Universidade do Minho
Licenciatura em Engenharia Informática

Unidade Curricular de Computação Gráfica

Ano Letivo de 2024/2025

Practical Assignment CG

Adélio Fernandes	A78778
Tiago Carneiro	A93207
Tiago Guedes	A97369

2 de março de 2025

CG

Índice

Introdução

Este relatório apresenta o desenvolvimento de um motor gráfico 3D, criado no âmbito da unidade curricular de Computação Gráfica (CG). O objetivo do trabalho é a construção de um sistema capaz de gerar um cenário gráfico em 3D.

O projeto está dividido em quatro fases, sendo que esta primeira fase envolve a implementação de duas aplicações principais: um **gerador de modelos** e um **motor gráfico**. O gerador de modelos é responsável por criar ficheiros contendo as coordenadas dos vértices das primitivas gráficas, enquanto o motor gráfico processa um ficheiro de configuração XML e exibe os modelos gerados.

As primitivas gráficas implementadas nesta fase são um plano, uma caixa, uma esfera e um cone, cada um definido por um conjunto de parâmetros específicos. Além disso, o motor gráfico permite configurar a câmara, definindo a posição, direção de visualização e projeção da cena.

2. Gerador

O gerador é responsável por criar ficheiros contendo as coordenadas dos vértices das primitivas gráficas. Por cada execução, terá que receber o tipo de modelo a desenhar, os parâmetros necessários para a geração de pontos e por fim um nome para o ficheiro de output, sendo que neste caso, o nome do ficheiro terá que ter a extensão .3d, onde todos os pontos serão guardados para que no futuro, o motor o possa de maneira a conseguir desenhar os modelos numa janela.

2.1. Plano

A geração dos planos segue uma lógica baseada na subdivisão de uma superfície em pequenos quadrados, que são representados como dois triângulos. O processo inicia-se com a definição dos parâmetros essenciais: o comprimento total do plano é dividido por dois para obter a metade da dimensão, garantindo que a figura fique centrada na origem. Em seguida, calcula-se o passo entre subdivisões, obtido dividindo o comprimento pelo número de divisões especificadas.

Para construir a malha do plano, utilizam-se dois loops aninhados que percorrem a estrutura linha por linha e coluna por coluna. Em cada iteração, são calculadas as coordenadas dos quatro vértices que compõem um quadrado da subdivisão. Esses vértices são então organizados para formar dois triângulos, que são adicionados à classe Figure, garantindo que o modelo seja corretamente armazenado para posterior renderização.

A lógica é aplicada a três variações do plano, cada uma correspondente a um sistema de coordenadas diferente. No caso do plano XZ, os valores de X e Z variam enquanto a coordenada Y permanece fixa. No plano XY, variam-se X e Y, mantendo a coordenada Z constante. Já no plano YZ, os valores de Y e Z são alterados enquanto X é mantido fixo. Essa abordagem garante flexibilidade na geração de planos, permitindo que sejam utilizados em diferentes orientações dentro da cena 3D.

2.2. Caixa

A função generateBox cria uma caixa tridimensional subdividida em pequenas faces quadradas, permitindo um controle detalhado da malha por meio do parâmetro divisions. A caixa é gerada a partir de seis faces, cada uma representando um dos lados do cubo, e cada face é dividida em pequenos quadrados compostos por dois triângulos.

O cálculo das posições dos vértices é baseado no tamanho da caixa, que é centrada na origem, e no número de divisões especificado. Para cada face, a posição dos vértices é determinada usando um deslocamento incremental definido pela variável step, que representa o tamanho de cada subdivisão da face. As normais de cada face são definidas para garantir que a iluminação funcione corretamente.

Cada iteração do loop adiciona quatro vértices para formar um quadrado, além de definir as coordenadas de textura associadas. Dois triângulos são criados para formar o quadrado, utilizando os índices apropriados dos vértices. Isso é feito para todas as seis faces da caixa, garantindo que todos os lados sejam cobertos.

