Projeto de Programação Multiparadigma [v1.1]
2021/2022

Enunciado

Um projeto de programação multiparadigma tira partido das vantagens de diferentes paradigmas de programação de forma a produzir uma solução mais adequada (p.e. desempenho, leitura/interpretação, manutenção). Uma divisão clássica consiste na utilização do paradigma de programação orientado aos eventos para o desenvolvimento da **camada de apresentação** (*User Interface*) e outro(s) paradigma(s) (funcional, orientado aos objetos, imperativo, lógico) para o desenvolvimento da **camada de negócio**. A **camada de dados** completa uma típica arquitetura de 3 camadas.

Aviso: O trabalho entregue deve corresponder ao esforço individual de cada grupo, e em nenhum caso deve ser copiado código que será entregue. A deteção de código copiado será realizada por software especializado bastante sofisticado. Casos de plágio óbvio serão penalizados com a anulação do projeto, o que implica a reprovação à Unidade Curricular (UC). Adicionalmente, a situação será reportada à Comissão Pedagógica da ISTA/Conselho Pedagógico do ISCTE-IUL. Serão penalizados da mesma forma tanto os alunos que fornecem código (se for o caso) como os que copiam código.

Grupos de Trabalho

O trabalho deverá ser realizado por grupos de **3 elementos** (exceções necessitam de aceitação explícita por parte do coordenador da UC).

Datas

- Entrega nº1: até 18 de abril (via e-learning) contendo a 1ª parte (no mínimo com as tarefas T1, T3 e T4 - no caso de não ter realizado T2 pode utilizar uma *octree* fixa como input para T3 e T4)
- Discussão nº1: durante uma das aulas da semana 8 ou 9 (21 a 29 de abril)
- Entrega nº2: até 16 de maio (via e-learning) contendo a 1ª parte (melhorada) e a 2ª parte
- Discussão nº2: durante uma das aulas (podem ser necessárias *slots* adicionais) da última semana de aulas – 16 a 20 de maio

Introdução

O objetivo deste projeto é o desenvolvimento de competências de programação que permitam desenvolver, em colaboração, aplicações multiparadigma interativas de média escala. O projeto divide-se em duas partes. A primeira parte corresponde ao desenvolvimento da camada de negócio utilizando, obrigatoriamente, a linguagem de programação **Scala** seguindo os princípios de programação funcional pura (exceções serão explicitamente indicadas). A segunda parte corresponde ao desenvolvimento da camada de apresentação (interativa) que deverá ser desenvolvida utilizando **JavaFX**.

Parte 1: Camada de Negócio

Objetivos: Desenvolver um conjunto de métodos, utilizando o paradigma de programação funcional, capazes de garantir os objetivos funcionais da solução proposta.

Parte 2: Camada de Apresentação Interativa

Objetivos: Desenvolver uma *text-based User Interface* e uma *Graphical User Interface* que permitam uma interação fácil e intuitiva com a aplicação.

Requisitos

O projeto deverá, adicionalmente, satisfazer os seguintes requisitos:

Funcionais

1. Permitir executar com sucesso as várias tarefas/métodos (ver secção Tarefas);
2. Manter estado entre execuções, i.e., guardar o resultado das operações (p.e. `scaleOctree, mapColourEffect)` realizadas para execuções futuras.

Não funcionais

1. Apresentar ambas interfaces: gráfica (*Graphical User Interface – GUI*) e textual (*text-based User Interface*);
2. Os utilizadores deverão conseguir interagir corretamente (com o menor número de erros cometidos) e com facilidade (equilíbrio adequado entre quantidade de informação apresentada e número de ações necessárias para realizar uma tarefa) sem necessidade de treino.

