

# Computação Gráfica (MIEIC)

## Aula Prática 6

### *Modelação Procedimental*

v1.1 - 2019/04/24

## Objetivos


No final deste trabalho o estudante deverá ser capaz de:


- Descrever o conceito de modelação procedimental e as ferramentas matemáticas aplicadas;
- Aplicar técnicas de modelação procedimental à criação de elementos orgânicos, como vegetação.

As técnicas de modelação procedimental geram, de forma automática, modelos tridimensionais através de processos computacionais que ampliam a informação base segundo um determinado algoritmo. De forma a não ser necessário reprogramar os sistemas de modelação procedimental, recorre-se a métodos matemáticos que operam sobre descrições simbólicas parametrizáveis que, de forma flexível, são capazes de modelar diferentes tipos de objetos. Os métodos matemáticos mais utilizados são a geometria fractal e as gramáticas formais, particularmente os sistemas L e as gramáticas de formas.

## Trabalho prático

Ao longo dos pontos seguintes são descritas várias tarefas a realizar. Algumas delas estão

anotadas com o ícone  (captura de imagem). Nestes pontos deverão capturar uma imagem da aplicação para disco (p.ex. usando Alt-PrtScr em Windows ou Cmd-Shift-3 em Mac OS X para capturar para a clipboard e depois gravar para ficheiro num utilitário de gestão de imagens à escolha). No final de cada aula, devem renomear as imagens para o formato **"ex6-t<turma>g<grupo>-n.png"**, em que **turma** e **grupo** corresponde ao número de turma e grupo definido no ficheiro de grupos TP, e **n** corresponde ao número fornecido no exercício (p.ex. **"ex6-t1g01-1.png"**).

Nas tarefas assinaladas com o ícone  (código), devem criar um ficheiro **.zip da pasta que contém o vosso código (tipicamente na pasta 'ex6', se tiverem código noutras pastas incluam-no também)**, e nomeá-lo como **"ex6-t<turma>g<grupo>-n.zip"**, (com turma, grupo e n identificados tal como descrito acima **"ex6-t1g01-1.zip"**).

No final (ou ao longo do trabalho), um dos elementos deverá submeter os ficheiros via Moodle, através do link disponibilizado para o efeito. Bastará apenas um elemento do grupo submeter o trabalho.

# Preparação do Ambiente de Trabalho

Devem descarregar o código disponibilizado para este trabalho do Moodle e colocar a pasta **example6** contida no ficheiro .zip, ao mesmo nível dos trabalhos anteriores. Devem garantir que têm a versão mais recente da WebCGF (2.0.1 ou superior) na pasta **lib**.

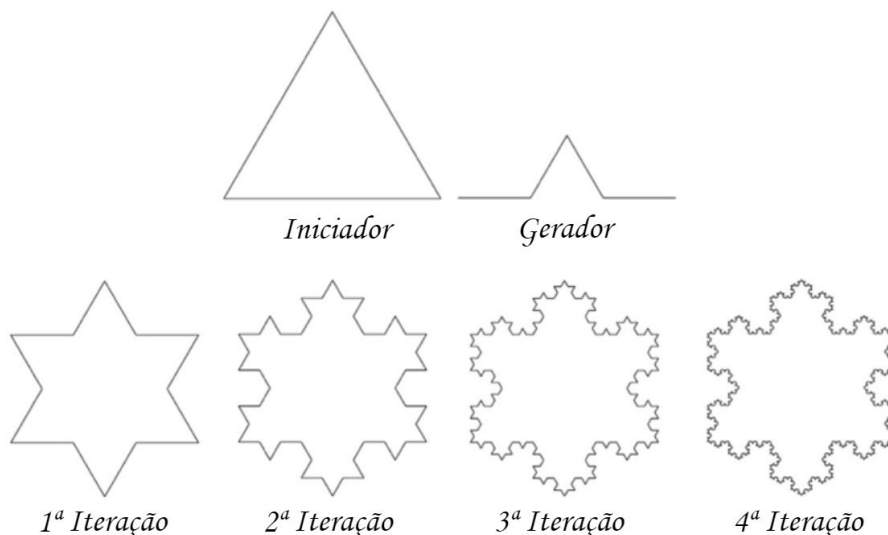
Deverá ter no final uma estrutura de pastas semelhante à seguinte:

- (pasta dos projetos)
  - lib -> atualizada
  - example1
  - (...)
  - example6 -> novo código

