Computação Gráfica (3º ano)

Trabalho Prático

Relatório

João Miguel Faria Leite (A94285)

Joel Vieira Barros (A94692)

Sandro Manuel Fernandes Duarte (A94731)

14 de abril de 2023

Resumo

Este documento visa apresentar o trabalho prático realizado no âmbito da Unidade Curricular de Computação Gráfica, que tem como objetivo o desenvolvimento de uma estrutura hierárquica de uma cena usando transformações geométricas.

Como teste desta hierarquia foi pedido um modelo de demonstração do sistema solar. No desenvolvimento da cena foi tirado partido da técnica de guardar e recuperar a matriz de transformação do OpenGl.

Após a conclusão desta fase foram obtidos os resultados esperados. As transformações geométricas apenas afetam os grupos aninhados dentro da sua estrutura, fazendo com que estas sejam relativas ao grupo que as envolve.

Conteúdo

Capítulo 1

Introdução

Na primeira fase deste projeto foram desenvolvidas duas aplicações: o gerador, que permite criar primitivas gráficas; e o motor, que lê ficheiros .xml de configuração, utilizando modelos criados pelo gerador, para exibir um modelo gráfico com a API OpenGl.

Para esta segunda fase do Trabalho Pratico foi pedido para alterar o motor de forma a que os modelos a desenhar sigam uma hierarquia de transformações geométricas.

Nesta fase do projeto foi proposta a implementação da cena como uma árvore, onde cada nodo contém uma série de transformações geométricas e, opcionalmente, um conjunto de modelos.

As transformações apenas existem dentro de um grupo e são aplicadas a todos os modelos e subgrupos da hierarquia. Só pode existir uma transformação de cada tipo.

Para a demonstração desta estrutura foi pedido um modelo estático do sistema solar, incluindo o sol, os planetas e as suas respetivas luas.

Capítulo 2

Estrutura do código

De modo a estruturar o projeto foram adicionadas novas diretorias. Vai ser dada apenas uma breve descrição sobre a funcionalidade do código. Para referência mais detalhada, consultar a documentação.

2.1 common

Nesta diretoria encontra-se todo o código utilizado por ambas aplicações.

2.1.1 cg_math

Contém código de estruturas e classes que abstraem componentes matemáticas.

- `Vec3f`

  Abstração das funcionalidades de um vetor do tipo (x, y, z).

2.1.2 cg_utils

Contém código que fornece estruturas utilitárias. Neste momento, apenas consistem em funções para auxilio no parsing básico.

- `Parser`

  Abstração das funcionalidades de parsing de inteiros e virgula flutuantes.

2.2 engine

Contém código utilizado pelo motor.

2.2.1 transform

Contém código que abstraí a aplicação das transformações geométricas da API OpenGl de modo a que sejam utilizadas pelos grupos.

- `Transform`

  Abstração de transformações geométricas. Utilizada como interface.

- `Translate`

  Classe concreta que move o sistema de eixos por um vetor.

- `Rotate`

  Classe concreta que roda o sistema de eixos por um ângulo e um vetor.

- `Scale`

  Classe concreta que escala de maneira uniforme o sistema de eixos por um escalar.

2.2.2 world

Contém código que representa a implementação das configurações presentes num ficheiro .xml.

- `World`

  Abstração da cena completa.

- `Window`

  Abstração da janela de visualização. Neste momento apenas guarda o tamanho original da janela.

- `Group`

  Abstração dos grupos a desenhar. Contém coleções de transformações geométricas, modelos e grupos aninhados. Foi concebido de maneira a ser uma espécie de lista ligada para garantir a hierarquia.

- `Model`

  Abstração do modelo. Contém uma coleção de vértices. São utilizados VBO, mas apenas iremos entrar em detalhe da sua implementação na terceira fase do projeto.

- `Camera`

  Abstração da camera e das suas funcionalidades.

Capítulo 3

Hierarquia de Transformações

Este projeto tem como objetivo a implementação de uma hierarquia de transformações geométricas. De modo a conseguir esse feito foi criada uma classe, Group, que contém as transformações e os modelos que estas afetam.

```
1 class Group {
2         // ...
3         std::vector<Transform*> mTransforms;
4         std::vector<Model*> mModels;
5         std::vector<Group*> mChildGroups;
6 };
```

Listing 3.1: Excerto de código de Group

Na criação do grupo, o algoritmo procura as transformações e modelos no ficheiro .xml e, se encontrar outro grupo, guarda o apontador para o seu sucessor.

Caso não encontre, fica com um vetor vazio.

Na altura de desenhar, guardamos a matriz de transformações original de forma a obter, no final, o sistema de eixos original.

São aplicadas as transformações seguidas pela renderização dos modelos.

Por fim, é aplicado o mesmo tratamento aos grupos "filhos".

```
1 void Group::Draw() {
2         glPushMatrix();
3         // Apply all transforms to the group
4         for (auto &transform: mTransforms) { transform->Apply(); }
5         // Draw all models in the group
6         for (auto &model: mModels) { model->Draw(); }
7         // Draw child group
8         for (auto &child: mChildGroups) { child->Draw(); }
9         glPopMatrix();
10 }
```

Listing 3.2: Excerto de código de Group::Draw

De notar que até ao processamento do grupos sucessores ainda não foi recuperada a matriz original pois queremos que os grupos "filhos"herdem a matriz do "pai", de forma a que as suas transformações sejam relativas.

Após terem sido tratados todos os sucessores, recuperamos a matriz. Estas operações ocorrem de forma recursiva e sequencial, permitindo assim a recuperação da matriz original apenas no fim do tratamento dos grupos todos, de forma a não influenciar os grupos não relacionados.

Capítulo 4

Conclusão

Com o desenvolvimento da segunda fase do projeto, foi-nos possível consolidar os conhecimentos adquiridos nas aulas, em particular, a importância da ordem das transformações geométricas assim como a funcionalidade da matriz de transformações.

Ao longo desta fase, enfrentamos alguns obstáculos, sobretudo na implementação da estrutura de dados da hierarquia de transformações. Foi ponderado o uso de uma Binary Space Partition Tree, de forma a utilizar o algoritmo do pintor mencionado nas aulas teóricas. Contudo, devido ao grau de dificuldade acrescido na sua implementação, optamos por utilizar vetores. Estes operam como arrays com elevado overhead computacional, mas facilitam o seu uso.

Outro percalço foi a utilização da câmara livre na cena, sendo que esta ainda não se encontra totalmente funcional. Conseguimos movimentar livremente, mas no primeiro movimento a câmara roda $180^{\underline{o}}$ sobre ela. É possível navegar com o rato para rodar e com as teclas w(andar em frente), a(andar para a esquerda), s(andar para trás) e d(andar para a direita).

Em suma, consideramos que foi elaborado um trabalho bastante positivo e que, apesar das dificuldades encontradas, foram superados muitos desafios, tendo-se concluído, dessa forma, todos os requisitos com sucesso.