Gráficos y Multimedia

Sesión 2: Iluminación y Shaders.

0. Modelos de Iluminación

INDICE

- Definición
- Fuentes de luz
- Modelo de iluminación de Phong
  - Definición
  - Tipos de luz: ambiental, difusa y especular
  - Combinación de luces
  - Atenuación de la intensidad
  - Transparencias
  - Fuentes de luz múltiples
  - Luces dirigidas: modelo de Warn
- Escala de color
  - Imágenes en niveles de gris
  - Imágenes en color

0. Modelos de Iluminación

DEFINICIÓN

- Entendemos por modelo de iluminación el cálculo de la intensidad de cada punto de la escena

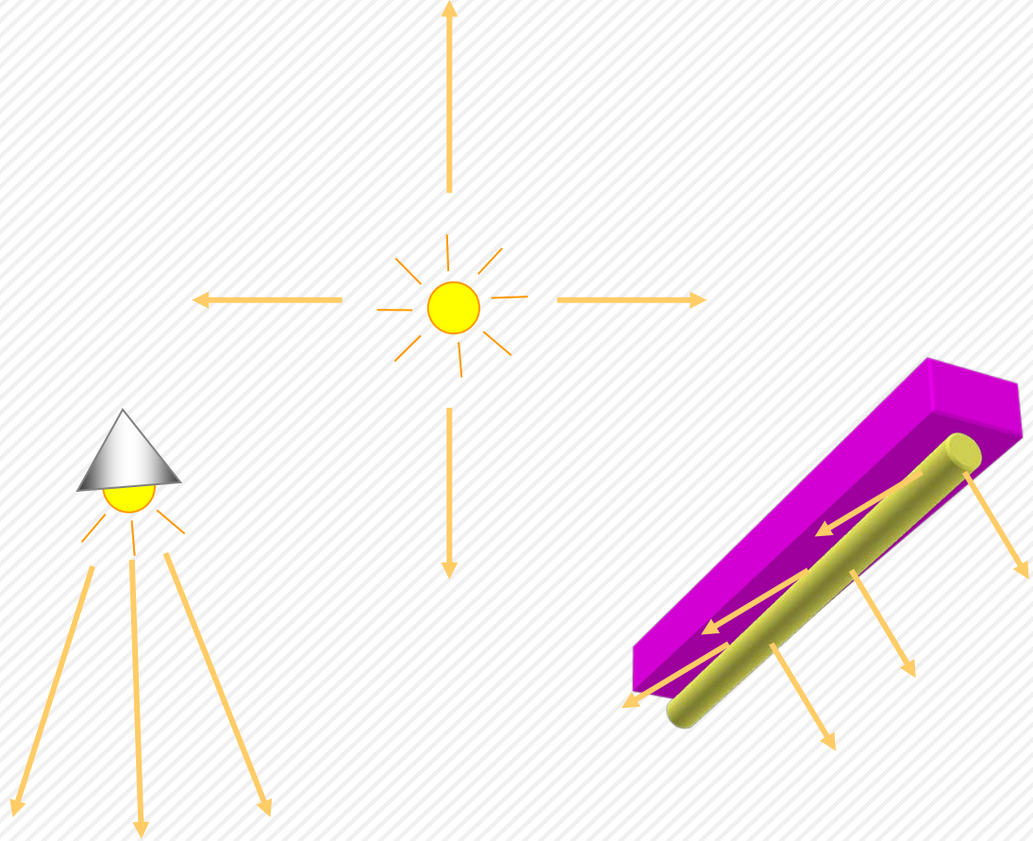
- En el cálculo de la intensidad de un punto intervienen:
  - El tipo e intensidad de la fuente de luz
  - El material del objeto
  - La orientación del objeto con respecto a la luz

- El modelo más utilizado es el modelo de ***Phong***

1. Fuentes de Luz

TIPOS

- Fuentes puntuales o puntos de luz
  - No tienen dimensión
  - No tienen dirección (emiten de forma radial)
  - Consideramos que una luz es puntual si su dimensión es muy pequeña comparada con la de los objetos de la escena
  - Ejemplos: sol, bombillas...
- Fuente de luz distribuidas
  - Tienen dimensión
  - Tienen dirección
  - Ejemplos: focos, tubos de luz...



2. Modelo de Iluminación de Phong

DEFINICIÓN

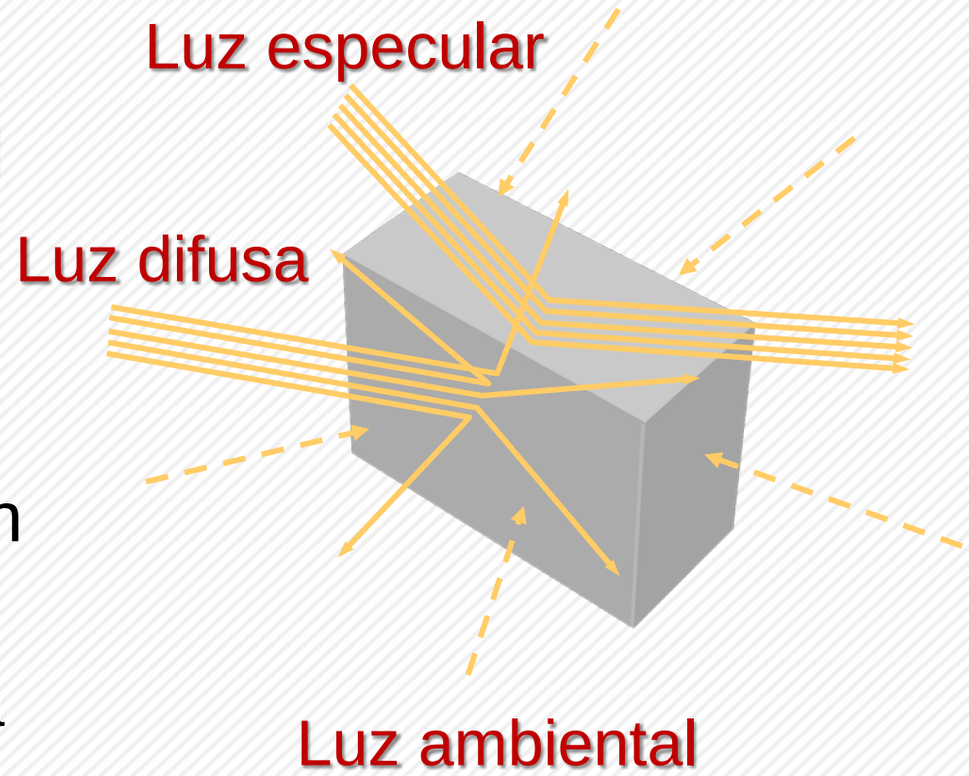
- Es un modelo empírico simplificado para iluminar puntos de una escena
- Los resultados son muy buenos en la mayoría de las escenas
- En este modelo, los objetos no emiten luz, sólo reflejan la luz que les llega de las fuentes de luz o reflejada de otros objetos

2. Modelo de Iluminación de Phong

DEFINICIÓN

- La luz que llega a un objeto puede ser de tres tipos
  - Luz ambiental: proviene de todas las direcciones e ilumina todas las caras del objeto por igual
  - Luz difusa: proviene de una dirección pero se refleja en todas direcciones
  - Luz especular: proviene de una dirección y se refleja sólo en una dirección
- La iluminación de un punto se calcula como la suma de los tres tipos de iluminaciones:

$$I_{Phong} = I_{amb} + I_{dif} + I_{esp}$$



2. Modelo de Iluminación de Phong

LUZ AMBIENTAL

- No proviene de una dirección concreta → incide sobre todas las partes del objeto
- Simula el proceso de reflexión de la luz sobre los demás objetos de la escena
- Se suele modelar como una constante → evita que las zonas sin luz directa se visualicen totalmente en negro.
- La iluminación ambiental depende de
  - Ia: constante de intensidad de la iluminación ambiental
  - ka: coeficiente empírico que depende de las propiedades ópticas del material del objeto ($0 \leq ka \leq 1$)

$$I_{ambiental} = I_a k_a$$

2. Modelo de Iluminación de Phong

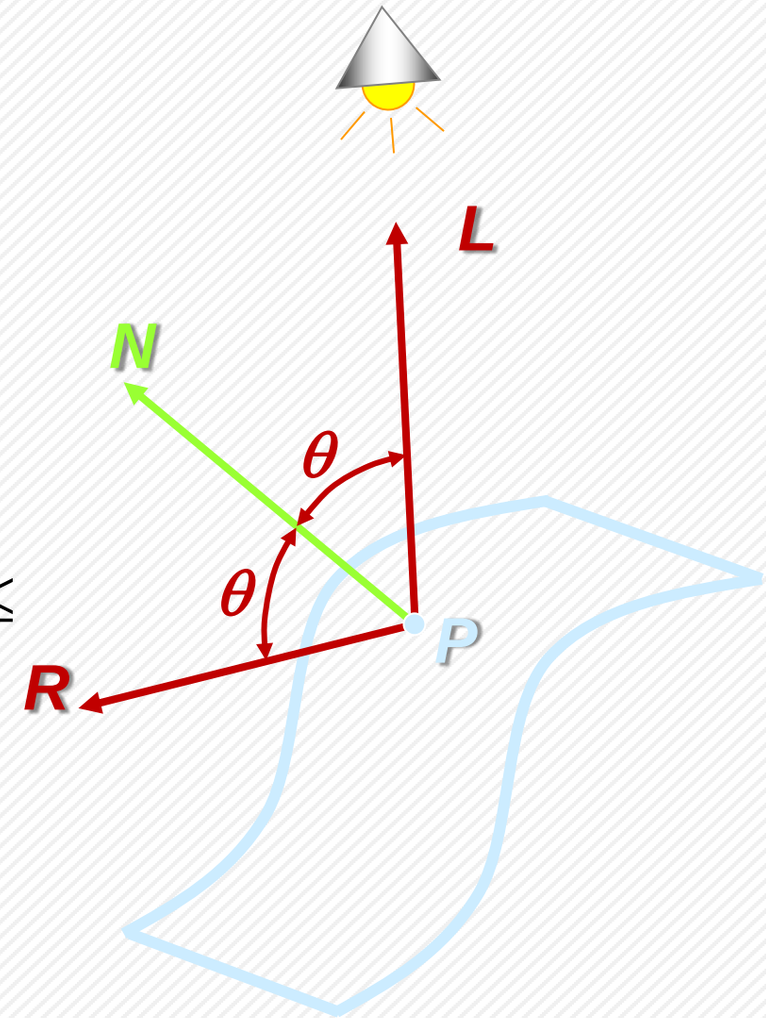
LUZ DIFUSA

- Proviene de una dirección concreta pero se refleja en todas direcciones
- Sólo afecta a las partes del objeto en las que la luz incide → se mostrarán más brillantes que las demás
- La iluminación depende del ángulo entre la superficie y la dirección de incidencia de la luz → iluminación máxima cuando la superficie y la fuente de luz son perpendiculares

