Universidade da Beira Interior
Departamento de Informática

Nome

Elaborado por:

Tiago Ramos 47883
Ricardo Matias 46586
Daniel Oliveira 48118
Francisco Gordino 48531

Orientador:

Professor Doutor Abel Gomes

9 de Janeiro

Agradecimentos

Agredecemos primeiramente ao nosso professor, D. Abel Gomes, pela oportunidade de podermos ampliar o nosso conhecimento na área de Computação Gráfica e também por podermos interagir com a linguagem C++ e com as bibliotecas do OpenGL, durante a realização deste trabalho assim como ao longo do semestre de modo a termos uma experiência mais técnica e próxima da realidade.

A sua abordagem didática, aliada à sua experiência na área, proporcionou um ambiente propício para o crescimento académico e profissional de cada membro do grupo. Para além disso gostariamos de deixar uma palavra de agradecimento aos nossos colegas de curso, pelas discussões saudáveis e ajuda fornecida em quaisquer dúvidas que tenham surgido e podessem ser debatidas de modo a interagirmos para um bem comum.

A todo o incentivo e apoio prestado, é guardado connosco em sentido de gratidão e cooperação.

Acrónimos

UBI - Universidade da Beira Interior
CG - Computação Gráfica
UC - Unidade Curricular

Conteúdo

1 / *Motivação*

Motivados por um semestre sob a orientação do Professor Doutor Abel Gomes, durante o qual fomos guiados no desenvolvimento de trabalhos e tarefas, adquirimos conhecimentos fundamentais sobre desenvolvimento gráfico. Sentindo a necessidade de aplicar todo o aprendizado adquirido, não poderíamos encontrar desafio mais adequado do que o presente projeto.

1.1 | Objetivos

Este projeto visa desenvolver uma representação gráfica do sistema solar por meio do OpenGL, em conjunto com diversas ferramentas e bibliotecas correlatas. A programação em C++ integra conceitos de renderização, manipulação de objetos e texturas, controles de câmera, e a utilização de shaders para criar uma experiência visual envolvente. Em projetos desse tipo, a aspiração comum é alcançar uma representação que se equipare a uma fotografia ou visão realista. Embora essa meta ainda não seja completamente atingida, a evolução nessa área está em constante avanço.

O tema selecionado para este projeto é a representação do sistema solar, aproveitando as ferramentas mencionadas para se aproximar ao máximo do objetivo proposto. A implementação apresenta uma aplicação interativa do sistema solar em escala, incorporando características realistas como rotação e órbita dos planetas. Este relatório explora diversos conceitos fundamentais, desde a configuração inicial do projeto até a renderização individual de cada planeta. Adicionalmente, são discutidas as estratégias adotadas para o controle de entrada do usuário, assim como a implementação de conceitos-chave, como transformações de matrizes e carregamento de texturas.

2 / *Tecnologias Utilizadas*

2.1 | Visual Studio Code

O Visual Studio Code é um ambiente de desenvolvimento leve e poderoso da Microsoft, conhecido por sua interface intuitiva e flexibilidade. Sua aparência limpa e moderna oferece uma experiência visual agradável. Destaca-se pela extensibilidade, permitindo aos usuários personalizarem o ambiente com diversas extensões. Oferece recursos avançados de edição de código, como destaque de sintaxe e sugestões inteligentes. Suporta integração com várias linguagens de programação, controle de versão e ferramentas de automação, tornando-o uma escolha popular entre os desenvolvedores.

2.2 | C++

C++ é uma linguagem de programação poderosa e amplamente utilizada, conhecida por sua eficiência e versatilidade. Projetada como uma extensão da linguagem C, o C++ oferece recursos adicionais, incluindo programação orientada a objetos, manipulação de memória e suporte para programação genérica.

Sua sintaxe é reconhecida por ser acessível aos programadores familiarizados com C, ao mesmo tempo em que introduz construções mais avançadas. C++ é frequentemente escolhido para o desenvolvimento de sistemas, jogos, aplicativos de desktop, e muitas outras aplicações.