Tema - *View Frustum Culling* com *Octree*

Neste trabalho prático ocupar-nos-emos com a aplicação de *octrees* na divisão e processamento de ambientes 3D. Considerando que um ambiente 3D pode ser representado por um paralelepípedo e, sendo uma *octree* uma estrutura em árvore que divide o espaço em oito partições espaciais (paralelepípedos) iguais, cada uma dessas partições representa em si um espaço que se divide de igual forma. O ambiente 3D assim dividido corresponde a um nó da *octree*, em que se armazena informação geométrica, nomeadamente as coordenadas de um dos vértices (à sua escolha) e dimensões. Os seus descendentes são (as árvores correspondentes a) os seus oito nós. Os nós que não são divididos correspondem a folhas da árvore, onde é guardado informação relativa aos modelos gráficos (i.e., objetos 3D p.e., cilindro) aí contidos. As folhas sem informação são consideradas nós vazios. Naturalmente, utilizando esta estrutura, reduz-se o número de testes necessários para decidir quais são os modelos gráficos que são enviados à placa gráfica apesar de presentes no ambiente 3D, mas fora do volume de visão da câmera não devem ocupar tempo de processamento para uma eficiência otimizada).
Deverá ser desenvolvida uma aplicação que permita carregar e visualizar modelos gráficos a partir de ficheiro textual. A aplicação deve ser invocada a partir da linha de comandos com o nome do ficheiro de configuração (p.e. conf.txt) como parâmetro. O ficheiro de configuração indicará os modelos gráficos (considerar somente os modelos de dimensão fixa *Box(1,1,1)* e *Cylinder(0.5, 1, 10)* - os parâmetros de *Cylinder* correspondem respetivamente ao raio, altura e ao número “fatias”) a serem carregados, componente difusa da cor (i.e. a cor do modelo gráfico) no formato *rgb,* a sua distribuição espacial assim com a escala a aplicar. Cada linha do ficheiro de configuração indica o nome do modelo gráfico e a respetiva translação e escala. A ausência da informação relativa à translação e escala implica utilizar uma translação (0,0,0) e uma escala (1,1,1). O formato para cada linha é o seguinte:

nomeDoModeloGráfico CorDifusa translX translY translZ escX escY escZ

por exemplo:

Cylinder (255,0,0) 10 10 0 0.1 0.1 0.1

o que corresponde a carregar o cilindro (considerar um cilindro fixo com raio 0.5 e altura 1) sendo a componente difusa da cor vermelho, e aplicar ao modelo gráfico a translação (10,10,0), e a escala (0.1,0.1,0.1).

Tendo como ponto de partida a aplicação acima descrita, pretende-se que os modelos gráficos previamente carregados sejam inseridos numa *octree.* Durante a visualização, e mediante movimento da câmera, devem ser visualmente identificados quais as partições espaciais da *octree* que são visíveis a partir da câmera ou seja, se encontram dentro do seu volume de visualização (para simplificar o volume de visualização é representado no código fornecido pelo cilindro azul). Esta é uma das técnicas, em computação gráfica, de otimização mais simples (*View Frustum Culling*) e, geralmente, utilizada em conjunto com *octree* (que permitem uma fácil divisão espacial) para evitar

enviar polígonos não visíveis para a placa gráfica com um número reduzido de testes. As partições espaciais da *octree* são testadas recursivamente para determinar a sua visibilidade e só polígonos dentro de partições espaciais visíveis são enviadas para a placa gráfica.

Simplificações:

- o ambiente 3D a considerar é um **cubo** (o que significa que as partições espaciais também são cubos) na origem e com dimensão máxima igual a 32 unidades em cada eixo (ver cubo vermelho fornecido). Modelos gráficos fora desse espaço podem ser ignorados;
- cada folha da *octree* contém os modelos gráficos que intersetam o seu espaço tridimensional o que significa que cada modelo gráfico poderia estar repetido em todos os nós que interseta (o que não é adequado em termos de gestão de memória). Numa implementação real os modelos gráficos podem ser divididos pelas folhas da *octree* para evitar repetições e permitir uma divisão espacial mais fina, no entanto, isso iria envolver um trabalho de computação gráfica fora do âmbito desta unidade curricular. No âmbito deste projeto, um modelo gráfico que intersete várias partições espaciais deverá ser incluído numa única *OcLeaf* que o possa conter na totalidade – ver anexo I.
- assume-se que o volume de visualização da câmera é representado pelo cilindro azul fornecido (no lugar da típica pirâmide truncada);

