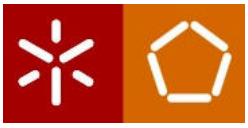

Universidade do Minho
Departamento de Informática

Computação Gráfica

Trabalho Prático - Fase 2

Grupo 5

João Coelho - A100596

José Rodrigues - A100692

Duarte Araújo - A100750

2024-04-05

Índice

Lista de Figuras

1. Introdução

A computação gráfica desempenha um papel fundamental na criação de ambientes virtuais imersivos e na visualização de objetos tridimensionais, sendo uma área de estudo essencial no campo da informática. No âmbito deste projeto, desenvolvemos um mecanismo 3D voltado para a construção de cenas complexas e a manipulação eficiente de modelos tridimensionais.

Nesta fase do projeto, o nosso foco principal foi na implementação de transformações geométricas hierárquicas e na organização eficaz de cenas num ambiente 3D. Essas transformações, que incluem translação, rotação e escala, são fundamentais para posicionar e orientar modelos tridimensionais de forma precisa e intuitiva.

A capacidade de manipular hierarquias de modelos é essencial para a criação de cenas realistas e complexas, onde diferentes objetos interagem entre si e com o ambiente circundante. Ao integrar transformações geométricas hierárquicas, podemos criar composições dinâmicas de modelos, permitindo a construção de ambientes virtuais cada vez mais detalhados.

Neste relatório, detalharemos as etapas de implementação das transformações geométricas hierárquicas e da organização de cenas no nosso mecanismo 3D. Abordaremos os desafios enfrentados, as soluções adotadas e os resultados alcançados durante esta fase do projeto.

2. Descrição dos Componentes Atualizados

Nesta fase do projeto, realizámos atualizações significativas na engine para suportar as transformações geométricas hierárquicas e a organização de cenas complexas num ambiente 3D. Os principais componentes atualizados incluem:

- **Engine:**
  - **Suporte a Transformações Hierárquicas:** Implementámos algoritmos para aplicar transformações de translação, rotação e escala de forma hierárquica em modelos individuais e grupos de modelos. Essas transformações são fundamentais para manipular a posição, orientação e escala dos objetos na cena, possibilitando a criação de animações e efeitos visuais complexos.
  - **Gestão de Hierarquia de Modelos:** Desenvolvemos estruturas de dados eficientes, como árvores de cena, para gerenciar a organização hierárquica de modelos. Isso permite uma representação eficaz das relações entre os objetos na cena, facilitando a manipulação e composição de cenas complexas.
- **Parser XML:**
  - **Atualização para Interpretar Transformações:** Aprimoramos o parser XML para interpretar e processar as novas tags relacionadas às transformações geométricas hierárquicas. Isso envolveu a implementação de algoritmos de análise sintática e semântica para extrair as informações relevantes dos arquivos de configuração XML.

3. Implementação das Transformações Geométricas Hierárquicas

Durante esta fase do projeto, concentramos os nossos esforços na implementação das transformações geométricas hierárquicas, uma funcionalidade crucial para manipular a posição, orientação e escala dos objetos numa cena 3D. A seguir, exploramos mais detalhadamente as principais características da implementação:

3.1. Representação Hierárquica de Modelos

Para alcançar uma manipulação eficiente de hierarquias de modelos, optamos por uma abordagem baseada em árvores de cena. No código fornecido, a estrutura `Tree` representa essa árvore de cena. Cada nó na árvore (Node) contém informações sobre as transformações a serem aplicadas aos modelos associados a esse nó e uma lista de modelos ou grupos de modelos.

```
struct Node {
    std::vector<Transformation> transformations;
    std::vector<std::string> model_name;
};
```

Figure 1: "Definição do Node"

Aqui, `transformations` armazena as transformações geométricas (como translação, rotação e escala) a serem aplicadas aos modelos neste nó específico. Cada modelo é identificado pelo seu nome (model_name). Isso permite que cada nó na árvore represente um conjunto específico de modelos e as transformações que serão aplicadas a esses modelos.