2. Modelo de Iluminación de Phong

LUZ DIFUSA

- La iluminación difusa depende de:
  - ***N***: normal de la superficie en el punto P
  - ***L***: vector de incidencia de la luz
  - I_l: intensidad de la fuente de luz
  - k_d: coeficiente empírico de reflexión que depende de la longitud de onda de la luz ($0 \leq k_d \leq 1$)
- El vector de reflexión ***R*** forma con la normal ***N***, un ángulo θ equivalente al que forman ***N*** y ***L***. *Ley de Lambert*



$$I_{difusa} = I_l k_d \cos\theta = I_l k_d (L \cdot N) \qquad 0 \leq \theta \leq 2\pi$$

2. Modelo de Iluminación de Phong

LUZ ESPECULAR

- Procede de una dirección concreta y se refleja en una única dirección → produce brillos intensos (por ejemplo, objetos metálicos)

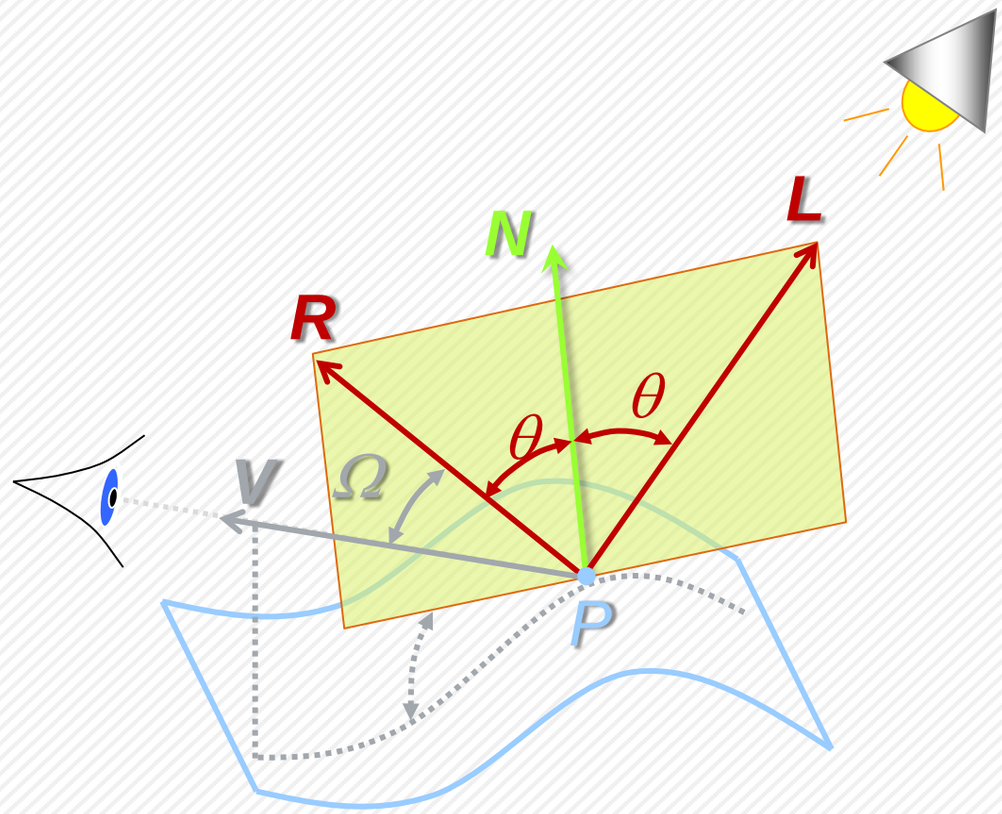
- Como la difusa, sólo afecta a las partes del objeto en las que la luz incide directamente

- La iluminación depende del ángulo entre la dirección de incidencia de la luz y la posición del observador

2. Modelo de Iluminación de Phong

LUZ ESPECULAR

- ***V**:* vector de posición del observador
- ***L**:* vector de incidencia de la luz
- ***R**:* vector reflejado de la luz
- I_l*:* intensidad de la fuente de luz
- k_e*:* coeficiente empírico de reflexión especular ($0 \leq k_e \leq 1$)
- ***n**:* un índice que simula la rugosidad de la superficie ($1 \leq n < \infty$, 1: mate, ∞: espejo)
- Ω*:* ángulo entre ***V*** y ***R***
- ***R=2N(L·N)-L***



$$I_{especular} = I_l \, k_e \cos^n \Omega = I_l \, k_e \, (\mathbf{R} \cdot \mathbf{V})^n$$

2. Modelo de Iluminación de Phong

LUZ ESPECULAR

- En la expresión anterior, los brillos (iluminación especular) sólo dependen del ángulo Ω entre el observador y el vector de incidencia de la luz

- En algunos materiales, los brillos sólo se producen cuando la luz incide con un ángulo θ cercano a los 90º con respecto a la normal de la superficie (por ejemplo, el cristal)

- Para modelar esto, podemos sustituir la constante k_e por una función W del ángulo θ

$$I_{especular} = I_l\, W(\theta) \cos^n \Omega = I_l\, W(\theta)\, (\mathbf{R}\cdot\mathbf{V})^n$$

2. Modelo de Iluminación de Phong

COMBINACIÓN DE ILUMINACIONES

- **Modelo de Phong**: la iluminación en un punto es una combinación lineal de los tres tipos de iluminación

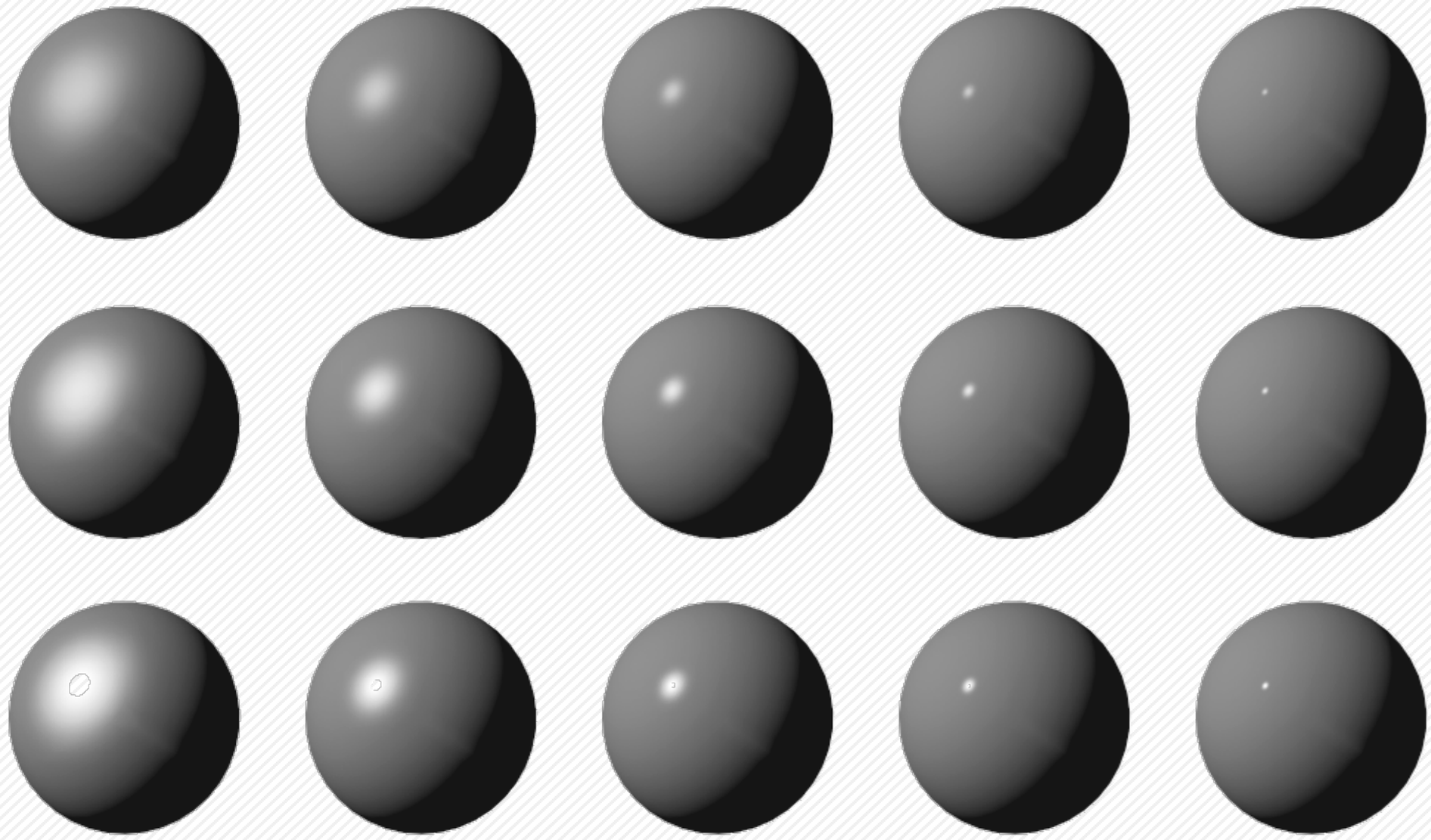
$$I_{Phong} = I_{ambiental} + I_{difusa} + I_{especular} = I_a k_a + I_l(k_d(\mathbf{L}\cdot\mathbf{N}) + k_e(\mathbf{R}\cdot\mathbf{V})^n)$$

