



LUCAS SIMÕES DE ALMEIDA

MATRÍCULA - 1712101

PROJETO 4

RENDERIZAÇÃO DE CENA COM REFLEXÃO E/OU SOMBRA

Relatório da Disciplina Computação
INF1761, Segundo Semestre do
ano de 2021.

Professor: Waldemar Celes

RIO DE JANEIRO

2021

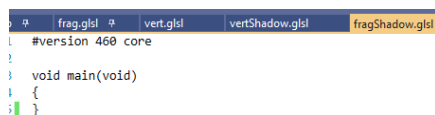
Neste projeto, tivemos que renderizar uma cena tridimensional, que fosse pelo menos composta de dois dos seguintes itens:

- Geração de reflexão planar
- Geração de sombra planar
- Textura projetiva (projeção de uma imagem nos objetos)
- Geração de sombra com shadow mapping

Desses itens eu escolhi focar na textura projetiva e na geração de sombra através de shadow mapping, no momento em que escrevo esse texto, apenas consegui implementar esses dois itens dos quatro, porém gostaria de implementar a reflexão planar também, caso haja tempo o suficiente.

Nesta cena será necessária a utilização de dois programas de shaders, um dedicado para as sombras, e o outro que compõe a cena com a textura projetiva, o programa de shader que faz as sombras é bem simples:

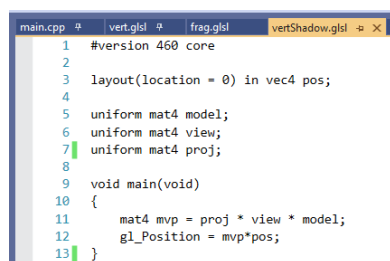
Imagem 1 -



```
#version 460 core

void main(void)
{
}
```

Imagem 2 -



```
#version 460 core

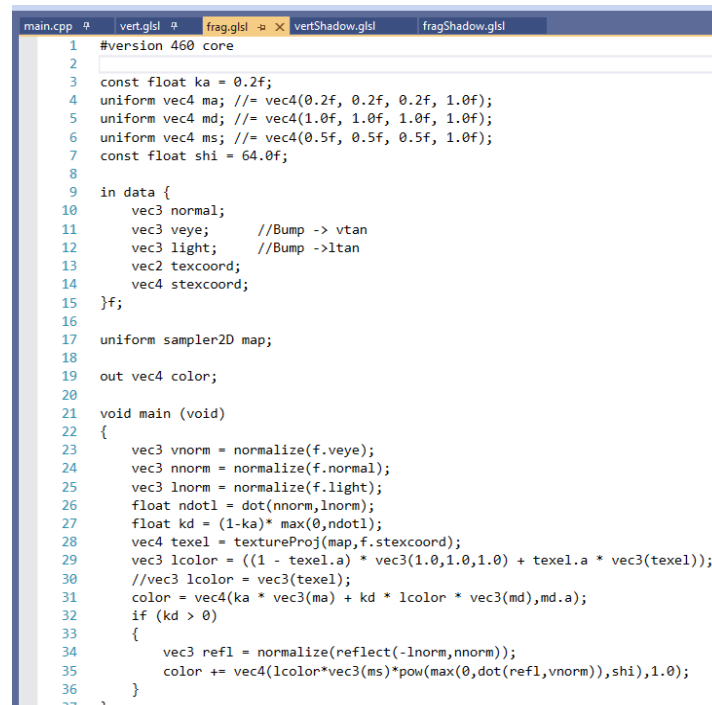
layout(location = 0) in vec4 pos;

uniform mat4 model;
uniform mat4 view;
uniform mat4 proj;

void main(void)
{
    mat4 mvp = proj * view * model;
    gl_Position = mvp*pos;
}
```

O programa que compõe a cena é um shader mais complexo, contendo a textura projetada, a reflexão de luz e todo o cálculo necessário para as luzes.

Imagem 3 -



```
#version 460 core

const float ka = 0.2f;
uniform vec4 ma; // = vec4(0.2f, 0.2f, 0.2f, 1.0f);
uniform vec4 md; // = vec4(1.0f, 1.0f, 1.0f, 1.0f);
uniform vec4 ms; // = vec4(0.5f, 0.5f, 0.5f, 1.0f);
const float shi = 64.0f;

in data {
    vec3 normal;
    vec3 veye; //Bump -> vtan
    vec3 light; //Bump -> ltan
    vec2 texcoord;
    vec4 stexcoord;
}f;

uniform sampler2D map;

out vec4 color;

void main (void)
{
    vec3 vnrm = normalize(f.veye);
    vec3 nnrm = normalize(f.normal);
    vec3 lnrm = normalize(f.light);
    float ndotl = dot(nnrm,lnrm);
    float kd = (1-ka)* max(0,ndotl);
    vec4 texel = textureProj(map,f.stexcoord);
    vec3 lcolor = ((1 - texel.a) * vec3(1.0,1.0,1.0) + texel.a * vec3(texel));
    //vec3 lcolor = vec3(texel);
    color = vec4(ka * vec3(ma) + kd * lcolor * vec3(md),md.a);
    if (kd > 0)
    {
        vec3 refl = normalize(reflect(-lnrm,nnrm));
        color += vec4(lcolor*vec3(ms)*pow(max(0,dot(refl,vnrm)),shi),1.0);
    }
}
```

