Computação Gráfica (MIEIC)

Aula Prática 1

Introdução a WebGL e configuração de projeto

Objetivos

- Instalar, explorar e aprender a utilizar as bibliotecas e exemplos de base para os trabalhos, bem como os procedimentos para a submissão de resultados
- Aprender a criar e utilizar uma interface gráfica (GUI) para controlar aspectos da cena e os seus objetos.
- Aprender a criar objetos simples

Introdução

Atualmente é possível gerar gráficos interativos 3D em browsers web, recorrendo à tecnologia **WebGL** e à linguagem **JavaScript**.
Esta forma de desenvolvimento 3D tem as vantagens de não necessitar da instalação de bibliotecas ou a compilação de aplicações, e de poder facilmente disponibilizar as aplicações em diferentes sistemas operativos e dispositivos (incluindo dispositivos móveis). No entanto, para aplicações mais exigentes, é recomendável a utilização de linguagens e bibliotecas mais eficientes, como o **C++** e o **OpenGL**. Porém, dado que a API **WebGL** é baseada em **OpenGL** (mais especificamente **OpenGL ES 2.0/3.0**), há uma série de conceitos comuns, pelo que o **WebGL** pode ser visto como uma boa plataforma de entrada nas versões atuais da tecnologia **OpenGL**.
No contexto de **CGRA**, iremos então recorrer ao **WebGL** e a **JavaScript** para criar pequenas aplicações gráficas que ilustrem os conceitos básicos de Computação Gráfica, e permitam a experimentação prática dos mesmos, e que podem ser corridas em *web browsers* recentes **(que suportem WebGL)**.
Para apoiar o desenvolvimento, a biblioteca **WebCGF (Web Computer Graphics @ FEUP)** foi desenvolvida pelos docentes da unidade curricular e por alguns estudantes, especificamente para as aulas de **CGRA** e **LAIG**, com o objetivo de abstrair alguma da complexidade de inicialização e criação de funcionalidades de suporte, permitindo a focalização nas componentes relevantes aos conceitos de Computação Gráfica a explorar.

Preparação do ambiente de desenvolvimento

Uma parte importante deste primeiro trabalho prático é a preparação do ambiente de desenvolvimento. Deve para isso garantir que tem configuradas as principais componentes necessárias, descritas abaixo, e garantir que consegue abrir no browser uma aplicação de exemplo.

Componentes necessárias

- **Web Browser com suporte WebGL 2.0**:
    - A aplicação será efetivamente executada através do browser. Uma lista atualizada dos browsers que suportam WebGL 2.0 pode ser encontrada em http://caniuse.com/#feat=webgl2. Atualmente, os browsers Firefox, Google Chrome e Opera têm suporte para esta versão de WebGL, tanto a versão desktop como versão para Android (embora nalguns casos nem todos os dispositivos que podem correr esses browsers tenham capacidades gráficas para executar as aplicações WebGL).

- **A estrutura de base do projeto**:
    - Inclui a biblioteca WebCGF e outras bibliotecas associadas, bem como a estrutura de pastas de base para o projeto e o ficheiro HTML que serve de base/ponto de entrada da aplicação.
    - Um ficheiro .zip com todos os ficheiros necessários, incluindo o código de base para este trabalho prático está disponibilizado no Moodle.
    - Este ficheiro deve ser descompactado para uma pasta própria, de forma a conter as pastas ***lib*** (a biblioteca) e ***example1*** (o código do projeto de exemplo).
    - Por questões relacionadas com restrições de segurança dos browsers, a pasta com esta estrutura deve ser disponibilizada por um servidor web/HTTP (ver ponto seguinte)

- **Servidor HTTP**:
    - Sendo as aplicações acessíveis via browser, e dadas as restrições de segurança dos mesmos que impedem o acesso a scripts através do sistema de ficheiros no disco local, é necessário que as aplicações sejam disponibilizadas através de um servidor web HTTP. No contexto de CGRA, não haverá a geração dinâmica de páginas (as componentes dinâmicas são corridas no interpretador de Javascript do browser), pelo que qualquer servidor HTTP que disponibilize conteúdo estático servirá. Existem várias soluções possíveis para este requisito, incluindo:

