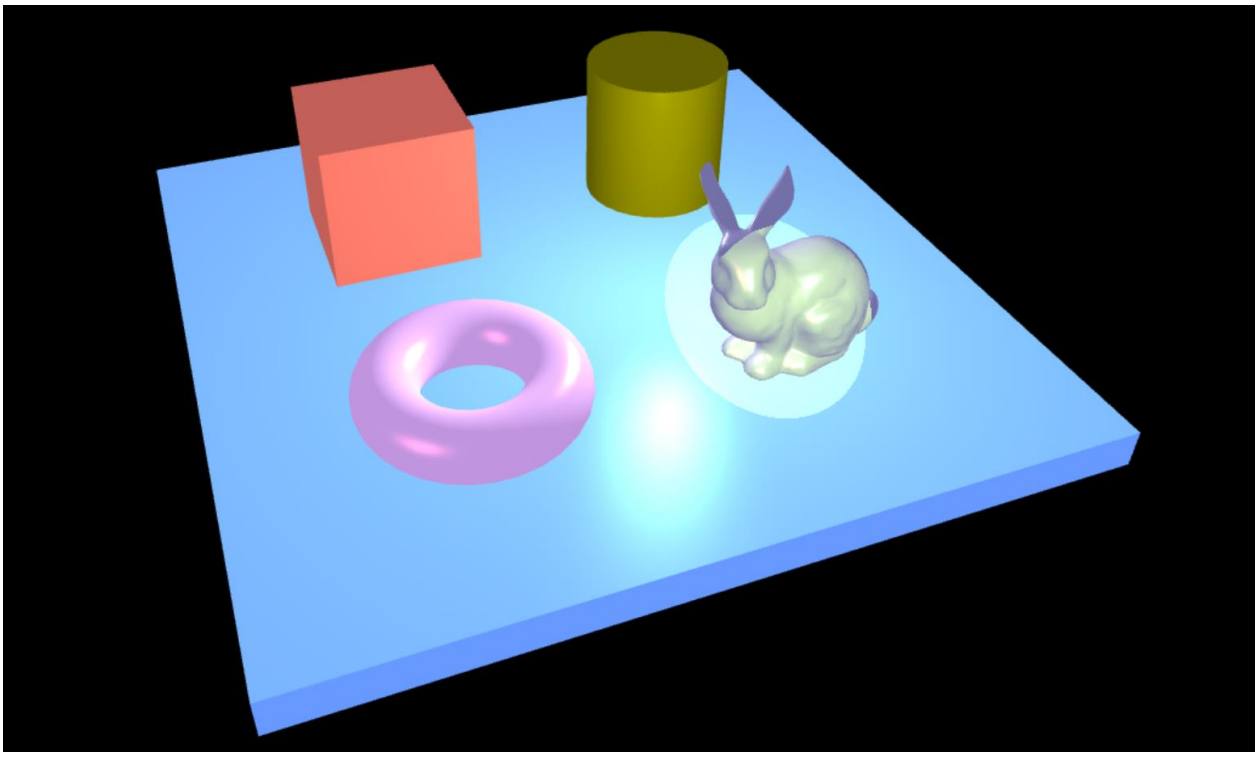
Relatório CGI - Projeto 3

Afonso Proença 59158

Francisco Freitas 60313



Tipos de dados usados no programa (JavaScript e shaders) para representar as luzes.

Shaders Structure

```
struct LightInfo {
    // Light colour intensities
    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
    // Light geometry
    vec4 position;  // Position/direction of light (in camera coordinates)
    vec3 axis;
    float apperture;
    float cutoff;
    int onState;
    int type; // Type 0 if Point, type 1 if Directional, type 2 if Spotlight
};
```

Shader Variables

```
uniform int uNLights; // Effective number of lights used
uniform LightInfo uLights[MAX_LIGHTS]; // The array of lights present in the scene
uniform MaterialInfo uMaterial;        // The material of the object being drawn
```

JavaScript

```
let lights = [
    {
    ambient : [128.5,128.5,128.5],
    diffuse : [128.5,128.5,128.5],
    specular : [200,200,200],
    position : [1.0,2.0,1.0,1.0],
    axis : [0.0,0.0,-1.0],
    apperture : 0.0,
    cutoff : -1,
    onState : true,
    type : 'Point'
    },
    {
    ambient : [100,0,0],
    diffuse : [100,0,0],
    specular : [150,0,0],
    position : [-10.0,5.0,5.0,0.0],
    axis : [10.0,-5.0,-5.0],
    apperture : 0.0,
    cutoff : -1,
    onState : false,
    type : 'Directional'
    },
    {
```

Foi utilizado um vetor de luzes, contendo informação sobre cada luz. Além da estrutura definida pelos professores, foram adicionados dois atributos associados a cada luz: o seu estado (ligado ou desligado) e o seu tipo (Pontual, Direcional ou *Spotlight*).

