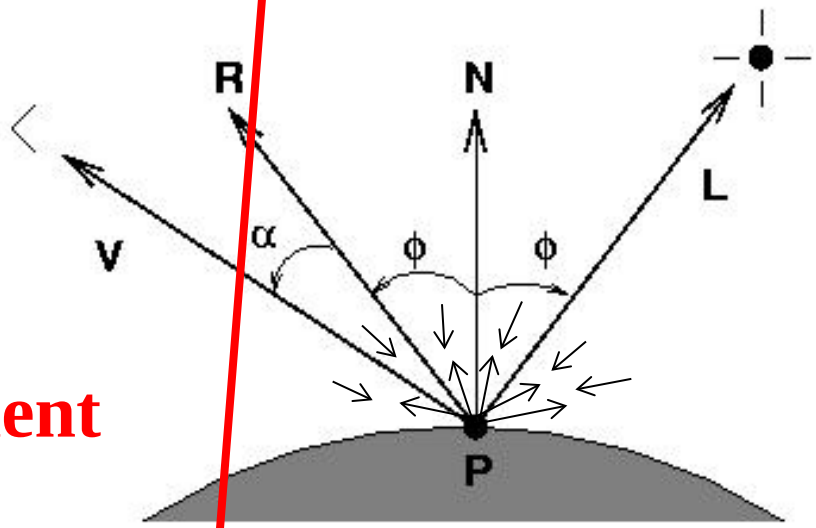
Classe 7: contingut

- Realisme: Il·luminació (2)
  - **Il·luminació en OpenGL 3.3 (1)**
    - Càlcul de color en vèrtexs (en el Vertex Shader)
    - Shading de polígons
  - Suavitzat d'arestes
  - Il·luminació en OpenGL 3.3 (2)
    - Càlcul de color en fragments

Recordem: Càlcul color en un punt

$$I_\lambda(P) = I_{a\lambda} k_{a\lambda} + \sum_i (I_{fi\lambda}\, k_{d\lambda}\, cos(\Phi_i)) + \sum_i (I_{fi\lambda}\, k_{s\lambda}\, cos^n(\alpha_i))$$

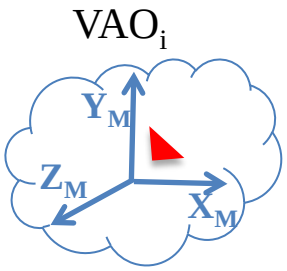


Model Ambient

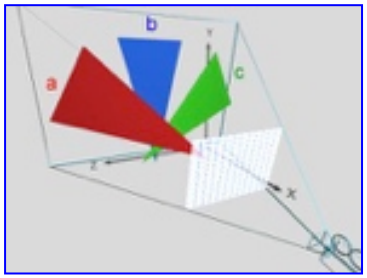
Model Difús (Lambert)

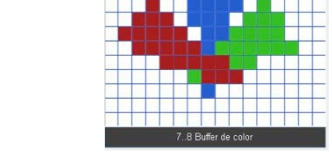
Model Especular (Phong)

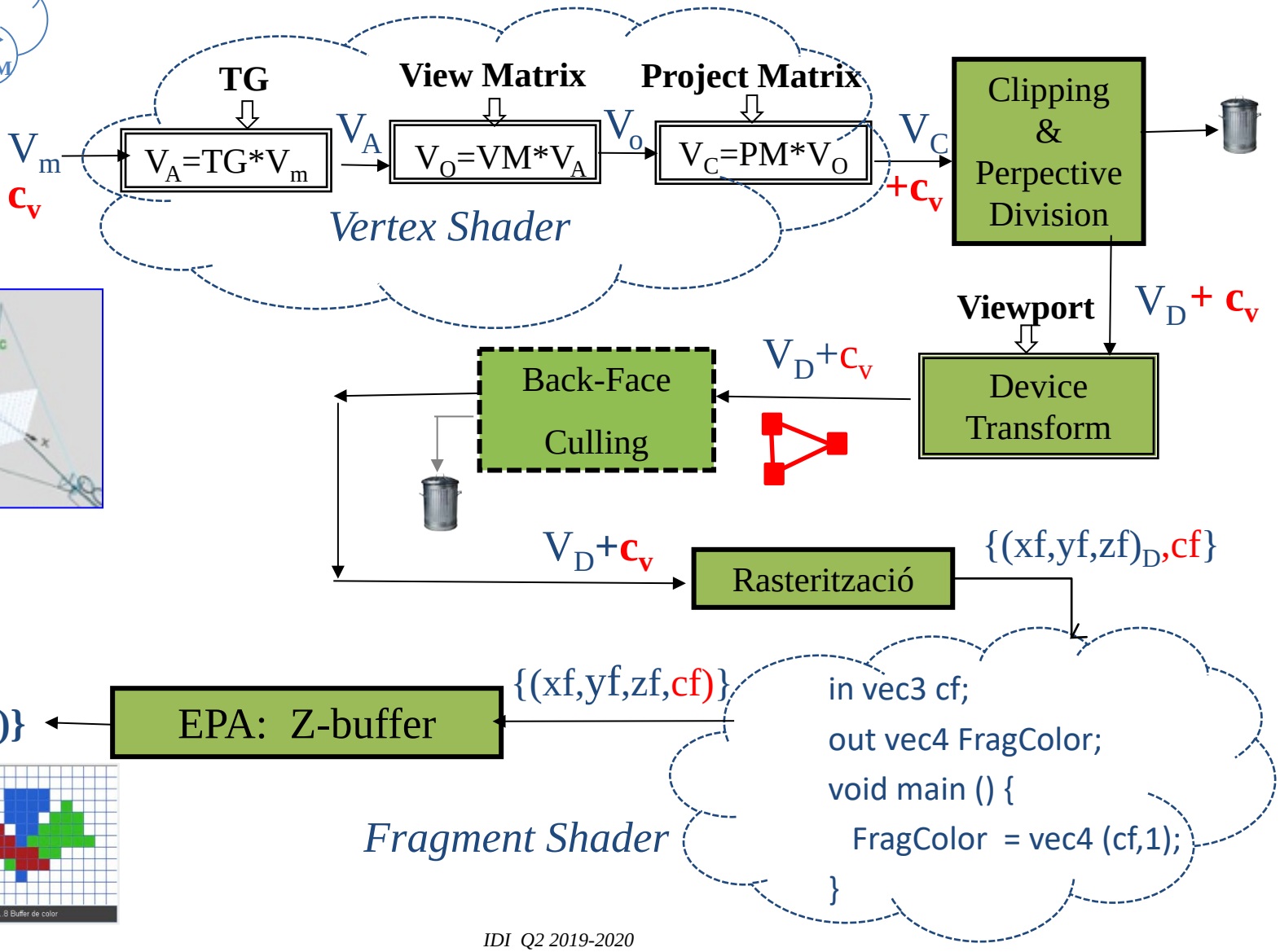
Procés de visualització



Com calcular el color aplicant fórmula de models empírics?