A principal característica do C++ é sua capacidade de suportar múltiplos paradigmas de programação, incluindo programação procedural, orientada a objetos e genérica. Isso proporciona aos desenvolvedores uma ampla gama de ferramentas para abordar diferentes problemas e estilos de programação.

Além disso, o C++ é conhecido por seu desempenho eficiente, sendo uma escolha popular para projetos que exigem alta performance, como jogos e sistemas embarcados.

2.3 | OpenGL

O OpenGL é uma biblioteca gráfica amplamente utilizada para renderização 2D e 3D em ambientes de desenvolvimento. Reconhecida por sua versatilidade, o OpenGL proporciona uma interface eficiente para interação com aceleração de hardware gráfico.

Sua interface é conhecida pela simplicidade e eficácia, permitindo que desenvolvedores criem representações visuais complexas. Com suporte para diversas plataformas, o OpenGL é utilizado em uma variedade de aplicações, desde jogos até visualizações científicas.

A flexibilidade do OpenGL se destaca ao permitir a configuração detalhada de cenas gráficas, manipulação de texturas e aplicação de efeitos visuais avançados. Além disso, é frequentemente combinado com outras bibliotecas e frameworks para aprimorar funcionalidades, tornando-o uma escolha popular em ambientes de desenvolvimento gráfico.

2.3.1 GLFW

GLFW, que significa "Graphics Library Framework", é uma biblioteca C/C++ dedicada à criação de janelas com contexto OpenGL, além de lidar com entrada de eventos, como teclado e mouse. Seu principal objetivo é fornecer uma interface simples e eficiente para a inicialização e gerenciamento de janelas, contextos OpenGL e dispositivos de entrada.

2.3.2 GLEW

GLEW, que significa "OpenGL Extension Wrangler Library", é uma biblioteca em C/C++ que auxilia os desenvolvedores no gerenciamento de extensões OpenGL. O principal propósito do GLEW é simplificar o processo de carregamento e gerenciamento de extensões OpenGL, proporcionando uma interface consistente e fácil de usar.

2.3.3 GLM

GLM, que significa "OpenGL Mathematics", é uma biblioteca de matemática para C++ projetada para ser utilizada em conjunto com o OpenGL. Ela fornece classes e funções que facilitam a manipulação de vetores, matrizes e outras operações matemáticas comumente usadas em gráficos 3D e programação de shaders.

2.4 | PowerPoint

O Microsoft PowerPoint é um aplicativo de software amplamente utilizado para criação de apresentações visuais. Parte integrante do pacote Microsoft Office, o PowerPoint oferece uma variedade de recursos que permitem aos usuários criar apresentações profissionais e atraentes.

2.5 | Overleaf

Overleaf é uma plataforma de edição colaborativa online projetada especificamente para a criação de documentos LaTeX. LaTeX é um sistema de preparação de documentos amplamente utilizado para produzir documentos científicos, acadêmicos e técnicos, especialmente em áreas como matemática, ciência da computação e engenharia.

3 / *Etapas de Desenvolvimento*

3.1 | Introdução

Entrámos agora na fase de desenvolvimento, onde os conceitos teóricos ganham vida através da implementação prática. Nesta etapa, iremos detalhar a configuração do ambiente de desenvolvimento, a estrutura do código e as decisões de implementação que deram origem à representação tridimensional do sistema solar.

Paralelamente, vamos explorar a metodologia utilizada e os resultados dos testes realizados, destacando a eficiência, estabilidade e a adesão aos objetivos propostos. Este capítulo encapsula o processo de transformação das ideias numa aplicação funcional, abordando cada etapa desde a codificação até à validação prática. Isso evidencia o equilíbrio entre a visão conceptual e a execução prática alcançada ao longo deste projeto.

