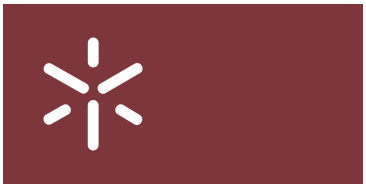


Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de Computação Gráfica

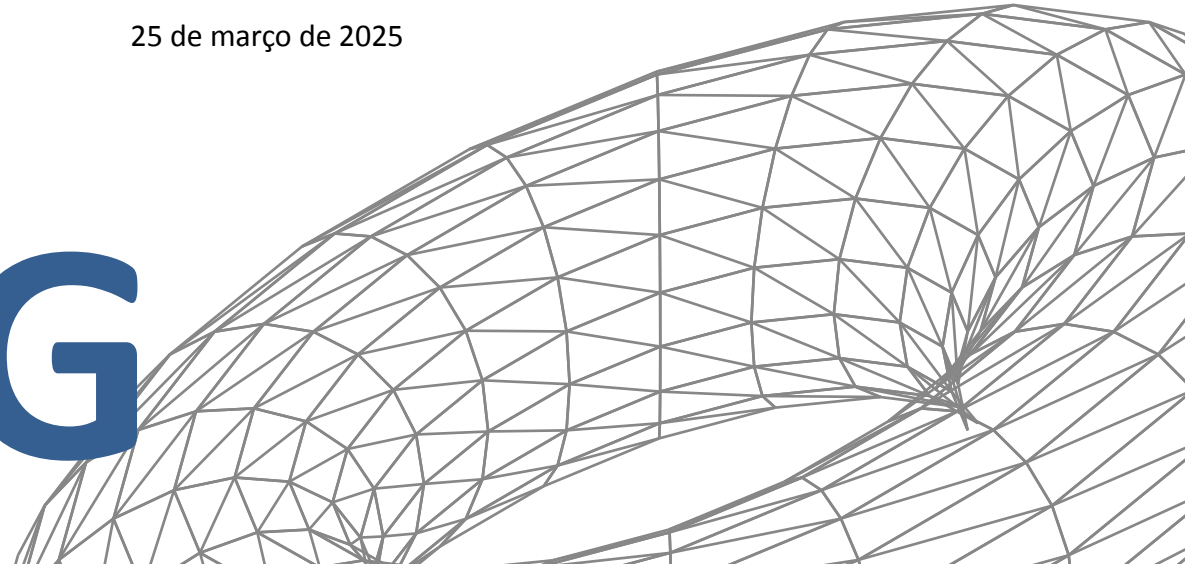
Ano Letivo de 2024/2025

Trabalho Prático - Fase 2

Eduardo Faria	**Hélder Gomes**	**Nuno Silva**	**Pedro Pereira**
a104353	a104100	a104089	a104082

25 de março de 2025

CG



Data da Receção	
Responsável	
Avaliação	
Observações	

Trabalho Prático - Fase 2

Eduardo Faria	**Hélder Gomes**	**Nuno Silva**	**Pedro Pereira**
a104353	a104100	a104089	a104082

25 de março de 2025

Resumo

Na segunda fase do projeto, o grupo desenvolveu cenas hierárquicas recorrendo a transformações geométricas. O trabalho teve início com a implementação de estruturas em memória que permitissem a gestão dos dados necessários para o desenho. Em seguida, foram concebidas as funcionalidades responsáveis pela renderização, com base nas instruções definidas no ficheiro de configuração. O grupo elaborou esse ficheiro, posicionando o Sol, os oito planetas e respetivas luas. Dada a complexidade associada às faixas de asteroides, optou-se pela criação de um gerador de ficheiros XML para configurar o sistema solar. Como funcionalidade extra, e com o objetivo de facilitar a visualização dos objetos renderizados, foi implementada uma câmara livre, que permite ao utilizador posicionar a sua visão em qualquer ponto do espaço.

Área de Aplicação: Computação Gráfica, Modelação Tridimensional e Motores de Renderização

Palavras-Chave:Computação Gráfica, OpenGL, Primitivas Gráficas, Modelação 3D, XML

Índice

1. Transformações Geométricas

Para a correta implementação de transformações geométricas, foi introduzida a noção de cenas hierárquicas com base no agrupamento de modelos e a sucessiva aplicação de transformações — *translate*, *rotate* e *scale*.