Imagem 4 -

```
main.cpp  x vert.glsl  x vertShadow.glsl  fragShadow.glsl
3 layout(location = 0) in vec4 pos;
4 layout(location = 1) in vec3 normal;
5 layout(location = 2) in vec2 texcoord;
6
7
8 uniform vec4 leye; //light pos in eyespace
9 uniform mat4 svp;
10 uniform mat4 model;
11 uniform mat4 view;
12 uniform mat4 proj;
13
14 out data {
15     vec3 normal;
16     vec3 veye;      //Bump -> vtan
17     vec3 light;     //Bump -> ltan
18     vec2 texcoord;
19     vec4 stexcoord;
20 }v;
21
22 void main(void)
23 {
24     //mat4 view2 = M*view;
25     vec3 peye = vec3(model * view * pos);
26     if (leye.w == 0)
27         v.light = normalize(vec3(leye));
28     else
29         v.light = normalize(vec3(leye)-peye);
30
31     v.veye = - peye;
32     mat4 nm = transpose(inverse(model*view));
33     v.normal = normalize(vec3(nm*vec4(normal,0.0f)));
34     mat4 mvp = proj * view * model;
35     v.texcoord = texcoord;
36     //vec4 modelPos = model*pos;
37     v.stexcoord = svp*pos;
38     gl_Position = mvp*pos;
39 }
```

Agora indo para a main, iremos precisar de duas funções de desenho, na primeira iremos desenvolver nosso shadow mapping, acionamos um frame buffer de objeto (fbo), criado na initialize, informando ao programa que tudo dessa função será escrito nesse buffer, enviamos ao shader da cena uma svp, uma matriz que criada com a sview e sproj, contendo informações necessárias para o mapeamento de sombra e a textura projetiva, em seguida aplicamos o depth texture criado na initialize em todos os objetos que serão desenhados.

Imagem 5 -

```
/*
static void createShadowMap(void)
{
    glBindFramebuffer(GL_FRAMEBUFFER, fbo);
    glDrawBuffer(GL_NONE); //sem buffer de cor
    glReadBuffer(GL_NONE);
    //light "camera"
    sproj = glm::perspective(
        glm::radians(40.0f), 1.0f, 1.0f, 10.0f
    );
    sview = glm::lookAt(
        glm::vec3(light[0], light[1], light[2]), //lightpos
        glm::vec3(0.0f, 0.0f, 0.0f), //center
        glm::vec3(0.0f, 2.0f, 0.0f) //up
    );
    glViewport(0, 0, DIM, DIM); //shadow map dimension
    glClear(GL_DEPTH_BUFFER_BIT);

    //-----Matriz de proj-----//
    //-----//
    loadProjectionMatrix(f_pid);

    glUseProgram(s_pid);
    GLint loc_view = glGetUniformLocation(s_pid, "view");
    GLint loc_proj = glGetUniformLocation(s_pid, "proj");

    glUniformMatrix4fv(loc_view, 1, GL_FALSE, glm::value_ptr(view));
    glUniformMatrix4fv(loc_proj, 1, GL_FALSE, glm::value_ptr(proj));

    //-----Bind da Textura de profundidade -----//
    //-----//
    GLint loc_sampler = glGetUniformLocation(f_pid, "map");
    glUniform1i(loc_sampler, 0);
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, depth);
}
```

A segunda função de desenho é a que estamos acostumados a manipular, precisamos primeiramente alterar o framebuffer para 0, e resetar nossas configurações de janela ao padrão, esta função irá aplicar textura com a imagem desejada, igual no projeto 2, porém com a significativa diferença de que as coordenadas de textura são disponibilizadas pela projeção.

Imagem 6 -

```
static void createScene(void)
{
    glBindFramebuffer(GL_FRAMEBUFFER, 0);
    glDrawBuffer(GL_BACK);
    glViewport(0, 0, w_s, h_s); //window dimension
    glClearColor(0.8f, 1.0f, 1.0f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glUseProgram(f_pid);
    GLint loc_view = glGetUniformLocation(f_pid, "view");
    GLint loc_proj = glGetUniformLocation(f_pid, "proj");
    GLint loc_sampler = glGetUniformLocation(f_pid, "map");
    glUniformMatrix4fv(loc_view, 1, GL_FALSE, glm::value_ptr(view));
    glUniformMatrix4fv(loc_proj, 1, GL_FALSE, glm::value_ptr(proj));

    //-----Bind Proj Texture Esferas-----//
    //-----//
    glUniform1i(loc_sampler, 1);
    glActiveTexture(GL_TEXTURE1);
    glBindTexture(GL_TEXTURE_2D, texSmile);
}
```

