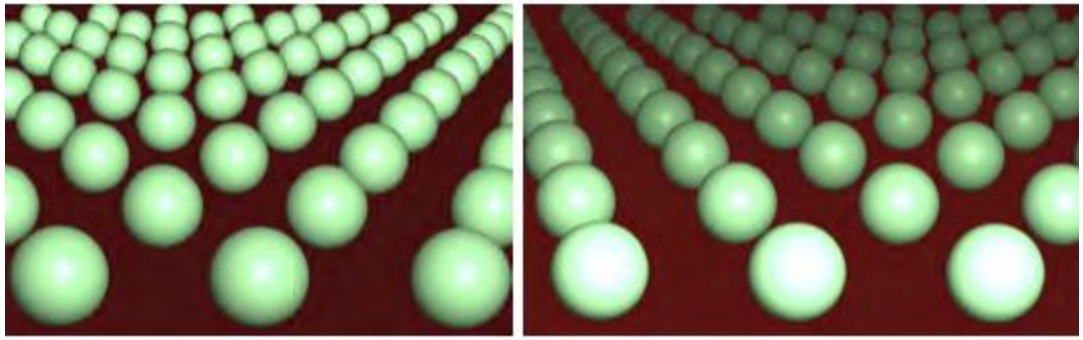
Iluminación y sombreado II

Atenuación

Para tener en cuenta la atenuación que sufre la luz al viajar desde su fuente de origen hasta la superficie del objeto situado a una distancia d, el modelo de Phong establece utilizar la siguiente ecuación donde los coeficientes a, b y c son constantes características de la fuente de luz:

$$attenuationFactor = \frac{1}{a + bd + cd^2}$$

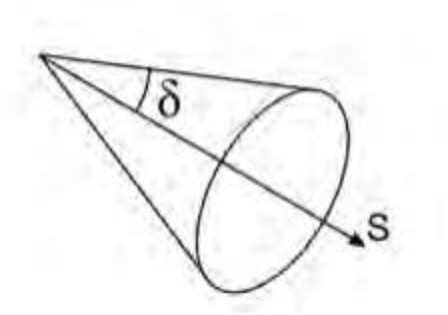


Tipos de fuentes de luz

En general, siempre se han considerado dos tipos de fuentes de luz dependiendo de su posición:

Posicional: la fuente emite luz en todas las direcciones desde un punto dado, muy parecido a como ilumina una bombilla, por ejemplo.

Direccional: la fuente está ubicada en el infinito, todos los rayos de luz son paralelos y viajan en la misma dirección. En este caso el vector L en el modelo de iluminación de Phong es constante