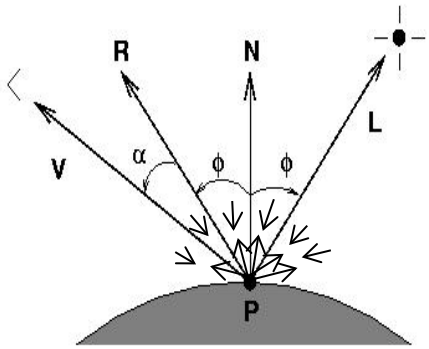
Càlcul del color per vèrtex en el Vèrtex Shader (1)

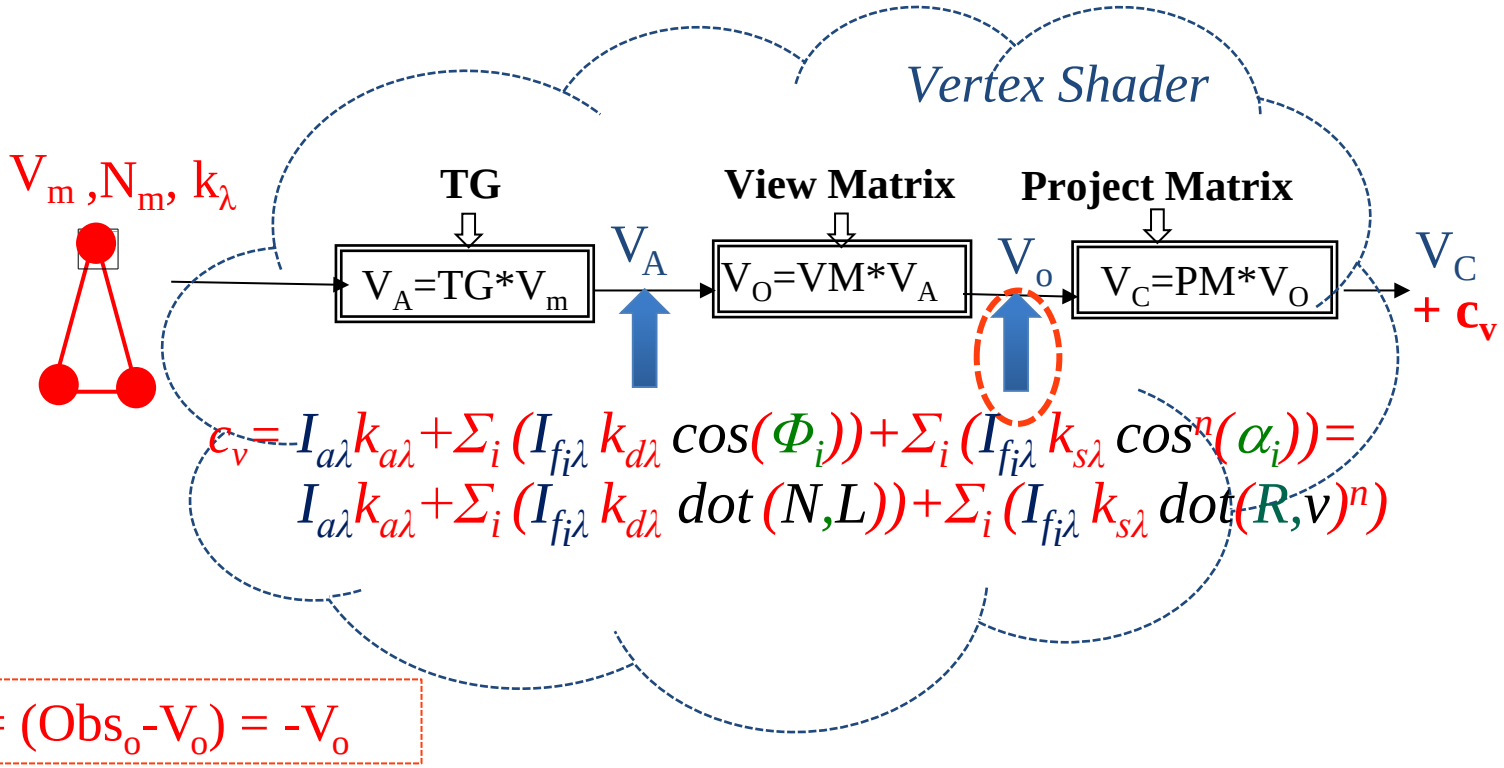
Atributs per cada model:

- Coordenades (V), normal (N) i constants de material (k_λ) per vèrtex en VBOs del seu VAO

Uniforms:

- Fonts de llum actives => color, posició
- Llum ambient





$\boldsymbol{v}_o = (Obs_o - V_o) = -V_o$

Càlcul del color per vèrtex en el Vèrtex Shader (3)

El càlcul el farem per cada vèrtex (al Vertex Shader)

I el farem en SCO, per tant:

- Cal passar la posició del vèrtex a SCO
  - multiplicant per (**view * TG)**
- Cal passar el vector normal a SCO
  - multiplicant per la matriu **inversa** de la **transposada de (view * TG),** -li direm **NormalMatrix-**

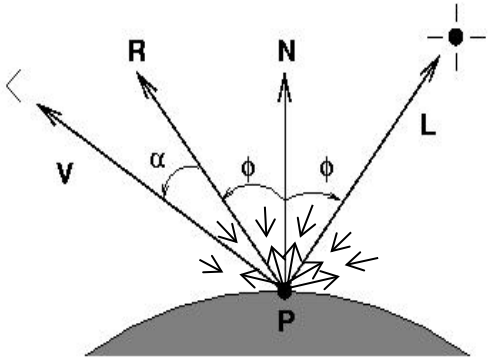
  *mat3 NormalMatrix = inverse (transpose (mat3 (view * TG)))*
- La posició del focus de llum també ha d'estar en SCO
  - Multiplicat per **view** **(si no la tenim directament en SCO)**

Càlcul del color per vèrtex en el Vèrtex Shader (2)

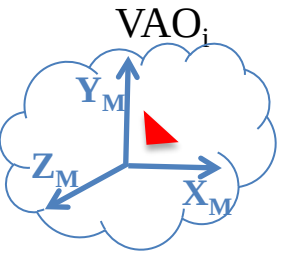
Pseudocodi de Vertex Shader per calcular color en SCO:

$$c_v = I_{a\lambda} k_{a\lambda} + \Sigma_i (I_{fi\lambda}\, k_{d\lambda} dot(N_o, L_o)) + \Sigma_i (I_{fi\lambda}\, k_{s\lambda}\, dot(R_o, v_o)^n)$$

```
in vec3 vertex, N;
in vec3 matamb, matdiff, matspec;
in float matshin;
uniform mat4 proj, view, TG;
uniform vec3 posFocus, Ia, If;
out vec3 fcolor;
...
void main()
{
   mat3 NormalMatrix = inverse (transpose (mat3 (view * TG)
   vec3 NormSCO = normalize (NormalMatrix * N);
   vec4 vertexSCO = view * TG * vec4 (vertex,1.0);
   vec4 focusSCO = view * vec4 (posFocus, 1.0);
   vec3  LSCO = normalize (focusSCO.xyz - vertexSCO.xyz);
   fcolor = color_vertex (NormSCO, LSCO, -vertexSCO);
   gl_Position = proj * view * TG * vec4 (vertex, 1.0);
}
```



Càlcul del color en el Vèrtex Shader + shading (1)