Imagem 7 -

```

//-----Bind Proj Texture Mesa-----//
//-----//
glUniform1i(loc_sampler, 0);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, depth);
//-----Criando Mesa-----//
//-----//
glm::mat4 mesa_model = glm::translate(glm::mat4(1.0),
    glm::vec3(0.0f, 0.0f, -1.0f));
mesa_model = glm::scale(mesa_model, glm::vec3(5.5f, 5.5f, 5.5f));
mesa_model = glm::rotate(mesa_model, 90.0f, glm::vec3(1.0, 0.0f, 0.0f));
glm::vec4 maMesa(0.2f, 0.2f, 0.2f, 1.0f);
glm::vec4 mdMesa(1.0f, 1.0f, 1.0f, 1.0f);
glm::vec4 msMesa(0.5f, 0.5f, 0.5f, 1.0f);

loadMaterial(f_pid, f_index, 3, maMesa, mdMesa, msMesa);
loadMatrices(f_pid, mesa_model);

//glBindVertexArray(cube_vao);
//glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_INT, 0);
//glBindVertexArray(quad_vao);
glUseProgram(f_pid);
drawQuadScreen();
}
```

Vale constar que é necessária declarar nessa função de desenho, em um dos objetos desenhados, uma textura depth que irá aplicar o mapeamento de sombras na cena.

Vale constar que tomei a liberdade de testar aplicações de materiais diferentes nos objetos desenhados e até cheguei a testar múltiplas texturas em uma cena.

Resultado Final:

Imagem 8 –

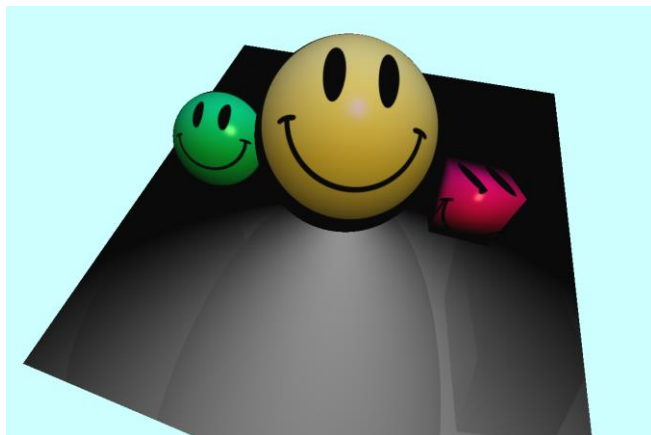


Imagem 9 –

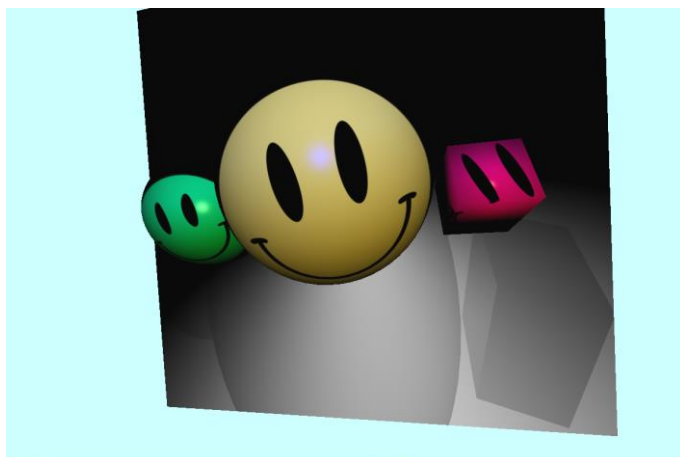
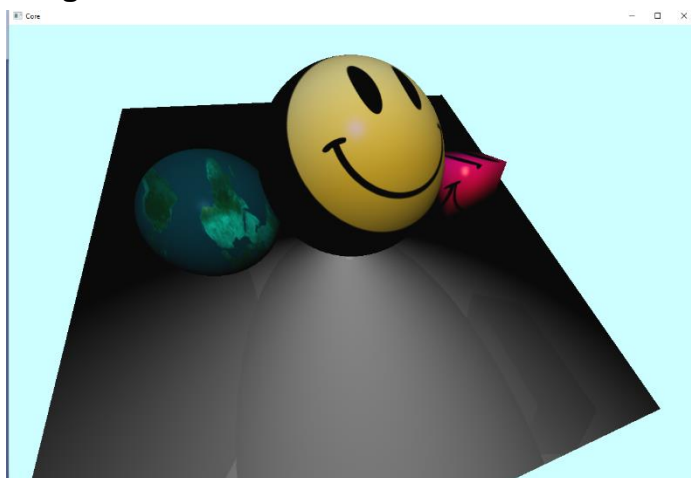


Imagem 10 –



Gostaria de agradecer ao professor **Waldemar Celes**, pela oportunidade e por estar sempre disponível para esclarecer dúvidas pontuais.