Tarefas

Pretende-se que sejam implementadas as seguintes tarefas/métodos:

- **T1** carregar os modelos gráficos (considerar somente cubos e cilindros de dimensão fixa) de acordo com ficheiro de configuração;
- **T2** criar uma *octree* de acordo com os modelos gráficos previamente carregados e permitir a sua visualização (as partições espaciais são representadas com *wired cubes*). A *octree* `oct1` presente no código fornecido poderá ajudar na interpretação;
- **T3** permitir que durante a visualização e mediante movimento da câmera (o movimento é obtido, no código fornecido, clicando no botão esquerdo do rato), sejam visualmente identificados, através de alteração da cor da partição, as partições espaciais da *octree* que sejam visíveis a partir da câmera (i.e., que intersetam o seu volume de visualização). O código dado também fornece uma *third person view* (canto inferior direito) que permite visualizar a *octree* de diferentes perspetivas (através do rato), independentemente da posição da câmera;
- **T4** `scaleOctree(fact:Double, oct:Octree[Placement]):Octree[Placement]` operação de ampliação/redução de uma *octree* e dos modelos gráficos nela inseridos, segundo o fator fornecido (assumir somente a utilização dos fatores 0.5 e 2 – para controlar a complexidade). A ampliação poderá resultar numa *octree* com dimensão máxima superior a 32 unidades;

- **T5** `mapColourEffect(func: Color => Color, oct:Octree[Placement]): Octree[Placement]` função de ordem superior que mapeia uma função em todos os modelos gráficos inseridos numa dada *octree*. Deverá utilizar este método para ilustrar a aplicação dos efeitos sépia[1] e "greenRemove" (efeito que remove a componente verde da cor);
- **T6** desenvolver uma *text-based User Interface* permitindo escolher o ficheiro de configuração, lançar a visualização do ambiente 3D e aplicar os métodos desenvolvido (p.e. `scaleOctree`);
- **T7** desenvolver uma GUI com as mesmas opções que a *text-based User Interface*.

Fatores de valorização:

- Permitir a escolha de diferentes opções de construção da *octree* (p.e. limite de profundidade da *octree*, dimensão da menor partição espacial);
- Utilização de conceitos de programação funcional "avançada" (p.e. *fold*, *tail recursion*);
- Otimizações e funcionalidades adicionais.

Alguns Tipos de Dados a Utilizar

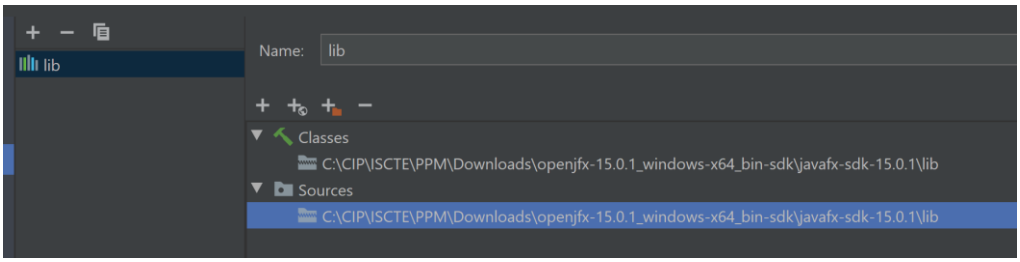
```
type Point = (Double, Double, Double)
type Size = Double
type Placement = (Point, Size) //1st point: origem, 2nd point: size
type Section = (Placement, List[Node])

trait Octree[+A]
case class OcNode[A](placement: A,
                  up_00: Octree[A], up_01: Octree[A],
                   up_10: Octree[A], up_11: Octree[A],
                   down_00: Octree[A], down_01: Octree[A],
                   down_10: Octree[A], down_11: Octree[A]
                  ) extends Octree[A]

case class OcLeaf[A, B](section: B) extends Octree[A]
case object OcEmpty extends Octree[Nothing]
```

A cada partição espacial e cada folha de uma *octree* estão associadas, tal como referido acima, as coordenadas de um dos vértices e a dimensão. Observe-se que a presença do *OcEmpty* se deve à eventual necessidade de dividir o ambiente 3D, p.e., com apenas um modelo gráfico numa partição espacial; neste caso as restantes sete partições espaciais serão vazias.

