Frag:

#version 330 core

out vec4 FragColor;

in vec3 matambFS;

in vec3 matdiffFS;

in vec3 matspecFS;

in float matshinFS;

in vec4 vertexFS;

in vec3 normalFS;

// Valors per als components que necessitem dels focus de llum

uniform vec3 colFocus;

uniform vec3 posFocus;

vec3 llumAmbient = vec3(0.2, 0.2, 0.2);

vec3 Lambert (vec3 NormSCO, vec3 L)

{

// S'assumeix que els vectors que es reben com a paràmetres estan normalitzats

// Inicialitzem color a component ambient

vec3 colRes = llumAmbient \* matambFS;

// Afegim component difusa, si n'hi ha

if (dot (L, NormSCO) > 0)

colRes = colRes + colFocus \* matdiffFS \* dot (L, NormSCO);

return (colRes);

}

vec3 Phong (vec3 NormSCO, vec3 L, vec4 vertSCO)

{

// Els vectors estan normalitzats

// Inicialitzem color a Lambert

vec3 colRes = Lambert (NormSCO, L);

// Calculem R i V

if (dot(NormSCO,L) < 0)

return colRes; // no hi ha component especular

vec3 R = reflect(-L, NormSCO); // equival a: normalize (2.0\*dot(NormSCO,L)\*NormSCO - L);

vec3 V = normalize(-vertSCO.xyz);

if ((dot(R, V) < 0) || (matshinFS == 0))

return colRes; // no hi ha component especular

// Afegim la component especular

float shine = pow(max(0.0, dot(R, V)), matshinFS);

return (colRes + matspecFS \* colFocus \* shine);

}

void main()

{

vec3 norm = normalize(normalFS);

vec4 L = normalize(vec4(posFocus,1) - vertexFS); // Vector entre la posicio del focus i el model

vec3 fcolor = Phong(norm, vec3(L), vertexFS); //Amb brillantors;

//fcolor = Lambert(normalSCO, vec3(L));

FragColor = vec4(fcolor,1);

}

Vertex:

#version 330 core

//ferran

in vec3 vertex;

in vec3 normal;

in vec3 matamb;

in vec3 matdiff;

in vec3 matspec;

in float matshin;

out vec3 matambFS;

out vec3 matdiffFS;

out vec3 matspecFS;

out float matshinFS;

out vec4 vertexFS;

out vec3 normalFS;

uniform mat4 proj;

uniform mat4 view;

uniform mat4 TG;

void main()

{ //SCModel --TG--> SC Aplicacio --view--> SCObservador

matambFS = matamb;

matdiffFS = matdiff;

matspecFS = matspec;

matshinFS= matshin;

vec4 vertexSCO = view \* TG \* vec4(vertex, 1.0); // En SCO

vertexFS = vertexSCO;

mat3 NormalMatrix = inverse(transpose (mat3 (view\*TG)));

vec3 normalSCO = normalize(NormalMatrix \* normal);

normalFS = normalSCO;

gl\_Position = proj \* view \* TG \* vec4 (vertex, 1.0);

}