        - **Usar um servidor web no próprio computador**: Nalguns casos os estudantes têm já um servidor web (ex: Apache, Node.js) a correr para suportar outros projetos. O mesmo pode ser usado para este efeito, desde que o servidor disponibilize a pasta com a aplicação através de um URL acessível por HTTP. No caso de não terem nenhum servidor, podem correr um mini-servidor usando uma das seguintes opções:
            - **Web server for Chrome:** Um mini-servidor web que corre no próprio Google Chrome: https://chrome.google.com/webstore/detail/web-server-for-chrome/ofhbbkphhbklhfoeikjpcbhemlocgigb
            - **Python**: Caso o Python esteja instalado, pode ser criado um servidor HTTP simples correndo, na pasta que se pretende partilhar via web, o comando seguinte (dependendo da versão de Python):

        - `python -m SimpleHTTPServer 8080` (para versões 2.x)
        - `python -m http.server 8080` (para versões 3.x).
    - **Node.js:** https://www.npmjs.com/package/http-server
    - **Outros servidores alternativos:**
        - https://cesanta.com/binary.html
        - http://nginx.org/en/download.html
        - https://www.ritlabs.com/en/products/tinyweb/
        - https://caddyserver.com/download
        - https://www.vercot.com/~serva/
        - https://aprelium.com/abyssws/

    - **Usar a área web de estudante da FEUP**: colocar o projeto uma pasta dentro da pasta public_html da conta do estudante, e acedendo à mesma através do endereço público **http://paginas.fe.up.pt/~login/mytest** (login será o código de estudante, upXXXX). Neste caso, ficará por omissão acessível a todos (o que pode ser contornado, p.ex. com um ficheiro de controlo de acesso *.htaccess*). Tem também a desvantagem de implicar a edição/atualização dos ficheiros no servidor da FEUP, e obrigar a uma ligação à rede da FEUP para poder editar/carregar a aplicação

- **Um editor de texto ou IDE**: O código que compõe as aplicações será escrito em JavaScript, e armazenado em ficheiros de texto. Para a sua edição existem várias alternativas também:
    - O próprio **Google Chrome** disponibiliza nas suas **"Developer Tools" (Ctrl-Shift-I)** um debugger de JavaScript, que permite fazer execução passo-a-passo, análise de variáveis, consulta da consola, etc. ao código que está a ser corrido no browser, e permite também mapear os ficheiros acessíveis por HTTP aos ficheiros originais armazenados em disco. Pode por isso ser usado como editor e debugger, e é no momento **a solução recomendada**.
    - Qualquer editor de texto pode servir para editar os ficheiros. No entanto, sugere-se fortemente um editor de texto que suporte uma estrutura de projeto com navegação numa árvore de ficheiros, para permitir alternar facilmente entre os diferentes ficheiros que constituirão o projeto (Ex: Visual Studio Code, WebStorm, Brackets, Sublime, Atom, Notepad++, …).

Teste do ambiente de desenvolvimento

Nesta fase deve ter já uma pasta com a aplicação/template, partilhada através de um servidor web. Deve por isso ter também o endereço URL através do qual a pasta é acessível **(NOTA: evitar pastas com espaços e acentos!)**. Abra o browser e direcione-o para o URL referido, e ao fim de alguns segundos deve surgir a aplicação de exemplo, podendo manipular o ponto de vista com o rato, usando o botão esquerdo para rodar a cena, o botão direito para a deslocar lateralmente, e carregando no botão central (ou Ctrl+botão esquerdo) para aproximar/afastar.

