PROGRAMACIÓN 3D

Máster en Programación de Videojuegos

TEMA 4.1: LUCES (OpenGL)

Juan Mira Núñez

U-tad
CENTRO UNIVERSITARIO
DE TECNOLOGÍA Y ARTE DIGITAL

Índice

- Introducción
- Gouraud Shading
- Phong Shading
- Blinn-Phong shading

Introducción

- Vamos a ver en profundidad el tema de los shaders y las luces.
- Vamos a ver, cómo se calcula la luz usando lo aprendido en el tema anterior.
- Veremos el cálculo de la luz usando el vertex shader y el fragment shader.
- Aprenderemos a usar structs y más cosas de los shaders.

Analicemos nuestro shader…

```
#version 430
layout (location=0) in vec3 vertPos;
layout (location=1) in vec3 vertNormal;
out vec4 varyingColor;
struct PositionalLight
{
        vec4 ambient;
        Vec4 diffuse;
        Vec4 specular;
        Vec3 position;
};
uniform PositionalLight light;
```

Número de versión de OpenGL

Forzamos un id Attribute

Variable de entrada (in) y salida (out)

Se pueden crear estructuras

Positional light and Gouraud Shading

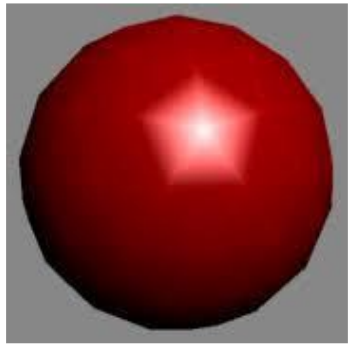
El **sombreado Gouraud** es una técnica usada en gráficos 3D por ordenador que simula efectos de luz y color sobre superficies de objetos.

Fue publicada por Henri Gouraud en 1971.

Esta técnica de sombreado permite suavizar superficies con una carga computacional menor que con otros métodos basados en el cálculo píxel a píxel.

1. Se calculan los valores de luminosidad de cada vértice promediando los valores de las superficies que en él convergen mediante el método de reflexión de Phong.
2. Se calculan las intensidades de cada píxel de las superficies implicadas mediante interpolación bilineal a partir de los valores estimados de los vértices. De esta manera, los valores de luminosidad de los polígonos consisten en el gradiente formado por los valores de los vértices

Problema: ejemplo

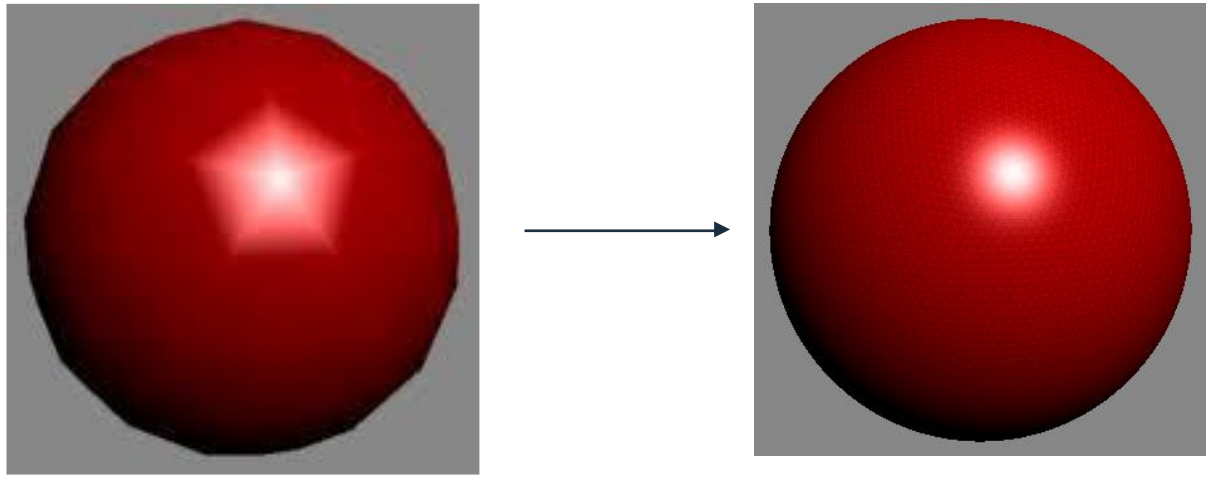


Positional light and Gouraud Shading

Ventajas:

- Interpolar la luz de una superficie completa a partir de tres valores originales que requieren bastante tiempo de computación siempre será más rápido que calcular dichos valores de luz píxel a píxel (Phong)

- A mayor densidad poligonal mejor resultado se obtiene.



