Universidade do Minho
Escola de Engenharia

Computação Gráfica

Fase 4
Grupo 7

Braga, Junho de 2023

Bernardo Amado Pereira da Costa, A95052
Eduardo Miguel Pacheco Silva, A95345
José Carlos Gonçalves Braz, A96168

Índice

1. Introdução

Nesta quarta fase, foi nos pedida a implementação das texturas e normais de forma a produzir os efeitos luminosos pretendidos no sistema através da declaração dos ficheiros de texturas e propriedades das normais no ficheiro de configuração com extensão *.xml.*

A implementação destas novas *features* assentou na utilização do pensamento lógico e infraestrutura criada em fases anteriores tais como o uso dos *VBOS* para implementação das texturas de forma mais eficiente bem como o suporte para as diferentes tipologias de luzes pela aplicação da mesma prática de extração da informação relativa no ficheiro de configuração.

2. Luzes

Os conceitos atómicos para a iluminação do cenário baseiam-se no cálculo das normais para cada figura geométrica (variação dependendo da tipologia da figura geométrica).

No caso da existência do terreno irregular (uma esfera com *specular mapping* por exemplo (uma das funcionalidades adicionais implementadas na fase anterior)), foi utilizada uma técnica de obter o vetor normal através do cálculo do produto escalar com quatro pontos.

2.1. Tipologias de luzes

Como é descrito no enunciado, é pretendido o suporte de três tipologias diferentes de luz: *Point Light, Directional Light e SpotLight.*

Dada a existência de propriedades similares na declaração das propriedades das luzes no ficheiro de configuração, de forma a tornar a implementação mais lógica, foi criada uma classe geral que, numa fase mais avançada, será referenciada na especificação de cada tipologia.

```
1  class Light {
2      public:
3          virtual ~Light() = default;
4          virtual void run(GLuint light) = 0;
5  };
```

Figura 1 - Classe Light

2.1.1. *Point Light*

A primeira tipologia de luz que pode ser especificada no ficheiro de configuração denomina-se *Point Light*. Este conceito pretende retratar a emissão da luz através de um único ponto fixo para todas as direções possíveis. Por norma, a intensidade da luz decresce consoante a distância de um dado objeto à fonte de luz.

```
1  class PointLight : public Light {
2      private:
3          GLfloat position[4];
4      public:
5          PointLight(XMLElement * pointLight);
6          void run(GLuint light) override;
7  };
8
9  PointLight::PointLight(XMLElement *point_light) {
10     this->position[0] = point_light->FloatAttribute("PosX");
11     this->position[1] = point_light->FloatAttribute("PosY");
12     this->position[2] = point_light->FloatAttribute("PosZ");
13     this->position[3] = 1.0f;
14 }
15
16 void PointLight::run(GLuint light) {
17     GLfloat amb[4] = {0.1, 0.1, 0.1, 1.0};
18     GLfloat diff[4] = {1.0, 1.0, 1.0, 1.0};
19
20     glLightfv(light, GL_POSITION, this->position);
21     glLightfv(light, GL_AMBIENT, amb);
22     glLightfv(light, GL_DIFFUSE, diff);
23 }
```

Figura 2 - classe PointLight e implementação

2.1.2. *Directional Light*

A segunda tipológica de luz denominada *Directional Light* baseia-se na presença uniforme da luz no cenário numa única direção. Em comparação com a *Point Light,* a intensidade da luz não é alterada com a distância de um dado objeto à fonte de luz.

```
class DirectionalLight : public Light {
    private:
        GLfloat direction[4];
    public:
        DirectionalLight(XMLElement  * directionalLight);
        void run(GLuint light) override;
};

DirectionalLight::DirectionalLight(XMLElement *directional_light) {
    this->direction[0] = directional_light->FloatAttribute("DirX");
    this->direction[1] = directional_light->FloatAttribute("DirY");
    this->direction[2] = directional_light->FloatAttribute("DirZ");
    this->direction[3] = 0.0f;
}

void DirectionalLight::run(GLuint light) {
    GLfloat amb[4] = {0.8, 0.8, 0.8, 1.0};
    GLfloat diff[4] = {0.8f, 0.8f, 0.8f, 1.0f};
    glLightfv(light, GL_POSITION, this->direction);
    glLightfv(light, GL_AMBIENT, amb);
    glLightfv(light, GL_DIFFUSE, diff);
}
```

