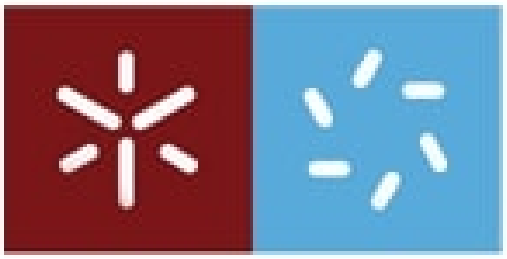

Universidade do Minho
Escola de Ciências

Computação Gráfica

Jorge Daniel Pereira Silva (a80931)
José Pedro Silva Ferreira (a96798)
Rui Alexandre Dias Neto (a80433)

26 de maio de 2024

Conteúdo

1 Introdução

Nesta quarta fase do trabalho prático, foi pedido que fossem geradas coordenadas para textura e normais para cada vértice. Foi também pedido que no *engine* fossem ativadas as funcionalidades de luz e textura, bem como ler e aplicar as normais e coordenadas de textura dos arquivos do modelo XML.

2 Engine

Nesta aplicação implementamos uma classe *colors* e uma classe *lights*.

2.1 Colors

Nesta classe continuamos com a mesma consistência das outras fases, onde criamos estruturas de dados para guardar a informação lida no ficheiro XML(figura 1).

```
vector<float> diffuse = {200, 200, 200};
vector<float> ambient = {50, 50, 50};
vector<float> specular = {0, 0, 0};
vector<float> emissive = {0, 0, 0};
float shininess = 0;
```

Figura 1: Colors

2.2 Lights

Nesta classe, tal como na *colors*, criamos estruturas de dados para guardar informação lida no ficheiro XML(figura 2).

```
vector<float> directional;
vector<float> positional;
string type;
float cutoff;
```

Figura 2: Lights

2.3 Main

Tivemos que criar uma função *DrawLigths*(Figura 3) para desenhar as luzes e a *DrawModels*(Figura 4) alteramos para conseguirmos desenhar as cores.

```
void drawLight(Lights light){
  string type = light.get_type();
  if(type=="directional"){
    glLightfv(GL_LIGHT0, GL_POSITION, fromVectorToArray(light.get_lights_directional()));
  } else if(type=="positional"){
    glLightfv(GL_LIGHT0, GL_POSITION, fromVectorToArray(light.get_lights_positional()));
  } else if(type=="spot"){
    glLightfv(GL_LIGHT0, GL_POSITION, fromVectorToArray(light.get_lights_positional()));
    glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, fromVectorToArray(light.get_lights_directional()));
    glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, light.get_cutoff());
  }

}
```

Figura 3: DrawLights

```
// desenhar o model atraves dos vbos
void drawModel(Model model) {
  Colors colors = model.get_colors();

  glMaterialfv(GL_FRONT, GL_DIFFUSE, fromVectorToArray(colors.get_diffuse()));
  glMaterialfv(GL_FRONT, GL_AMBIENT, fromVectorToArray(colors.get_ambient()));
  glMaterialfv(GL_FRONT, GL_SPECULAR, fromVectorToArray(colors.get_specular()));
  glMaterialfv(GL_FRONT, GL_EMISSION, fromVectorToArray(colors.get_emissive()));
  glMaterialf(GL_FRONT, GL_SHININESS, colors.get_shininess());
  // Set buffer active and define the semantics
  glBindBuffer(GL_ARRAY_BUFFER, model.get_points());
  glVertexPointer(3, GL_FLOAT, 0, 0);
  // Draw the model using the index buffer
  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, model.get_index());
  glDrawElements(GL_TRIANGLES, model.get_numPoints() * 3, GL_UNSIGNED_INT,
                 NULL);
}
```

Figura 4: DrawModels

3 Conclusão

Ao longo da realização desta fase do projeto, lidamos com as metodologias e procedimentos necessários para a implementação da iluminação, culminando na criação e utilização da nossa classe *Lights*.
Embora gostássemos de ter conseguido completar os requisitos apresentados no enunciado do trabalho prático, a nossa má gestão do tempo tornou esse objetivo inalcançável.
Enunciaremos agora alterações/adições ao nosso projeto que gostaríamos de ter implementado. Algumas dessas mudanças seriam a aplicação de texturas específicas a cada planeta, o cálculo das normais (através das fórmulas conhecidas do nosso material de estudo e das aulas) e a sua respetiva inclusão num número adequado de VBOs.
Concluindo, com a realização deste trabalho prático, independentemente das dificuldades encontradas, sentimos que desenvolvemos competências na área da Computação Gráfica, no que toca à leitura de ficheiros XML, à geração e renderização de primitivas gráficas, à animação das nossas *scenes* e também à utilização das ferramentas do OpenGL.