Recursos disponibilizados

A biblioteca 'WebCGF'

Estrutura

A biblioteca **WebCGF** (Web Computer Graphics @ FEUP) - tem como classes principais as seguintes:

- ***CGFapplication* (+)** - Gere as questões genéricas de inicialização da aplicação e bibliotecas de apoio, e interliga os outros componentes
- ***CGFscene* (*)** - É responsável pela inicialização, gestão e desenho da cena
- ***CGFinterface* (*)** - É usada para construir e gerir a interface com o utilizador; pode aceder ao estado interno da cena para, por exemplo, ativar ou desativar funcionalidades (p.ex. luzes, animações). Na sua base está a biblioteca **dat.GUI**:
  http://workshop.chromeexperiments.com/examples/gui

A biblioteca contempla também as seguintes classes que representam entidades que podem integrar uma cena (lista não exaustiva):

- ***CGFobject* (*)** - Representa um objeto genérico, que deve implementar o método ***display()***; os objetos a serem criados devem ser sub-classes de ***CGFObject***
- ***CGFlight* (+)** - Armazena alguma informação associada a uma fonte de luz (poderá ser estendida por sub-classes para implementar características adicionais)
- ***CGFcamera* (+)** - Armazena a informação associada a uma câmara

Para a correta execução dos trabalhos, espera-se que **venham a estender as classes assinaladas com (*)**, de forma a implementar as cenas, interface e objetos requeridos em cada um dos trabalhos, tal como exemplificado na secção seguinte.
As classes assinaladas com **(+)** são **classes utilitárias de exemplo, não exaustivas**, a instanciar para facilitar a gestão e armazenamento das entidades associadas (podendo no entanto ser estendidas por sub-classes, se desejarem acrescentar funcionalidades). A biblioteca inclui ainda alguns objetos pré-definidos, como é o caso dos eixos (***CGFaxis***), e mais algumas classes auxiliares, mas que não deverão ser necessárias para os primeiros trabalhos práticos de CGRA.

Interação de base

Em termos de interação, por omissão é possível manipular a vista utilizando o rato da seguinte forma:

- Botão esquerdo - rodar a cena em torno da origem
- Botão central (roda pressionada) - aproximar/afastar; em alternativa, pode ser utilizado CTRL + Botão esquerdo
- Botão direito - "deslizar" a câmara lateralmente

Código de base do exercício

O código de base fornecido para o exercício estende as classes referidas na secção anterior de forma a implementar o desenho de uma cena muito simples.
A classe ***MyScene*** estende ***CGFscene***, e implementa os métodos ***init()*** e ***display()***:

- ***init()*** : Contém o código que é executado uma única vez no início, depois da inicialização da aplicação. É aqui que tipicamente se inicializam variáveis, criam objetos ou são feitos cálculos intensivos cujos resultados podem ser armazenados para posterior consulta.
- ***display()*** : Contém o código que efetivamente desenha a cena repetidamente. Este método será o foco deste primeiro trabalho.

Leia com atenção os comentários disponíveis no código desses dois métodos, pois fornecem informação importante sobre o seu funcionamento e utilização.
Em particular, o código de exemplo contido no método ***display()*** está dividido em três secções:

- Inicialização do fundo, câmara e eixos
- Transformações geométricas
- Desenho de primitivas

Este trabalho focar-se-á no desenho de primitivas e na sua organização, na declaração e uso de transformações geométricas, e na combinação de ambas para produzir uma geometria composta.
A classe ***MyInterface*** estende ***CGFinterface***, e implementa o método ***init()***:

- ***init()*** : Nesta função, a interface gráfica é inicializada. É possível adicionar *inputs* de texto, *checkboxes*, *sliders*, menus *drop-down*, entre outros elementos que podem ser utilizados para interagir com a cena criada. Mais informação sobre a sua utilização está disponível em http://workshop.chromeexperiments.com/examples/gui.

Trabalho prático

Os próximos pontos descrevem os tópicos abordados durante esta aula prática, assim como as tarefas a realizar.