- Los coeficientes k_a, k_d y k_e modulan la influencia de cada tipo de luz
- Se suele utilizar una aproximación que simplifica los cálculos:
  - $R\cdot V \cong N\cdot H$
  - H = (L+V)/2 (H tiende a ser constante cuando las luces y el observador están lejos del objeto)
  - Se puede utilizar el parámetro n para mejorar la aproximación

$$I_{Phong} = I_a k_a + I_l(k_d(\mathbf{L}\cdot\mathbf{N}) + k_e(\mathbf{N}\cdot\mathbf{H})^n)$$

2. Modelo de Iluminación de Phong

EJEMPLO



Para todas las esferas k_a =0.1 y k_d =0.45

2. Modelo de Iluminación de Phong

ATENUACIÓN DE LA INTENSIDAD

- Cuando la luz viaja por el espacio se va atenuando según una función de la distancia d entre el foco de luz y el objeto

$$f(d) = 1/d^2$$

- Una mejor aproximación empírica utiliza la siguiente función de atenuación, donde los coeficientes a0, a1 y a2 pueden manipularse para conseguir diferentes efectos de iluminación

$$f(d) = \frac{1}{a_0 + a_1 d + a_2 d^2}$$

- La iluminación de Phong queda como sigue:

$$I_{Phong} = I_a k_a + f(d)\, I_l (k_d (\mathbf{L} \cdot \mathbf{N}) + k_e (\mathbf{N} \cdot \mathbf{H})^n)$$

2. Modelo de Iluminación de Phong

TRANSPARENCIAS

- Podemos considerar una nueva componente asociada a la luz que se ve a través de un objeto transparente → **luz transmitida**

$$I_{transm} = k_t(N \cdot H')^n$$

- H' se calcula a partir de L, V y dos coeficientes de refracción (eta) $\eta 1$ y $\eta 2$, dependientes de las características del material

$$\mathbf{H}' = \frac{-\mathbf{L} - (\eta_2/\eta_1)\mathbf{V}}{\eta_2/\eta_1 - 1}$$

- El modelo de Phong queda como sigue:

$$I_{Phong} = I_a k_a + f(d)\, I_l (k_d(\mathbf{L} \cdot \mathbf{N}) + k_e(\mathbf{N} \cdot \mathbf{H})^{n_e} + k_t(\mathbf{N} \cdot \mathbf{H}')^{n_t})$$

2. Modelo de Iluminación de Phong

MÚLTIPLES FUENTES DE LUZ

- Sea un conjunto de fuentes de luz en una escena, cada una con una intensidad I_{li} y emitiendo en la dirección L_i

- El modelo de Phong queda como sigue:

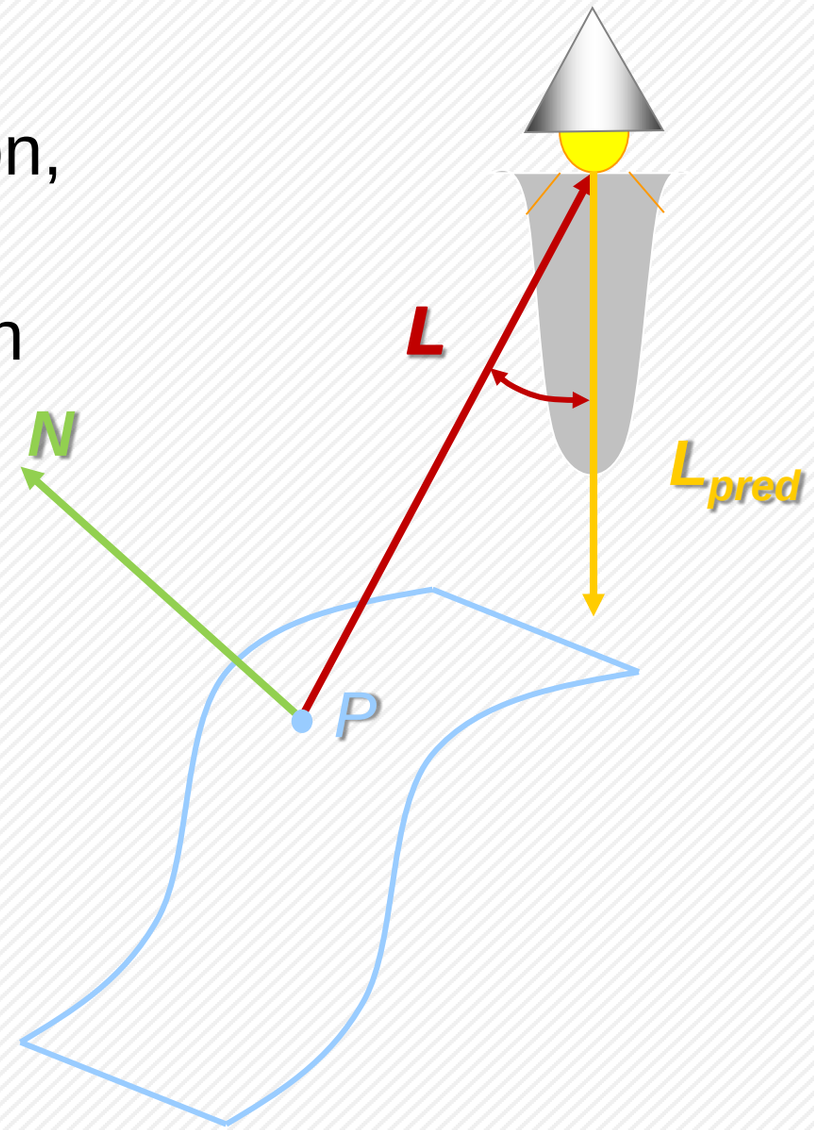
$$I_{Phong} = I_a k_a + \sum_i f(d_i) I_{li} (k_d (\mathbf{L}_i \cdot \mathbf{N}) + k_e (\mathbf{N} \cdot \mathbf{H}_i)^{n_e} + k_t (\mathbf{N} \cdot \mathbf{H'}_i)^{n_t})$$

2. Modelo de Iluminación de Phong

LUZ DIRIGIDA: MODELO DE WARN

- Técnica para simular fuentes de luz dirigidas
- Cada fuente de luz emite en una dirección predominante $\boldsymbol{L_{pred}}$. Al separarnos de esa dirección, la luz se debilita
- Definimos la iluminación según el modelo de Warn en función de:
  - I_l: intensidad del punto de luz
  - $\boldsymbol{L}$: posición de la luz
  - $\boldsymbol{L_{pred}}$: dirección predominante de iluminación
  - s: coeficiente de concentración de la luz (valores altos indican luz más concentrada)
- En el modelo de Phong, se sustituye I_l por I_w

$$I_w = I_l (\boldsymbol{L_{pred}} \cdot \boldsymbol{L})^s$$



4. Visualización de Polígonos

INDICE

- Definición
- Visualización de objetos poliédricos
    - Sombreado plano o de intensidad constante
- Visualización de aproximación de objetos curvos
    1. Sombreado de Gouraud
    2. Sombreado de Phong
    3. Mejoras del sombreado de Phong

4. Visualización de Polígonos

DEFINICIÓN

- El sombreado o visualización de polígonos es la asignación de intensidades a cada punto de los polígonos que forman un objeto
- Un objeto formado por polígonos puede representar:
  - Un objeto poliédrico: al ser una representación exacta interesa marcar las aristas entre los polígonos
  - Una aproximación de un objeto curvo: interesa eliminar en los posible las aristas entre caras
- Veremos técnicas para representar estos dos tipos de objetos

4. Visualización de Poliedros

SOMBREADO PLANO

- Se calcula un único valor de intensidad para cada polígono mediante el modelo de iluminación de *Phong*, y se asigna a todos sus puntos.
- Es exacto cuando:
  - El objeto es un poliedro.
  - Las fuentes de luz se encuentran alejadas del objeto (en esos casos N·L y la función de atenuación se pueden considerar constantes).
  - El observador está lejos del objeto (V·R se puede considerar constante).
- Inconvenientes del sombreado plano
  - Si el objeto es una aproximación poliédrica de un objeto curvo, se visualizan las aristas entre polígonos.
  - No pueden representarse brillos interiores a los polígonos, pues todos sus puntos tienen intensidad constante.
  - Para obtener una buena representación de objetos aproximados, son necesarios muchos polígonos de pequeño tamaño.

4. Objetos Aproximados

SOMBREADO DE GOURAUD

- Es un método incremental que realiza una interpolación de intensidades.