As arestas na árvore de cena são representadas pelas relações pai-filho entre os nós. Cada nó pode ter zero ou mais nós filhos, indicando a relação hierárquica entre os modelos na cena. Por exemplo, no código fornecido, a estrutura Tree contém uma lista de filhos (children), onde cada filho é uma nova instância da estrutura `Tree`.

```
struct Tree {
    Node node;
    std::vector<Tree> children;
};
```

Figure 2: "Definição da Tree"

Essa estrutura recursiva permite uma representação flexível e hierárquica dos modelos na cena. Quando aplicamos transformações a um nó na árvore, essas transformações são propagadas recursivamente para todos os nós filhos, permitindo a composição eficiente de transformações hierárquicas.

Essa abordagem baseada em árvores de cena simplifica a organização dos modelos na cena e facilita a aplicação de transformações hierárquicas. Além disso, ela se alinha bem com os princípios de programação orientada a objetos, permitindo uma representação intuitiva e modular da cena 3D. Essa conformidade com os princípios da programação orientada a objetos também aproveita todo o potencial da linguagem de programação utilizada (C++), permitindo uma implementação mais eficiente e fácil de manter.

3.2. Aplicação de Transformações

Para aplicar as transformações geométricas hierárquicas, desenvolvemos algoritmos eficientes que operam em cada nó da árvore de cena. Utilizamos operações matriciais, como multiplicação de matrizes, para calcular as transformações resultantes com base nas transformações definidas em cada nó. Isso nos permite manipular a posição, orientação e escala dos modelos de forma precisa e flexível.

Por exemplo, ao percorrer a árvore de cena e renderizar os modelos, aplicamos as transformações definidas em cada nó. Aqui está um trecho de código que ilustra como as transformações são aplicadas durante a renderização dos modelos:

```
void render_models(Tree tree, std::vector<Transformation> transformations = {}) {
    // Push novas transformações
    for (auto& transform : tree.node.transformations) {
        transformations.push_back(transform);
    }

    glPushMatrix();

    // Aplicar todas as transformações acumuladas
    for (const auto& transformation : transformations) {
        switch (transformation.type) {
            case TransformationType::TRANSLATE:
                glTranslatef(transformation.values[0], transformation.values[1], transformation.values[2]);
                break;

            case TransformationType::ROTATE:
                glRotatef(transformation.values[0], transformation.values[1], transformation.values[2], transformation.values[3]);
                break;

            case TransformationType::SCALE:
                glScalef(transformation.values[0], transformation.values[1], transformation.values[2]);
                break;
        }
    }

    // Renderizar os modelos associados a este nó
    for (const auto& model_name : tree.node.model_name) {
        // Renderizar modelo
        // não interessa para esta secção
    }

    glPopMatrix();

    // Renderizar modelos dos nós filhos
    for (auto& child : tree.children) {
        render_models(child, transformations);
    }
}
```

Figure 3: “Trecho de código da aplicação das transformações”

Neste código, percorremos a árvore de cena recursivamente e aplicamos as transformações definidas em cada nó. As transformações são acumuladas à medida que descemos na hierarquia da árvore, permitindo que cada modelo herde as transformações dos seus nós pais. Essa abordagem eficiente permite-nos renderizar cenas complexas com hierarquias de modelos e transformações dinâmicas.

Como pode ter notado, para aplicar as transformações são utilizadas as funções `glTranslatef`, `glRotatef` e `glScalef` do OpenGL. Essas funções são parte integrante do pipeline de renderização do OpenGL e operam multiplicando as coordenadas dos vértices dos modelos por matrizes de transformação apropriadas.

A função `glTranslatef` é usada para aplicar uma transformação de translação nos modelos. Ela desloca os modelos ao longo dos eixos x, y e z da cena. Matematicamente, essa função multiplica as coordenadas dos vértices do modelo por uma matriz de translação, que é construída com base nas quantidades de deslocamento especificadas.

Por sua vez, `glRotatef` é empregada para aplicar uma transformação de rotação aos modelos. Essa função rotaciona os modelos em torno de um eixo específico, conforme especificado pelo programador. Matematicamente, ela multiplica as coordenadas dos vértices do modelo por uma matriz de rotação, que é gerada com base no ângulo de rotação e no eixo de rotação definidos.

Por fim, glScalef é utilizada para aplicar uma transformação de escala aos modelos. Ela altera as dimensões dos modelos ao longo dos eixos x, y e z. Matematicamente, essa função multiplica as coordenadas dos vértices do modelo por uma matriz de escala, que é construída com base nos fatores de escala especificados.