Ao final do processo, a função retorna um objeto Figure contendo todas as informações necessárias para renderizar a caixa, incluindo vértices, normais, coordenadas de textura e índices dos triângulos. O método é eficiente e permite a geração de cubos detalhados com subdivisões controladas pelo usuário, proporcionando maior precisão visual.

2.3. Esfera

A geração da esfera segue uma abordagem baseada na subdivisão da sua superfície em pequenos triângulos utilizando um sistema de coordenadas esféricas. O processo começa com a definição de dois parâmetros fundamentais: stacks e slices. As stacks representam as divisões ao longo do eixo vertical da esfera, enquanto as slices representam as divisões ao longo da sua circunferência. Dessa forma, a superfície da esfera é descrita em pequenas secções triangulares.

Para construir a malha da esfera, percorrem-se os valores de stacks e slices através de dois loops aninhados. Cada iteração desses loops calcula os ângulos phi (latitude) e theta (longitude) para determinar a posição de quatro pontos adjacentes na superfície da esfera. A partir desses pontos, são formados dois triângulos que preenchem a região correspondente da esfera.

Os cálculos das coordenadas dos vértices baseiam-se nas fórmulas das coordenadas esféricas:

X = raio × cos(phi) × cos(theta)
Y = raio × sin(phi)
Z = raio × cos(phi) × sin(theta)

A normal de cada vértice é calculada normalizando o vetor posição, ou seja, dividindo cada componente das coordenadas pelo raio da esfera. Além disso, são geradas coordenadas de textura associadas a cada vértice, permitindo a aplicação de texturas na esfera posteriormente.

Para evitar redundâncias na definição de triângulos, são aplicadas condições para garantir que a primeira stack (topo da esfera) e a última stack (base da esfera) sejam tratadas corretamente, garantindo uma conexão precisa entre as divisões. Essa abordagem assegura que a esfera seja construída de maneira eficiente e detalhada, respeitando os parâmetros especificados para a sua resolução.

2.4. Cone

A função generateCone cria um cone tridimensional a partir dos parâmetros radius, height e divisions, que definem respectivamente o raio da base, a altura total do cone e o número de divisões da base, influenciando a suavidade do modelo. O cone é gerado utilizando triângulos e dividido em duas partes principais: a base circular e as faces laterais.

Para criar a base, a função define um ponto central fixo em (0, -halfHeight, 0), onde halfHeight é metade da altura total do cone. Os pontos ao redor da base são calculados usando as funções trigonométricas seno e cosseno para determinar suas coordenadas x e z em relação ao ângulo correspondente e ao raio especificado. A base é formada por triângulos onde um dos vértices é o centro e os outros dois são pontos consecutivos ao redor do círculo.

Já as faces laterais do cone são formadas pela ponta do cone, localizada em (0, halfHeight, 0), e por pares de pontos consecutivos da base. Cada iteração do loop adiciona dois triângulos à figura, um para a base e outro para a lateral, garantindo que os triângulos sejam definidos no sentido anti-horário, de modo que as normais fiquem corretamente orientadas para o exterior do modelo.

Ao final do processo, a função retorna um objeto Figure contendo todos os vértices necessários para renderizar o cone. Esse método assegura que o cone possa ser gerado de forma eficiente, permitindo ajustes no nível de detalhe através do parâmetro divisions, que quanto maior for, mais suavizada será a aparência da estrutura.