Figura 3 - classe DirectionalLight e implementação

2.1.2. *SpotLight*

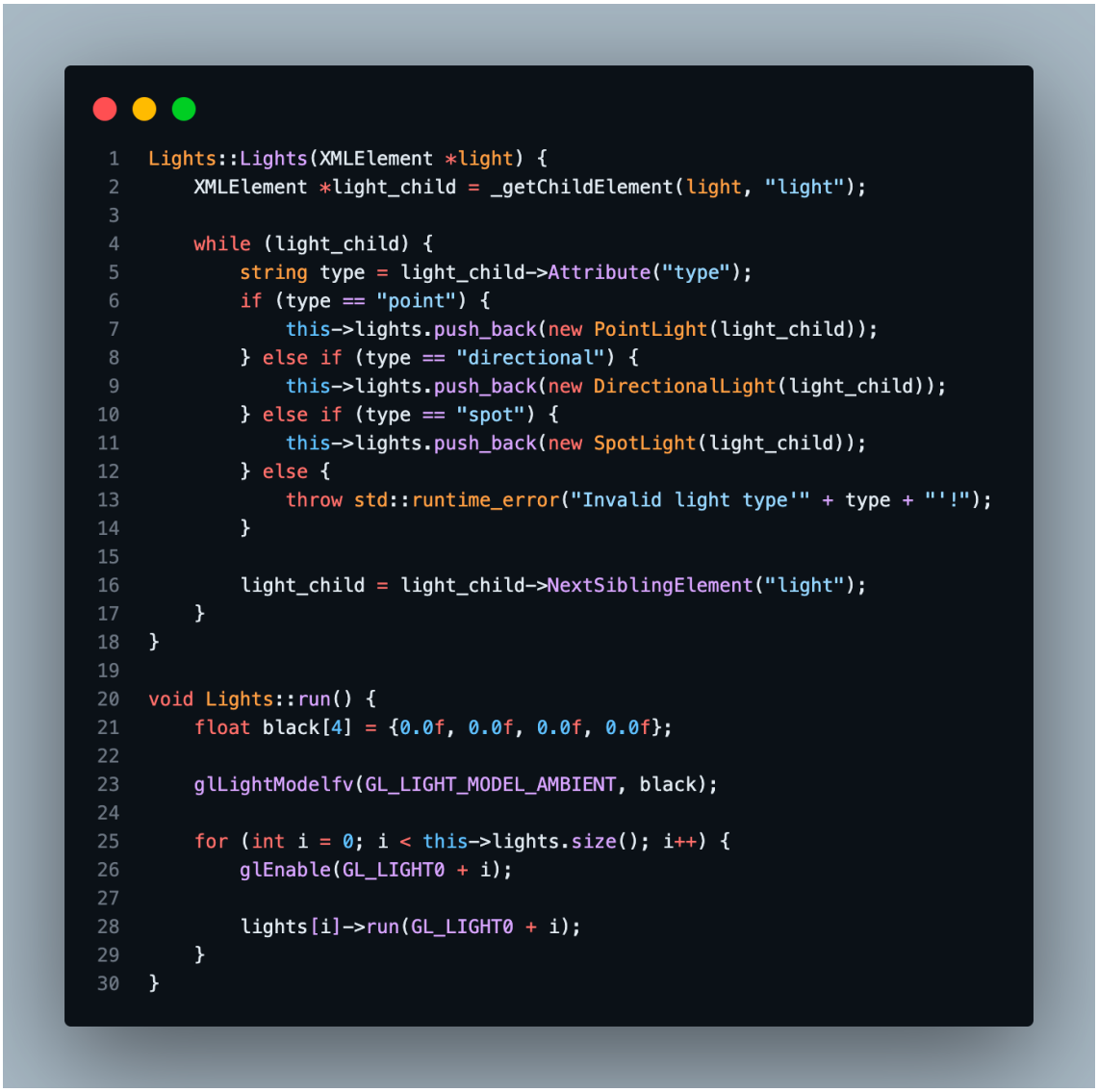
A terceira tipológica de luz denominada *SpotLight* apresenta características similares à *Point Light* dada a transmissão da luz através de um ponto fixo com a variação da intensidade dependente da distância do objeto à fonte luminosa. As diferenças baseiam-se no formato de um "cone" similar a uma lanterna.

```
class SpotLight : public Light {
    private:
        GLfloat point[4];
        GLfloat direction[4];
        float cutoff;
    public:
        SpotLight(XMLElement * spotLight);
        void run(GLuint light) override;
};

SpotLight::SpotLight(XMLElement *spot_light) {
    this->position[0] = spot_light->FloatAttribute("PosX");
    this->position[1] = spot_light->FloatAttribute("PosY");
    this->position[2] = spot_light->FloatAttribute("PosZ");
    this->position[3] = 1.0f;

    this->direction[0] = spot_light->FloatAttribute("DirX");
    this->direction[1] = spot_light->FloatAttribute("DirY");
    this->direction[2] = spot_light->FloatAttribute("DirZ");
    this->direction[3] = 0.0f;

    this->angle = spot_light->FloatAttribute("Angle");
    this->exponent = spot_light->FloatAttribute("Exponent");
}

void SpotLight::run(GLuint light) {
    GLfloat amb[4] = {0.1, 0.1, 0.1, 1.0};
    GLfloat diff[4] = {1.0, 1.0, 1.0, 1.0};

    glLightfv(light, GL_POSITION, this->position);
    glLightfv(light, GL_SPOT_DIRECTION, this->direction);
    glLightf(light, GL_SPOT_CUTOFF, this->angle);
    glLightf(light, GL_SPOT_EXPONENT, this->exponent);
    glLightfv(light, GL_AMBIENT, amb);
    glLightfv(light, GL_DIFFUSE, diff);
}
```