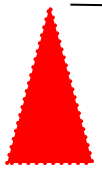


Uniforms:*posFocus,colorFocus*

TG, VM, PM

Vertex Shader

V_m,N, k_λ



- Càlcul coordenades de clipping
- Càlcul del **color** en el vèrtex

V_c+c_v

Clipping & Perpective Division

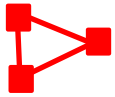
Viewport

V_n+c_v

V_D+c_v

Device Transform

Back-Face Culling



V_D+c_v

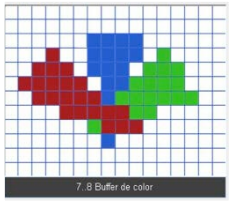
Rasterització **SHADING**

{(xf,yf,zf,cf)}

{(xf,yf,zf,cf)}

EPA: Z-buffer

{(xf,yf,cf)}



Fragment Shader

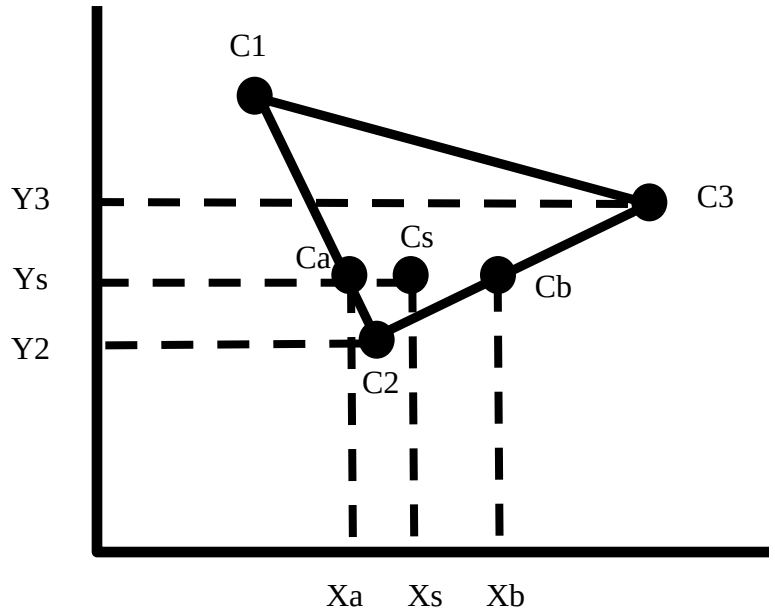
```
in vec3 cf;
out vec4 FragColor;
void main () {
  FragColor  = vec4 (cf,1);
}
```

Recordem: Shading (colorat) de polígons

- Colorat Constant ≡ **Flat *shading*** ➔ $C_f = C1$

  color uniforme per tot el polígon (funció del color calculat en un vèrtex); cada cara pot tenir diferent color.

- Colorat de Gouraud ≡ ***Gouraud shading* ≡ *Smooth shading***

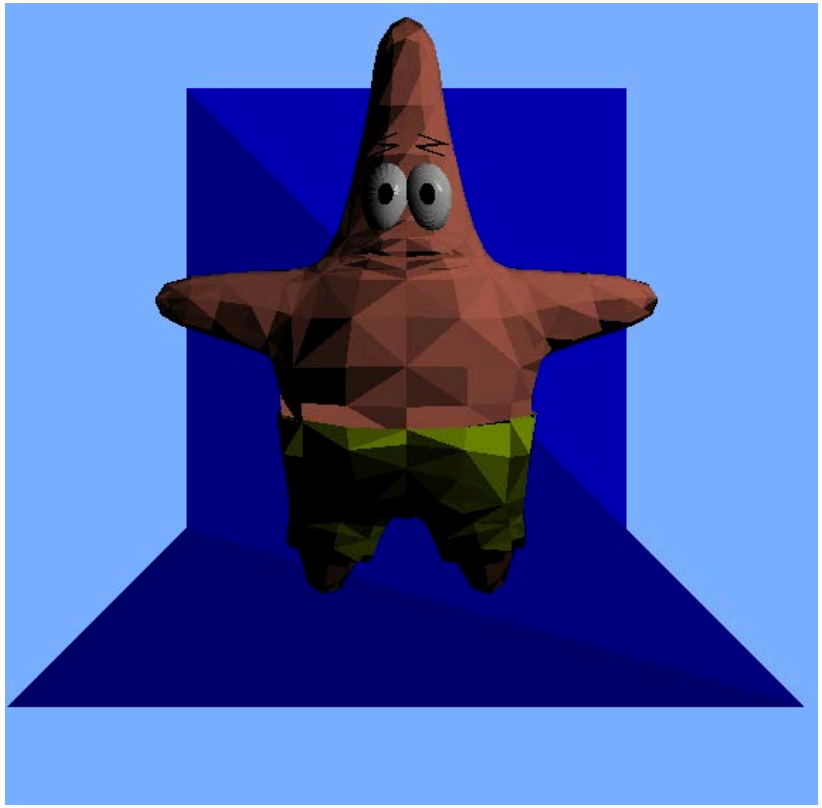


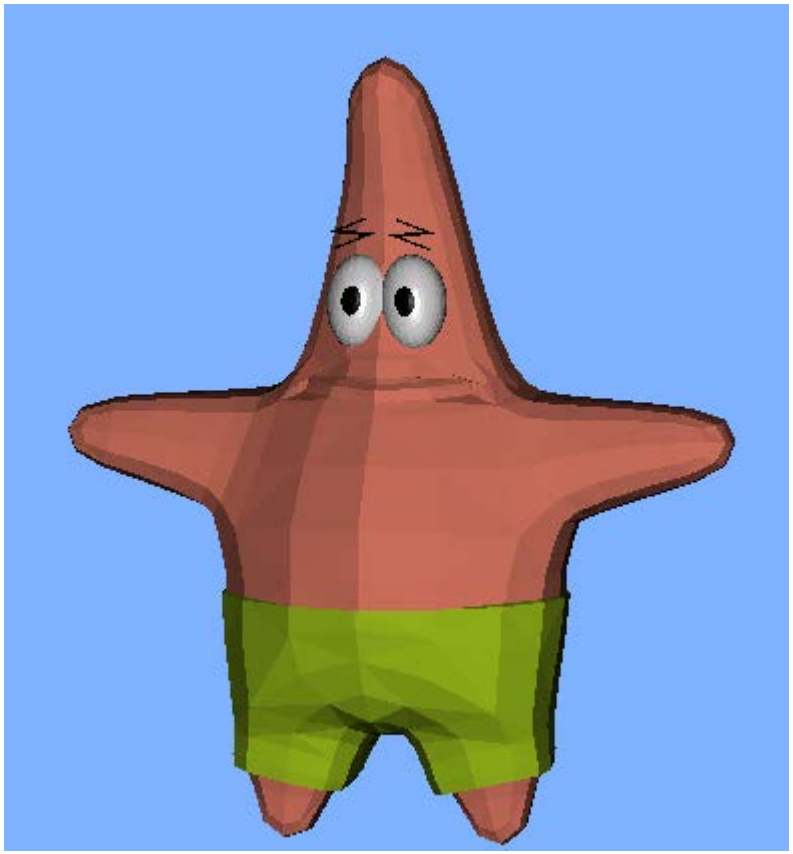
$$Ca = \frac{1}{Y1 - Y2}(C1(Ys - Y2) + C2(Y1 - Ys))$$