3.2 | Divisão de tarefas

Começamos desenhar os objetos do sistema solar e a atribuir os seus movimentos, com destaque para a contribuição do Daniel e do Ricardo. O Francisco e o Tiago foram responsáveis pelo carregamento de texturas, enquanto o controle da câmera ficou a cargo do Ricardo e o Francisco. Quanto à iluminação e sombras, o Tiago e Daniel desempenharam papéis significativos.

3.3 | Etapas

3.3.1 Desenho dos objetos e a sua movimentação

```
void renderSphere(float r, int sectors, int stacks) {
    Sphere sphere(r, sectors, stacks);
    sphere.Draw();
}
```

Excerto de Código 3.1: Função para desenhar a esfera

3.3.2 Atribuição dos movimentos aos objetos

```
ang[2] += angular_speed(365.25);
          x[2] = radius * sin(3.14159 * 2 * ang[2] / 360);
          y[2] = radius * cos(3.14159 * 2 * ang[2] / 360);
          velocidade[2] += 0.0787f;
```

Excerto de Código 3.2: Rotação e translação

3.3.3 Atribuição dos movimentos aos objetos

```
ang[2] += angular_speed(365.25);
          x[2] = radius * sin(3.14159 * 2 * ang[2] / 360);
          y[2] = radius * cos(3.14159 * 2 * ang[2] / 360);
          velocidade[2] += 0.0787f;
```

Excerto de Código 3.3: Rotação e translação

3.3.4 Atribuição de texturas

```
unsigned int loadTexture(char const* path)
{
    unsigned int textureID;
    glGenTextures(1, &textureID);

    int width, height, nrComponents;
    unsigned char* data = stbi_load(path, &width, &height, &nrComponents
        , 0);
    if (data)
    {
        GLenum format;
        if (nrComponents == 1)
            format = GL_RED;
        else if (nrComponents == 3)
            format = GL_RGB;
        else if (nrComponents == 4)
            format = GL_RGBA;

        glBindTexture(GL_TEXTURE_2D, textureID);
        glTexImage2D(GL_TEXTURE_2D, 0, format, height, width, 0, format,
             GL_UNSIGNED_BYTE, data);
        glGenerateMipmap(GL_TEXTURE_2D);

        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_MIN_FILTER,
            GL_LINEAR_MIPMAP_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)
            ;

        stbi_image_free(data);
    }
    else
    {
        std::cout << "Texture failed to load at path: " << path << std::
            endl;
        stbi_image_free(data);
    }

    return textureID;
}
```

Excerto de Código 3.4: Função de leitura de texturas

```
GLuint earthTextureID = loadTexture("earth.jpg");
    GLuint marsTextureID = loadTexture("mars.jpg");
    GLuint sunTextureID = loadTexture("sun.jpg");
    GLuint moonTextureID = loadTexture("moon.jpg");
    GLuint venusTextureID = loadTexture("venus.jpg");
    GLuint jupiterTextureID = loadTexture("jupiter.jpg");
    GLuint uranusTextureID = loadTexture("uranus.jpg");
    GLuint mercuryTextureID = loadTexture("mercury.jpg");
    GLuint neptuneTextureID = loadTexture("neptune.jpg");
    GLuint saturnTextureID = loadTexture("saturn.jpg");
    GLuint saturnRingTextureID = loadTexture("saturn_ring.png");
    GLuint ceresTextureID = loadTexture("ceres.jpg");
    GLuint jupiterMoonTextureID = loadTexture("ceres.jpg");
    GLuint fundoTextureID = loadTexture("estrelas.jpg");
```