Figura 4 - classe SpotLight e implementação

2.2. Ficheiro de configuração

Após a implementação da infraestrutura, é usada a classe *Lights* de forma a haver a capacidade de suportar as diferentes tipologias apresentadas anteriormente.



```
Lights::Lights(XMLElement *light) {
    XMLElement *light_child = _getChildElement(light, "light");

    while (light_child) {
        string type = light_child->Attribute("type");
        if (type == "point") {
            this->lights.push_back(new PointLight(light_child));
        } else if (type == "directional") {
            this->lights.push_back(new DirectionalLight(light_child));
        } else if (type == "spot") {
            this->lights.push_back(new SpotLight(light_child));
        } else {
            throw std::runtime_error("Invalid light type'" + type + "'!");
        }

        light_child = light_child->NextSiblingElement("light");
    }
}

void Lights::run() {
    float black[4] = {0.0f, 0.0f, 0.0f, 0.0f};

    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, black);

    for (int i = 0; i < this->lights.size(); i++) {
        glEnable(GL_LIGHT0 + i);

        lights[i]->run(GL_LIGHT0 + i);
    }
}
```

Figura 5 - classe Lights

3. Texturas

Repetindo o processo relatado anteriormente na secção das luzes, de forma a suportar a extração de informação relativa às texturas do ficheiro de configuração, procedemos à criação da classe *Texture.*

```
1  class Texture {
2     private:
3      GLuint texture;
4
5     public:
6      Texture() = default;
7      Texture(XMLElement *texture);
8      GLuint getTexture() { return texture; };
9  };
10
11 Texture::Texture(XMLElement *texture) {
12     this->texture =
13         GeometricShape::loadTextureImageVBO(texture->Attribute("file"));
14 }
```