1.1. Ficheiro de configuração de cena

Nesta fase, o ficheiro de configuração de cenas sofreu uns ajustes para acomodar as transformações geométricas hierárquicas. As entradas do tipo *group* passaram a opcionalmente receber transformações do tipo *transform* e modelos da terminologia *models*.

Os elementos *transform* definem o conjunto de operações de transformações geométricas a aplicar nos modelos do grupo/subgrupos. As transformações permitidas são as operações de *translate*, *rotate* e *scale*, sendo cada um parametrizada de acordo com o seu tipo.

Os modelos definidos dentro do elemento *models* estão sujeitos às transformações aplicadas no próprio grupo ou em grupos pais.

Em seguida segue um exemplo de uma estrutura simples e válida para um ficheiro de configuração de cena.

```
<world>
    <window width="1080" height="720" />
    <camera>
        <position x="10" y="10" z="10" />
        <lookAt x="0" y="0" z="0" />
        <up x="0" y="1" z="0" />
        <projection fov="60" near="1" far="1000" />
    </camera>
    <group name="SolarSystem">
        <group name="Sun">
            <transform>
                <scale x="3" y="3" z="3" />
                ...
            </transform>
            <models>
                <model file="../../objects/sphere.3d" />
                ...
            </models>
            ...
        </group>
    ...
    </group>
```

Figura 1: Exemplo da estrutura de um ficheiro de configuração de cena.

1.2. Implementação das transformações

Naturalmente, as estruturas de dados associadas ao armazenamento da informação dos modelos necessitaram de ser reestruturadas. Introduziu-se a noção da estrutura de dados `struct GroupConfig {...}`, que armazena as transformações a ele associadas e os modelos e conjunto de subgrupos (da mesma estrutura de dados) que irão sofrer as alterações.

A renderização dos modelos passou a ser feita de modo recursivo. De modo geral, os modelos eram desenhados (iterativamente por cada grupo) da seguinte forma:

1. Fazer uso da função `glPushMatrix()` para colocar o estado da matriz atual na primeira posição da *stack* de matrizes.
2. Aplicar todas as transformações associadas ao grupo.
3. Renderizar os modelos.
4. Repetir o processo, recursivamente, para cada subgrupo do grupo.

1.3. Exemplo de configuração

Com a implementação das alterações previamente mencionadas, é possível recorrer a cenas com vários conjuntos de modelos com transformações geométricas aplicadas. Segue um exemplo de uma cena possível com a atual implementação:

