PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

LUCAS SIMÕES DE ALMEIDA

MATRÍCULA - 1712101

PROJETO 2

CENA RENDERIZADA COM PIPELINE PROGRÁMAVEL

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 utilizando um pipeline programável e através da biblioteca openGL, programar em placa gráfica, esta cena deve ter obrigatoriamente: instâncias de cubos e esferas, iluminação por pixel, e mapeamento de textura de pelos menos 2 objetos da cena.

Meu maior problema inicial eram os shaders, eu criei diversos tipos de shaders para teste, pois acreditava que meus shaders eram o problema principal do código, afinal quando comecei, se quer conseguia gerar um objeto.

```
static void initialize()
{
    GLenum err = glewInit();
    if (GLEW_OK != err) {
        fprintf(stderr, "GLEW Error: %s\n", glewGetErrorString(err));
        exit(-1);
    }
    printf("OpenGL version: %s\n", glGetString(GL_VERSION));

    glClearColor(1.0f,1.0f,1.0f,1.0f);          //Inicializa com a cor de tela branca
    glEnable(GL_DEPTH_TEST);

    sphere_vao = createSphere(R,R);             // Cria vao de esfera
    cube_vao = createCube(0.0f,0.0f,0.0f,0.5f); // Cria Vao de cubo no centro
    setProgram();                               // Cria f_pid

}
```

Essa é a função de inicializar, ela cria tudo que será necessário na função display, o programa de shader, os VAO dos objetos utilizados e o mais importante, inicializa a glew.

```
static void display(void)//GLFWwindow * win
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    setCamera();
    createScene();
    glutSwapBuffers();
}
```

A função de display ficou extremamente curta, afinal ela só chama funções, a definição da câmera,a que cria a cena e a que atualiza a janela em tempo real.

Tive bastante dificuldade para enteder o processo de criação de objeto através do pipeline programável, mas quando finalmente entendi a coisa ficou bem mais natural:

```
static void createScene(void)
{
    //glm::mat4 model = glm::mat4(1.0f); // identity

    glm::vec4 leye = glm::vec4(0.0f, 0.0f, 0.0f, 1.0f);

    //----------------------Definindo material-------------------------------//
    //-----------------------------------------------------------------------//

    GLuint ubuffer = CreateBuffer(GL_UNIFORM_BUFFER,sizeof(Material), (GLvoid*)&mat);
    GLuint f_index = glGetUniformBlockIndex(f_pid, "Material");
    glBindBufferBase(GL_UNIFORM_BUFFER, 0, ubuffer);

    //-------------------------Criando esfera1-------------------------------//
    //-----------------------------------------------------------------------//

    //frag - based illumination
    glm::mat4 f_model = glm::translate(glm::mat4(1.0), // Altera posicao do desenho 3D
        glm::vec3(1.0f, 0.42f, 0.0f));
    f_model = glm::scale(f_model, glm::vec3(0.3f, 0.3f, 0.3f)); // altera escala do desenho
    glm::vec4 maTexture (0.2f, 0.2f, 0.2f, 1.0f);
    glm::vec4 mdTexture(1.0f, 1.0f, 1.0f, 1.0f);
    glm::vec4 msTexture(1.0f, 1.0f, 1.0f, 1.0f);

    GLuint tex = CreateTexture2D("earth.jpg");
    GLint loc_sampler = glGetUniformLocation(f_pid, "earth");
    glUseProgram(f_pid);
    glUniform1i(loc_sampler, 0);
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, tex);

    loadMaterial(f_pid, f_index, 0, maTexture, mdTexture, msTexture);
    loadMatrices(f_pid, f_model);

    GLint f_loc_leye = glGetUniformLocation(f_pid, "leye");
    glUseProgram(f_pid);
    glUniform4fv(f_loc_leye, 1, glm::value_ptr(leye));

    glBindVertexArray(sphere_vao);
    glDrawElements(GL_TRIANGLES, 6 * R * R, GL_UNSIGNED_INT, 0);
```