$$Cb = \frac{1}{Y3 - Y2}(C2(Y3 - Ys) + C3(Ys - Y2))$$

$$Cs = \frac{1}{Xb - Xa}(Ca(Xb - Xs) + Cb(Xs - Xa))$$

Flat versus Gouraud/smooth Shading

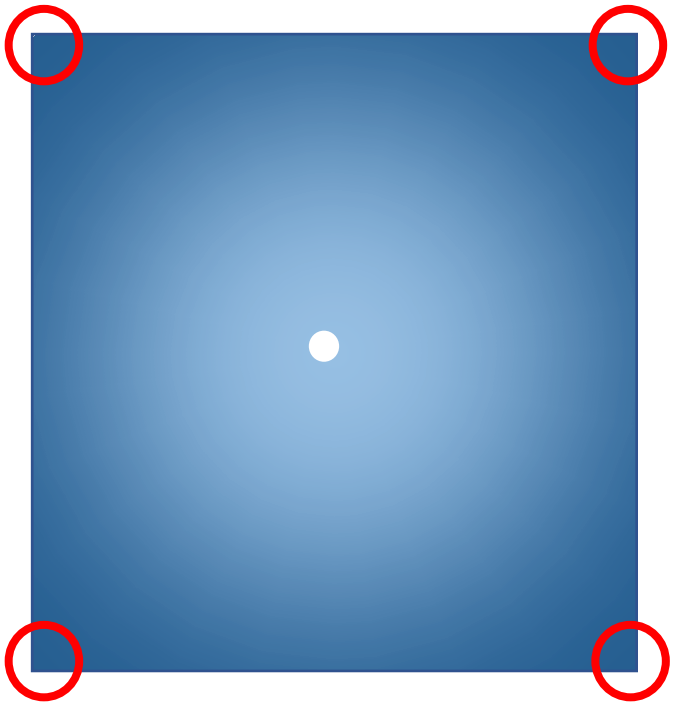


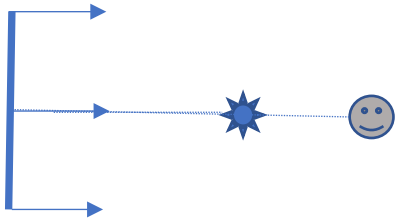


Càlcul color en VS: limitacions

Efecte del càlcul de Lambert i Phong en Vertex Shader:

Com s'hauria de veure



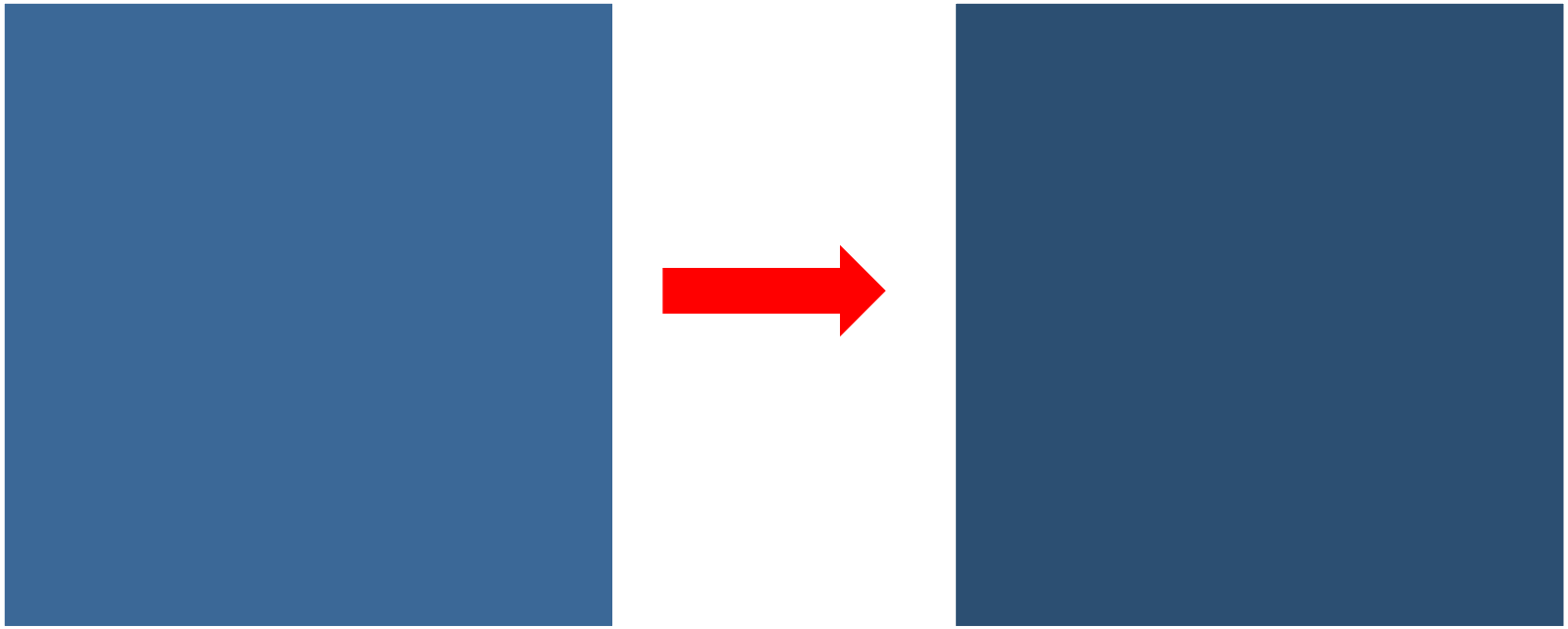


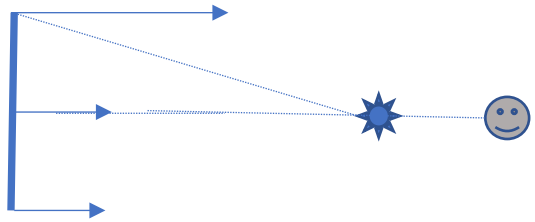
Com es veu

Possible solució: **Discretitzar més**

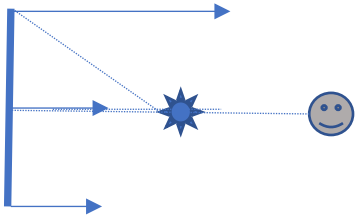
Càlcul color en VS: limitacions

Què passa si apropem el focus de llum al quadrat?





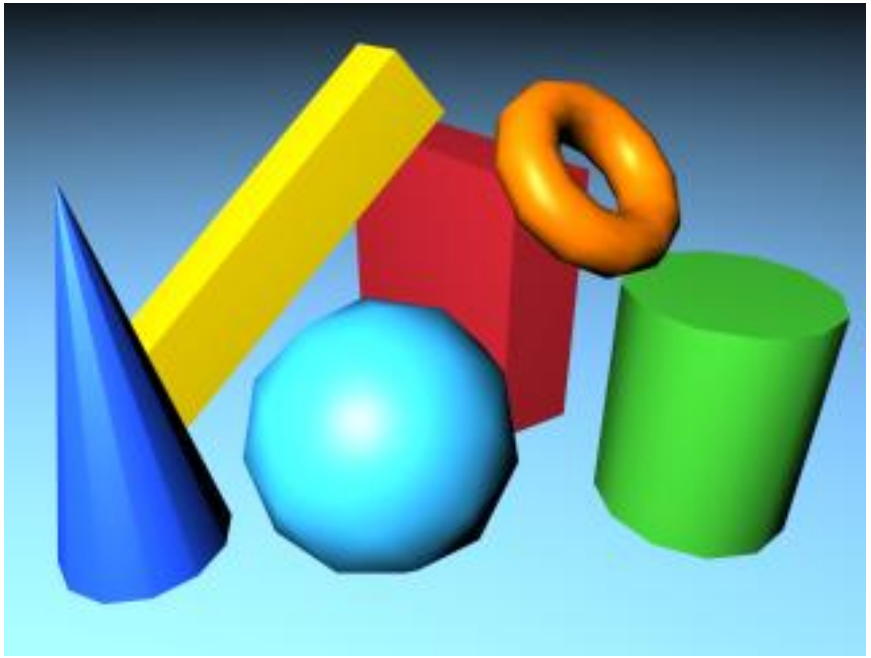
angle Φ creix



Classe 7: contingut

- Realisme: Il·luminació (2)
  - Il·luminació en OpenGL 3.3 (1)
    - Càlcul de color en vèrtexs (en el Vertex Shader)
    - Shading de polígons
  - **Suavitzat d'arestes**
  - Il·luminació en OpenGL 3.3 (2)
    - Càlcul de color en fragments