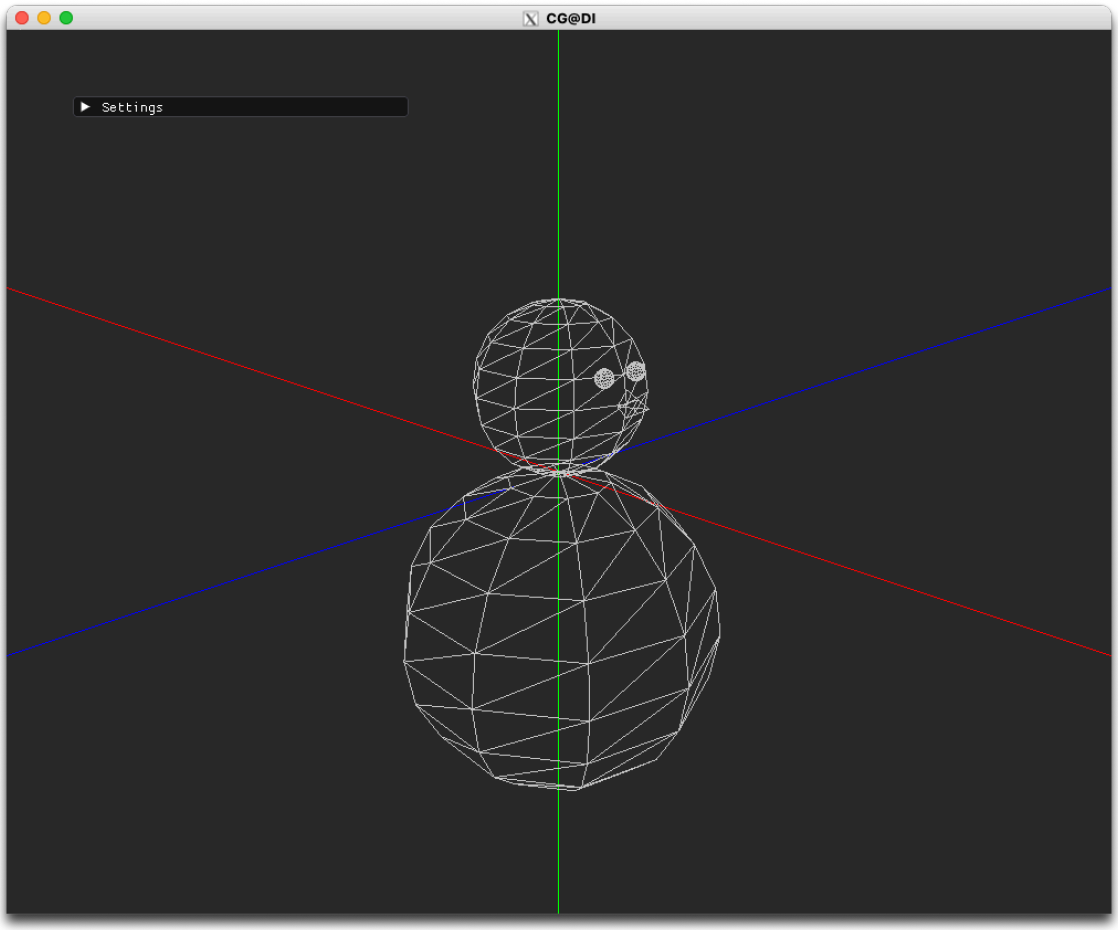


Figura 2: Exemplo de uma configuração de cena.

2. Sistema Solar

Foi desenvolvido um ficheiro de configuração de cena que define a estrutura hierárquica de objetos representativos de um sistema solar. Este ficheiro especifica transformações geométricas aplicadas a grupos de modelos, respeitando a ordem e herança dessas transformações, conforme descrito no enunciado.

2.1. Escalonamento e Posicionamento Inicial

Numa fase inicial, a equipa procedeu à pesquisa das dimensões reais e distâncias relativas entre o Sol, os planetas e suas luas. Contudo, rapidamente se concluiu que a utilização de valores reais tornava o sistema inviável para visualização no motor gráfico. Para resolver esta limitação, foi definido um fator de escala que permitisse uma representação proporcional, mas visualmente adequada. Com base nessa análise, foi criado manualmente o ficheiro `config.xml`, contendo a definição hierárquica do Sol e dos planetas.

2.2. Geração Automática do Sistema Solar

A adição das luas e das duas faixas de asteroides (Cintura de Asteroides e Cintura de *Kuiper*) revelou-se demasiado complexa para ser feita manualmente, tornando inviável a continuidade do processo por essa via. Como solução, e em alinhamento com o requisito de que apenas a aplicação *engine* deve ser modificada, foi desenvolvido um gerador de XML em *Python*, devido à sua simplicidade e flexibilidade.

Este *script* permite configurar:

- O número de luas por planeta;
- A quantidade de asteroides por faixa;
- A escala individual de cada planeta;
- A distância ao Sol e respetiva rotação em torno deste (representando a translação orbital).

2.3. Transformações Aplicadas

A ordem das transformações geométricas foi cuidadosamente definida, uma vez que a sequência tem impacto direto no resultado visual. A abordagem adotada foi a seguinte:

1. **Rotação** – para definir o novo referencial, simulando a órbita do planeta ou lua;
2. **Translação** – para posicionar o objeto à distância correta em relação ao Sol ou ao planeta correspondente;
3. **Escala** – para ajustar o tamanho final do modelo tridimensional.

2.4. Aprimoramentos Visuais

Com o objetivo de obter uma representação mais realista e visualmente apelativa, foram introduzidos alguns detalhes adicionais:

- Uma rotação sobre o próprio eixo foi aplicada ao anel de Saturno (modelo *torus*), simulando o seu movimento natural;
- Nos asteróides, foi aplicada uma variação aleatória no eixo Y, criando uma distribuição com maior volume e profundidade, aproximando-se de uma representação mais caótica e realista das faixas.

Segue-se uma imagem ilustrativa da versão inicial do sistema solar, ainda sem animações, iluminação ou texturas, mas que permite uma perceção clara da estrutura e organização hierárquica implementada.