É possível alterar o tipo de cada luz no decorrer do programa, com manipulação dos valores, interagindo com o menu de controlo da biblioteca *dat.gui.*.

Indicar o que teria que mudar no programa para que, em vez de 3 luzes, se pudesse ter um número maior. Explique as alterações nos shaders e código JavaScript, incluindo a construção da interface para a manipulação dos respetivos parâmetros

De acordo com o programa desenvolvido, as únicas mudanças necessárias a efetuar para adicionar mais luzes, são no próprio vetor inicial de luzes. Todo o programa está escrito de forma extensível, inclusivamente na manipulação dos parâmetros e dos shaders, podendo suportar até a 8 luzes - número limite de luzes pelo fragment shader, sendo este também alterável.

```
let lights = [
    {
    ambient : [128.5,128.5,128.5],
    diffuse : [128.5,128.5,128.5],
    specular : [200,200,200],
    position : [1.0,2.0,1.0,1.0],
    axis : [0.0,0.0,-1.0],
    apperture : 0.0,
    cutoff : -1,
    onState : true,
    type : 'Point'
    },
    {
    ambient : [100,0,0],
    diffuse : [100,0,0],
    specular : [150,0,0],
    position : [-10.0,5.0,5.0,0.0],
    axis : [10.0,-5.0,-5.0],
    apperture : 0.0,
    cutoff : -1,
    onState : false,
    type : 'Directional'
    },
    {
    ambient : [75,75,100],
    diffuse : [75,75,100],
    specular : [150,150,175],
    position : [0,3.0,0.0,0.0],
    axis : [0.0, 0.0, -1.0],
    apperture : 15.0,
    cutoff : 30,
    onState : false,
    type : 'Spotlight'
    }
];
```

Como se encontra visível na imagem ao lado, seria apenas necessário adicionar uma luz diretamente, no vetor *"lights"*, especificando cada parâmetro inicial, visto que este poderá ser alterando durante a execução da animação.

Explicar como foi implementada a luz do tipo spotlight, recorrendo ao código respetivo no fragment shader.

```
for(int i=0; i<MAX_LIGHTS; i++) {
    if(i == uNLights) break;
    if(uLights[i].onState == 1) {

        vec3 L;
        if(uLights[i].type == 1) // this is, if uLights[i].position.w == 0 (Directional light)
            L = normalize((mViewNormals * uLights[i].position).xyz);
        else
            L = normalize((mView * uLights[i].position).xyz - fPosition);

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

        vec3 ambientColor = uLights[i].ambient/255.0 * material.Ka/255.0;
        vec3 diffuseColor = uLights[i].diffuse/255.0 * material.Kd/255.0;
        vec3 specularColor = uLights[i].specular/255.0 * material.Ks/255.0;

        float intensity;
        if(uLights[i].type == 2) {
            float dotLVector = dot(L, -uLights[i].axis)/length(L) * length(-uLights[i].axis);
            float angle = acos(dotLVector);
            intensity = 0.0;
            if(radians(uLights[i].apperture) > angle)
                intensity = pow(cos(angle), uLights[i].cutoff);
        }
        else {
            intensity = 1.0;
        }

        vec3 diffuse = max(dot(L,N), 0.0) * diffuseColor * intensity;
        vec3 specular = pow(max(dot(N,H), 0.0), uMaterial.shininess) * specularColor * intensity;

        if(dot(L,N) < 0.0) {
            specular = vec3(0.0);
        }

        gl_FragColor += vec4(ambientColor + diffuse + specular, 1.0);
```

A luz *Spotlight*, à semelhança da luz pontual, tem um ponto emissor de luz, daí o “L” ser atualizado juntamente com o da luz pontual. Além disto, como analisado nas aulas teóricas, os seus valores *“diffuse”* e *“specular”* são alterados consoante o ângulo da direção central que define para onde a luz aponta, o ângulo de abertura (“*apperture”*) que define no espaço uma região cónica, em graus e convertido para radianos e o parâmetro de decaimento (“intensity”) que atenua a intensidade da luz à medida que nos afastamos da direção central.