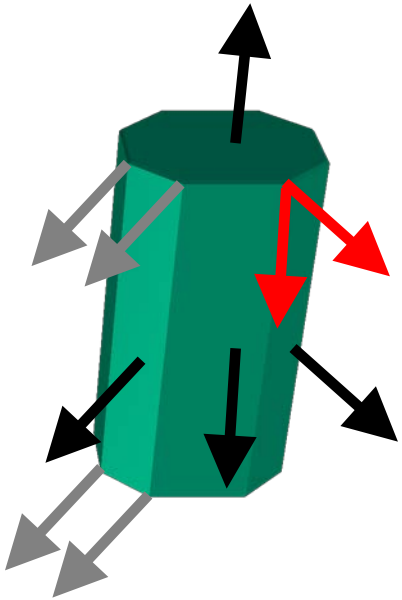
Suavitzat d'arestes (1)

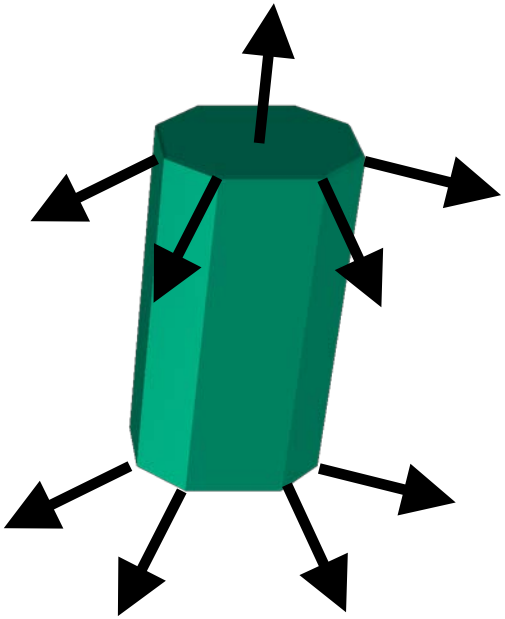


Quin model d'il·luminació i shading s'utilitza?
Per què no es veuen les arestes?
Noteu la forma de les siluetes

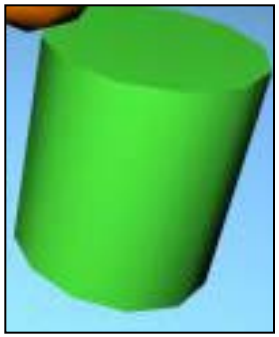
Suavitzat d'arestes (2)

- Normal per cara vs normal per vèrtex



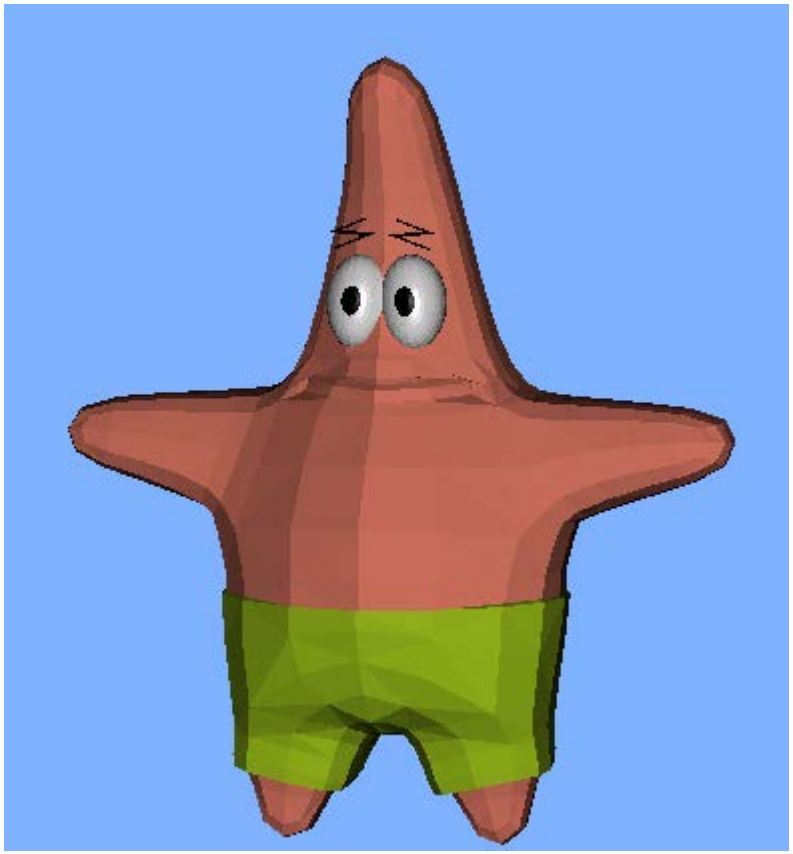


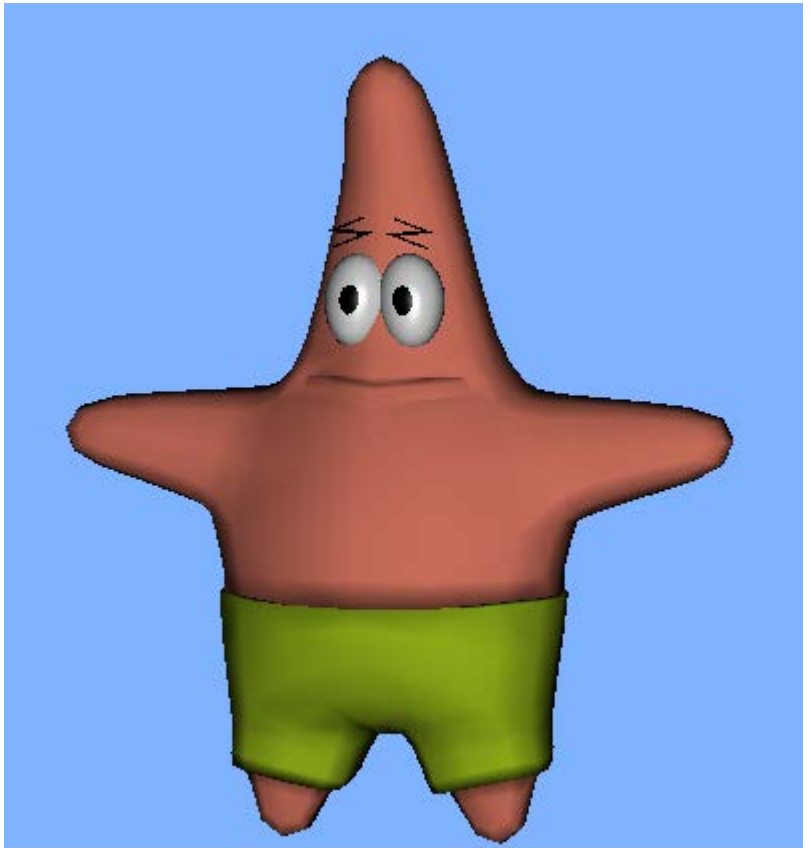
$$\overrightarrow{Nv} = \frac{\sum_i \overrightarrow{Ni}}{\left\| \sum_i \overrightarrow{Ni} \right\|}$$



