# Aula prática 2

## Objetivos

Estudo de algumas aplicações dos shaders

Tipos de variáveis: constantes (uniform), atributos (attribute) e interpoladas (varying)

Na aula primeira aula prática necessitámos de usar 2 *fragment shaders* para podermos pintar com cores diferentes, quer o interior, quer a fronteira dum polígono. Por causa disso, fomos também obrigados a usar dois programas GLSL e o código tornou-se mais verboso e complicado. O que seria bom mesmo era termos a possibilidade de enviar para o programa GLSL, a partir da nossa aplicação javascript, a cor com que pretendemos pintar o polígono. Felizmente tal é possível...

## Exercício 2.1 - uniform

Pegue no exemplo do triângulo, disponível aqui, e modifique-o por forma a que o *fragment shader* aceite uma variável **uniform** do tipo **vec4** e com o nome `color`:

```
uniform vec4 color;
```

As alterações no código da aplicação passam por:

- Obter a localização da variável **uniform** usando `gl.getUniformLocation()`
- Adaptar a função `render()` para pintar o interior do triângulo e desenhar a sua fronteira
- Usar a função `gl.uniform4fv()` para enviar para o programa GLSL a cor que o *fragment shader* irá usar

### Revisão

A solução do exercício 1.7 da aula prática 1 passava por usar também uma variável **uniform**. Por exemplo, a variável poderia conter informação acerca do deslocamento em x:

```
uniform float dx;
```

O *vertex shader* poderia assim mudar a coordenada dos vértices de acordo:

```
attribute vec4 vPosition;
uniform float dx;
void main(){
    gl_Position = vPosition;
    gl_Position.x += dx;
}
```

Já na função `render()` da nossa aplicação webgl/javascript, necessitaríamos de duas coisas:

- implementar um ciclo de animação
- passar para o programa GLSL o valor do deslocamento em x (`dx`)

Para o primeiro bastaria colocar no final da função `render()` a seguinte linha de código:

```
function render() {
    ...
    requestAnimFrame(render);
}
```

A função `requestAnimFrame()` pede ao browser para invocar a função passada por argumento da próxima vez que o browser atualizar o conteúdo da página (tipicamente a cada 1/60 segundo).

A segunda alteração à função `render()`, para passar a informação acerca do deslocamento, poderia ser qualquer coisa do tipo:

```
    ...
    gl.uniform1f(..., 0.5*Math.sin(frame));
    ...
```

sendo `frame` uma variável global incrementada a cada passagem pela função `render()`.

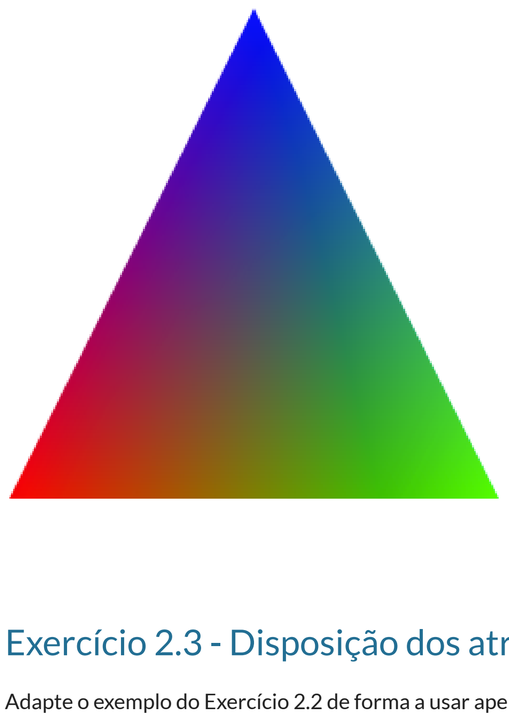
## Exercício 2.2 - varying

Os atributos associados a cada vértice não se limitam à posição dos mesmos no espaço. Neste exercício queremos passar ao programa, não apenas uma posição 2D por cada vértice, mas também uma cor RGB a ele associada. O objetivo é pintarmos um triâgulo com uma cor que varia gradualmente no seu interior, por interpolação das cores associadas a cada vértice.

**Passos necessários:**

- Criar um *buffer* adicional para conter as cores de cada vértice
- Declarar um novo atributo no *vertex shader*, de nome `vColor`
- Declarar uma variável **varying** no *vertex shader*, de nome `fColor`
- Declarar a mesma variável `fColor` no *fragment shader*
- Adaptar o código de ambos os *shaders* de acordo com o pretendido.

Eis um possível output do programa:



## Exercício 2.3 - Disposição dos atributos em memória

Adapte o exemplo do Exercício 2.2 de forma a usar apenas um *buffer*. Experimente fazer de duas formas distintas:

- Guardando primeiro no *buffer* os dados relativos às coordenadas de todos os vértices, seguidos dos dados relativos às cores desses mesmos vértices
- Guardando no buffer a informação de cada vértice em posições de memória contíguas, alternando a informação relativa à posição com a da cor para cada vértice.

Experimente variantes onde a posição dos vértices no programa javascript é constituída por pontos 2D vs. pontos 3D. Faça o mesmo para a cor, mas agora com coordenadas 3D (RGB) e 4D (RGBA). A coordenada A representa a opacidade da cor, o valor 1 significa que a cor é totalmente opaca e 0 significa que é totalmente transparente, logo invisível.

## Exercício 2.4 - TPC

Escreva um programa que faça *morphing* dum polígono noutro, com o mesmo número de vértices.

**Ajuda**: necessita associar a cada vértice duas posições: a inicial e a final e misturá-las no shader com a função `mix()`.