Figura 6 - classe Texture e implementação

3.1. VBOS

Partindo da implementação dos *Virtual Buffer Objects* da fase anterior, decidimos aplicar os mecanismos relacionados com as texturas através dos *VBOS* dada a notável melhoria na eficiência no processo de *render* do cenário, processo que pode ser bastante exigente dada a maior carga de pixéis.

```
void GeometricShape::drawObjectVBOMode(vector<VBOStruct> vbos, GLuint texture) {
    glEnableClientState(GL_VERTEX_ARRAY);
    glEnableClientState(GL_NORMAL_ARRAY);

    for (auto vbo : vbos) {
        glBindBuffer(GL_ARRAY_BUFFER, vbo.points);
        glVertexPointer(3, GL_FLOAT, 0 ,0);

        if (vbo.normals != 0) {
            glBindBuffer(GL_ARRAY_BUFFER, vbo.normals);
            glNormalPointer(GL_FLOAT, 0, 0);
        }

        if (texture) {
            glBindBuffer(GL_ARRAY_BUFFER, texture);
            glTexCoordPointer(3, GL_FLOAT, 0, 0);
        }

        glDrawArrays(vbo.primitive, 0, vbo.size);
    }
}
```

Figura 7 - Função drawObjectVBOMode adaptada para texturas.

3.2. Aplicação

Após a integração com os *VBOS,* procedemos ao uso de recursos presentes nas bibliotecas *DevIL* e *OpenGL* para a possível integração no cenário final.

```
GLuint GeometricShape::loadTextureImageVBO(string pathFile) {
    GLuint texture;
    unsigned int t, tw, th;
    unsigned char *texData;
    ilGenImages(1, &t);
    ilBindImage(t);
    ilLoadImage((ILstring)pathFile.c_str());
    tw = ilGetInteger(IL_IMAGE_WIDTH);
    th = ilGetInteger(IL_IMAGE_HEIGHT);
    ilConvertImage(IL_RGBA, IL_UNSIGNED_BYTE);
    texData = ilGetData();

    glGenTextures(1, &texture);

    glBindTexture(GL_TEXTURE_2D, texture);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, tw, th, 0, GL_RGBA, GL_UNSIGNED_BYTE, texData);
    glGenerateMipmap(GL_TEXTURE_2D);

    return texture;
}
```

Figura 8 - leitura da textura do ficheiro para VBO

3. Funcionalidades adicionais

No enunciado anterior foi referida a implementação da estratégia de *picking*. Esta implementação serviu principalmente para permitir a identificação dos planetas do sistema solar através de um *click* por parte do utilizador no cenário.

Nesta fase, decidimos adaptar esta funcionalidade de forma a suportar os conceitos implementados anteriormente como as luzes e as texturas.

```
int picking(int xx, int yy, Camera *camera, Group *group) {
    // Turn off lighting and texturing
    glDisable(GL_LIGHTING);
    glDisable(GL_TEXTURE_2D);

    // Clear the frame buffer and place the camera
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    _3f p = camera->getPosition();
    _3f l = camera->getLookAt();
    _3f u = camera->getUp();
    gluLookAt(p.x, p.y, p.z, l.x, l.y, l.z, u.x, u.y, u.z);

    // Draw coded version of objects taking advantage of the values stored on
    // the depth buffer
    glDepthFunc(GL_LEQUAL);
    // draw
    mtEnable(MT_CODE_COLOUR_DRAW);
    group->run();
    mtDisable(MT_CODE_COLOUR_DRAW);
    glDepthFunc(GL_LESS);

    // Read pixel under mouse position
    GLint viewport[4];
    unsigned char res[4];
    glGetIntegerv(GL_VIEWPORT, viewport);
    glReadPixels(xx, viewport[3] - yy, 1, 1, GL_RGBA, GL_UNSIGNED_BYTE, res);

    // Reactivate lighting and texturing
    glEnable(GL_LIGHTING);
    glEnable(GL_TEXTURE_2D);

    if (picked) cout << "entrei picked: " << picked << endl;

    picked = res[0];
    // Return red color component
    return res[0];
}
```