Suavitzat d'arestes: exemple

- Normal per cara vs normal per vèrtex

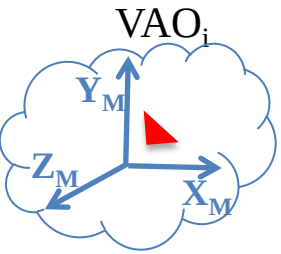




Classe 7: contingut

- Realisme: Il·luminació (2)
  - Il·luminació en OpenGL 3.3 (1)
    - Càlcul de color en vèrtexs (en el Vertex Shader)
    - Shading de polígons
  - Suavitzat d'arestes
  - **Il·luminació en OpenGL 3.3 (2)**
    - **Càlcul de color en fragments**

Càlcul del color per fragment en en Fragment Shader (1)



Uniforms: *posFocus,colorFocus, TG, VM,PM*

Vertex Shader

V_M,N, mat

- Càlcul coordenades de clipping
- ??

V_c

Clipping & Perpective Division

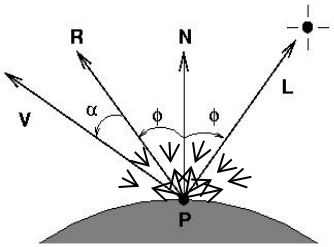
Viewport

V_n

Device Transform

V_D

Back-Face Culling



V_D

Rasterització **SHADING**

{(xf,yf,zf)}

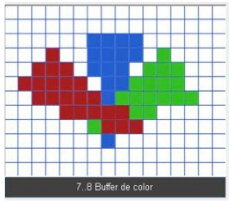
Fragment Shader

```
....
out vec4 FragColor;
void main () {
...
FragColor = ??
}
```

{(xf,yf,zf,cf)}

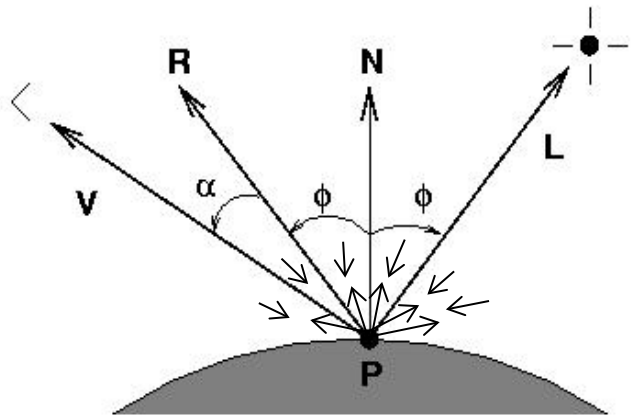
EPA: Z-buffer

{(xf,yf,cf)
}



$$FragColor = I_{a\lambda} k_{a\lambda} + \Sigma_i (I_{fi\lambda} k_{d\lambda} \cos(\Phi_i)) + \Sigma_i (I_{fi\lambda} k_{s\lambda} \cos^n(\alpha_i))$$

Càlcul del color per fragment en en Fragment Shader (2)



$$FragColor\ \ I_{a\lambda}k_{a\lambda} + \Sigma_i (I_{fi\lambda}\, k_{d\lambda}\, cos(\Phi_i)) + \Sigma_i (I_{fi\lambda}\, k_{s\lambda}\, cos^n(\alpha_i)$$

$cos(\Phi)$ => dot (L,N) en SCO

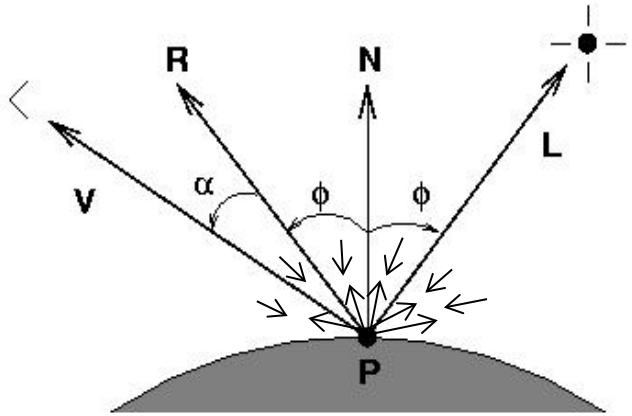
$cos(\alpha)$ => dot (R, v) en SCO

Idea 1: Per cada píxel (fragment)

- ✓ Tenim la informació de les llums, són *uniforms*
- Requereix el vèrtex, altres vectors en SCO o SCA i les constants material
  - ✓ Tenim el fragment en SCD i matrius (uniforms)=> *podríem calcular les coordenades del punt que es projecte en SCO o SCA aplicant inversa de les matrius.*
  - ➢ Procés costós
  - ➢ No tenim accés a la Normal ni a les constants empíriques del material (eren atributs/in en VS)

ALTRE IDEA?

Càlcul del color per fragment en en Fragment Shader (3)



$$FragColor \;\; I_{a\lambda}k_{a\lambda} + \Sigma_i (I_{fi\lambda}\, k_{d\lambda}\, cos(\Phi_i)) + \Sigma_i (I_{fi\lambda}\, k_{s\lambda}\, cos^n(\alpha_i)$$

$cos(\Phi)$ => dot (L,N) en SCO

$cos(\alpha)$ => dot (R, v) en SCO

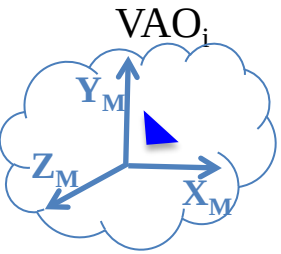
Proposta de solució:

- ✓ Tenim la informació de les llums, són *uniforms*
- ✓ Podem fer "out" en el VS dels atributs associats al vèrtex: N i V *en SCO*, i de les constants empíriques de material . La rasterització calcularà el seu valor pel fragment interpolant la informació dels vèrtexs del triangle ➔ tindrem els seus valors en el FS com variables "in" ☺

Observació:

- La normal del fragment és una aproximació de la normal del punt del triangle que es projecta en el fragment. Aquesta interpolació de la normal es coneix com "*shading de phong*"

Càlcul del color per fragment en en Fragment Shader (4)



Uniforms: *posFocus,colorFocus,TG, VM, PM*

Vertex Shader

V_M, N_M,
$k_{a\lambda}$, $k_{d\lambda}$, $k_{s\lambda}$, n

- Càlcul coordenades de clipping
- Càlcul de N_o i V_o

V_c

$k_{a\lambda}$, $k_{d\lambda}$, $k_{s\lambda}$, n
N_o , V_o

- Clipping
- Perspective Division
- Device Transform
- Back-Face Culling

V_D

$k_{a\lambda}$, $k_{d\lambda}$, $k_{s\lambda}$, n
N_o , V_o

Rasterització

interpolació

$\{(x_f,y_f,z_f)$, $k_{a\lambda f}$, $k_{d\lambda f}$, $k_{s\lambda f}$, n_f, N_{of} , $V_{of}\}$