Em conjunto, essas funções permitem a manipulação precisa das posições, orientações e tamanhos dos modelos na cena renderizada, conforme definido pelas transformações hierárquicas especificadas na árvore de cena.

4. Implementação da Câmera FPS (Extra)

A câmara no estilo "first-person shooter" (FPS) adiciona uma dimensão de imersão e interatividade à experiência do utilizador, permitindo-lhe explorar a cena a partir da perspetiva de um observador em primeira pessoa. Abaixo está uma explicação detalhada de como esta funcionalidade foi implementada:

4.1. Funcionalidades Implementadas

- **Movimento Livre:** A câmara FPS permite aos utilizadores moverem-se livremente pela cena utilizando os controlos do teclado. Podem avançar, recuar e movimentar-se para os lados.
- **Rotação da Câmara:** Os utilizadores podem rodar a câmara em torno do seu eixo vertical e horizontal para alterar a direção do olhar. Isto proporciona uma sensação de realismo e liberdade de movimento ao explorar a cena.
- **Restrições de Movimento:** Foram implementadas restrições para limitar o movimento da câmara dentro dos limites da cena, evitando que os utilizadores saiam da área de visualização.
- **Suavização de Movimento:** Foi adicionada suavização ao movimento da câmara para evitar transições bruscas e proporcionar uma experiência de exploração mais natural e confortável.

4.2. Matemática e Algoritmos Utilizados

4.2.1. Movimento e Rotação

O movimento da câmara é calculado com base na posição e na direção para a qual ela está olhando. Isto é realizado utilizando vetores de direção para frente e para os lados, os quais são calculados a partir da posição da câmara e do ponto para o qual ela está olhando. As transformações de movimento são então aplicadas usando esses vetores e a velocidade de movimento configurada.

A rotação da câmara é controlada pelo movimento do rato. Quando o rato se move, a diferença na posição atual e anterior do rato é utilizada para calcular a alteração na direção do olhar da câmara. Esta direção é então atualizada para a câmara, permitindo que o utilizador gire livremente a câmara.

4.2.2. Restrições de Movimento

Para evitar que a câmara saia da área de visualização, foram implementadas restrições de movimento. Isto é feito verificando se a nova posição calculada da câmara está dentro dos limites da cena. Se a câmara estiver prestes a ultrapassar esses limites, o movimento é ajustado para mantê-la dentro da área visível da cena.

5. Demonstração do Sistema Solar

A configuração do sistema solar no XML define as disposições e características dos corpos celestes que compõem o nosso sistema solar dentro da cena 3D. Abaixo está uma explicação detalhada de como essas informações estão estruturadas e o que cada parte representa:

5.1. Grupos de Modelos dos Planetas

Cada planeta no sistema solar é representado por um grupo de modelos dentro da cena 3D. Cada grupo inclui um modelo esférico que representa o planeta e as transformações necessárias para posicionar e dimensionar corretamente o modelo na cena. Aqui está a disposição dos planetas no sistema solar:

- **Sol:** O grupo principal representa o Sol, posicionado no centro da cena com uma escala maior (3 vezes maior) em comparação com os outros planetas.
- **Mercúrio, Vénus, Terra, Marte, Júpiter, Saturno, Úrano e Neptuno:** Cada um destes planetas é representado por um grupo separado, com a sua própria posição e escala relativas dentro da cena.

5.2. Transformações de Posição e Escala

Cada grupo de modelos de planeta contém transformações de posição (translate) e escala (scale) para posicionar e dimensionar corretamente o modelo esférico do planeta na cena. Estas transformações são aplicadas em relação às coordenadas da cena para garantir que os planetas estejam corretamente posicionados e dimensionados relativamente ao Sol e uns aos outros.

5.3. Modelos Esféricos

Cada grupo de modelos de planeta inclui um modelo esférico representado pelo ficheiro "template-Sphere.3d". Este modelo esférico é utilizado para representar visualmente os planetas na cena 3D.