Excerto de Código 3.5: Atribuiçao de texturas

3.3.5 Controlos da câmara

```
if (glfwGetKey( window, GLFW_KEY_RIGHT ) == GLFW_PRESS){
    horizontalAngle -= 0.05;
  }

  if (glfwGetKey( window, GLFW_KEY_LEFT ) == GLFW_PRESS){
    horizontalAngle += 0.05;
  }

  if (glfwGetKey( window, GLFW_KEY_UP ) == GLFW_PRESS){
    verticalAngle += 0.05;
  }

  if (glfwGetKey( window, GLFW_KEY_DOWN ) == GLFW_PRESS){
    verticalAngle -= 0.05;

  }

  if (glfwGetKey( window, GLFW_KEY_LEFT_SHIFT ) == GLFW_PRESS){
    position[1] += 1.0f;

  }

  if (glfwGetKey( window, GLFW_KEY_LEFT_CONTROL ) == GLFW_PRESS){
    position[1] -= 1.0f;
```

```
  }

  // Up vector
  glm::vec3 up = glm::cross( right, direction );

  // Move forward
  if (glfwGetKey( window, GLFW_KEY_W ) == GLFW_PRESS){
    position += direction * deltaTime * speed;
  }
  // Move backward
  if (glfwGetKey( window, GLFW_KEY_S ) == GLFW_PRESS){
    position -= direction * deltaTime * speed;
  }
  // Strafe right
  if (glfwGetKey( window, GLFW_KEY_D ) == GLFW_PRESS){
    position += right * deltaTime * speed;
  }
  // Strafe left
  if (glfwGetKey( window, GLFW_KEY_A ) == GLFW_PRESS){
    position -= right * deltaTime * speed;
  }

  if (glfwGetKey(window, GLFW_KEY_Q) == GLFW_PRESS) {
    if (initialFoV >= 4 && initialFoV < 120) {

            initialFoV += 1.0f;

    }
}



if (glfwGetKey(window, GLFW_KEY_E) == GLFW_PRESS) {
    if (initialFoV > 4 && initialFoV <= 120) {

            initialFoV -= 1.0f;

    }


}
```

Excerto de Código 3.6: Função para desenhar a esfera

3.3.6 Iluminação dos objetos

```
#version 330 core

in vec2 UV;
in vec3 FragPos;
in vec3 Normal;

out vec4 FragColor;

uniform sampler2D myTextureSampler;
uniform vec3 lightPos;          // Example light position
uniform vec3 lightColor;        // Example light color
uniform vec3 viewPos;           // Example camera position

uniform float ambientStrength;  // Example ambient strength
uniform float specularStrength; // Example specular strength
uniform float shininess;        // Example shininess

void main(){
    // Ambient component
    vec3 ambient = ambientStrength * texture(myTextureSampler, UV).xyz;

    // Diffuse component
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(lightPos - FragPos);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = diff * lightColor;

    // Specular component
    vec3 viewDir = normalize(viewPos - FragPos);
    vec3 reflectDir = reflect(-lightDir, norm);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), shininess);
    vec3 specular = specularStrength * spec * lightColor;

    // Final color with texture
    vec3 result = ambient + diffuse + specular;
    FragColor = vec4(result, 1.0);
}
```

Excerto de Código 3.7: Codigo que junta varios fatores para criar a iluminçao de cada fragmento da superficie dos objetos

```
void setShaderUniforms(GLuint programID, const glm::vec3& lightColor,
    const glm::vec3& lightPos, const glm::vec3& viewPos,
    float ambientStrength, float specularStrength, float shininess,
    const glm::mat4& projection, const glm::mat4& view, const glm::mat4&
        model) {
    glUniform3f(glGetUniformLocation(programID, "lightColor"),
        lightColor.r, lightColor.g, lightColor.b);
    glUniform3f(glGetUniformLocation(programID, "lightPos"), lightPos.x,
        lightPos.y, lightPos.z);
    glUniform3f(glGetUniformLocation(programID, "viewPos"), viewPos.x,
        viewPos.y, viewPos.z);

    glUniform1f(glGetUniformLocation(programID, "ambientStrength"),
        ambientStrength);
    glUniform1f(glGetUniformLocation(programID, "specularStrength"),
        specularStrength);
    glUniform1f(glGetUniformLocation(programID, "shininess"), shininess)
        ;

    glUniformMatrix4fv(glGetUniformLocation(programID, "projection"), 1,
        GL_FALSE, &projection[0][0]);
    glUniformMatrix4fv(glGetUniformLocation(programID, "view"), 1,
        GL_FALSE, &view[0][0]);
    glUniformMatrix4fv(glGetUniformLocation(programID, "model"), 1,
        GL_FALSE, &model[0][0]);
}
```

