gl.TEXTURE_2D, texturesIds[i%texturesIds.length]);
    gl.drawElements(gl.TRIANGLES, 36, gl.UNSIGNED_INT, 0);
}
```

4.1. Colores

Los colores de los cubos se seleccionan de forma aleatoria cada vez que se carga la página.

4.2. Texturas

En las imágenes mostradas en esta memoria no se puede observar porque quedaron desactualizadas, pero hay una gran variedad de texturas puestas entre todos los cubos.

4.2.1. Solucionar errores por CORS

Se abre una consola en el directorio del proyecto y se ejecuta esta línea:

```
python3 -m http.server
```

A continuación, visitar: localhost:8000/template.html

4.3. Controles

- F5 : actualiza la página y realeatoriza los colores de los cubos.