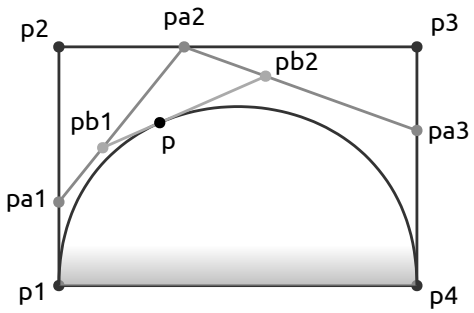
Computação Gráfica - Trabalho 2

Atividade Remota

Em Dupla ou Trio | Estimativa: 8h

Trabalho 2 – Implementação A

A famosa **Curva de Bézier** é muito usada na Computação Gráfica. Ela pode ser construída de diferentes maneiras. Uma dessas maneiras é o chamado **Algoritmo De Casteljau**. Para entendê-lo, considere o ponto **p** na figura à seguir:



O ponto **p** pode ser obtido como uma interpolação linear entre os pontos **pb1** e **pb2** usando um parâmetro **t**. O ponto **pb1** pode ser obtido de forma análoga: via interpolação linear entre os pontos **pa1** e **pa2**, usando o mesmo parâmetro **t** anterior.

De forma semelhante, o ponto **pb2** pode ser obtido via interpolação linear entre **pa2** e **pa3**, usando novamente o mesmo parâmetro **t**.

Cada **pa1**, **pa2**, **pa3** também poderá ser obtido via interpolação linear no seu respectivo segmento (**pa1** como interpolação linear entre **p1** e **p2**, usando **t** como parâmetro; **pa2** como interpolação linear entre **p2** e **p3**; **pa3** por interpolação linear entre **p3** e **p4**). Em todos os casos, usa-se o mesmo parâmetro **t** na interpolação.

a) Faça um algoritmo para calcular **p** à partir de **p1**, **p2**, **p3**, **p4** por meio de chamadas à rotina **LERP**(A, B, t). Escreva o pseudo-código aqui.

b) Implemente seu algoritmo manualmente com **HTML, JavaScript e a API Canvas 2D**.

Obs.: sua implementação deve mostrar os 4 pontos (p1, p2, p3, p4) em posições já predeterminadas e permitir ainda que o usuário possa escolher um deles e mover, mudando qualquer um dos 4 pontos. Você pode usar os eventos *onmousedown / onmousemove / onmouseup*.

