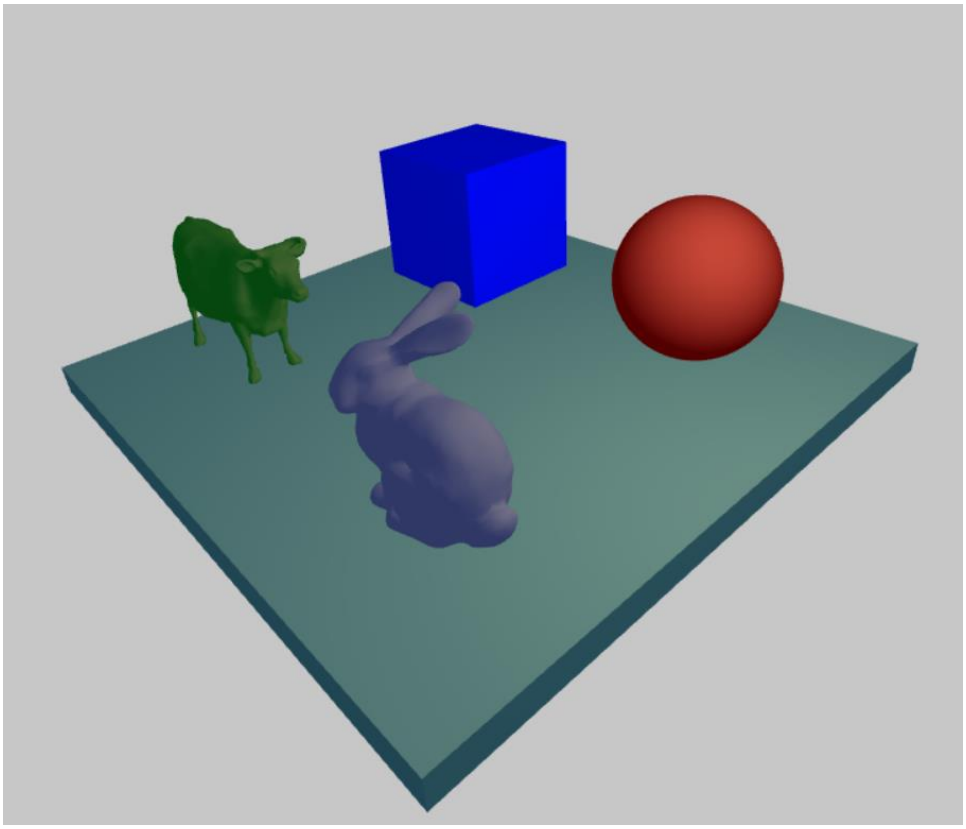
CGI - 23/24

Projecto 3 - Iluminação



António Cinca Festas – 63739 – ac.festas@campus.fct.unl.pt

Gonçalo Carvalho – 61605 – gmm.carvalho@campus.fct.unl.pt

Shader Principal

Este programa é utilizado para desenhar os objetos iluminados pelas luzes activas.

Vertex Shader

```
attribute vec4 vPosition;
attribute vec3 vNormal;

uniform mat4 mProjection;
uniform mat4 mModelView;
uniform mat4 mNormals;

varying vec3 fNormal;
varying vec3 fViewer;

void main()
{
    gl_Position = mProjection * mModelView * vPosition;

    //fNormal = (mNormals * vec4(vNormal, 0.0)).xyz;
    fNormal = mat3(mModelView) * normalize(vNormal);

    fViewer = -(mModelView * vPosition).xyz; // - position of the camera
}
```

mProjection: Matriz da projeção

mModelView; Matriz view * Matriz model

mNormals: É a matriz normal do model view

fNormal: varying para passar para o fragment shader

fViewer: O contrário da posição da camara

Fragment Shader

```
precision highp float;

const int MAX_LIGHTS = 8;

struct LightInfo {
    vec4 pos;
    vec3 Ia;
    vec3 Id;
    vec3 Is;
};

struct MaterialInfo {
    vec3 Ka;
    vec3 Kd;
    vec3 Ks;
    float shininess;
};

varying vec3 fNormal;
varying vec3 fViewer;

uniform int uNLights; // The number of lights active or inactive
uniform LightInfo uLight[MAX_LIGHTS]; // The array of lights present in the scene
uniform MaterialInfo uMaterial; // The material of the object being drawn
uniform mat4 mView;
uniform mat4 mViewNormals;

// varying mat4 fModelView;

void main()
{
    vec4 tempColor = vec4(0.0, 0.0, 0.0, 1.0);

    for (int i = 0; i < MAX_LIGHTS; i++) {
        if (i >= uNLights) {
            break ; // no more lights to process
        }

        vec3 L;

        if(uLight[i].pos.w == 0.0)  // if is a vector
            L = normalize((mViewNormals * uLight[i].pos).xyz);
        else                        // if is a point
            L = normalize((mView * uLight[i].pos).xyz + fViewer);


        vec3 V = normalize(fViewer);
        vec3 N = normalize(fNormal);
        vec3 H = normalize(L + V);

        vec3 ambient = uLight[i].Ia * uMaterial.Ka;
        vec3 diffuse = uLight[i].Id * uMaterial.Kd * max(dot(L,N), 0.0);

        vec3 specular;
        if( dot(L,N) < 0.0 ) {
            specular = vec3(0.0, 0.0, 0.0);
        }
        else {
            specular = uLight[i].Is * uMaterial.Ks * pow(max(dot(N, H), 0.0), uMaterial.shininess);
        }

        tempColor += vec4(ambient + diffuse + specular, 0);
    }
    gl_FragColor = tempColor;
}
```

fNormal: Normal do objeto

fViewer: Contrário da posição da câmara

uNLights: número de luzes ativas

LightInfo: Um array das luzes ativas para aplicar nos objetos

MaterialInfo: O struct que representa o material (ka, kd, ks, shininess) do objeto que está a ser desenhado.

mView: a matriz do output da função lookAt

mViewNormals: Inversa da transposta da matriz mView

Wireframe Shader

Este programa é utilizado apenas em duas situações:

No desenho das esferas das luzes, onde o *uColor* recebe a componente *diffuse* da luz a ser desenhada.

No desenho da wireframe do objeto selecionado, onde o *uColor* recebe sempre a cor branca

Vertex Shader

```
attribute vec4 vPosition;

uniform mat4 mProjection;
uniform mat4 mModelView;


void main()
{
    gl_Position = mProjection * mModelView * vPosition;
}
```

Este shader utiliza uma versão muito básica de um shader que apenas aplica a projeção, com o modelView e com a posição do ponto a tratar.

Fragment Shader

```
precision highp float;

uniform vec4 uColor;

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

Este shader vai receber um uColor que vai ser a cor