[1] A nova cor é obtida calculando os componentes RGB da seguinte forma (sendo r, g e b, os valores originais): (R = 40%r + 77%g + 20%b; G = 35%r + 69%g + 17%b; B = 27%r + 53%g + 13%b)

Material Fornecido

- *CameraTransformer.java* – contendo métodos utilitários para lidar com transformações à câmera (exemplo de ilustração da transparência de integração do paradigma de Programação Orientado aos Objetos com o paradigma Funcional);
- CameraView.java – contendo métodos para lidar com uma camaraView
- Javafx-sdk-15.0.1 library (https://gluonhq.com/products/javafx/) adicionado ao projeto:
  File/ProjectStructure/Libraries/+/Java/ 'path to lib'



- Octree.scala – contendo alguns tipos de dados a utilizar;
- Main.scala – contendo a estrutura base para lançar o projeto;

Exemplo de Divisão de um ambiente 3D numa *octree*

A raiz da árvore representa o ambiente 3D representado via *octree*.

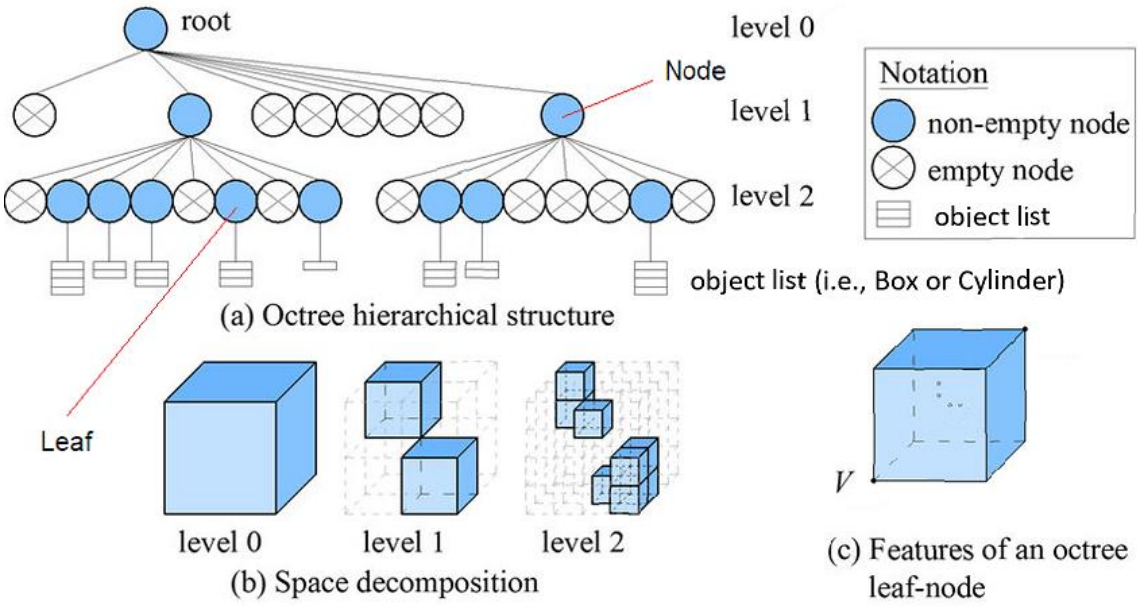


Figura 1 – Representação gráfica e respetiva representação em árvore de uma *octree*[2]

[2] Imagem adaptada de: Vo, A. V., Truong-Hong, L., Laefer, D. F., & Bertolotto, M. (2015). Octree-based region growing for point cloud segmentation. *ISPRS Journal of Photogrammetry and Remote Sensing*, *104*, 88-100