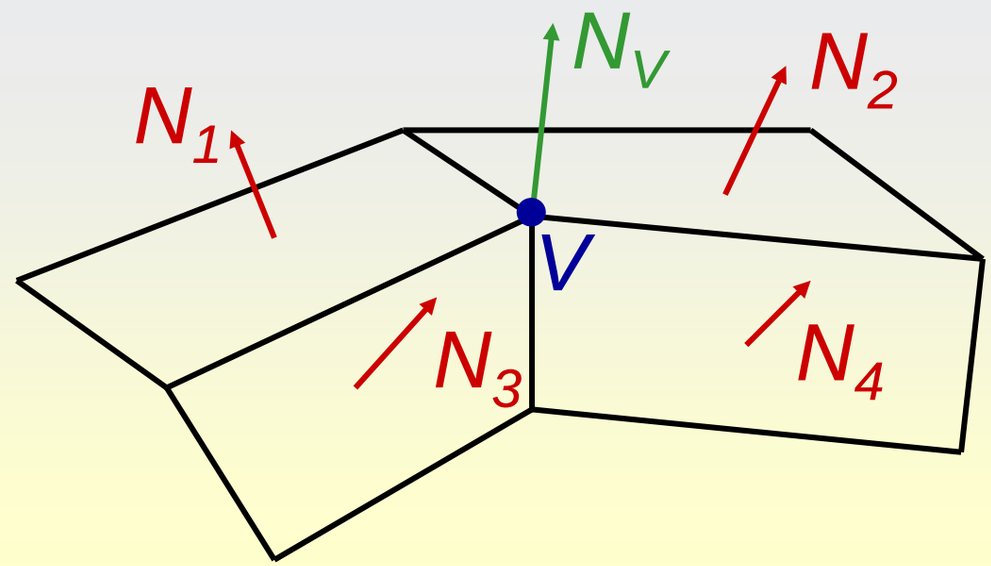
- En cada vértice del polígono se calcula la intensidad. La intensidad de los puntos intermedios se calcula por interpolación lineal.

4. Objetos Aproximados

SOMBREADO DE GOURAUD

- 1. Calcular intensidad en los vértices
  - 1.1. Calcular normal en el vértice: para evitar visualizar las aristas, la normal se calcula como la media de las normales de polígonos adyacentes:

$$N_V = \frac{\sum_{k=1}^{n} N_k}{\left| \sum_{k=1}^{n} N_k \right|}$$



  - 1.2. Calcular la intensidad del vértice V según el modelo de iluminación de Phong

4. Objetos Aproximados

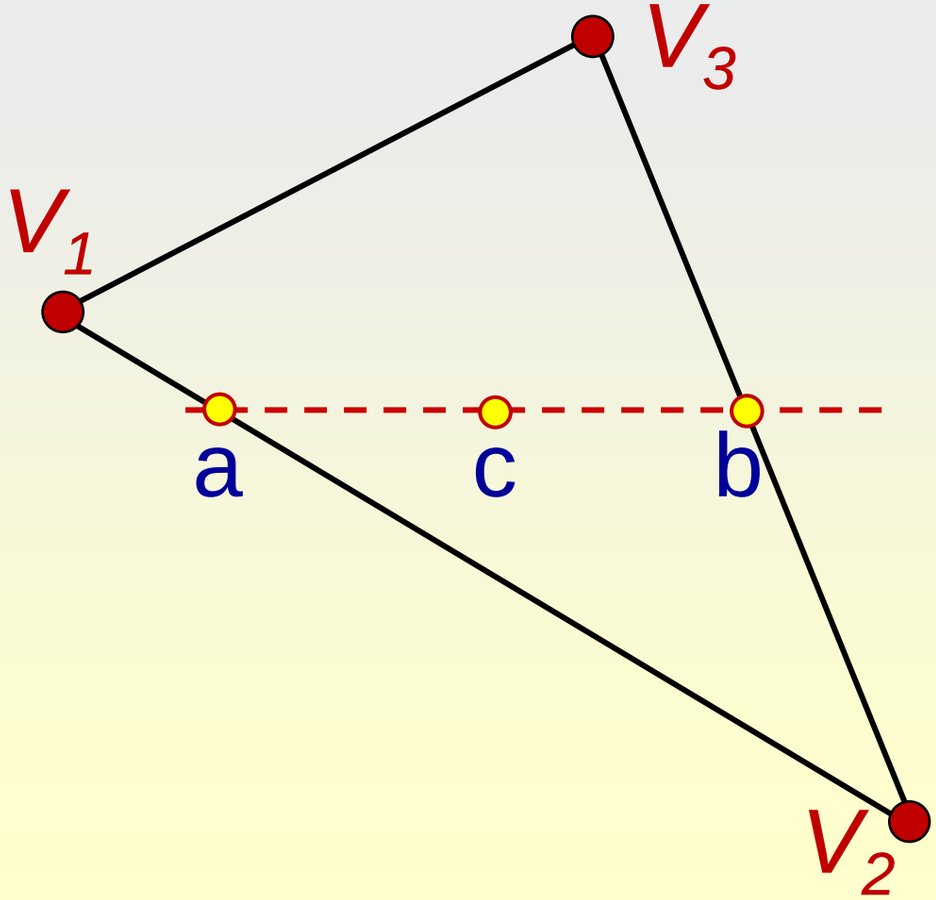
SOMBREADO DE GOURAUD

- 2. Calcular la intensidad de los puntos interiores por interpolación lineal de las de los vértices:

$$I_a = \frac{y_a - y_{V_2}}{y_{V_1} - y_{V_2}} I_{V_1} + \frac{y_{V_1} - y_a}{y_{V_1} - y_{V_2}} I_{V_2}$$

$$I_b = \frac{y_b - y_{V_2}}{y_{V_3} - y_{V_2}} I_{V_3} + \frac{y_{V_3} - y_b}{y_{V_3} - y_{V_2}} I_{V_2}$$

$$I_c = \frac{x_b - x_c}{x_b - x_a} I_a + \frac{x_c - x_a}{x_b - x_a} I_b$$



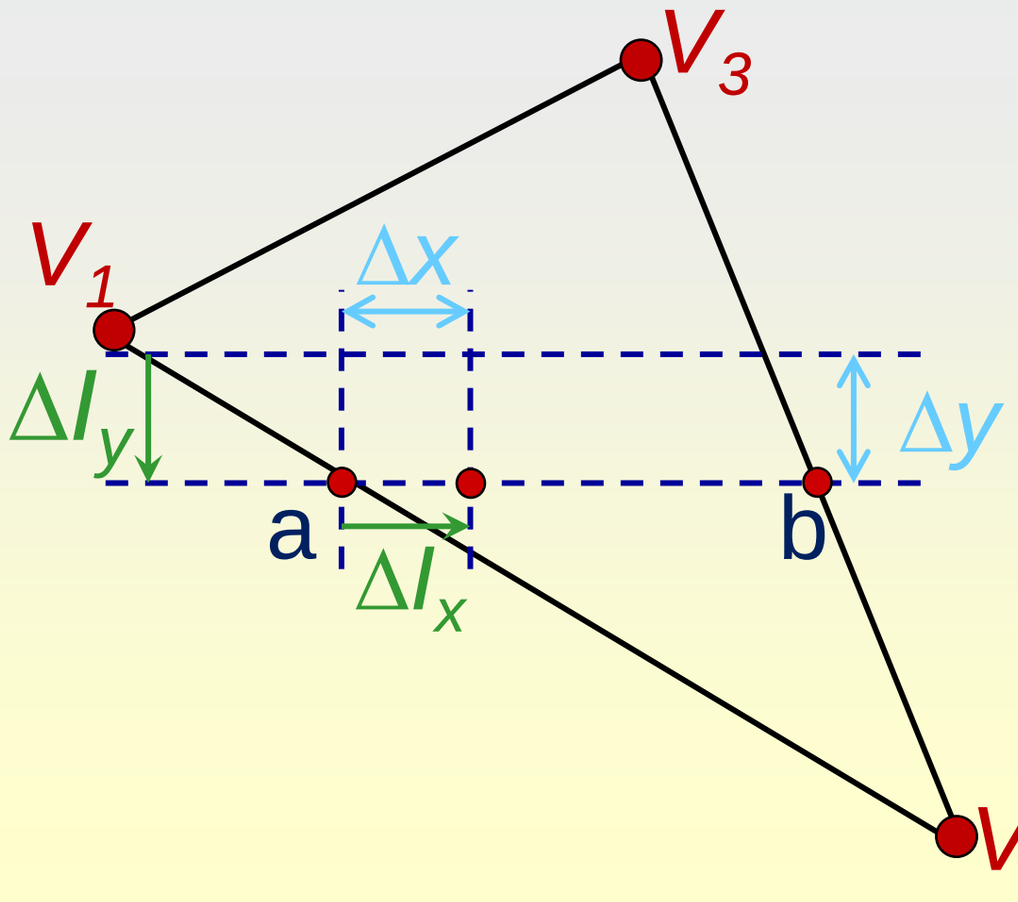
4. Objetos Aproximados

SOMBREADO DE GOURAUD

- Las intensidades pueden calcularse de forma incremental, rellenando el polígono por líneas de rastreo:

$$\Delta I_y = \Delta y \frac{I_{V_2} - I_{V_1}}{y_{V_2} - y_{V_1}}$$

$$\Delta I_x = \Delta x \frac{I_b - I_a}{x_b - x_b}$$



4. Objetos Aproximados

SOMBREADO DE GOURAUD

- Ventajas:
  - Rápido
  - Elimina las aristas: mejora la visualización de las aproximaciones poliédricas de objetos curvos
- Inconvenientes:
  - Bandas de Mach
  - No representa bien los brillos especulares, por lo que se suele utilizar sólo para reflexión difusa

4. Objetos Aproximados

SOMBREADO DE PHONG

- Es un método incremental que realiza una interpolación de normales (en vez de interpolación de intensidades).