Excerto de Código 3.8: Função que atribui a luz aos objetos

3.3.7 Renderização final dos objetos

```
glm::mat4 earthModelMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(
    x[2], 0.0f, y[2]));
        earthModelMatrix = glm::rotate(earthModelMatrix, velocidade[2],
            glm::vec3(0.0f, 1.0f, 0.0f));
        earthModelMatrix = glm::rotate(earthModelMatrix, velocidade[11],
             glm::vec3(1.0f, 0.0f, 0.0f));

        Projection = getProjectionMatrix();
        View = getViewMatrix();
        MVP = Projection * View * earthModelMatrix;
        glm::vec3 viewPos = getCameraPosition();

        setShaderUniforms(programID, lightcolor, lightpos, viewPos, 0.5f
            , 0.5f, 3.0f, Projection, View, earthModelMatrix);

        glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &MVP[0][0]);
        setTexture(earthTextureID, programID);
        renderSphere(1.0f, 36, 18);
```

Excerto de Código 3.9: Renderização da Terra

```
glm::mat4 moonModelMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(x
    [2] + x[8], 0.0f, y[2] + y[8]));
        moonModelMatrix = glm::rotate(moonModelMatrix, velocidade[2],
            glm::vec3(0.0f, 1.0f, 0.0f));
        moonModelMatrix = glm::rotate(moonModelMatrix, velocidade[11],
            glm::vec3(0.5f, 0.0f, 0.0f));

        Projection = getProjectionMatrix();
        View = getViewMatrix();
        MVP = Projection * View * moonModelMatrix;
        viewPos = getCameraPosition();

        setShaderUniforms(programID, lightcolor, lightpos, viewPos, 0.5f
            , 0.5f, 3.0f, Projection, View, moonModelMatrix);

        glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &MVP[0][0]);
        setTexture(moonTextureID, programID);
        renderSphere(0.55f, 36, 18);
```

Excerto de Código 3.10: Renderização da Lua

```
glm::mat4 sunModelMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(0.0
    f, 0.0f, 0.0f));
        sunModelMatrix = glm::rotate(sunModelMatrix, velocidade[8], glm
            ::vec3(0.0f, 1.0f, 0.0f));
        sunModelMatrix = glm::rotate(sunModelMatrix, velocidade[11], glm
            ::vec3(1.0f, 0.0f, 0.0f));

        Projection = getProjectionMatrix();
        View = getViewMatrix();
        MVP = Projection * View * sunModelMatrix;
        viewPos = getCameraPosition();

        setShaderUniforms(programID, lightcolor, lightpos, viewPos, 1.f,
             0.1f, 51.2f, Projection, View, sunModelMatrix);


        glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &MVP[0][0]);
        setTexture(sunTextureID, programID);
        renderSphere(10.0f, 36, 18);
```

Excerto de Código 3.11: Renderização do Sol

```
glm::mat4 skyModelMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(0.0
    f, 0.0f, 0.0f));
        Projection = getProjectionMatrix();
        View = getViewMatrix();
        MVP = Projection * View * skyModelMatrix;
        viewPos = getCameraPosition();

        setShaderUniforms(programID, lightcolor, glm::vec3(), viewPos,
            1.0f, 0.f, 0.0f, Projection, View, skyModelMatrix);
        glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &MVP[0][0]);
        setTexture(fundoTextureID, programID);
        renderSphere(2000.0f, 36, 18);
```