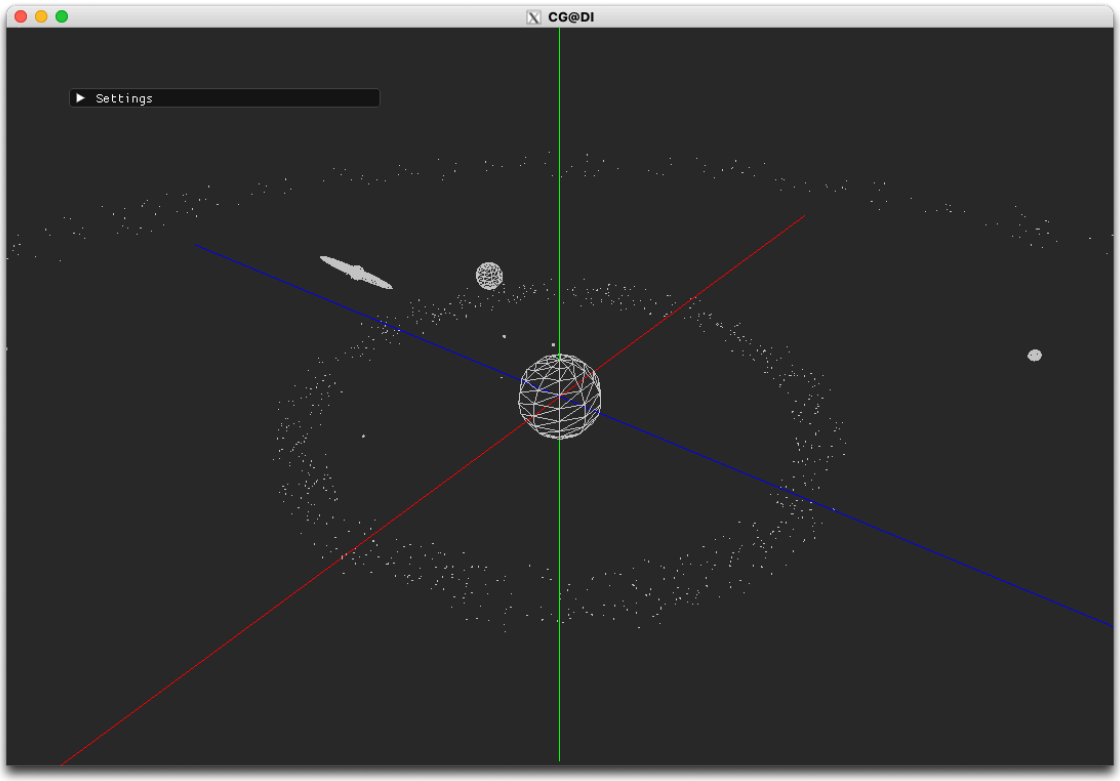


Figura 3: Resultado da geração de cena automática.

3. Extras

3.1. Movimentação da Câmara

Com a inclusão de cenas de larga escala, o grupo implementou um novo tipo de movimentação de câmara. Em adição à movimentação de câmara do estilo *click and drag*, desenvolveu-se uma câmara com movimentação semelhante a câmaras de jogos FPS.

O utilizador do *engine* pode usar a tecla ESC para alternar entre os dois tipos de câmara. Uma vez no modo de movimentação de câmara FPS, o utilizador pode movimentar o rato para alterar o ponto para o qual a câmara está a olhar e fazer uso das teclas WASD para se movimentar em torno do eixo de visão.

Esta movimentação livre da câmara possibilita o utilizador do *engine* explorar a cena através de ângulos previamente inalcançáveis.

3.2. Remoção da Renderização sem VBOs

3.2.1. Contexto

Na fase inicial do projeto, implementámos um menu com uma checkbox que permitia alternar entre a renderização dos modelos com ou sem o uso de VBOs no GLUT. Esta funcionalidade foi concebida para demonstrar a eficiência dos VBOs, conforme comprovado nos testes da Fase 1.