Positional light and Gouraud Shading: Paso a Paso (Vertex shader)

1. Definimos Versión de Open GL

```
#version 430
```

2. Creamos variables de entrada para la posición del vertice y para su normal:

```
layout (location=0) in vec3 vertPos;
layout (location=0) in vec3 vertNormal;
```

3. Creamos variable de salida al fragment shader:

```
out vec4 varyingColor;
```

4.Definimos un struct para pasar los datos de la luz y creamos una varible de ese tipo para comunicar los valores al shader:

```
struct PositionalLight
{
        vec4 ambient;
        vec4 diffuse;
        Vec4 specular;
        Vec3 position;
};
uniform PositionalLight light;
```

Positional light and Gouraud Shading: Paso a Paso (Vertex shader)

5. Hacemos lo propio para pasar valores al material:

```
struct Material
{
	vec4 ambient;
	vec4 diffuse;
	vec4 specular;
	float shininess;
};
uniform Material material;
```

6. Creamos el resto de variables que necesitamos, la luz ambiental, la matriz mv , la matriz de proyección y la matriz de normales:

```
uniform vec4 globalAmbient;
uniform mat4 mv_matrix;
uniform mat4 proj_matrix;
uniform mat4 norm_matrix;
```

Positional light and Gouraud Shading: Paso a Paso (Vertex shader)

7. Creamos el programa principal del shader

```
Void main (void)
{
vec4 color;
```

8. Convertimos la posición del vertice al espacio de la vista:

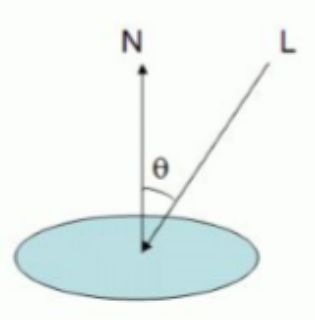
```
vec4 P = mv_matrix * vec4(vertPos, 1.0);
```

9. Convertimos la normal al vertice al espacio de la vista:

```
vec3 N = normalize((norm_matrix * vec4(vertNormal,1.0)).xyz);
```

10. Calculamos el vector de el vertice a la luz en espacio de vista

```
vec3 L = normalize(light.position - P.xyz);
```



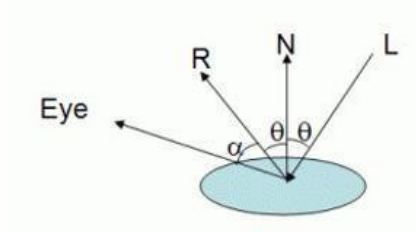
Positional light and Gouraud Shading: Paso a Paso (Vertex shader)

11. Vector Eye:

vec3 Eye= normalize(-P.xyz);

12. Vector R:

vec3 R =reflect(-L,N);



13. I. Ambiente = Luz Ambiente * Material Ambiente

vec3 ambient= ((globalAmbient* material.ambient) + (light.ambient*material.ambient)).xyz;

14. **I. Difuso = Luz Difusa * Material difuso * cos(θ)**

vec3 diffuse= light.diffuse.xyz* material.diffuse.xyz* max(dot(N,L), 0.0);

15. **I. Especular= (R·Eye) * Brillo* Material Especular**

vec3 specular= material.specular.xyz* light.specular.xyz* pow(max(dot(R,Eye), 0.0f),material.shininess);

Positional light and Gouraud Shading: Paso a Paso (Vertex shader)

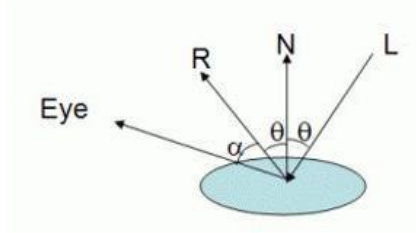
16. Mandamos el color al fragment shader:

varyingColor= vec4((ambient+ diffuse+ specular), 1.0);