1. Utilização da Interface Gráfica (GUI)

A interface gráfica (GUI) permite interagir com a cena criada, fornecendo diferentes tipos de elementos para controlar determinados aspetos da dita cena.
A GUI surge como um menu colapsável na página da cena, no canto superior direito. No código de base fornecido com o trabalho prático, existe um controlador do tipo ***checkbox*** que controla a visibilidade do eixo de coordenadas.
Para esse efeito, a cena foi preparada para mostrar condicionalmente os eixos, na classe **MyScene**, onde se:

- Inicializa uma variável na cena - *showAxis* - na função **init()** com um valor booleano
- Implementa a funcionalidade que altera a cena de acordo com o valor dessa variável.

A interface gráfica é definida na classe **MyInterface** (sub-classe de *CGFinterface*), da seguinte forma:

- Inicializa o objeto da GUI (classe **dat.GUI (**http://workshop.chromeexperiments.com/examples/gui**)**) na função **init()**
- Adiciona um controlador à GUI, associado à cena e à variável *showAxis* da mesma. Dado que esta foi inicializada com um valor booleano, a GUI infere que deverá ser representada por uma checkbox.

2. Geometria básica e estruturação

O desenho de objetos em **WebGL** e nas versões modernas de **OpenGL** é tipicamente baseado na definição de um conjunto de triângulos com um conjunto de características associadas.
Esses triângulos são definidos por um conjunto de vértices (e possivelmente algumas características associadas a cada vértice), e a forma como os vértices se ligam para formar os triângulos.
No código de base fornecido com o trabalho prático é providenciado o código necessário para criar um **losango (MyDiamond.js)**, e incluído na cena para que possa ser visível.
Considere que esse losango é definido pelos vértices A, B, C e D, que formam dois triângulos de vértices ABC e DCB (Figura 1).

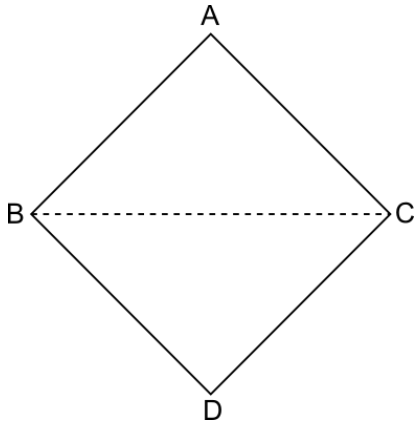

Figura 1: Geometria de exemplo (quadrilátero).

Para criarmos essa geometria, temos que, em primeiro lugar, criar um *array* de vértices com as coordenadas dos quatro cantos.

```
vertices = [
      xA, yA, zA,
      xB, yB, zB,
      xC, yC, zC,
      xD, yD, zD
];
```

Em seguida, devemos indicar como estes vértices serão ligados entre si para formar triângulos.
Para isso criamos um novo *array* que indique, através de índices referentes à ordem dos vértices, como agrupá-los três a três. Neste caso, sendo os triângulos definidos pela ordem ABC e DCB, teremos:

```
indices = [
      0, 1, 2,
      3, 2, 1
];
```

O uso de índices permite reduzir a quantidade de informação necessária para definir a geometria. Em vez de repetirmos 3 coordenadas na lista de vértices quando o mesmo vértice é usado mais do que uma vez, apenas repetimos o seu índice na lista de índices.
Quanto maior for a geometria e o número de vértices partilhados (algo bastante comum em modelos 3D compostos por uma malha de triângulos), mais benefício há em usar os índices para representar a conectividade.
Tendo esta informação definida, o desenho efetivo da geometria implica passar a informação assim declarada em **JavaScript** para buffers do **WebGL** (já alocados na memória gráfica), e instruir o mesmo para os desenhar considerando a sua conectividade como sequências de triângulos.