5.4. Visualização da Cena

Além das descrições acima, abaixo está uma representação visual da cena 3D gerada pela configuração do sistema solar no XML:

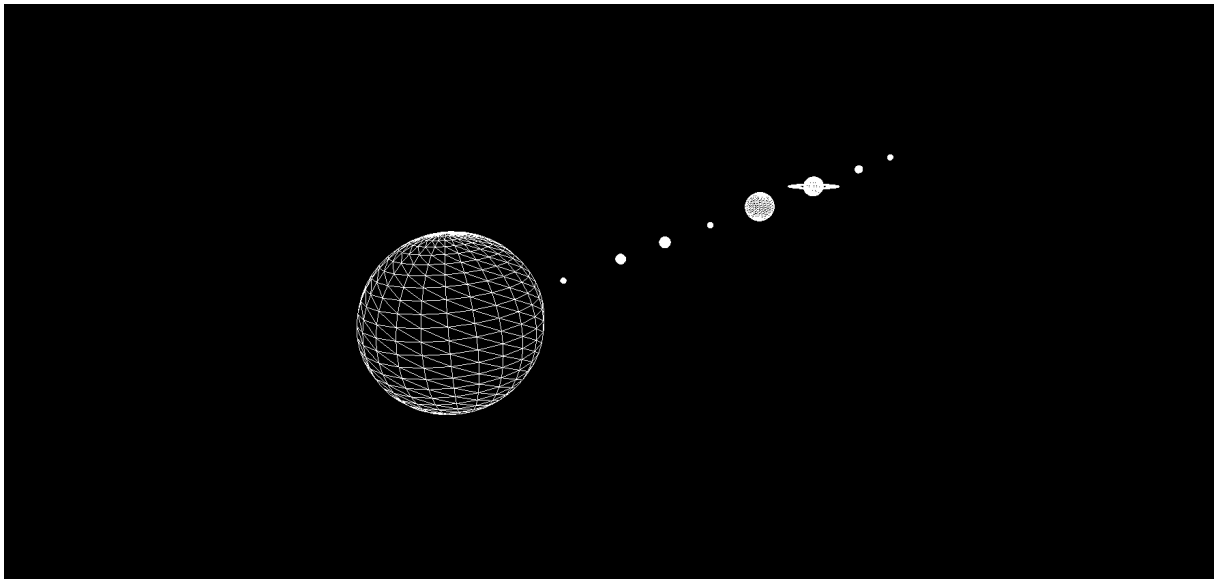

Figure 4: "Demo do Sistema Solar"

Este texto explica a demo do sistema solar alinhado. No entanto, o grupo também desenvolveu uma demo em que os planetas não estão alinhados, proporcionando diferentes perspectivas e experiências visuais.

6. Resultados Obtidos

Após a implementação das transformações geométricas hierárquicas, organização de cenas complexas e aprimoramento da câmara no estilo "first-person shooter" (FPS), realizamos uma série de testes para avaliar a funcionalidade e integridade do sistema. Estes testes foram projetados para garantir que o sistema atendesse aos requisitos estabelecidos para a Fase 2 do projeto. Vamos agora apresentar os resultados desses testes.