3. Motor

O Motor, é a aplicação responsável pela leitura do ficheiro ”.XML”, ficheiro que dentro do seu formato contém as particularidades e especificidades das primitivas que vamos construir através do OpenGL e é responsável por construir estas mesmas primitivas através dos modelos (”.3d”) gerados pela aplicação previamente mencionada, o Gerador.

```
<world>
    <window width="512" height="512" />
    <camera>
        <position x="3" y="2" z="1" />
        <lookAt x="0" y="0" z="0" />
        <up x="0" y="1" z="0" />
        <projection fov="60" near="1" far="1000" />
    </camera>
    <group>
        <models>
            <model file="plane.3d" />
            <model file="cone.3d" />
        </models>
    </group>
</world>
```

Fig. Exemplo da Estrutura do ficheiro XML

3.1. XML Parser

O código implementa um parser XML para carregar configurações de um arquivo XML e utilizá-las na renderização de um ambiente 3D. O xml_parser é a função principal responsável por carregar o arquivo e processar seus elementos. Primeiramente, o arquivo XML é carregado e validado. Caso ocorra algum erro, uma mensagem é exibida.

Dentro do XML, o nó raiz <world> contém as configurações que serão carregadas. O parser primeiro busca o nó <window>, onde define as dimensões da janela de renderização. Em seguida, o nó <camera> é processado para definir a posição da câmera, a direção para onde está olhando e as configurações de projeção. Depois, o código busca os nós <group> e

<models> para carregar modelos 3D a partir de arquivos externos. Para cada modelo encontrado, a função loadModels adiciona o objeto à lista de modelos que serão desenhados.

O parser também inclui funções auxiliares, como getFloatAttribute e getIntAttribute, que facilitam a extração de valores numéricos dos atributos XML. Esse processo garante que todas as configurações necessárias para a renderização sejam carregadas corretamente antes do início do programa.

3.2. Camera

A configuração da câmera no código é baseada na leitura do XML e na manipulação dos parâmetros que controlam sua posição, direção e projeção. A função loadCameraConfiguration é responsável por carregar esses valores a partir do XML, garantindo que a câmera seja posicionada corretamente na cena.

A posição da câmera é definida pelos atributos x, y, z dentro do nó <position>. Além disso, o raio da câmera (cradius) é calculado com base na distância entre o ponto da câmera e a origem, utilizando a fórmula da magnitude de um vetor tridimensional.

O nó <lookAt> define para onde a câmera está olhando, enquanto o nó <up> define a orientação vertical. Já a configuração de projeção é carregada a partir do nó <projection>, contendo os valores fov (campo de visão), near (plano próximo) e far (plano distante).

Além do carregamento inicial, a câmera pode ser movimentada em tempo real usando o teclado. As teclas W, A, S, D ajustam a posição, enquanto as setas direcionais alteram o ponto para onde a câmera está olhando. Esse comportamento é implementado nas funções processKeys e processSpecialKeys.

No loop de renderização, a posição da câmera é recalculada dinamicamente para permitir interatividade. A função gluLookAt define a câmera na cena, garantindo que ela esteja sempre apontada para a direção correta.

3.3. Leitura e Desenho

O carregamento e a renderização dos modelos 3D ocorrem em duas etapas principais: leitura dos modelos e desenho na tela.

A função loadModels percorre o nó <models> no XML e extrai os caminhos dos arquivos de modelos 3D. Esses arquivos são então carregados utilizando a função figureFromFile, e os modelos são armazenados em uma lista global chamada models. Esse processo permite que múltiplos modelos sejam carregados e renderizados dentro da cena.

A função drawScene é responsável pelo desenho dos modelos. Ela percorre a lista de modelos carregados e extrai os vértices de cada um. O OpenGL é utilizado para desenhar os triângulos que formam a superfície dos objetos, chamando glBegin(GL_TRIANGLES) e passando os vértices através de glVertex3f.

O loop de renderização é gerenciado pela função renderScene, que limpa o buffer, posiciona a câmera e desenha os modelos carregados. O OpenGL também aplica transformações de escala (glScalef) e define o modo de renderização dos polígonos (glPolygonMode), permitindo alternar entre renderização preenchida ou em wireframe.

A renderização ocorre continuamente dentro do loop do GLUT, garantindo que os modelos sejam redesenhados em tempo real conforme a interação do usuário.

5. Conclusões e Trabalho Futuro