Excerto de Código 3.12: Renderização do espaço sideral

3.4 | Resultado Final

3.4.1 Plano geral

Figura 3.1: Sistema solar renderizado

3.4.2 Visualização de planetas e as suas sombras e os seus satelites em orbita

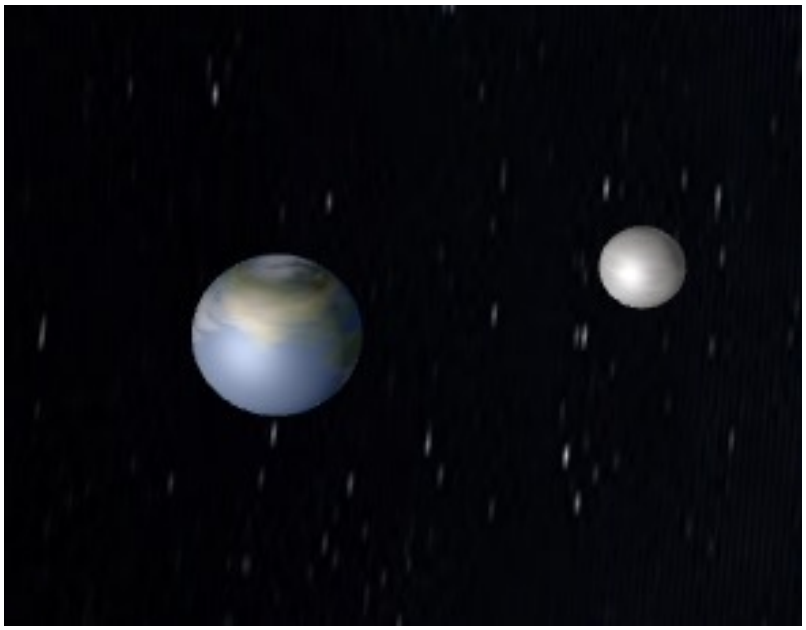

Figura 3.2: Terra e a lua

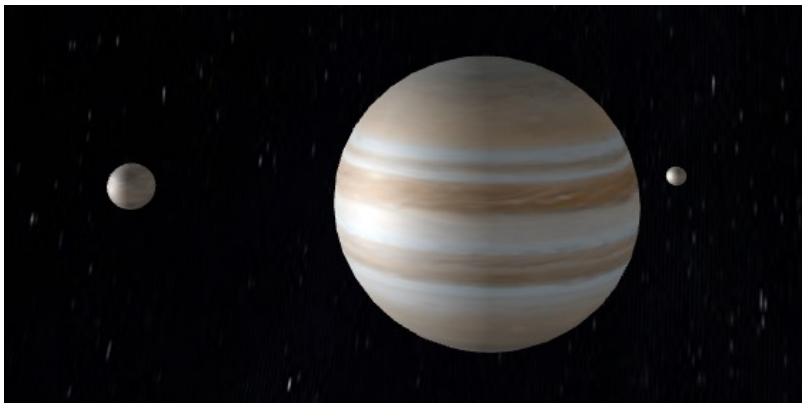

Figura 3.3: Saturno e os seus satelites

3.4.3 Visualização do ponto de reflexão dependendo de onde está posicionada a câmara.

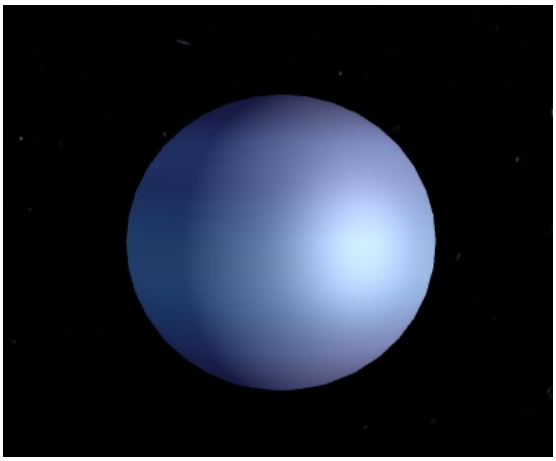

Figura 3.4: Neptuno visto de lado

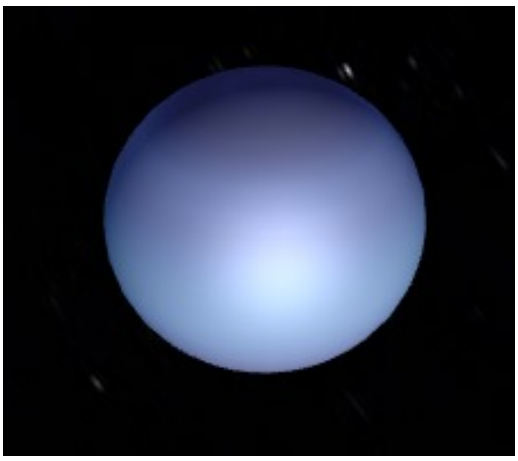

Figura 3.5: Neptuno visto de frente

4 / *Trabalhos Futuros*

Num trabalho futuro, podemos considerar a implementação de elementos adicionais para enriquecer a experiência visual. Entre as possíveis adições, destacam-se a inclusão de menus iterativos, implementação da capacidade de desenhar as linhas das órbitas dos planetas, a representação detalhada do anel de Saturno, a reprodução visual da cintura de asteroides e a incorporação de controladores que permitam ajustar a velocidade das órbitas dos corpos celestes.

4.1 | Menus Iterativos

A introdução de menus iterativos pode proporcionar uma interação mais dinâmica com o sistema solar simulado. Esses menus podem conter opções para explorar informações detalhadas sobre cada corpo celeste, como características físicas, dados históricos e curiosidades astronômicas.

4.2 | Linha das orbitras

Integrar a representação visual das órbitas dos planetas acrescentaria um elemento gráfico essencial à simulação. Essas linhas podem ser desenhadas de forma elegante, destacando as trajetórias elípticas que os planetas percorrem em torno do Sol. A inclusão dessa característica não apenas aprimoraria a precisão da simulação, mas também ofereceria uma perspectiva mais intuitiva sobre os padrões de movimento orbital.

4.3 | Anel de Saturno

A adição do anel de Saturno ofereceria uma representação mais fiel e completa desse planeta fascinante. A modelagem precisa do anel, juntamente com a capacidade de ajustar a visibilidade e orientação do anel, seria uma contribuição valiosa para a simulação.