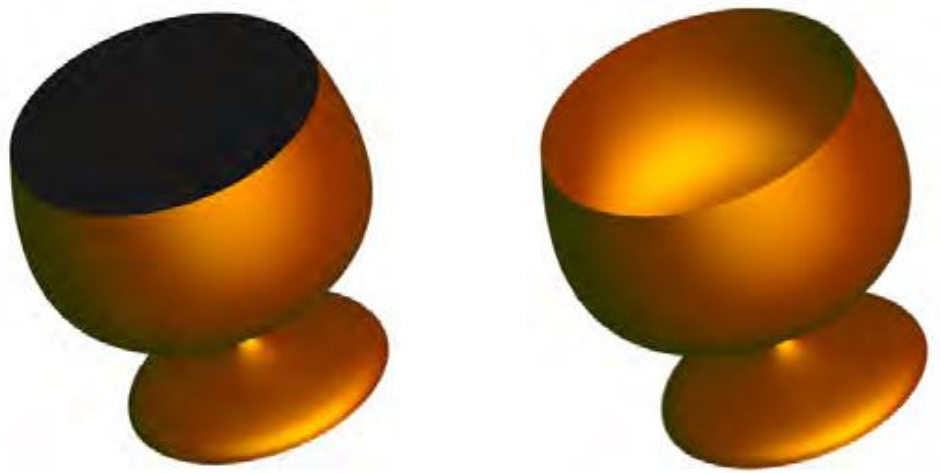
Tipos de fuentes de luz

Shader para calcular la iluminación como un foco de luz

```
struct LightData {  // Solo figuran los campos nuevos
    ...
    vec3 Direction;   // Dirección de la luz en coordenadas del ojo
    float Exponent;   // Atenuación
    float Cutoff;     // Ángulo de corte en grados
};

uniform LightData Light;

vec3 phong(vec3 N, vec3 L, vec3 V) {
    vec3 ambient  = Material.Ka * Light.La;
    vec3 diffuse  = vec3(0.0);
    vec3 specular = vec3(0.0);

    float NdotL = dot(N, L);
    float spotFactor = 1.0;

    if (NdotL > 0.0) {
        vec3 S = normalize(Light.Position - ec);
        float angle = acos(dot(-S, Light.Direction));
        float cutoff = radians(clamp(Light.Cutoff, 0.0, 90.0));

        if (angle < cutoff) {
            spotFactor = pow(dot(-S, Light.Direction), Light.Exponent);

            vec3 R = reflect(-L, N);
            float RdotV_n = pow(max(0.0, dot(R, V)), Material.alpha);

            diffuse  = NdotL * (Light.Ld * Material.Kd);
            specular = RdotV_n * (Light.Ls * Material.Ks);
        }
    }

    return (ambient + spotFactor * (diffuse + specular));
}
```

Iluminación por ambas caras

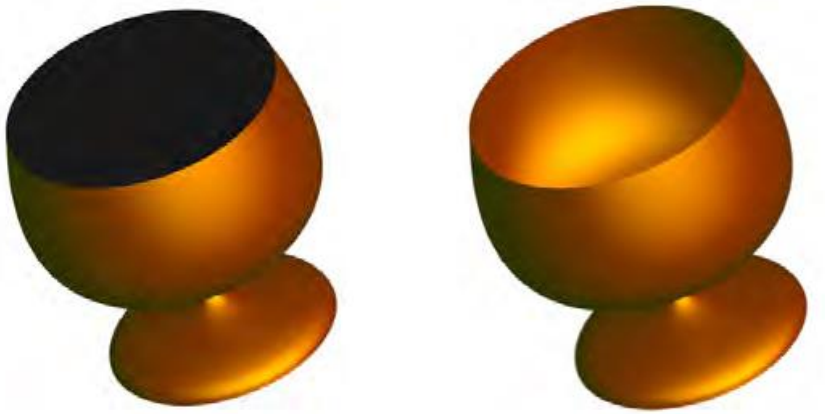
Cuando la escena incorpora modelos abiertos, es posible observar los polígonos que se encuentran en la parte trasera de los objetos, como, por ejemplo, ocurre en la copa. Para una correcta visualización es necesario utilizar la normal contraria en los polígonos que forman parte de la cara trasera.



Iluminación por ambas caras

En shader de fragmentos:

```
if (gl_FrontFacing)
    fragmentColor = vec4(phong(n, L, V), 1.0);
else
    fragmentColor = vec4(phong(-n, L, V), 1.0);
```



Iluminación por ambas caras

Ejercicio:

1. Implementa un shader que te permita sombrear objetos por ambas caras. Esto es muy útil en el caso de que los objetos sean abiertos como, por ejemplo, ocurre con las primitivas del cilindro, el cono o el plano.
2. Tratar las dos caras de los polígonos te permitiría utilizar un tipo de material diferente para cada cara. Por ejemplo, material oro para las caras delanteras y material plata para las trasera. Piensa como lo harías y trata de implementarlo en tu shader.

Toon shader

Unity Toon Shader Tutorial

El objetivo es simular el sombreado típico en cómics. Este efecto se consigue haciendo que la componente difusa del color de un fragmento se restrinja a solo un número determinado de posibles valores

```
vec3 toonShading(vec3 N, vec3 L) {
 vec3 ambient = Material.Ka * Light.La;
 float NdotL = max(0.0, dot(N, L));
 float levels = 3.0; float scaleFactor = 1.0 / levels;
 vec3 diffuse = ceil(NdotL * levels) * scaleFactor *
(Light.Ld * Material.Kd);
 return (ambient + diffuse);
}
void main() {
 vec3 n = normalize(N);
 vec3 L = normalize(Light.Position - ec);
 fragmentColor = vec4(toonShading(n, L), 1.0);
}
```

Toon shader

Unity Toon Shader Tutorial

Ejercicio: Añade la función toonShading a tu shader de fragmentos y haz que se llame a esta en lugar de a la función phong. Observa que la componente especular se ha eliminado de la ecuación, por lo que la función toonShading solo necesita los vectores N y L.