3.2.2. Problemas Identificados

Com a introdução de cenas mais complexas, repletas de triângulos e modelos, foi possível notar uma perda de desempenho por parte do *engine*:

- **Sem VBOs:** Aproximadamente **25 FPS**.[1]
- **Com VBOs (sem índices):** Cerca de **413 FPS**.[1]

A renderização da cena sem VBOs, para uma cena complexa, tornou-se insustentável. Para além disso, para possibilitar a renderização sem VBOs, o programa mantinha listas de pontos para cada modelo, o que se tornava redundante e consumia memória desnecessariamente.

Ficou claro que a alternância entre renderizar com e sem VBOs já não era necessária, uma vez que a comparação feita na primeira fase do projeto já tinha comprovado a superioridade dos VBOs, e manter as listas de pontos para a renderização sem VBOs implicava num uso excessivo de memória, devido à duplicação de dados.

3.2.3. Resultados Obtidos

Assim, o grupo decidiu remover a funcionalidade de alternância. Com esta decisão, deixámos de guardar em memória o conjunto de pontos associado a cada modelo, otimizando o desempenho global.

[1] Testes realizados num *MacBook Air* M1 2020.

A eliminação das listas redundantes permitiu um uso mais eficiente da memória mas mantendo o número médio de FPS. A cena de geração automática conta com **365080 triângulos**. Sabemos que cada triângulo conta com **3 pontos**, e cada ponto com **3 *floats***. O tamanho de um *float* na versão C++11[2] assume um valor de **4 *bytes***[3]. É possível então fazer a assumpção de que, para a cena em questão, foi possível reduzir o consumo de memória em **13.14 MB**.

[2] `printf("C++ version: %ld", __cplusplus).`
[3] `printf("Size of float: %lu", sizeof(float)).`

4. Conclusão

Nesta fase do projeto, alcançámos importantes melhorias tanto nas capacidades quanto na eficiência do *engine*. A implementação de cenas hierárquicas com transformações geométricas permitiu uma estrutura mais clara e modular, a inclusão de um novo modelo de navegação da cena fortaleceu as funcionalidades do *engine*, e a remoção da renderização sem VBOs reduziu o consumo desnecessário de memória e melhorou significativamente o desempenho. Em suma, a fase consolidou as bases para uma renderização mais eficiente e escalável, preparando o terreno para as futuras expansões e melhorias.

Referências

CMake. (2025). CMake. https://cmake.org/

The Khronos Group. (2025). OpenGL. https://www.opengl.org/

John F. Fay, John Tsiombikas, and Diederick C. Niehorster. (2025). freeGlut. https://freeglut.sourceforge.net/

Lee Thomason. (2025). TinyXML2 [Source code]. GitHub. https://github.com/leethomason/tinyxml2

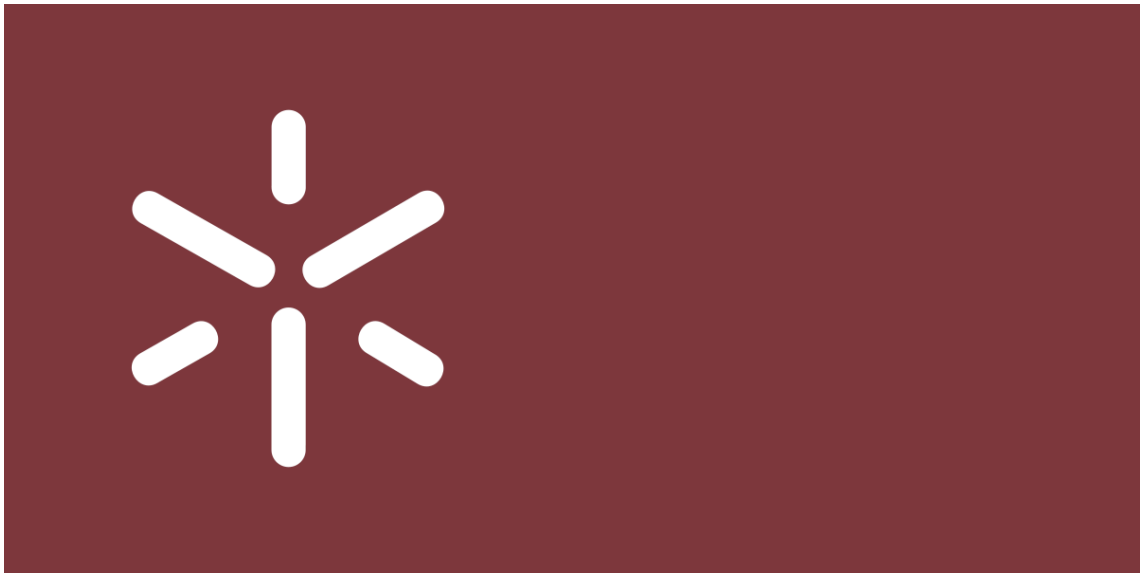
orconut. (2025). Dear imGui [Source code]. GitHub. https://github.com/ocornut/imgui