Avaliação

A realização do projeto é obrigatória para obter aprovação à UC. Não existe qualquer possibilidade de obter aprovação à UC sem realizar o projeto. O projeto é composto por duas partes e realizado em grupo, no entanto, as discussões e aferições são individuais. As classificações possíveis na aferição individual são A, B, C ou D, definindo a nota final (*nf*):

- A: *nf* igual à classificação do projeto;
- B: *nf* igual a 85% da classificação do projeto;
- C: *nf* igual a 70% da classificação do projeto;
- D: reprovação à UC.

A **primeira parte** do projeto (com peso de **75%**) será inicialmente avaliado em termos funcionais (i.e., se os métodos produzem os resultados esperados) e se cumpre os requisitos funcionais (para ser usado numa primeira fase via *text-based User Interface* onde a GUI fornecida será nesta primeira parte utilizada somente para efeitos de visualização do ambiente 3D). A solução desenvolvida deverá, obrigatoriamente, aplicar as seguintes matérias de programação funcional introduzidos na UC:

- Recursividade (no lugar de iteratividade) e *Pattern matching*;
- Imutabilidade (exceto na camada interativa);
- *Pure functions* (exceto na camada interativa);
- Funções de ordem superior;
- *Functional Object-Oriented programming*.

Ainda que a funcionalidade exigida tenha sido concretizada, a não aplicação ou aplicação inadequada das matérias mencionadas poderão diminuir a classificação. A avaliação também poderá ser penalizada em função da reduzida qualidade do código (p.e. difícil manutenção, baixa eficiência, dificuldade de interpretação, duplicação, complexidade desnecessária).

A **segunda parte** do projeto (com peso de **25%**) será avaliada em termos da qualidade e usabilidade das interfaces (*text-based User Interface* e GUI) desenvolvidas, do cumprimento dos requisitos e da correta aplicação da matéria sobre engenharia de sistemas interativos (*Event-driven Programming*, *UI patterns* e *Tree Testing*). A avaliação da GUI via *Tree Testing* deve ser realizada com 2 tarefas e 6 participantes[3].

Apesar de não ter um peso na avaliação a utilização de tecnologias para suporte à programação em equipa (e.g. Git, GitHub) é aconselhado.

Os alunos poderão obter feedback juntos dos professores das respetivas turmas sobre o progresso do projeto, e deverão seguir as recomendações dadas.

[3] Os elementos de cada grupo devem participar no *Tree Testing* de pelo menos dois outros grupos

Os projetos são classificados de acordo com os seguintes pesos:

1ª parte (75%):

- Requisitos funcionais – 40%
- Correta e completa aplicação da matéria de Programação Funcional – 25%
- Qualidade do código – 10%

2ª parte (25%):

- Qualidade das interfaces desenvolvidas – 15%
- Correta aplicação da matéria de sistemas interativos – 10%

Entrega

Submissões: ambas as entregas deverão ser, obrigatoriamente, realizadas por cada grupo através do e-learning.

Dados e formatação:

- Projeto com todo o código fonte desenvolvido em formato *Zip* ("File/Export/Project to Zip File…") **contendo todos os dados necessários para poder testá-lo**. O nome do projeto deve incluir o **número do grupo e nome dos vários membros** (*NúmeroGrupo_Nome1_Nome2_Nome3*).
  Sugestão: verifiquem que a importação do vosso projeto numa máquina diferente ocorre com sucesso antes de efetuarem a submissão;
- Resultados do *Tree Testing* (ficheiro: Link_TreeTesting.txt) contendo o link para os resultados obtidos (optimalworkshop link);
- Possíveis comentários (ficheiro: Readme.txt).

Os ficheiros (Link_TreeTesting.txt, Readme.txt) devem ser adicionados à raiz do projeto.