Explicar como implementou o desafio. Ou seja, o tratamento que é dado ao input gerado pelo rato e qual a sua implicação nas variáveis do programa.

```
//Camera Movement
//Adjust natural camera movement
const MVIEW_LIM = 0.25;
//Rotation angle limit
const CAM_SPINS = 1;
const ROT_LIM = CAM_SPINS * 180; // 180 means one spin (rotation from -180 to 180)
//Sensivity factor Attenuant
const SENSE_FACTOR = 45;
```

```
function getCursorPosition(canvas, event)
{
    const mx = event.offsetX;
    const my = event.offsetY;

    const x = ((mx / canvas.width * 2) - 1);
    const y = (((canvas.height - my)/canvas.height * 2) - 1);

    return vec2(x,y);
}

canvas.addEventListener("mousedown", function(event) {
    mouseIsDragging = true;
    initMousePos = getCursorPosition(canvas, event);
});

canvas.addEventListener("mouseup", function(event) {
    mouseIsDragging = false;
});

canvas.addEventListener("mousemove", function(event) {
    const pos = getCursorPosition(canvas, event);
    if (mouseIsDragging) {
        var dy = (pos[1] - initMousePos[1]) * options.mouseSensivity * SENSE_FACTOR/2;
        var dx = (pos[0] - initMousePos[0]) * options.mouseSensivity * SENSE_FACTOR;
        // update the latest angle
        camera.Theta > ROT_LIM * camera.Agility ? camera.Theta = ROT_LIM * camera.Agility : camera.Theta -= dy;
        camera.Theta < -ROT_LIM * camera.Agility ? camera.Theta = -ROT_LIM * camera.Agility : camera.Theta -= dy;
        camera.Gama < -ROT_LIM * camera.Agility ? camera.Gama = -ROT_LIM * camera.Agility : camera.Gama += dx;
        camera.Gama > ROT_LIM * camera.Agility ? camera.Gama = ROT_LIM * camera.Agility : camera.Gama += dx;
    }
    initMousePos[0] = pos[0];
    initMousePos[1] = pos[1];
});
```

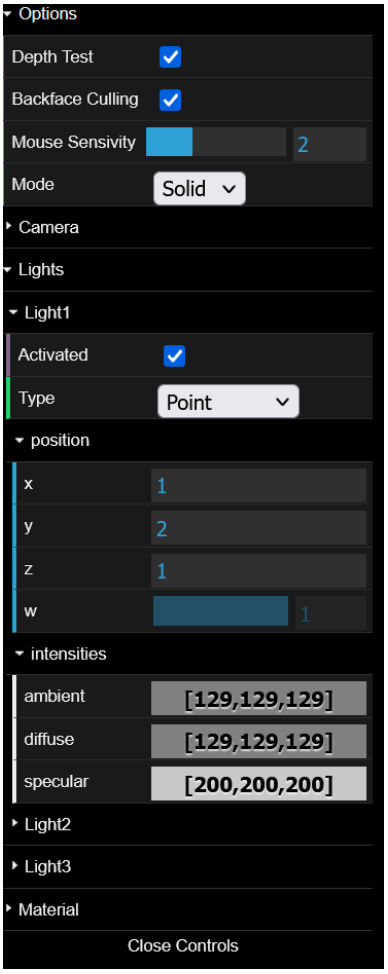
```
function render()
{
    mProjection = perspective(camera.Fovy, aspect, camera.Near, camera.Far);

    if(mView[0][0] < MVIEW_LIM || mView[0][0] > -MVIEW_LIM) { //So that the rotation feels natural
        mView = mult(lookAt([camera.eye[0], camera.eye[1], camera.eye[2]],
        [camera.at[0], camera.at[1], camera.at[2]],
        [camera.up[0], camera.up[1], camera.up[2]]),
        mult(rotateZ(camera.Theta), rotateY(camera.Gama)));
    } else {
        mView = mult(lookAt([camera.eye[0], camera.eye[1], camera.eye[2]],
            [camera.at[0], camera.at[1], camera.at[2]],
            [camera.up[0], camera.up[1], camera.up[2]]),
            mult(rotateY(camera.Gama), rotateX(camera.Theta)));
    }
```