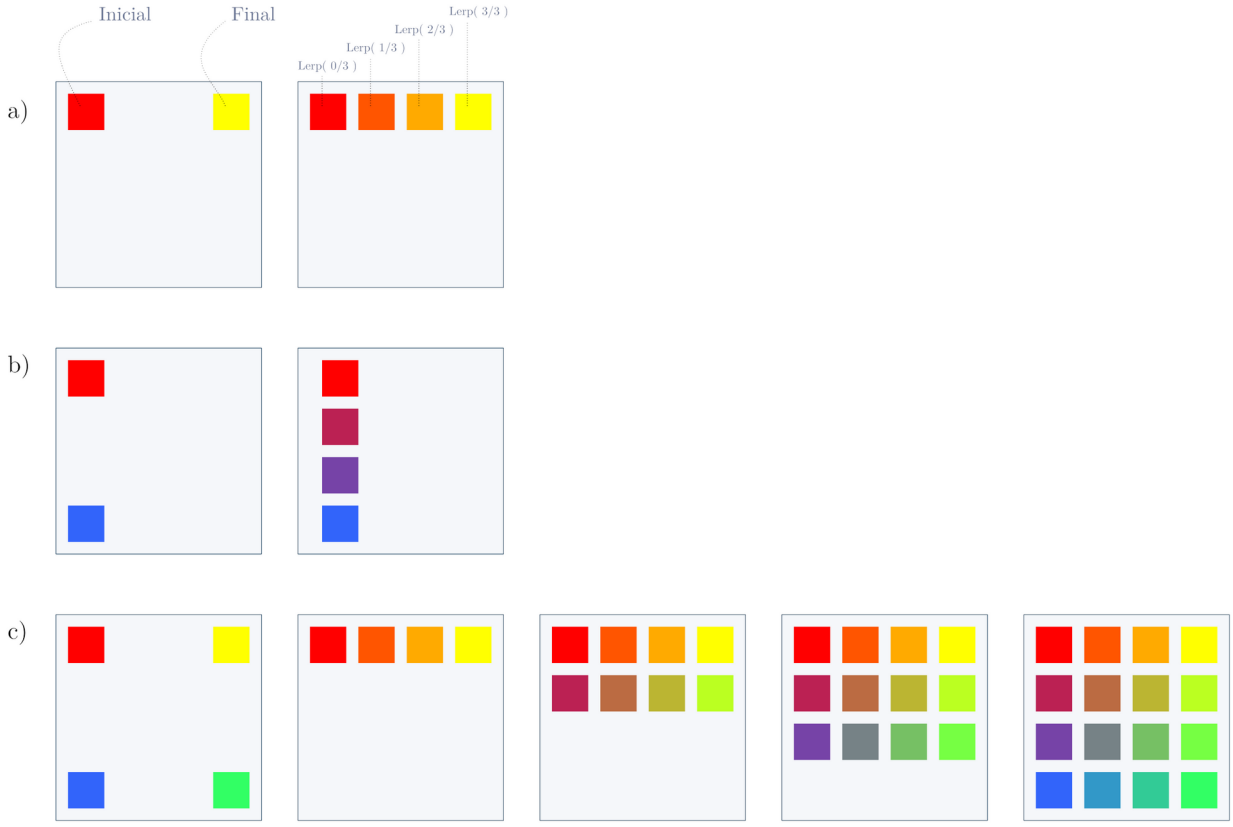
Trabalho 2 – Implementação B

Observe as interpolações abaixo:

No **item (a)**, uma interpolação linear é feita de uma cor inicial até uma cor final. Para N = 4 DIVs, o looping percorre fazendo $i = 0$ até 3. Em cada iteração i, chama a função LERP de forma adequada, construindo assim as cores nessa fileira.

O **item (b)** faz a mesma coisa mas num *layout* vertical.

O **item (c)** trabalha com interpolação linear em ambos os sentidos: num looping faz da esquerda para a direita e no looping maior vai fazendo de cima para baixo. Assim, vai sendo feita uma fileira em cima, depois outra fileira embaixo, depois outra fileira mais embaixo ainda, etc.



Faça um código de **HTML e JavaScript** que crie DIVs com uma determinada cor de fundo (*background*) sendo especificada por você. Com isso, você poderá criar DIVs de cores diversas, podendo fazer um looping para criar as interpolações apresentadas acima. Na sua implementação:

(1) Você deve criar um campo na tela para o usuário escolher quantos DIVs intermediários quer na interpolação. Ao dar *<enter>* nesse campo, o gráfico deve ser atualizado automaticamente.

(2) Você deve criar campos para o usuário escolher as cores da interpolação (cores inicial e final).

(3) Você deve fazer a interpolação linear do item (a) acima e também a implementação do item (c) (interpolação em ambas as direções).

(4) Você deve fazer também uma versão que trabalhe com **HSL** ao invés de RGB: a interpolação deve ser feita de uma cor HSL até outra cor HSL.

Trabalho 2 – Implementação C

Faça uma implementação com **HTML e JavaScript e WebGL** com os seguintes recursos:

(1) Um triângulo deve ser desenhado numa posição fixada por você.

(2) Botões de **Translação** (um para esquerda, outro para direita, outro para cima e outro para baixo), onde cada botão ao ser clicado altera uma variável de deslocamento que deverá ser acessada no **Vertex Shader** (através de uma variável **uniform**) e aplicada em todos os vértices.

(3) Botões de **Escala** (um para diminuir e outro para aumentar). Ao clicar nesses botões, uma variável de "fator de escala" (inicialmente com valor = 1) deve ser atualizada e usada devidamente no **Vertex Shader** por meio de uma variável **uniform**. Ao clicar no botão de "aumentar", pode-se fazer: fator = fator x 2 (e algo análogo para o botão "diminuir").

(4) Botões de **Rotação** (um para aumentar o ângulo e outro para diminuir o ângulo). Ao clicar num dos botões, um ângulo deve ser atualizado e o **Vertex Shader** deverá estar aplicando uma rotação seguindo esse ângulo por meio de uma variável **uniform**.

(5) Um botão de **Resetar** tudo. Ao clicar nesse botão, o deslocamento, o fator de escala e o ângulo de rotação deverão ser resetados para o valor original padrão.

Você poderá criar o layout da interface da sua implementação de múltiplas maneiras. Poderá usar ícones nos botões, etc. Faça caprichado! Bom trabalho!