Cada membro do grupo deve submeter individualmente na entrega final a autoavaliação do trabalho em grupo (ficheiro: grupo_*NúmeroGrupo*.pdf) – ver tabela no anexo II. Este ficheiro deverá ser submetido através do e-learning num outro link de submissão que será criado para o efeito.

Discussão: a discussão é individual e todos os alunos terão de demonstrar ser capazes de fazer um trabalho com o nível de qualidade igual ao que entregaram. Ao entregar o trabalho, os alunos implicitamente afirmam que são os únicos responsáveis pelo código entregue e que todos os membros do grupo participaram de forma equilibrada na sua execução, tendo todos adquirido os conhecimentos necessários para produzir, individualmente, um trabalho equivalente.

Código que pode ser útil:

```
val b = new Box(10,10,10) //origem está no meio da Box
b.setTranslateX(5)
b.setTranslateY(5)
b.setTranslateZ(5) //origem passa a ser canto inferior da Box
val h:Node = new Box(3,3,3)
h.setTranslateX(2)
h.setTranslateY(2)
h.setTranslateZ(2)

//verificar se um modelo contém outro
b.getBoundsInParent.contains(h.asInstanceOf[Shape3D].getBoundsInParent)

//verificar se um modelo interseta outro
h.asInstanceOf[Shape3D].getBoundsInParent.intersects(b.getBoundsInParent)
```

Anexo I - Representação gráfica do algoritmo

Se durante o processo de divisão da *octree* em partições espaciais, cada modelo gráfico (p.e. o cilindro na imagem abaixo) não couber totalmente numa das partições criadas (p.e. cubos 1 e 2 na figura 2) então, estes devem ficar na *OcLeaf* "ascendente" onde possam ficar totalmente contidos (i.e., cubo 3 da figura 2). No exemplo da figura 2, a *octree* correta será composta somente por uma *OcLeaf* contendo o cilindro (não existindo neste caso partições espaciais).
Nota: Este algoritmo pode, em certos casos, não reduzir o número de testes necessários para identificar os modelos gráficos presentes no volume de visualização da câmera. Esta é uma desvantagem que advém da considerada não divisão dos modelos gráficos e manutenção nas folhas da *octree*.
A figura 3 representa a situação em que o modelo gráfico cabe totalmente numa das partições criadas.

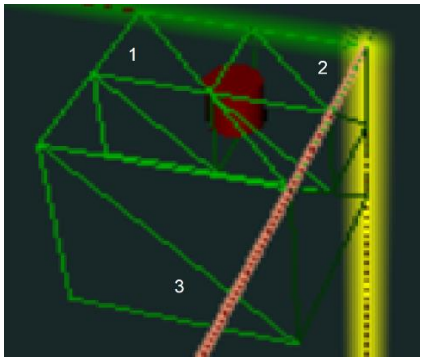


Figura 2 – partição espacial errada

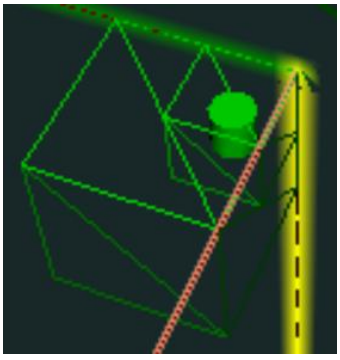


Figura 3 – partição espacial

Anexo II - Avaliação do trabalho em grupo

Classifique os membros do seu grupo (incluindo-o a si próprio) acerca dos itens seguintes numa escala até 20. Coloque o nome de cada membro no cabeçalho da respetiva coluna.

Responsabilidades	***Minha (preencher)***	***MembroA (preencher)***	***MembroB (preencher)***
Contribuiu para a sua parte do trabalho			
Assistiu a todas as reuniões calendarizadas			
Foi cooperativo			
Teve uma atitude positiva			
Foi tolerante a outros pontos de vista			
Teve espírito de grupo			
Completou todo o trabalho a si atribuído e no tempo estipulado			
Esforçou-se para produzir resultados de qualidade			
Incluiu os outros membros nas decisões tomadas e discussões			
Ajudou os outros membros do grupo			