4.4 | Cintura de Asteroides

A inclusão visual da cintura de asteroides entre as órbitas de Marte e Júpiter proporcionaria um elemento adicional de realismo à representação do sistema solar. Pode-se explorar opções para destacar asteroides significativos ou apresentar informações sobre composição e características individuais.

4.5 | Controladores de Velocidade

A implementação de controladores interativos permitiria aos usuários ajustar a velocidade das órbitas dos planetas, proporcionando uma visão mais dinâmica do movimento celestial. Isso poderia ser realizado através de barras deslizantes ou caixas de diálogo que permitiriam a personalização da velocidade de rotação de cada corpo celeste. Essas adições não apenas enriqueceriam a simulação, mas também proporcionariam uma experiência educativa mais envolvente, permitindo aos utulizadores explorar e compreender melhor os elementos do sistema solar de uma maneira interativa e informativa.

5 / *Considerações Finais*

Ao chegarmos à conclusão do desenvolvimento deste projeto, experimentamos uma sensação de parcial satisfação, motivada pela conquista de praticamente todos os objetivos inicialmente propostos. Este percurso culminou em uma simulação cativante e realista do nosso sistema solar, proporcionandonos uma visão envolvente dos movimentos celestiais que regem os corpos que compõem nosso sistema planetário.

Ao refletirmos sobre o tempo dedicado a esse empreendimento, ganhamos uma profunda apreciação pela importância intrínseca da Computação Gráfica (CG). A habilidade de visualizar e representar graficamente fenômenos astronômicos complexos ampliou consideravelmente nossa compreensão sobre o funcionamento interno do sistema solar. A complexidade dos cálculos orbitais, a aplicação de texturas realistas e a incorporação de elementos interativos, como menus e controladores, destacam-se como áreas nas quais a Computação Gráfica desempenha um papel crucial.