17. Calculamos la posición del vertice:

gl_Position= proj_matrix* mv_matrix* vec4(vertPos,1.0);

}



Positional light and Gouraud Shading: Paso a Paso (Fragment shader)

```
#version 430
in vec4 varyingColor;
out vec4 fragColor;
struct PositionalLight
{ vec4 ambient;
vec4 diffuse;
vec4 specular;
vec3 position;
};
struct Material
{ vec4 ambient;
vec4 diffuse;
vec4 specular;
float shininess;
};
uniform vec4 globalAmbient;
uniform PositionalLight light;
uniform Material material;
uniform mat4 mv_matrix;
uniform mat4 proj_matrix;
uniform mat4 norm_matrix;
void main (void)
{
fragColor = varyingColor;
}
```

Positional light and Gouraud Shading: Paso a Paso (Configurar desde C++)

Localización de variables uniform de dentro de un struct:

```
ambientLoc= glGetUniformLocation(idProgram, "light.ambient");

diffuseLoc= glGetUniformLocation(idProgram, "light.diffuse");
```

Seteo de variables:

```
glProgramUniform4fv(idProgram, ambientLoc, 1, lightAmbient);

glProgramUniform4fv(idProgram, diffuseLoc, 1, lightDiffuse);
```

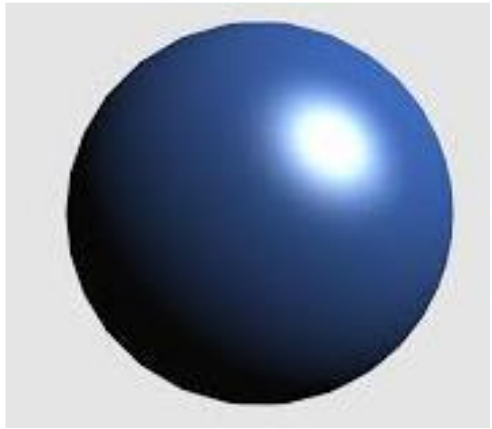
Matriz de normales:

```
normalLoc= glGetUniformLocation(idProgram, "norm_matrix");

...

inverseTMat= glm::transpose(glm::inverse(mvMat));

glUniformMatrix4fv(normalLoc, 1, GL_FALSE, glm::value_ptr(inverseTMat));
```

Phong Shading

Se calculan las normales a cada vértice, luego se interpolan en cada pixel de los polígonos rasterizados para finalmente calcular el color del pixel basándose en la normal interpolada y el método de iluminación.

Este método de sombreado presenta un costo computacional más elevado que el sombreado de Gouraud ya que se debe calcular la iluminación en cada pixel, pero ofrece una mejor calidad con reflejos especulares más exactos.



Phong Shading: Paso a paso (Vertex shader)

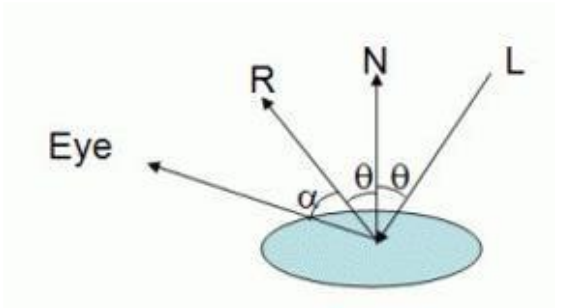
```
#version430
layout (location=0) in vec3 vertPos;
layout (location=1) in vec3 vertNormal;
outvec3 varyingNormal;      // Normal al vertice en espacio de vista
outvec3 varyingLightDir;    // Vector de la luz al vertice
outvec3 varyingVertPos;     // Posición del vertice en espacio de vista
... // El resto de variables, son igual al anterior:
void main(void)
{
varyingNormal= (norm_matrix * vec4(vertNormal,1.0)).xyz;
varyingLightDir= light.position - varyingVertPos;
varyingVertPos= (mv_matrix* vec4(vertPos,1.0)).xyz;
gl_Position= proj_matrix* mv_matrix* vec4(vertPos,1.0);
}
```

Phong Shading: Paso a paso (Fragment shader)

Variables:

```
#version 430
in vec3 varyingNormal;
in vec3 varyingLightDir;
in vec3 varyingVertPos;
out vec4 fragColor;
```