Esse trecho, é o ínicio da create scene, a parte mais importante desse trecho é que utilizo do “leye” para definir a sua posição, mas como só precisamos definir ela uma vez para cada programa de shader utilizado, a criação de objetos depois disso pode ser resumida dessa forma:

```
//-------------------------Criando esfera2-------------------------------//
//-----------------------------------------------------------------------//
glm::mat4 v_model = glm::translate(glm::mat4(1.0),
    glm::vec3(-1.0f, 0.42f, 0.0f));
v_model = glm::scale(v_model, glm::vec3(0.3f, 0.3f, 0.3f));

GLuint tex1 = CreateTexture2D("java-logo.jpg");
GLint loc_sampler1 = glGetUniformLocation(f_pid, "java-logo");
glUseProgram(f_pid);
glUniform1i(loc_sampler1, 0);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, tex1);

loadMaterial(f_pid, f_index, 1, maTexture, mdTexture, msTexture);
loadMatrices(f_pid, v_model);

glBindVertexArray(sphere_vao);
glDrawElements(GL_TRIANGLES, 6 * R * R, GL_UNSIGNED_INT, 0);
```

Precisamos sempre bindar após definir essas mudanças glUniform*, pois o shader precisa saber a qual objeto aquela alteração foi aplicada. Se não utilizarmos do bind, a ultima alteração, modificara todo o projeto.

Podemos ver o resultado final abaixo:

Imagem 1 -

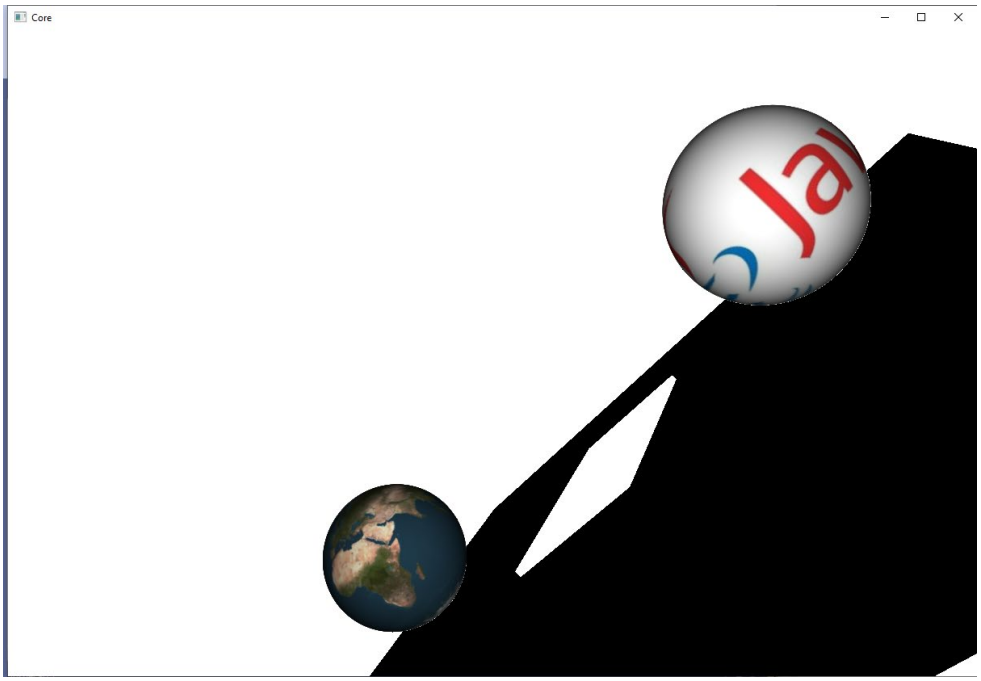


Imagem 2 -

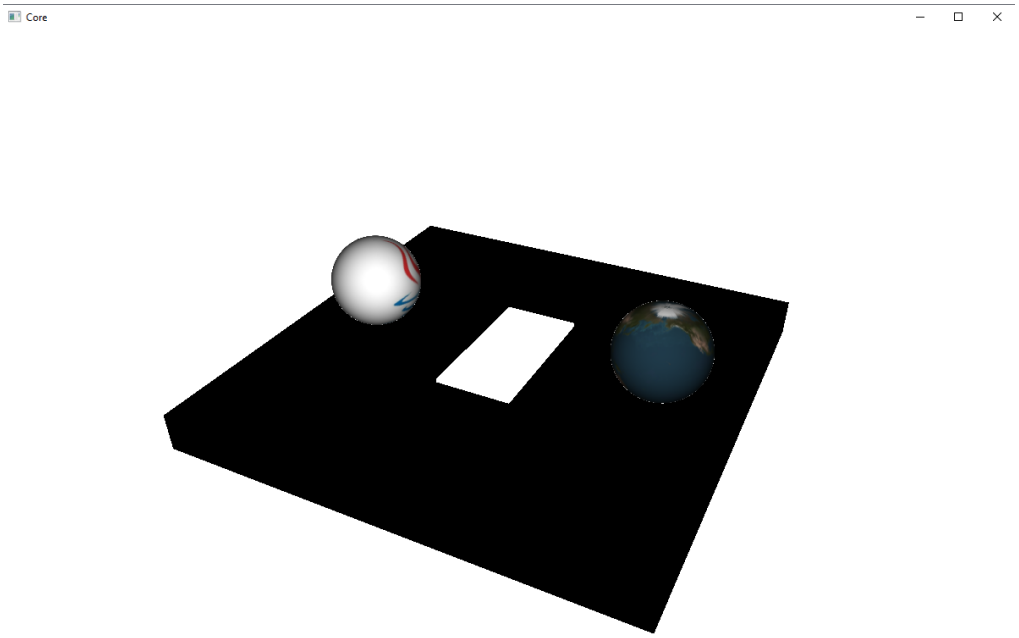
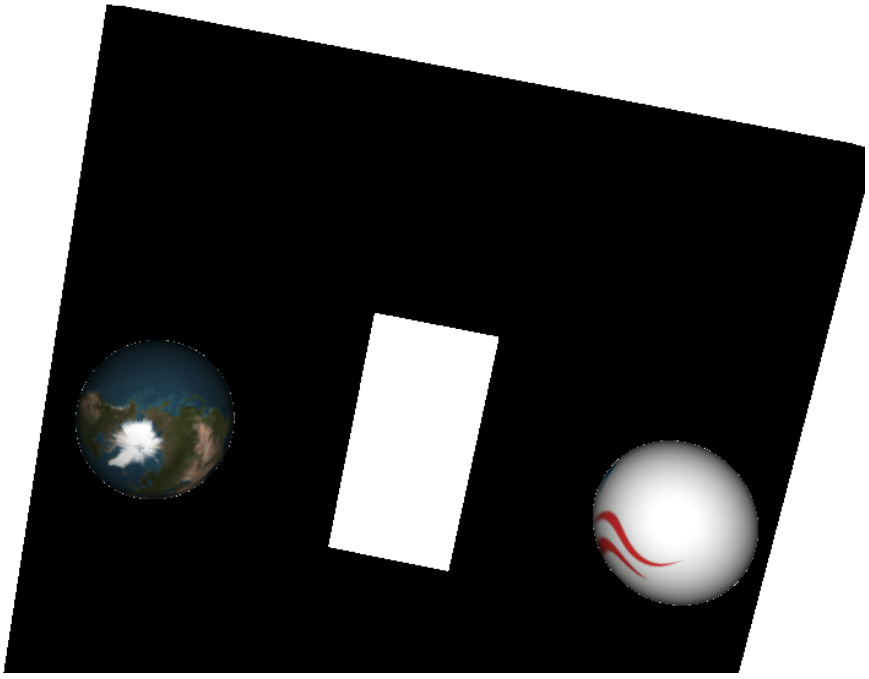


Imagem 3 -



Ficou parecendo uma tomada analisando pela vista superior, graças a implementação do controle de câmera arcball, é possível ver as cenas de qualquer ângulo, utilizei a estrutura da minha implementação do projeto 1 porém a modifiquei para funcionar no novo pipeline.

Sobre as texturas testei diversas imagens, inclusive uma com o meu rosto, e todas deram certo, porém como já estava atrasado com a entrega, não desenvolvi uma maneira de aplicar texturas aos objetos cubo escalonados, ou seja, a mesa e a folha de papel. Mas consegui aplicar às esferas, então utilizei 2 texturas diferentes para ter certeza que estava tudo okay.

Concluindo, o trabalho foi extremamente complexo no início, tive diversas dificuldades, e eu quase desisti, mas o resultado final valeu muito a pena e fico muito feliz que tenha seguido em frente. É bem complicado entender como o pipeline programável funciona, mas as peças acabam se encaixando eventualmente e a coisa fica bem mais natural.

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