Opcional: Para complementar el shader de sombreado cómic, sería muy interesante reforzar también los bordes del objeto dibujado utilizando el color negro. Dirígete a la bibliografía y encuentra técnicas que te permitan conseguir este segundo efecto utilizando shaders. También puedes seguir el siguiente tutorial para Unity pero que es fácilmente adaptable a WebGL:

Unity Outline Shader Tutorial - Roystan

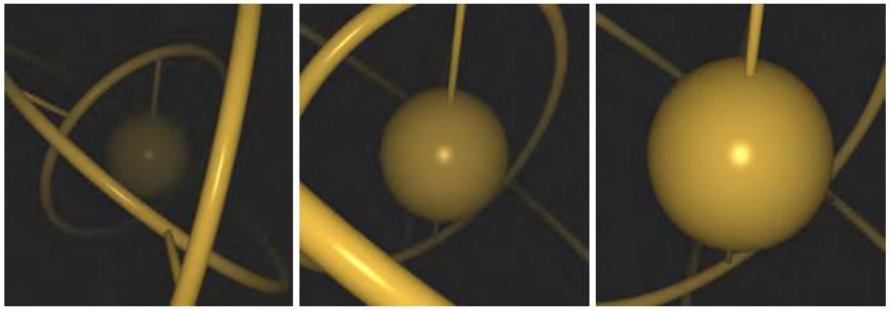
Niebla

El efecto de niebla se consigue mezclando el color de cada fragmento con el color de la niebla. A mayor distancia de un fragmento respecto a la cámara, mayor peso tiene el color de la niebla y al contrario, a menor distancia, mayor peso tiene el color del fragmento.

En primer lugar hay que obtener el color del fragmento y después, a modo de posproceso, se mezcla con el color de la niebla dependiendo de la distancia a la cámara. Para implementar este efecto se definen tres variables: distancia mínima, distancia máxima y color de la niebla. Así, dado un fragmento, tenemos tres posibilidades:

- Si el fragmento está a una distancia menor que la distancia mínima, no se ve afectado por la niebla, el color definitivo es el color del fragmento.
- Si el fragmento está a una distancia mayor que la máxima, el color definitivo es el color de la niebla (es importante que el color de fondo coincida con el de la niebla).
- Si el fragmento está a una distancia entre la mínima y la máxima, su color definitivo depende del color del fragmento y del de la niebla. Esta variación puede ser lineal con la distancia, pero suele producir mucho mejor resultado utilizar una función exponencial.

Niebla



```
struct FogData {
    float maxDist;
    float minDist;
    vec3 color;
};

FogData Fog;
```

```
void main() {
    // Parámetros de la niebla
    Fog.minDist = 10.0;
    Fog.maxDist = 20.0;
    Fog.color   = vec3(0.15, 0.15, 0.15);

    // Cálculos de vectores básicos
    vec3 n = normalize(N);
    vec3 L = normalize(Light.Position - ec);
    vec3 V = normalize(-ec);

    // Distancia del fragmento a la cámara
    float dist = abs(ec.z);

    // --- Niebla lineal ---
    float fogFactor = (Fog.maxDist - dist) /
(Fog.maxDist - Fog.minDist);

    // --- Niebla exponencial (alternativa) ---
    // float fogFactor = exp(-pow(dist, 2.0));

    fogFactor = clamp(fogFactor, 0.0, 1.0);

    // Cálculo del color con iluminación Phong
    vec3 phongColor = phong(n, L, V);

    // Mezcla entre el color del objeto y el de la
niebla
    vec3 myColor = mix(Fog.color, phongColor,
fogFactor);

    fragmentColor = vec4(myColor, 1.0);
}
```

Ejercicio

Implementa varios shaders de los que has visto en este apartado en la escena que quieras. Prueba a experimentar con las diferentes técnicas que has visto. Si quieres puedes añadir código extra para activar y desactivar los shaders con el teclado.