Figura 9 - função picking

5. Resultados obtidos

5.1. *Luminosidade num objeto*

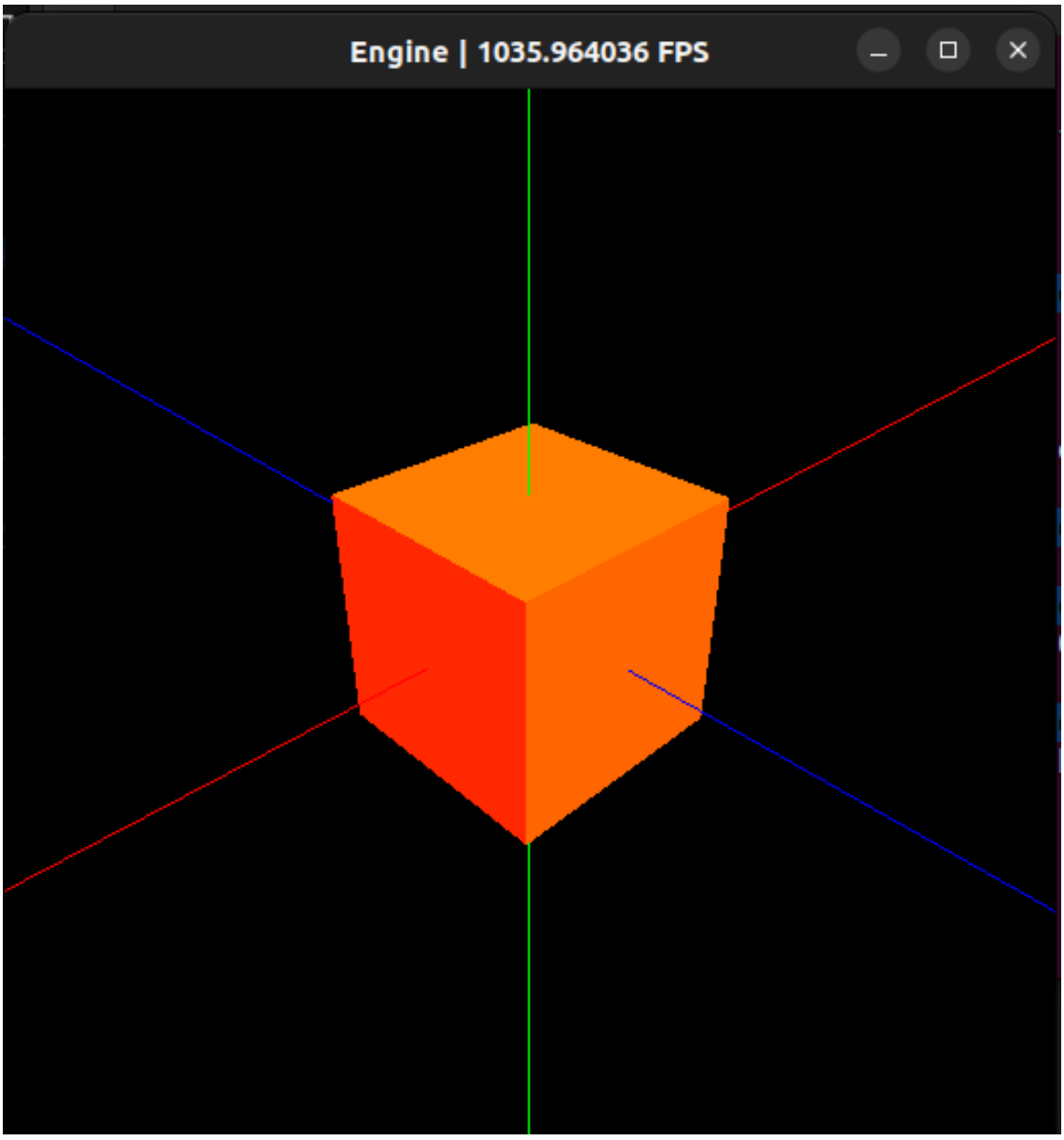


Figura 10 - Figura geométrica após implementação das luzes

5.2. *Sistema Solar*

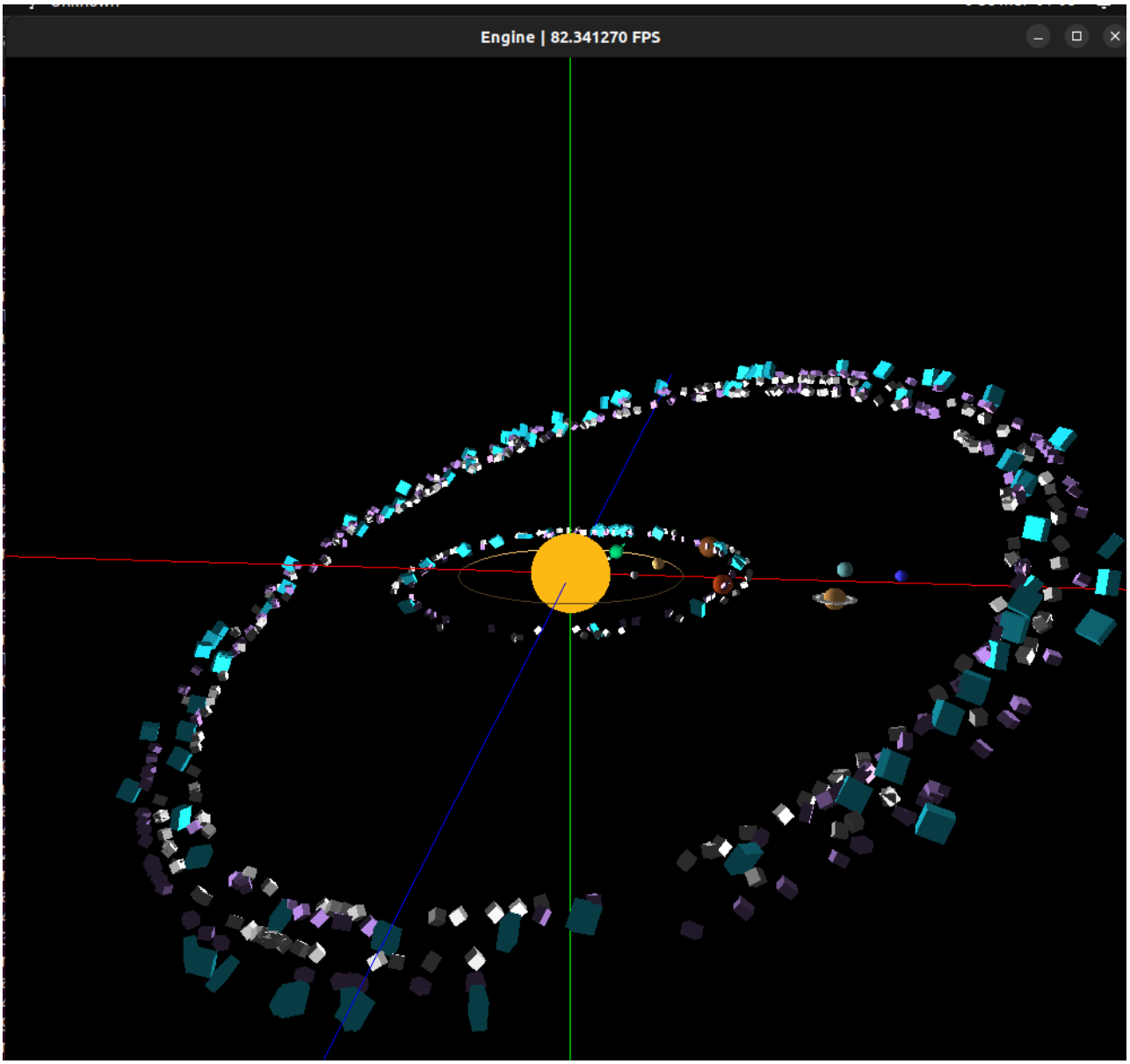


Figura 11 - Sistema Solar

6. Conclusão

Nesta fase final, exploramos em detalhe os conceitos relacionados com a iluminação do cenário e aplicação de texturas aos diferentes componentes presentes. Com esta implementação, desenvolvemos um conhecimento sólido dos conceitos teóricos tais como as diferentes tipologias de luz e as técnicas usadas para a replicação de texturas em larga escala sem pôr de parte a eficiência do processo.

Um aspeto que nos surpreendeu quando comparamos o trabalho que resultou desta fase final em relação à anterior é a grande diferença visual que a aplicação de luz no cenário pode causar.

Em suma, julgamos ter cumprido com os objetivos propostos e ter consolidado bastante a nossa aprendizagem na UC.