- En cada vértice del polígono se calcula la normal como media de las normales de los polígonos adyacentes. La normal de los puntos intermedios se calcula por interpolación lineal.

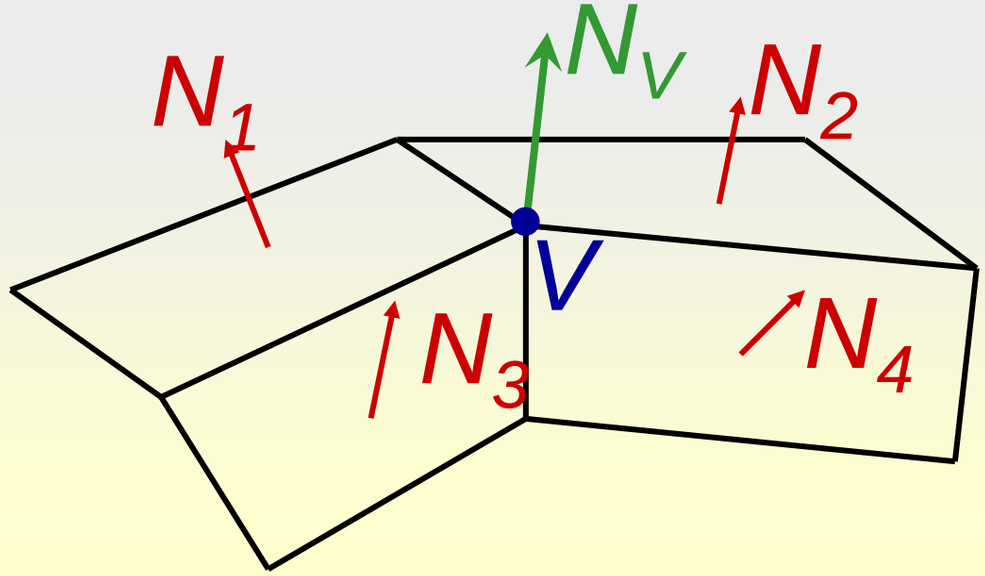
- En cada punto se aplica el modelo de iluminación de Phong

4. Objetos Aproximados

SOMBREADO DE PHONG

- 1. Calcular normal en los vértices
  - Como en el sombreado de Gouraud, para evitar visualizar las aristas, la normal se calcula como la media de las normales de los polígonos adyacentes:

$$N_V = \frac{\sum_{k=1}^{n} N_k}{\left| \sum_{k=1}^{n} N_k \right|}$$



4. Objetos Aproximados

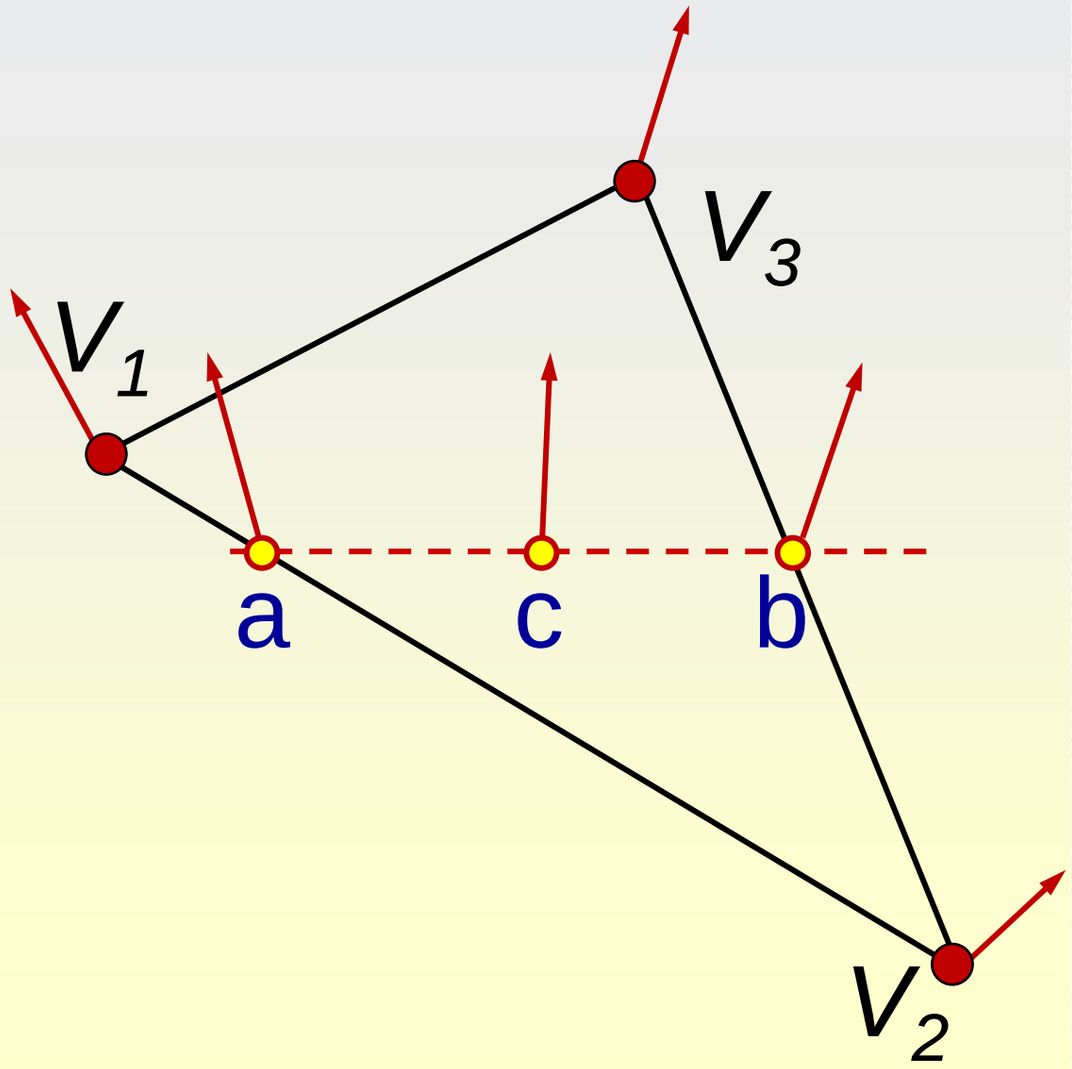
SOMBREADO DE PHONG

- 2. Calcular la normal de los puntos interiores por interpolación lineal de las de los vértices:

$$N_a = \frac{y_a - y_{V_2}}{y_{V_1} - y_{V_2}} N_{V_1} + \frac{y_{V_1} - y_a}{y_{V_1} - y_{V_2}} N_{V_2}$$

$$N_b = \frac{y_b - y_{V_2}}{y_{V_3} - y_{V_2}} N_{V_3} + \frac{y_{V_3} - y_b}{y_{V_3} - y_{V_2}} N_{V_2}$$

$$N_c = \frac{x_b - x_c}{x_b - x_a} N_a + \frac{x_c - x_a}{x_b - x_a} N_b$$



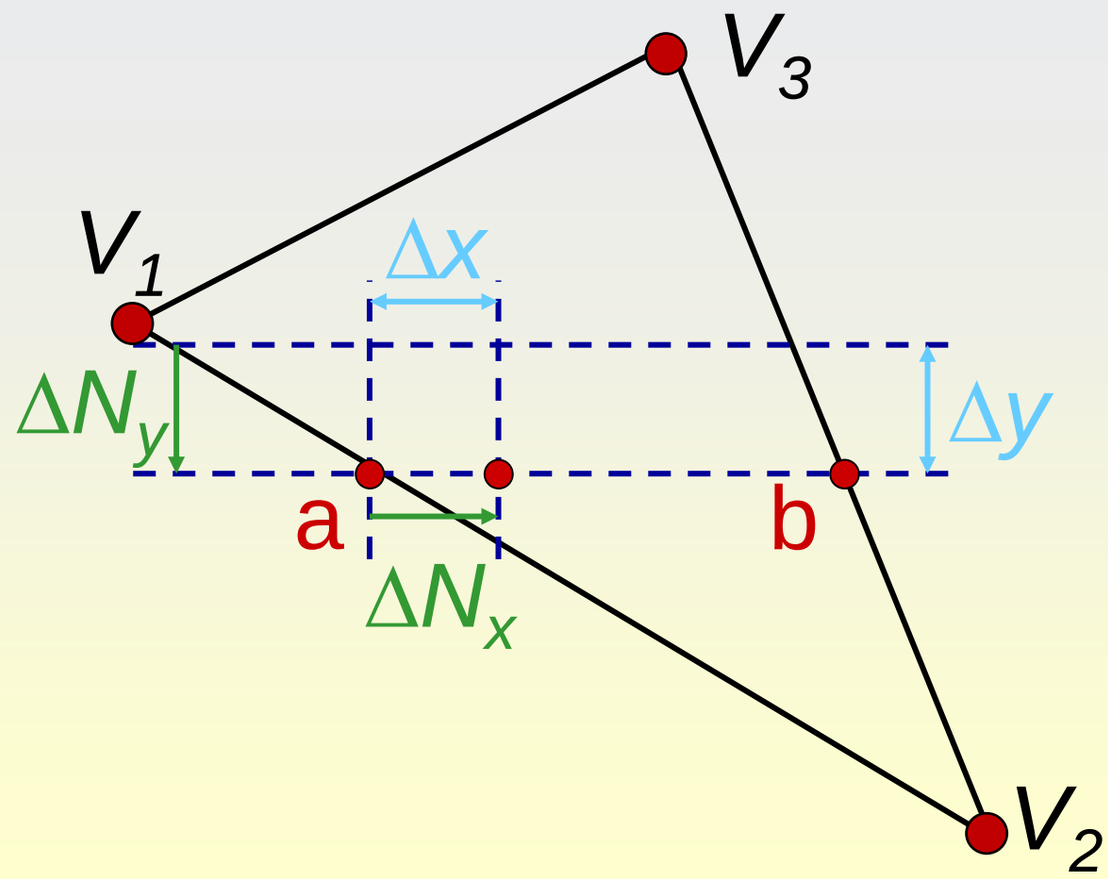
4. Objetos Aproximados

SOMBREADO DE PHONG

- Las normales pueden calcularse de forma incremental, rellenando el polígono por líneas de rastreo:

$$\Delta N_y = \Delta y \frac{N_{V_2} - N_{V_1}}{y_{V_2} - y_{V_1}}$$

$$\Delta N_x = \Delta x \frac{N_b - N_a}{x_b - x_b}$$



4. Objetos Aproximados

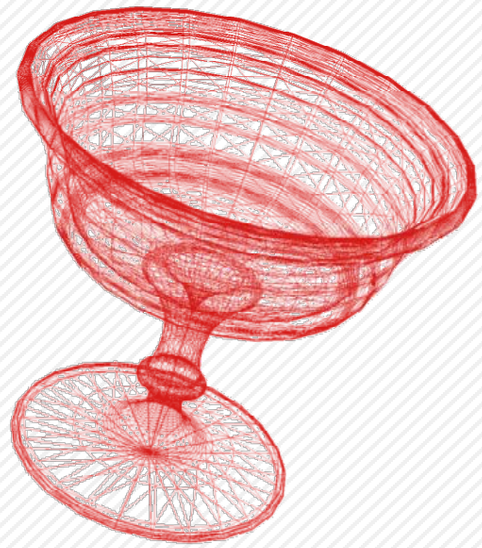
SOMBREADO DE PHONG

- 3. Calculo de la intensidad:
  - Calcularla en cada punto mediante el modelo de iluminación de Phong.
  - Ahora se puede introducir también la reflexión especular

- Ventajas:
  - Obtiene mejores resultados que el sombreado de Gouraud
  - Elimina casi completamente las bandas de Mach
  - Representa bien los brillos especulares

- Inconvenientes:
  - Tiene mucho mayor coste que el de Gouraud:
    - Ahora se interpolan tres componentes en vez de una
    - Para cada punto es necesario aplicar el modelo de iluminación

4. Objetos Aproximados



EJEMPLOS DE SOMBREADOS

Flat

Gouraud

Phong

4. Objetos Aproximados

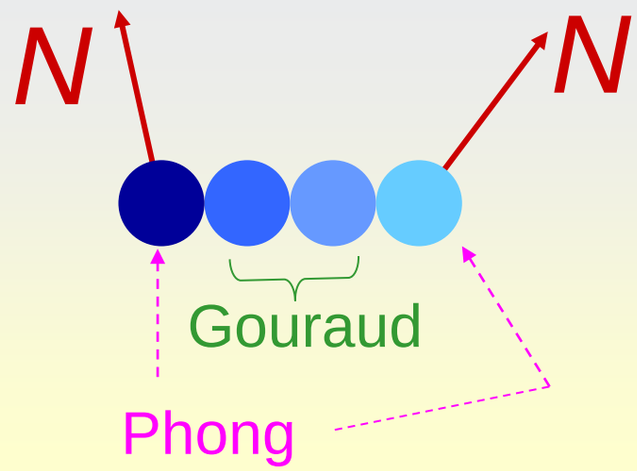
Mejoras del sombreado de Phong

- Combinación de los sombreados de Gouraud y Phong:
  - Reducción del número de normales
  - Detección de brillos
- Sombreado de Phong rápido

4. Objetos Aproximados

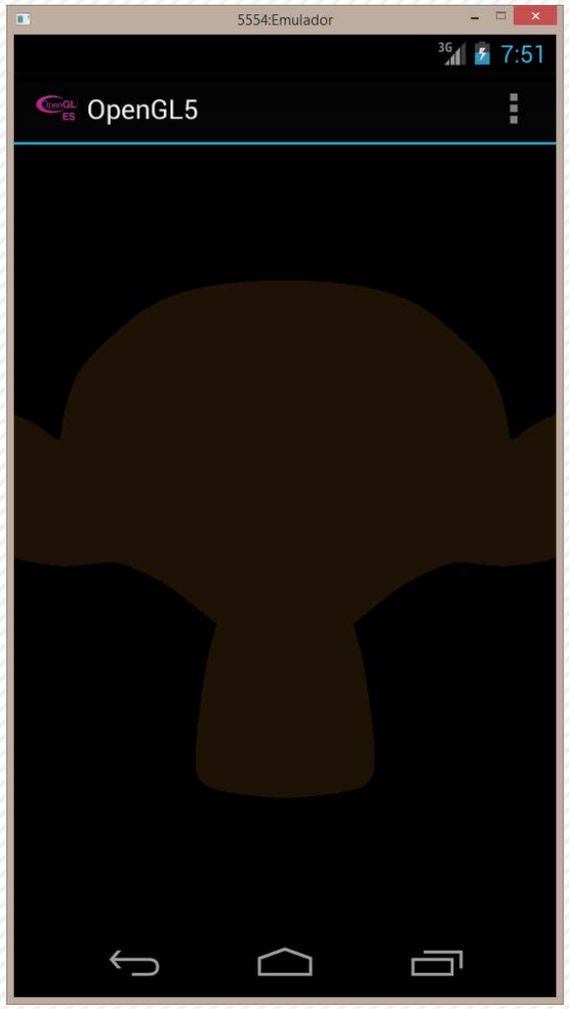
Reducción de número de normales

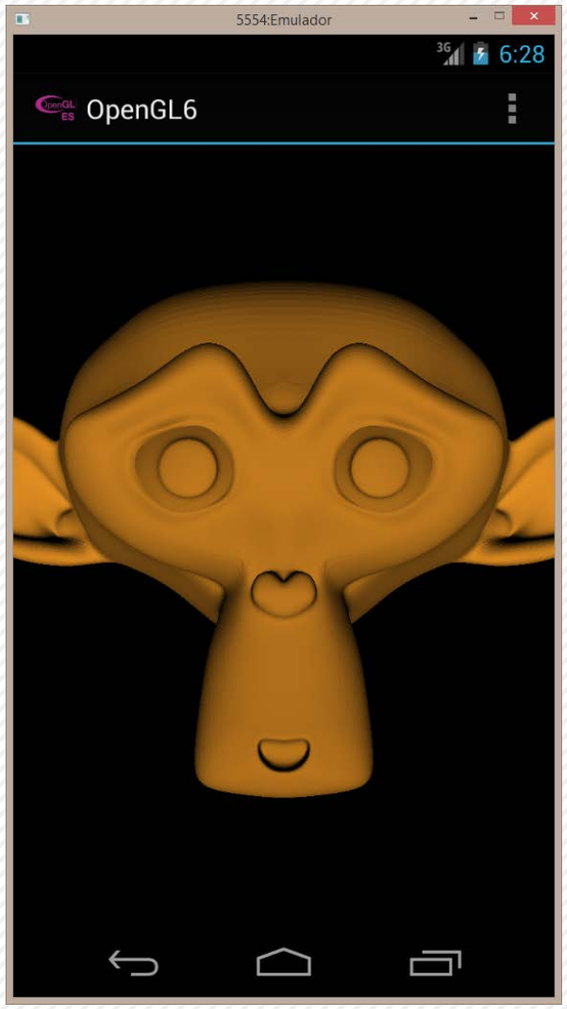
- Consiste en calcular normales en píxeles alternos
- La intensidad en esos puntos se calcula utilizando la media de los adyacentes
- También pueden saltarse más de un píxel y utilizar interpolación lineal
- No existe pérdida de calidad apreciable

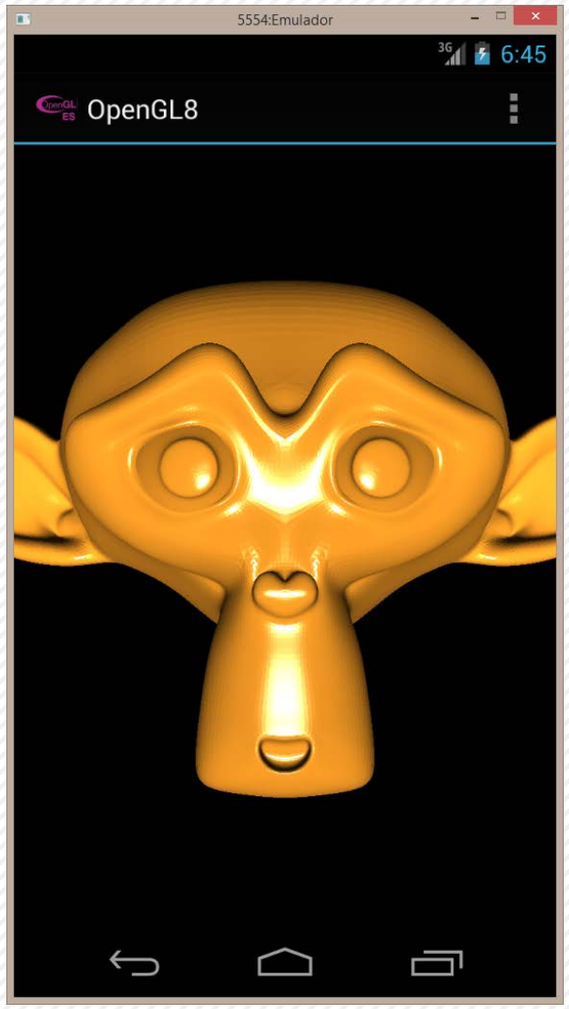


5. Iluminación con Shaders

Suma de los tres tipos de iluminación:







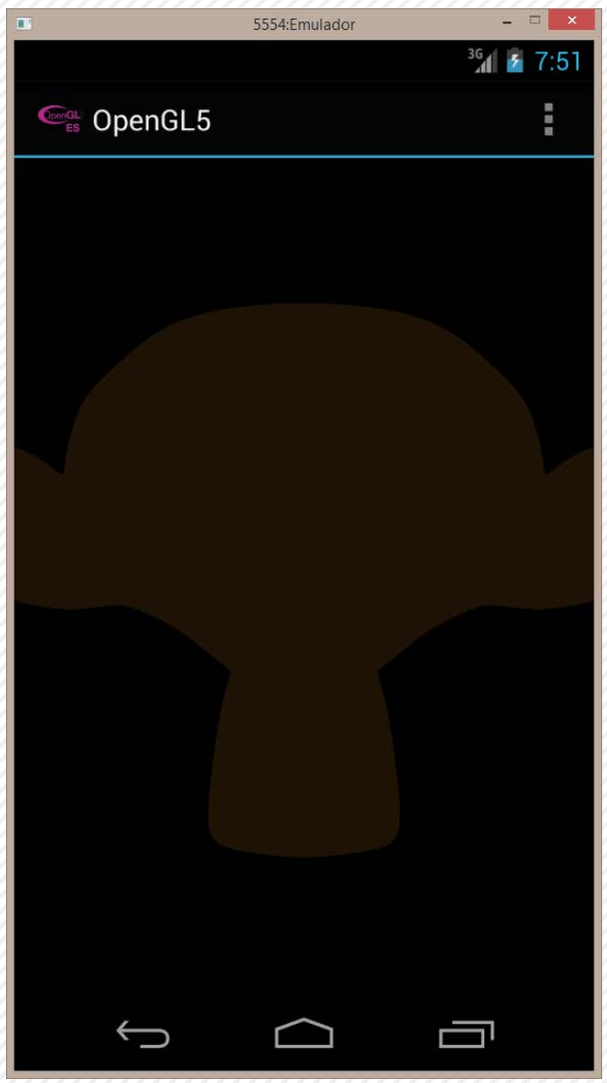
Ambiente + Difusa + Especular = Phong

5. Iluminación con Shaders

AMBIENTE

- La iluminación **ambiente** se consigue sumando básicamente un vector con el color ambiente al valor final de iluminación.
- En el *fragment shader*:

```
precision mediump float;  // Precisión media

varying vec4 v_Color;     // in: color del vertex shader

void main()
{
    gl_FragColor = v_Color;
}
```



- Este fragment shader contiene un `varying` con el color que calculará el vertex shader.

5. Iluminación con Shaders

AMBIENTE

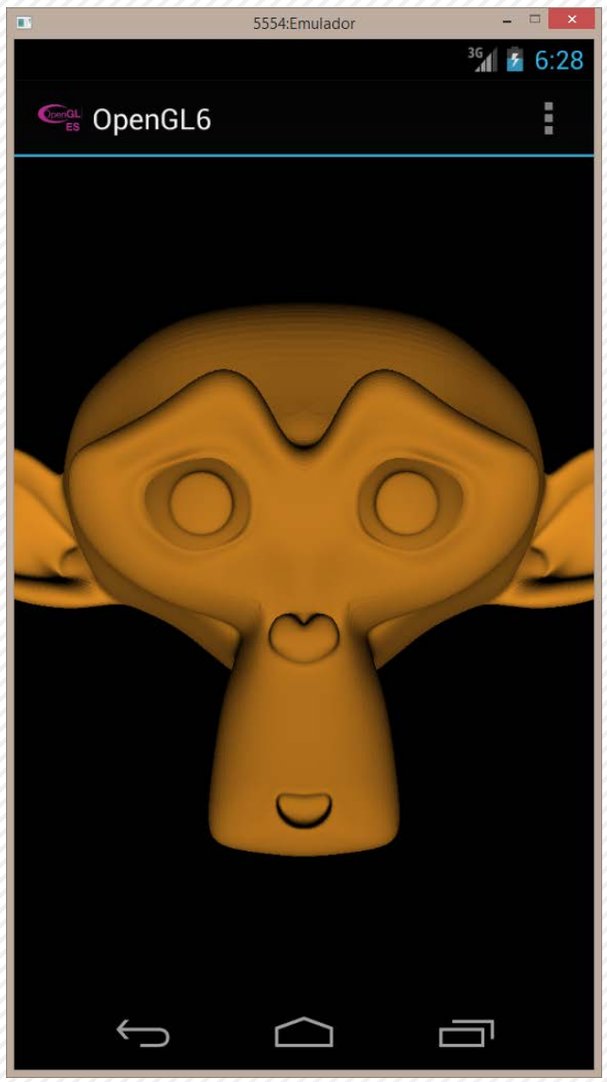
- Para el cálculo del valor ambiente se suele establecer un valor entre el 10% y el 20%:

```
uniform mat4 u_MVPMatrix;          // in: Matriz Projection*ModelView
uniform vec4 u_Color;              // in: Color del objeto
attribute vec4 a_Position;         // in: Posición del vértice

varying vec4 v_Color;              // out: color calculado

void main()
{
    float ambient = 0.15;          // 15% de intensidad ambiente
    v_Color = u_color * ambient;

    gl_Position = u_MVPMatrix * a_Position;
}
```

5. Iluminación con Shaders

DIFUSA

- La iluminación **difusa** se consigue empleando el mismo *fragment shader*, pero calculando la intensidad difusa como se vió en la transparencia 9 en el vertex shader, las variables necesarias son:




```
uniform mat4 u_MVPMatrix; // in: Matriz Projection*ModelView
uniform mat4 u_MVMatrix;  // in: Matriz ModelView
uniform vec4 u_Color;     // in: color del objeto

attribute vec4 a_Position;// in: Posición de cada vértice
attribute vec3 a_Normal;  // in: Normal de cada vértice

varying vec4 v_Color;     // out: Color de salida al fragment
                          //      shader
```

5. Iluminación con Shaders

DIFUSA

- Código principal:

```
void main()
{
    vec3 LightPos = vec3(0, 1, -7);                  // Posición de la luz [fija]
    vec3 P = vec3(u_MVMatrix * a_Position);          // Posición del vértice
    vec3 N = vec3(u_MVMatrix * vec4(a_Normal, 0.0)); // Normal del vértice

    float d = length(LightPos - P);                  // distancia
    vec3  L = normalize(LightPos - P);               // Vector Luz
    float diffuse = max(dot(N, L), 0.15);            // Cálculo de la int. difusa
                                                     // (15% de intensidad ambiente)
    float attenuation = 1.0/(0.3+(0.1*d)+(0.01*d*d)); // Cálculo de la atenuación
    diffuse = diffuse*attenuation;

    v_Color = u_Color * diffuse;

    gl_Position = u_MVPMatrix * a_Position;
}
```

5. Iluminación con Shaders

ESPECULAR

- La iluminación **especular** se consigue empleando el mismo *fragment shader*, pero calculando la misma como se vió en la transparencia 11 a 13, podemos emplear N·H en vez de R·V, al ser H=(V+L)/2, un cálculo mucho más sencillo y rápido.
- Todos los vectores de los productos escalares deben estar normalizados, por ello no es necesario dividir entre 2.
- Las variables que vamos a emplear son las mismas que en el *shader* difuso.

5. Iluminación con Shaders

ESPECULAR (per-vertex)