## 1. Sistemas L (*L-systems*)

O mecanismo de reescrita é uma técnica que possibilita a definição de objetos complexos através da sucessiva substituição de partes de um objeto inicial, utilizando um conjunto de regras de reescrita - as **regras de produção**. A figura seguinte ilustra graficamente o princípio de funcionamento de um sistema de reescrita, aplicado à construção da curva “flocos de neve”. O sistema inicia-se com dois objetos geométricos: o iniciador e o gerador. A cada passo de desenvolvimento, cada aresta do objeto geométrico em desenvolvimento, a começar com o iniciador, é substituída pela sequência de arestas que compõem o objeto gerador. O processo vai-se repetindo recursivamente até se atingir o resultado gráfico pretendido.

**Figura 1:** Exemplo de Sistema L com reescrita. Contém 4 iterações, 1 iniciador e 1 gerador.

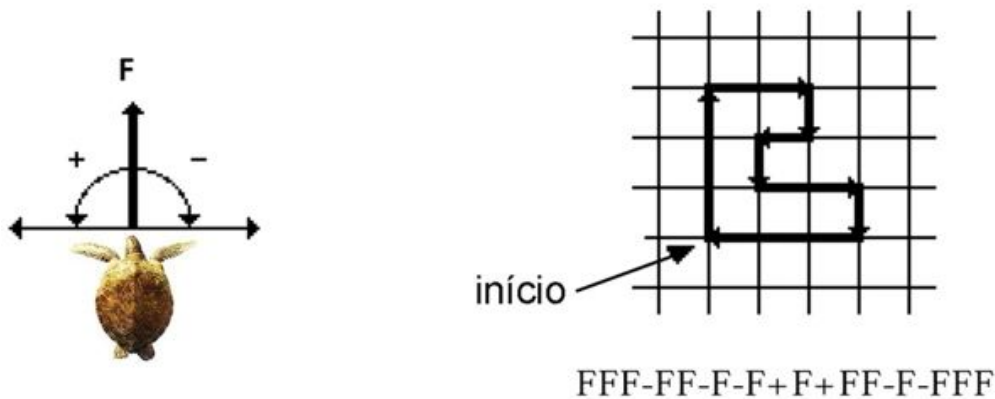


Um sistema L é definido por:

- Uma **gramática** que define os símbolos que fazem parte do sistema. Nos sistemas L é habitual associar-se a cada símbolo um comando gráfico, à semelhança da metáfora da

tartaruga na linguagem LOGO (ver fig. 2). “Avançar”, “Rodar para a esquerda”, “Rodar para a direita” são alguns dos comandos típicos. No nosso exemplo, vamos definir que:

- “F” representa avançar um passo (equivalente a desenhar um traço)
- “-” representa uma rotação de 60° no sentido dos ponteiros do relógio (negativa)
- “+” representa uma rotação de 60° no sentido contrário dos ponteiros do relógio (positiva)
- um **axioma**, definido como uma cadeia de caracteres inicial que descreve o iniciador. No nosso exemplo:
  - Axioma: F--F--F
- um conjunto de **regras de produção** (geradores) que definem como o sistema irá evoluir ao longo da sequência de desenvolvimento. No exemplo, definimos que em cada iteração, “F” será substituído pelo gerador definido pela regra de produção seguinte:
  - Regra de produção:
    - 1:  $F \rightarrow F+F--F+F$



**Figura 2:** Mecanismo de interpretação gráfica de sistemas L, baseada em LOGO.

Um cursor (uma tartaruga) é controlado através de um conjunto de comandos, que o deslocam sobre um plano desenhando figuras geométricas. Neste exemplo, os comandos permitem avançar o cursor (“F”), e rodá-lo 90°, para a esquerda ou para a direita (“+” ou “-”).

A Figura 3 ilustra os dois primeiros passos da sequência de desenvolvimento do mesmo sistema. Como se pode observar, a sequência de desenvolvimento resultante pode iterar indefinidamente dado que, a partir do primeiro passo de derivação, a regra de produção é sempre aplicável, devido ao facto de existir sempre um carácter “F” na cadeia de caracteres. Na implementação de um sistema L deve assim ser definido um número máximo de passos de derivação, para que a sua sequência de desenvolvimento seja finita.



Altere assim o código na função *init()* de **MyScene.js** para refletir estes valores iniciais:

```
this.axiom = "X";
this.ruleF = "FF";
this.ruleX = "F[-X] [X] F[-X]+FX";
this.angle = 30.0;
this.iterations = 4;
this.scaleFactor = 0.5;
```

Verifique a figura gerada... É uma planta! Interaja com os valores de **Angle** e **Iterations** para avaliar as alterações na planta gerada.