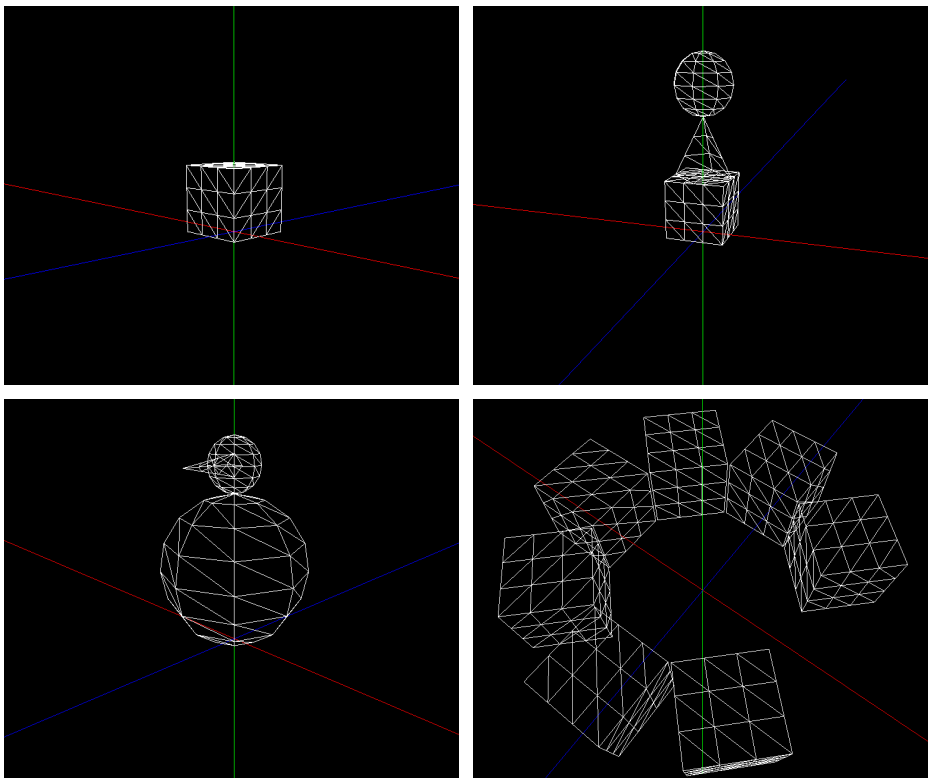

Figure 5: "Resultados Obtidos nos Testes da Fase 2"

Todos os quatro testes planejados foram executados com sucesso, demonstrando que o sistema atende aos requisitos estabelecidos. Os modelos tridimensionais foram corretamente posicionados, rotacionados e dimensionados conforme esperado, refletindo a aplicação adequada das transformações geométricas hierárquicas. Além disso, a organização das cenas complexas permitiu uma representação fiel dos objetos num ambiente 3D. Aprimoramentos na câmara FPS proporcionaram uma experiência de visualização mais imersiva e interativa. Em resumo, os resultados obtidos validam a conclusão bem-sucedida da Fase 2 do projeto, demonstrando a funcionalidade e integridade do sistema desenvolvido.

7. Controlos Importantes

Aqui estão os controlos essenciais para utilizar a aplicação:

- **Movimento:**
  - Utilize as teclas W, S, A e D para mover-se na cena.
  - As setas direcionais permitem rotacionar a câmera.
  - Pressione PgUp para mover-se para cima e PgDown para mover-se para baixo.

- **Visualização:**
  - As teclas numéricas 1, 2 e 3 alteram os modos de visualização: linha, preenchimento e ponto, respectivamente.
  - Use o sinal de adição (+) para aumentar a velocidade de movimento e o sinal de subtração (-) para diminuí-la.
  - Pressione F1 para remover/adicionar o referencial da cena.

Para entrar no modo de visualização de primeira pessoa (FPS) e ocultar o cursor, clique na janela do programa. Para sair deste modo e exibir o cursor novamente, pressione a tecla ESC.

8. Conclusões

Após a implementação das transformações geométricas hierárquicas e da câmara FPS, estamos satisfeitos por termos alcançado os objetivos estabelecidos para a Fase 2 do projeto. Essas funcionalidades representam não apenas avanços significativos na capacidade do sistema em lidar com ambientes 3D complexos, mas também marcam um progresso substancial em direção aos objetivos finais do projeto.

A introdução das transformações geométricas hierárquicas e a organização eficiente de cenas destacam-se como elementos fundamentais para a criação de ambientes virtuais imersivos e realistas. A capacidade de manipular hierarquias de modelos de forma intuitiva abre caminho para a construção de cenários cada vez mais detalhados e interativos.

A implementação da câmara FPS adicionou uma nova dimensão à interatividade do sistema, permitindo aos utilizadores explorar as cenas 3D a partir de uma perspectiva mais imersiva. O movimento livre proporciona uma experiência de visualização mais natural e dinâmica, melhorando significativamente a usabilidade do sistema.

No entanto, reconhecemos que ainda há espaço para melhorias e refinamentos nas próximas fases do projeto. Algumas áreas que merecem atenção incluem:

- **Melhorias na Interface do Utilizador:** Refinar a interface do utilizador para tornar o sistema mais intuitivo e fácil de usar, permitindo aos mesmos aceder e controlar as funcionalidades de forma mais eficiente.
- **Aprimoramento da Interatividade:** Explorar formas de aprimorar a interatividade do sistema, como adição de recursos de detecção de colisão e interação física entre objetos na cena.
- **Otimização de Desempenho:** Uma melhoria identificada é a renderização contínua de pontos mesmo quando não há movimento na cena (re-render é provocado pelo movimento do rato, apesar de não haver movimento por parte da câmara). Isso resulta em re-renders desnecessários, impactando o desempenho do sistema.