... El resto igual que en el shader anterior



Phong Shading: Paso a paso (Fragment shader)

```
void main(void)
{
```

1-Normalizamos Vectores:

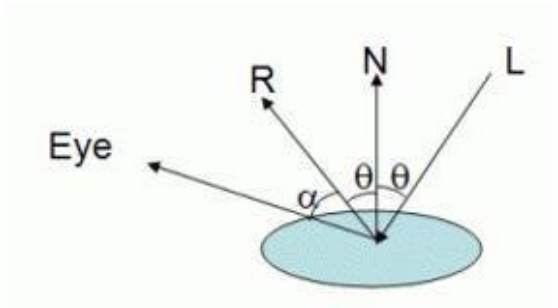
```
vec3 L   = normalize(varyingLightDir);
vec3 N   = normalize(varyingNormal);
vec3 Eye= normalize(-varyingVertPos);
```

2-Calculamos vector R

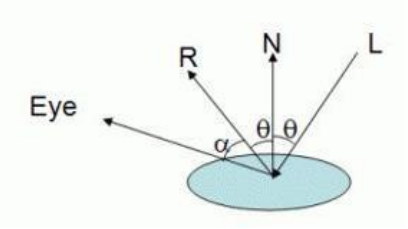
```
vec3 R = normalize(reflect(-L, N));
```

Calculamos ángulos

```
floatcosTheta= dot(L,N);
floatcosAlpha= dot(Eye,R);
```



Phong Shading: Paso a paso (Fragment shader)



3-Calculamos la contribución de cada componente de la luz:

3.1- I. Ambiente = Luz Ambiente * Material Ambiente

```
vec3 ambient= ((globalAmbient * material.ambient) + (light.ambient * material.ambient)).xyz;
```

3.2- I. Difuso = Luz Difusa * Material difuso * cos(θ)

```
vec3 diffuse= light.diffuse.xyz * material.diffuse.xyz * max(cosTheta, 0.0);
```

3.3- I. Especular= (R·Eye) * Brillo * Material Especular

```
vec3 specular= material.specular.xyz * light.specular.xyz * pow(max(cosAlpha, 0.0f),material.shininess);
```

4-Valor de color final

```
fragColor= vec4((ambient+ diffuse+ specular), 1.0);
}
```

Para añadir Texturas:

```
textureColor= texture(sampler, texCoord)
fragColor= textureColor* ( ambientLight+ diffuseLight+ specularLight)
```

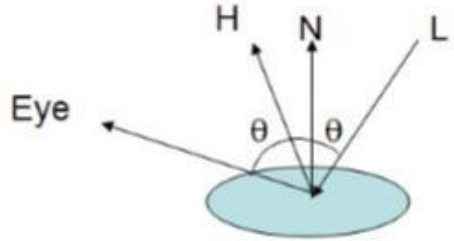
Blinn-Phong Shading: Paso a paso (Vertex shader)

1–Añadimos variable de salida del Vector H:

```
out vec3 varyingHalfVector;
```

2-Calculamos su valor:

```
voidmain(void)
{
        ...
        varyingHalfVector = (varyingLightDir + (-varyingVertPos)).xyz;
        ...
}
```



Blinn-Phong Shading: Paso a paso (Fragment shader)

1–Añadimos variable de entrada del Vector H:

```
in vec3 varyingHalfVector;
void main(void)
{
```

2-Normalizamos Vectores (ya no calculamos R):

```
vec3 L     = normalize(varyingLightDir);
vec3 N     = normalize(varyingNormal);
vec3 Eye= normalize(-varyingVertPos);
vec3 H     =normalize(varyingHalfVector);
```

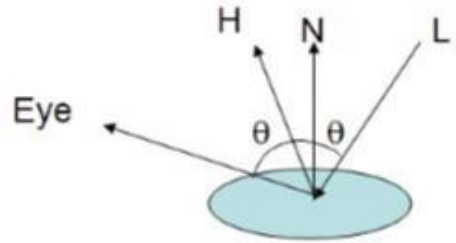
…

Calculamos nuevo Alpha:

```
float cosAlpha= dot(H,N);
```

3- I. Especular= (R·Eye) * Brillo* Material Especular

```
vec3 specular= material.specular.xyz * light.specular.xyz * pow(max(cosAlpha, 0.0f),material.shininess);
```



¿Dudas?