Lista de Siglas e Acrónimos

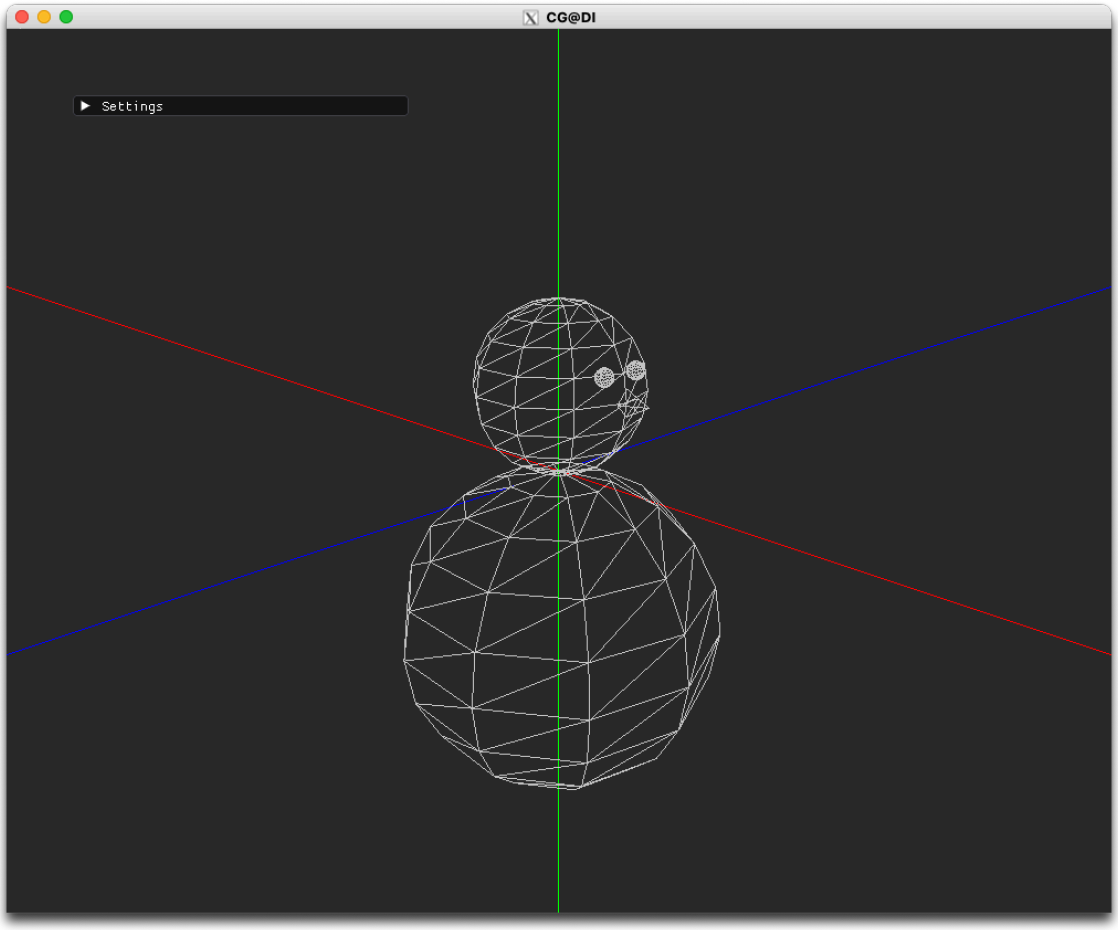
CG Computação Gráfica
XML *Extensible Markup Language*
GLUT *OpenGL Utility Toolkit*
VBO *Vertex Buffer Object*
FPS *Frames per Second*
FPS *First Person Shooter*

Anexos

Anexo 1: Logo da Universidade do Minho



Anexo 2: Exemplo de uma configuração de cena



Anexo 3: Resultado da geração de cena automática