Fragment Shader

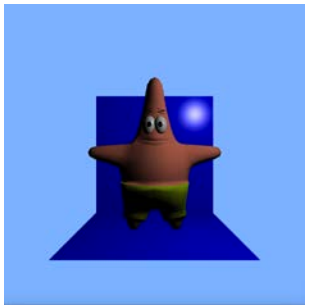
$cf = I_{a\lambda}k_{a\lambda} + \Sigma_i (I_{fi\lambda}\, k_{d\lambda}\, cos(\Phi_i)) + \Sigma_i (I_{fi\lambda}\, k_{s\lambda}\, cos^n(\alpha_i))$

FragColor = cf

$\{(x_f,y_f,z_f,cf)\}$

Z-buffer

(xf,yf,cf)

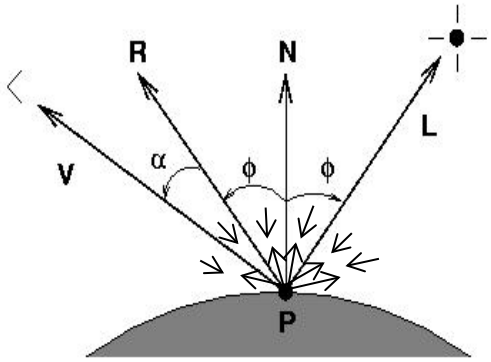


Càlcul del color per vèrtex en el Vèrtex Shader (5)

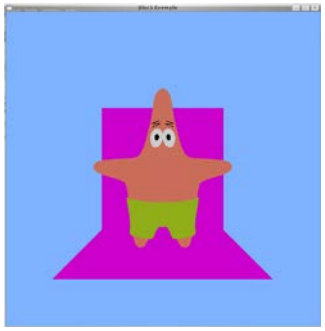
Pseudocodi de Fragment Shader per calcular color en SCO:

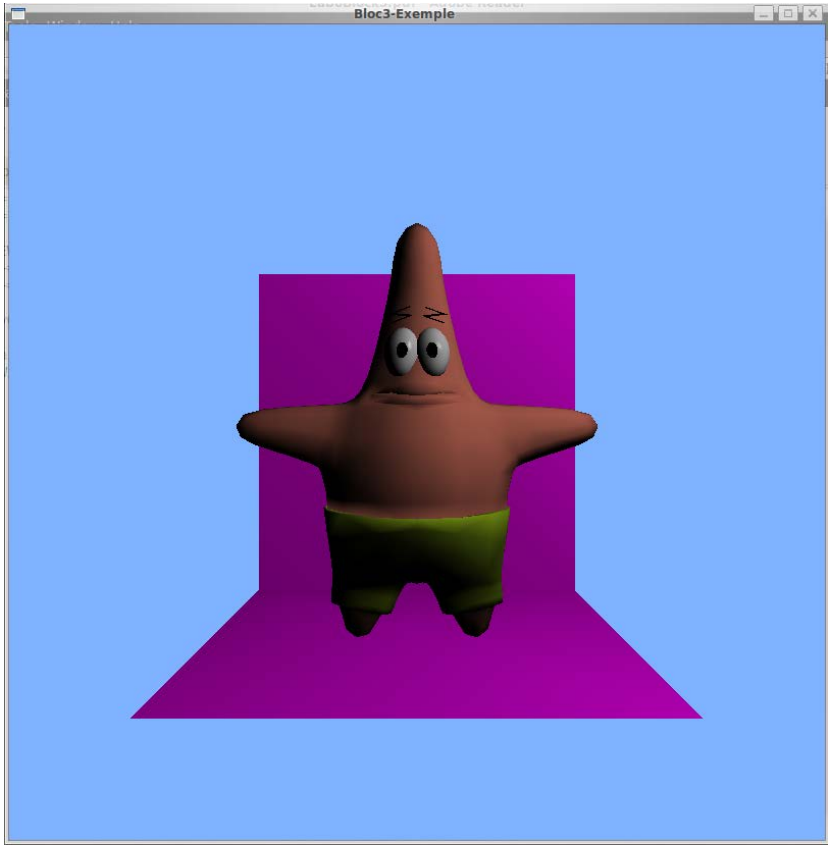
$$c_v = I_{a\lambda} k_{a\lambda} + \Sigma_i (I_{fi\lambda}\, k_{d\lambda} dot\,(N_o, L_o)) + \Sigma_i (I_{fi\lambda}\, k_{s\lambda}\, dot(R_o, v_o)^n)$$

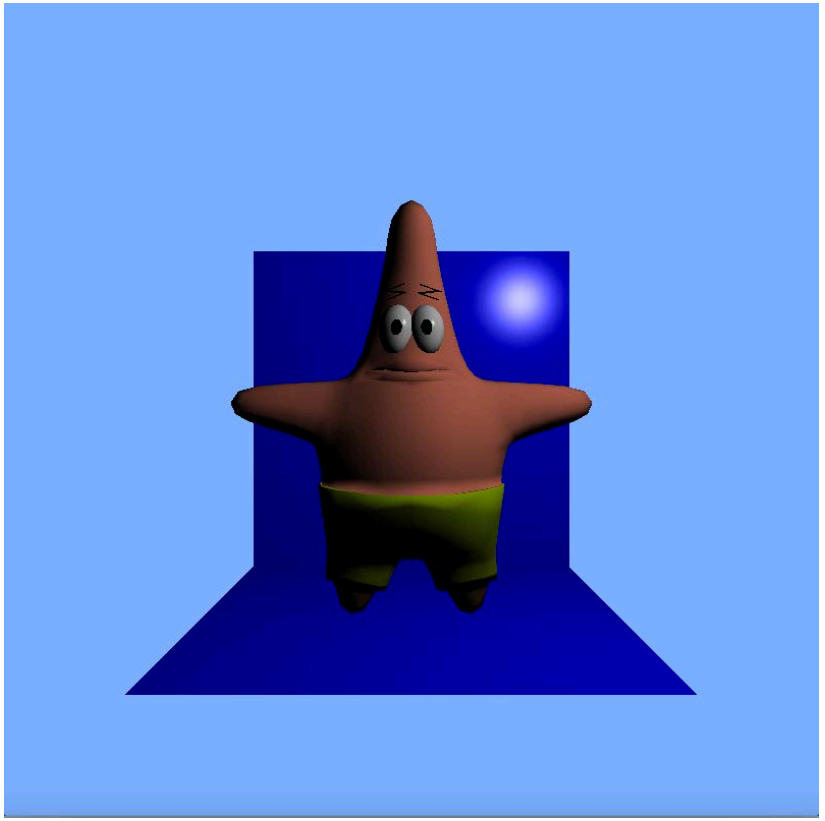
```
in vec3 NSCO, VSCO;
in vec3 fmatamb, fmatdiff, fmatspec;
in float fmatshin;
uniform mat4 proj, view, TG;
uniform vec3 posFocus, Ia, If;
out vec4 FragColor;
...
void main()
{
   vec4 focusSCO = view * vec4 (posFocus, 1.0);
   vec3 NSCO_norm = normalize (NSCO);
   vec3 L = normalize (focusSCO.xyz - VSCO.xyz);
   FragColor = vec4 (color_vertex (NSCO_norm, L, -VSCO), 1);
}
```