Na **WebCGF**, a complexidade desta fase final do desenho está encapsulada na classe ***CGFobject***. Dessa forma, para criar um determinado objeto 3D, podemos simplesmente:

- criar uma sub-classe da ***CGFobject***, p.ex. MyObject
- implementar o método **initBuffers**, onde
    - declaramos os arrays acima referidos,
    - invocamos a função **initGLBuffers** para a informação ser passada para o **WebGL**
- Na nossa cena:
    - Criar e inicializar uma instância do novo objeto no método ***init()*** da cena
    - Invocar o método ***display()*** dessa instância do objeto no método *display()* da cena

Na classe **MyDiamond** encontram um exemplo completo, que servirá de base para a criação de outras formas.

Exercício 1

1. Crie uma nova subclasse de *CGFobject* chamada ***MyTriangle***, num ficheiro **MyTriangle.js**, que defina um triângulo rectângulo no plano XY, com lado de 2 unidades, demonstrado na Figura 2, onde o ponto vermelho representa a origem (0,0,0).
2. Na GUI definida na classe ***MyInterface***, adicione dois controladores do tipo ***checkbox***, que deverão controlar a visibilidade do quadrado já apresentado na cena, assim como o triângulo criado na alínea anterior.
3. Crie uma nova subclasse de *CGFobject* chamada ***MyParallelogram***, que cria um paralelogramo como demonstrado na Figura 3, com o vértice mais à esquerda na origem da cena (0,0,0). A sua altura é de 1 unidade, e largura total é de 3 unidades. Deverá ser *double-sided*, isto é, visível em ambos lados (sugestão: deve ser explorada a repetição e ordem dos índices).
4. Adicione outra *checkbox* para controlar a visibilidade do paralelogramo.

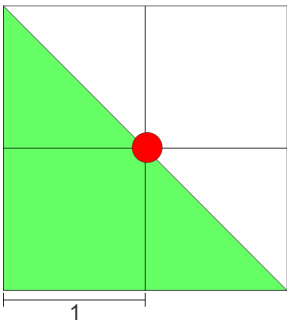

Figura 2: Triângulo retângulo

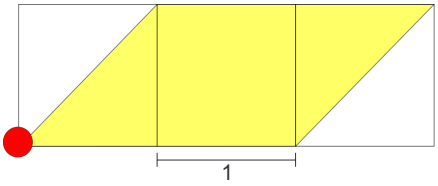

Figura 3: Paralelogramo inversível

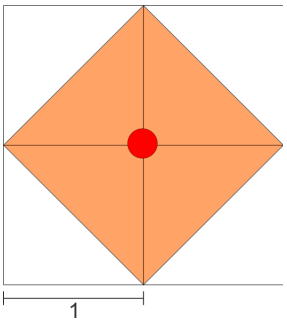

Figura 4: Losango já criado (MyDiamond)

Exercício 2

Crie as peças adicionais representadas na figura 5 e 6.

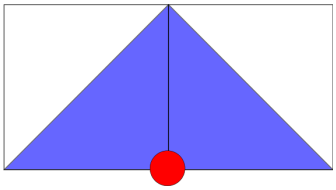

Figura 5: Triângulo rectângulo (pequeno) (MyTriangleSmall**)**

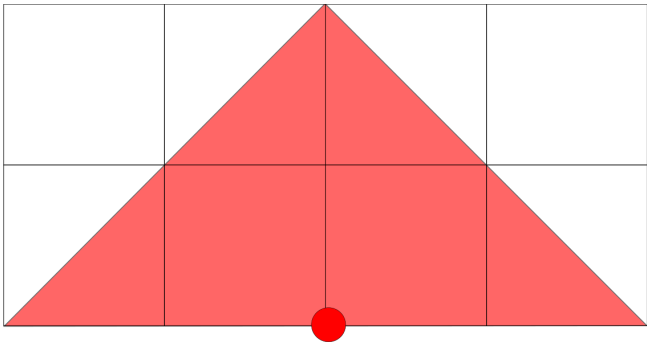

Figura 6: Triângulo rectângulo (grande)(MyTriangleBig)

Submissão

Os exercícios resolvidos nesta aula prática, em conjunto com os exercícios da próxima aula, serão submetidos no Moodle. As instruções de entrega serão fornecidas na segunda aula prática.