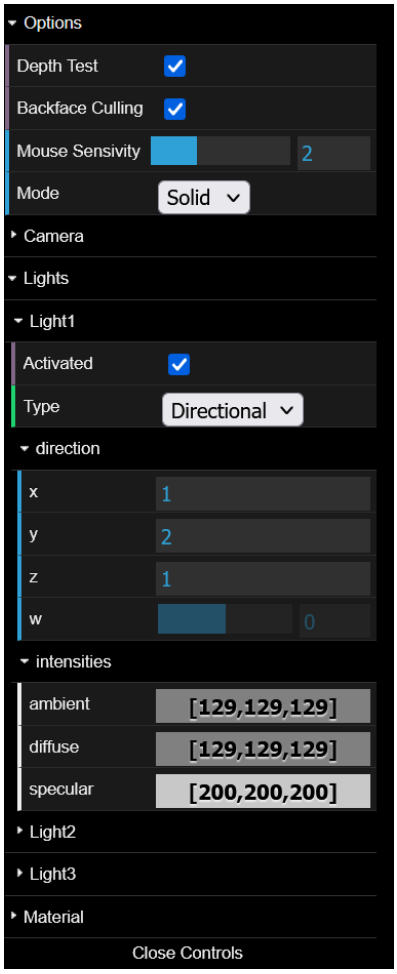
Primeiramente, para a implementação deste desafio, foi utilizada a função *"getCursorPosition"* para obtermos a posição do rato no momento em que o utilizador clica no botão do rato. De seguida, cada vez que há um movimento do rato enquanto o botão está premido no canvas, é registada a posição que é multiplicada por um fator de sensibilidade, após ter sido subtraída a posição do clique inicial. Desta forma conseguimos ajustar o ângulo

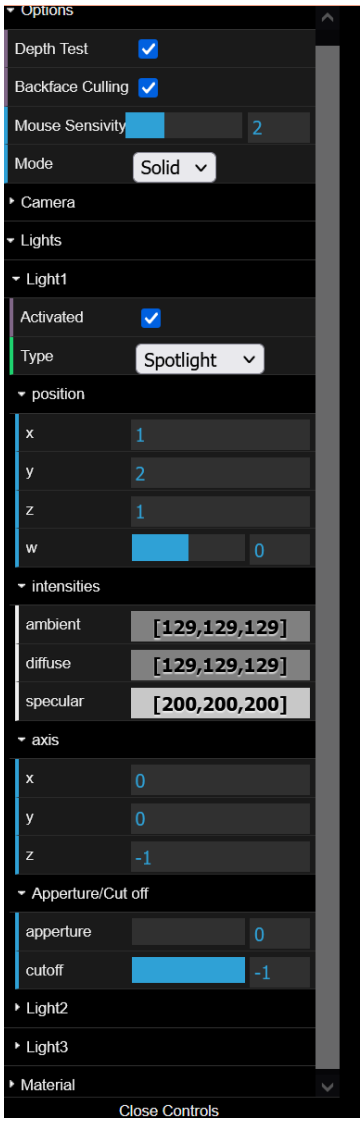
do movimento à mView atual. No render é ainda aplicada uma verificação para que o movimento seja sempre consistente com o input do utilizador, fazendo com que o movimento sinta natural ao arrastar. Esta alteração foi necessária, visto que o plano inclinava para um lado diferente ao arrastado em determinadas posições, devido à utilização implementada da mView.

Eventuais comentários que julgue pertinentes para entender o programa.

Neste menu é possível verificar a escolha do tipo de luz implementado, sendo as pastas e os parâmetros são alterados, consoante o mesmo. (3 imagens da mesma luz, variando o *"Type"* escolhido.







Foi adicionado ainda uma forma de controlar a sensibilidade do rato ao arrastar, controlando a velocidade do movimento da câmara em *"Mouse Sensitivity"* e o número de rotações que o plano consegue fazer em *"Max Rotations"*.