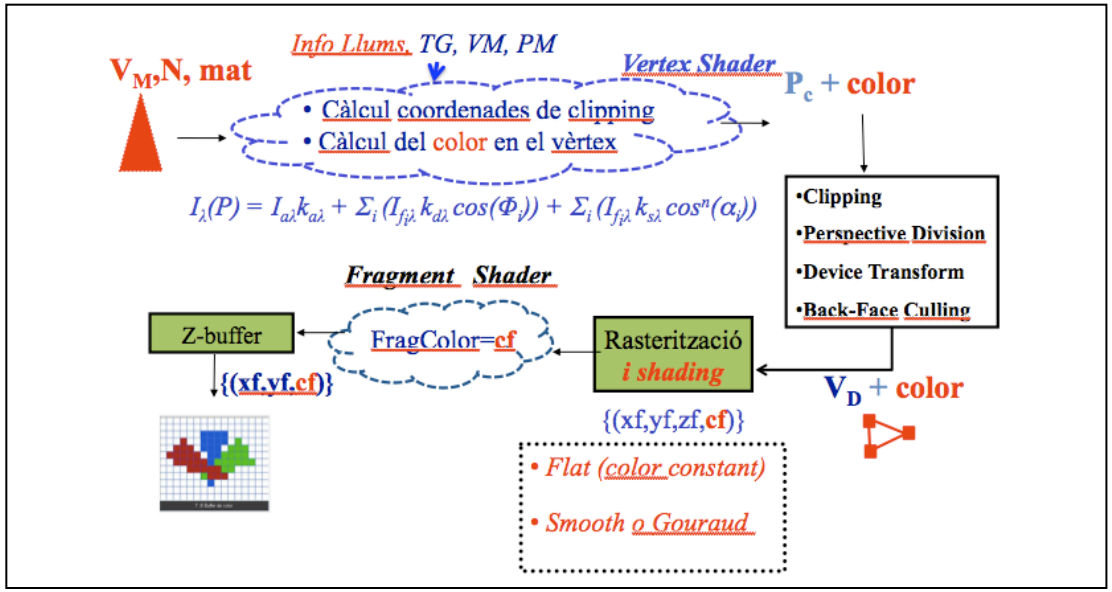


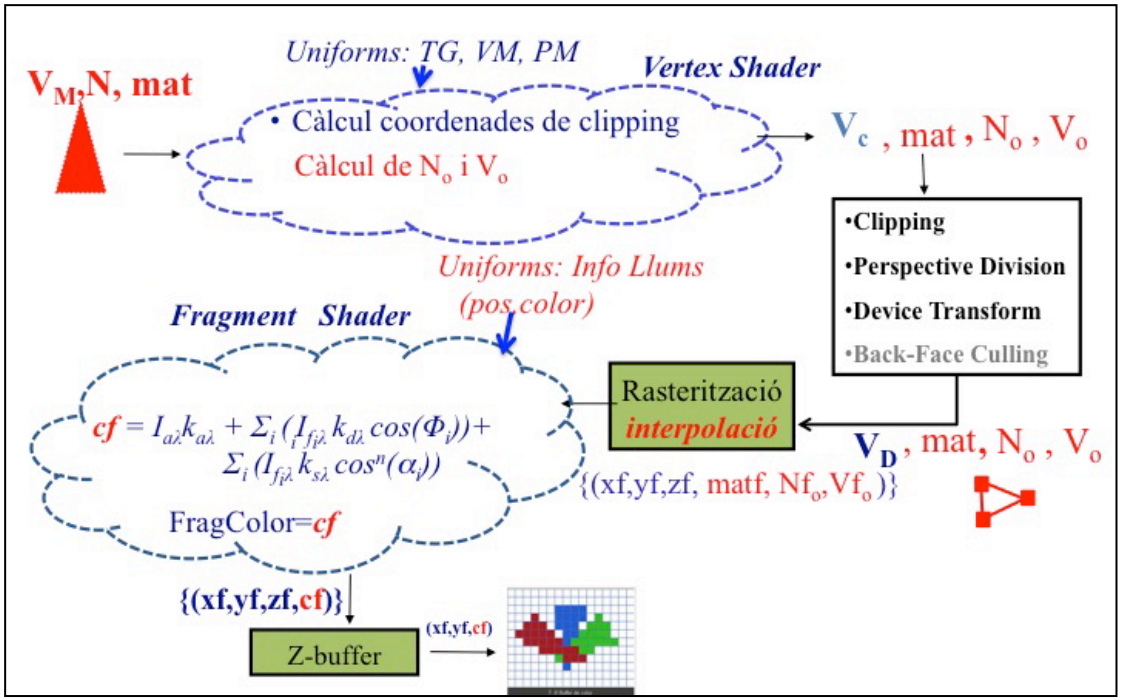
Exemple final: Sense il·luminació i Il·luminació en el VS versus FS








Info Llums, TG, VM, PM
V_M,N, mat
Vertex Shader
P_c + color
• Càlcul coordenades de clipping
• Càlcul del color en el vèrtex
$I_\lambda(P) = I_{a\lambda}k_{a\lambda} + \Sigma_i (I_{fi\lambda}\, k_{d\lambda} \cos(\Phi_i)) + \Sigma_i (I_{fi\lambda}\, k_{s\lambda} \cos^n(\alpha_i))$
•Clipping
•Perspective Division
•Device Transform
•Back-Face Culling
Fragment Shader
Z-buffer
FragColor=cf
Rasterització
i shading
{(xf,yf,cf)}
V_D + color
{(xf,yf,zf,cf)}
• Flat (color constant)
• Smooth o Gouraud


Uniforms: TG, VM, PM
V_M,N, mat
Vertex Shader
• Càlcul coordenades de clipping
Càlcul de N_o i V_o
V_c , mat , N_o , V_o
•Clipping
•Perspective Division
•Device Transform
•Back-Face Culling
Uniforms: Info Llums
(pos,color)
Fragment Shader
$cf = I_{a\lambda}k_{a\lambda} + \Sigma_i (I_{fi\lambda}\, k_{d\lambda} \cos(\Phi_i)) + \Sigma_i (I_{fi\lambda}\, k_{s\lambda} \cos^n(\alpha_i))$
FragColor=cf
Rasterització
interpolació
V_D , mat, N_o , V_o
{(xf,yf,zf, matf, Nf_o,Vf_o)}
{(xf,yf,zf,cf)}
Z-buffer
(xf,yf,cf)

Conceptes i preguntes

- Càlcul del color en el vèrtex shader. Quants VBOs requereix per VAO d'un model? Quina informació requereix? Cal fer algun càlcul en el fragment shader? La geometria en quin sistema de coordenades de referència ha d'estar?
- Limitacions del càlcul del color en el vèrtex shader.
- Suavitzat d'arestes. Quan cal fer-lo? Quina informació requereix? Degut a quin procés del procés de visualització es pot realitzar? Requereix algun càlcul extra en el vèrtex i/o en el fragment shader?
- Càlcul del color en el fragment shader. Quina informació requereix? Com accedeix a aquesta informació?
- Perquè el càlcul de la il·luminació d'una escena és més costós si es realitza en el FS que en el VS (la fórmula és la mateixa)?
- Quines limitacions en el realisme d'una escena es resolen calculant la il·luminació en el FS enlloc del VS? Quin experiment faries per assegurar-te executant un programa si calcula la il·luminació en el VS o en FS?
- Fer els exercicis proposats de la col·lecció de problemes.