## 2. Desenvolvimento de Plantas (MyLSPlant)

Vamos agora criar uma nova classe derivada de **MyLSystem**, designada **MyLSPlant**. Esta nova classe irá modelar plantas com base nas técnicas já desenvolvidas na classe **MyLSystem**.

A modelação de plantas será realizada através de dois símbolos terminais:

- “F”, que gerará um ramo (objeto da classe **MyBranch**);
- “X”, que gerará as folhas (objeto da classe **MyLeaf**).

Deverá criar apenas o construtor da classe e substituir os seguintes métodos:

- **Construtor:**

```
constructor(scene) {
    super(scene);
}
```

- **initGrammar**, que inicializa os símbolos a ser interpretados graficamente e as primitivas correspondentes:

```
initGrammar(){
    this.grammar = {
        "F": new MyBranch(this.scene),
        "X": new MyLeaf(this.scene)
    };
};
```

Deverá também criar as classes **MyBranch** e **MyLeaf**. A classe **MyBranch** deverá conter um cilindro com um material castanho e um número de lados reduzido (3 ou 4). Para a classe **MyLeaf**, poderão utilizar um polígono simples à escolha (visível dos dois lados), que se assemelhe a uma folha (por exemplo, um triângulo), com um material verde aplicado. Os materiais deverão ser aplicados na função *display()* das classes criadas.

Substitua o objeto de **MyLSystem** por um **MyLPlant**, utilizando os valores definidos no fim da secção 2 para gerar a nova planta.

(1  )

### 3. Desenvolvimento estocástico

Dado que não existem duas plantas idênticas, deveremos criar mais do que uma regra que se aplique a cada caractere, para obter variedade. Nesta situação deve-se escolher uma das regras aplicáveis, de forma aleatória.

Altere a função **doGenerate** na inicialização da cena, de forma que a lista de produções usadas para (re)generar o Sistema L de **MyLSPlant** sejam as seguintes (em vez de usar as variáveis de interface):

```
1: F → FF
2: X → F[-X][X]F[-X]+X
3: X → F[-X][x]+X
4: X → F[+X]-X
```

Obs.: Repare que as produções 2, 3 e 4 se aplicam ao mesmo carácter (“X”), tratando-se assim de um Sistema L estocástico. Se analisar o código da classe **MyLSystem** verifica que o método *iterate* seleciona aleatoriamente qual a regra a aplicar. Desta forma, sempre que premir o botão “Generate!” irá gerar uma planta distinta.

### 4. Desenvolvimento em 3D

Repare que as plantas geradas até agora só se desenvolvem no plano XY. Para que estas passem a ser geradas em 3D, temos que ampliar a capacidade de aplicar rotações com mais 4 caracteres:

- “\” - rotação em sentido positivo sobre o eixo dos XX;
- “/” - rotação em sentido negativo sobre o eixo dos XX;
- “^” - rotação em sentido positivo sobre o eixo dos YY;
- “&” - rotação em sentido negativo sobre o eixo dos YY;

Altere o método **display** da classe **MyLSystem** para incluir estes novos caracteres e operações de rotação.

Altere as regras de produção a serem fornecidas a **MyLSPlant** para as seguintes:



```
1: F → FF
2: X → F[-X][X]F[-X]+X
```

- 3:  $X \rightarrow F[-X][x] + X$
- 4:  $X \rightarrow F[+X] - X$
- 5:  $X \rightarrow F[/math> $\backslash$  $X$  $][X] F[\backslash \backslash X] + X$$
- 6:  $X \rightarrow F[\backslash X][X] / X$
- 7:  $X \rightarrow F[/math> $\backslash$  $X$  $]\backslash X$$
- 8:  $X \rightarrow F[^X][X] F[\&X]^X$
- 9:  $X \rightarrow F[^X] \&X$
- 10:  $X \rightarrow F[\&X]^X$

(2  )(1  )

## Checklist

Até ao final do trabalho deverá submeter as seguintes imagens e versões do código via Moodle, **respeitando estritamente a regra dos nomes**:

-  **Imagens (2): 1, 2 (nomes do tipo "ex6-t<turma>g<grupo>-n.png")**
-  **Código em arquivo zip (1):1 (nomes do tipo "ex6-t<turma>g<grupo>-n.zip")**