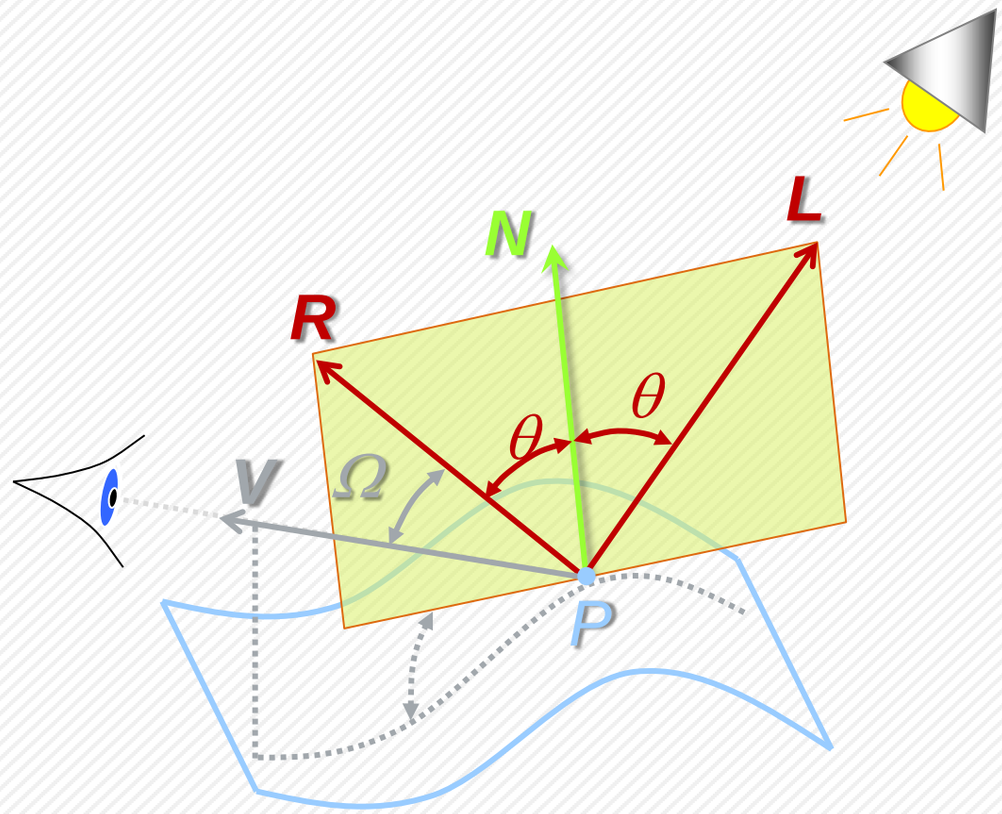
- Código principal:

```
void main()
{
    vec3 LightPos = vec3(-2, 0, -6);                    // Posición de la luz [fija]
    vec3 P = vec3(u_MVMatrix * a_Position);             // Posición del vértice
    vec3 N = vec3(u_MVMatrix * vec4(a_Normal, 0.0));    // Normal del vértice
    vec4 specularColor = vec4(1, 1, 1, 1);              // Color especular (brillos blancos)

    float d = length(LightPos - P);                     // distancia
    vec3  L = normalize(LightPos - P);                  // Vector Luz
    vec3  V = normalize(P-vec3(0,0,0));                 // Vector Visión
    vec3  H = normalize(V+L);                           // Vector Half H=(V+L)/2

    float attenuation = 1.0/(0.3+(0.1*d)+(0.01*d*d));   // Cálculo de la atenuación
    float ambient  = 0.15;                              // 15% de intensidad ambiente
    float diffuse  = max(dot(N, L), 0.0);               // Cálculo de la intensidad difusa
    float specular = pow(max(dot(N, H), 0.0), 200.0);   // Exponente de Phong (200)

    v_Color = u_Color*ambient+attenuation*(u_Color*diffuse + specularColor*specular);
    gl_Position = u_MVPMatrix * a_Position;
}
```

5. Iluminación con Shaders. Phong

LUZ ESPECULAR

- ***V**:* vector de posición del observador
- ***L**:* vector de incidencia de la luz
- ***R**:* vector reflejado de la luz
- I_l*:* intensidad de la fuente de luz
- k_e*:* coeficiente empírico de reflexión especular ($0 \leq k_e \leq 1$)
- ***n**:* un índice que simula la rugosidad de la superficie ($1 \leq n < \infty$, 1: mate, ∞: espejo)
- Ω*:* ángulo entre ***V*** y ***R***
- ***R=2N(L·N)-L***



$$I_{especular} = I_l \, k_e \cos^n \Omega = I_l \, k_e \, (\mathbf{R} \cdot \mathbf{V})^n$$

5. Iluminación con Shaders

VARIABLES DEL SHADER

- Desde nuestra clase debemos pasar toda la información al shader, en la clase OpenGLRenderer declaramos:

```
// Nombre de los uniform
private static final String U_MVPMATRIX = "u_MVPMatrix";
private static final String U_MVMATRIX = "u_MVMatrix";
private static final String U_COLOR = "u_Color";
private static final String U_MATERIAL = "u_Material";
private static final String U_DIRECTIONALLIGHT = "u_DirectionalLight";

// Nombre de los attribute
private static final String A_POSITION = "a_Position";
private static final String A_NORMAL = "a_Normal";

// Handles para los shaders
private int uMVPMatrixLocation;
private int uMVMatrixLocation;
private int uColorLocation;
private int aPositionLocation;
private int aNormalLocation;
```

5. Iluminación con Shaders

VARIABLES DEL SHADER

- Continuación:

```
private static final int POSITION_COMPONENT_COUNT = 3;
private static final int NORMAL_COMPONENT_COUNT = 3;
private static final int UV_COMPONENT_COUNT = 2;
// Cálculo del tamaño de los datos (3+3+2 = 8 floats)
private static final int STRIDE =
(POSITION_COMPONENT_COUNT + NORMAL_COMPONENT_COUNT + UV_COMPONENT_COUNT) *
BYTES_PER_FLOAT;

// Matrices de proyección y de vista
private final float[] projectionMatrix = new float[16];
private final float[] modelMatrix = new float[16];
private final float[] MVP = new float[16];

Resource3DSReader obj3DS;
```

5. Iluminación con Shaders

VARIABLES DEL SHADER

- En el método onSurfaceCreated() añadimos este código al final:

```
        // Activamos el programa OpenGL
        glUseProgram(program);

        // Capturamos los uniforms
        uMVPMatrixLocation = glGetUniformLocation(program, U_MVPMATRIX);
        uMVMatrixLocation = glGetUniformLocation(program, U_MVMATRIX);
        uColorLocation = glGetUniformLocation(program, U_COLOR);

        // Capturamos los attributes
        aPositionLocation = glGetAttribLocation(program, A_POSITION);
        glEnableVertexAttribArray(aPositionLocation);
        aNormalLocation = glGetAttribLocation(program, A_NORMAL);
        glEnableVertexAttribArray(aNormalLocation);

        // Creamos la matriz de modelo
        setIdentityM(modelMatrix, 0);
        translateM(modelMatrix, 0, 0f, 0.0f, -4f);
    }
```

5. Iluminación con Shaders

VARIABLES DEL SHADER

- En el método onDrawFrame( ) añadimos este código:

```
multiplyMM(MVP, 0, projectionMatrix, 0, modelMatrix, 0);
//System.arraycopy(temp, 0, projectionMatrix, 0, temp.length);

// Rotación de 1º alrededor del eje y
rotateM(modelMatrix, 0, -1.0f, 0f, 1f, 0f);
```

```
// Envía la matriz de proyección multiplicada por modelMatrix al shader
glUniformMatrix4fv(uMVPMatrixLocation, 1, false, MVP, 0);
// Envía la matriz modelMatrix al shader
glUniformMatrix4fv(uMVMatrixLocation, 1, false, modelMatrix, 0);
// Actualizamos el color (Marrón)
glUniform4f(uColorLocation, 0.78f, 0.49f, 0.12f, 1.0f);
```

5. Iluminación con Shaders

VARIABLES DEL SHADER

- En el método `onDrawFrame()` para cada objeto pasamos los vértices y las normales:

```
// Dibujamos el objeto
for (int i=0; i<obj3DS.numMeshes; i++) {
    // Asociando vértices con su attribute
    obj3DS.dataBuffer[i].position(0);
    glVertexAttribPointer(aPositionLocation, POSITION_COMPONENT_COUNT,
        GL_FLOAT, false, STRIDE, obj3DS.dataBuffer[i]);

    // Asociamos el vector de normales con su attribute
    Obj3DS.dataBuffer[i].position(POSITION_COMPONENT_COUNT);
    glVertexAttribPointer(aNormalLocation, NORMAL_COMPONENT_COUNT, GL_FLOAT,
        false, STRIDE, obj3DS.dataBuffer[i]);
    // Dibuja la malla de triángulos
    glDrawArrays(GL_TRIANGLES, 0, obj3DS.numVertices[i]);
}
```

5. Iluminación con Shaders

VARIABLES DEL SHADER

- En el método `onDrawFrame()` para cada objeto pasamos los vértices y las normales (versión multi-mesh):

```
// Dibujamos el objeto
for (int i=0; i<obj3DS.numMeshes; i++) {
    // Asociando vértices con su attribute
    obj3DS.dataBuffer[i].position(0);
    glVertexAttribPointer(aPositionLocation, POSITION_COMPONENT_COUNT,
        GL_FLOAT, false, STRIDE, obj3DS.dataBuffer[i]);

    // Asociamos el vector de normales con su attribute
    Obj3DS.dataBuffer[i].position(POSITION_COMPONENT_COUNT);
    glVertexAttribPointer(aNormalLocation, NORMAL_COMPONENT_COUNT, GL_FLOAT,
        false, STRIDE, obj3DS.dataBuffer[i]);
    // Dibuja la malla de triángulos
    glDrawArrays(GL_TRIANGLES, 0, obj3DS.numVertices[i]);
}
```

5. Iluminación con Shaders

ESPECULAR (empleando reflect())

- Código principal:

```
void main()
{
    vec3 LightPos = vec3(2, 0, 6);                      // Posición de la luz [fija]
    vec3 P = vec3(u_MVMatrix * a_Position);             // Posición del vértice
    vec3 N = vec3(u_MVMatrix * vec4(a_Normal, 0.0));    // Normal del vértice
    vec4 specularColor = vec4(1, 1, 1, 1);              // Color especular (brillos blancos)

    float d = length(LightPos - P);                     // distancia
    vec3  L = normalize(LightPos - P);                  // Vector Luz
    vec3  V = normalize(P-vec3(0,0,0));                 // Vector Visión
    vec3  R = normalize(reflect(L, N));                 // Vector reflejado R=2N(N.L)-L;
                                                        // L se debe negar !!!
    float attenuation = 1.0/(0.3+(0.1*d)+(0.01*d*d));   // Cálculo de la atenuación
    float ambient  = 0.15;                              // 15% de intensidad ambiente
    float diffuse  = max(dot(N, -L), 0.0);              // Cálculo de la intensidad difusa
    float specular = pow(max(dot(R, V), 0.0), 200.0);   // Exponente de Phong (200)

    v_Color = u_Color*ambient+attenuation*(u_Color*diffuse + specularColor*specular);
    gl_Position = u_MVPMatrix * a_Position;
}
```