

”

**E-fólio A** | Folha de resolução para E-fólio



**UNIDADE CURRICULAR:** Computação Gráfica

**CÓDIGO:** 21020

**DOCENTE:** António Araújo e Pedro Pestana

**A preencher pelo estudante**

**NOME:** Hélio Emanuel Soares de Sousa

**N.º DE ESTUDANTE:** 2000027

**CURSO:** Licenciatura em Engenharia Informática

**DATA DE ENTREGA:** 11 de novembro de 2022

## TRABALHO / RESOLUÇÃO:

### PARTE I- Implementação do algoritmo do ponto médio (2 valores):

No módulo *"lineMP.mjs"*, foi implementado o caso geral do algoritmo do ponto médio como indicado em (Foley et al., 1995, p. 78). A função *"lineMP"* recebe valores literais como solicitado no enunciado {x: int, y: int}. No início da função, define-se duas variáveis auxiliares que guardam os valores originais de P e Q em *"newP"* e *"newQ"*, respetivamente. Esta opção permite transformar os valores de X e Y, de cada ponto, sem a ocorrência de efeitos colaterais.

Como (Foley et al., 1995, p. 75) afirma, o algoritmo é adequado ao primeiro octante (declive entre 0 e 1) e os segmentos de reta com outros declives podem ser definidos a partir das reflexões adequadas em relação aos eixos principais. Assim sendo, optou-se por executar o algoritmo de redução ao primeiro octante definido em (Brisson L., 2013, pp. 12-13) e implementado na função *"reduceToFirstOctant"*.

Se o segmento de reta se encontrar num octante diferente do primeiro, é realizada uma transformação geométrica de redução ao primeiro octante, tendo por base três verificações:

1. Se o declive é negativo, ou seja, verifica a simetria.
2. Se y cresce mais depressa do que x, isto é, se o declive é  $> |1|$ .
3. Se o valor de X no extremo inicial é maior do que no extremo final.

Com base nestas verificações, são realizadas trocas de sinal e de posição dos valores de x e y, tal como detalhado em (Brisson L., 2013, pp. 12-13) e comentado no código linha a linha.

Após essa redução ao primeiro octante, o caso geral é executado tal como definido em (Foley et al., 1995, p. 78), devolvendo um conjunto de pontos discretos.

Se o octante original for diferente de 1, esse conjunto de pontos têm de ser devolvidos ao octante original, pelo que é preciso realizar a transformação inversa tendo em conta a simetria e o declive. É indiferente desenhar o segmento do extremo final para o extremo inicial ou vice-versa, o simples desenhar dos pontos não altera as suas posições (Brisson L., 2013, p. 13).

No final, a função *"lineMP"* retorna uma lista pontos discreta para o octante original que poderá ser utilizada para calcular os pixéis a acender, em conformidade com algoritmo do ponto médio.

Em suma, a função *"lineMP"* recebe dois pontos e chama a função *"reduceToFirstOctant"* para os verificar e transformar para o primeiro octante, quando aplicável. De seguida, corre o algoritmo do ponto médio devolvendo um conjunto de pontos, os quais, sempre que aplicável, são sujeitos a uma transformação geométrica inversa, após a qual se obtém um conjunto de pontos discretos finais. Esses pontos finais são o retorno da função e correspondem aos pixéis que devem ser "acessos" segundo este algoritmo.

### PARTE II- Interface Gráfico (2 valores)

Começamos pelo *index.html*, definiu-se um texto explicativo da interface gráfica que está incorporado diretamente no *"viewport"* da aplicação *"three.js"*, realizado como na AF2 desta Unidade Curricular, mas limitado o texto a 30% de *"width"* para evitar conflitos com os comandos do *"orbit controls"*.

Em *"head"*, é chamada uma única função, a *"main.js"*.

Analisando agora o código presente no ficheiro *"main.js"*:

Verifica-se a importação do *"three.js"* e do *"orbit controls"* através do *"CDN"* *"skypack"*, e o *"import"* do *"lineMP.mjs"* que se encontra na raiz do projeto.

A inicialização da função principal *"start"* é feita ao carregar a página com o evento *'DOMContentLoaded'*.

De seguida são definidas as seguintes variáveis globais:

Variáveis P e Q, que guardam os pontos inicial e final do segmento de reta a calcular.

A variável *"sizePixel"* define o tamanho do pixel, um plano quadrado de tamanho unitário, esta opção pelo tamanho unitário facilita os cálculos e permite utilizar a posição do objeto como coordenadas discretas num eixo cartesiano.

A variável *"sizeRasterDisplay"* define a quantidade de quadrículas que irá conter o tabuleiro, depois define-se *"rows"* e *"cols"* (linhas e colunas) para melhorar a legibilidade do código.

A seguir temos 4 variáveis que definem as cores indicadas no enunciado.

Por fim, temos a instanciação do *"raycaster"*, que permite obter a posição de um dado objeto que, neste caso, a obtém através da posição do rato. Inicializa-se com propriedade *"layer = 1"*, para que o *"raycaster"* apenas intercepe os planos que compõem o tabuleiro, pois serão os únicos na *"layer 1"*.

Como precisamos de dois pontos, foi criada a variável booleana *"firstXPressed"* que inicializa a falso, e é utilizada como um interruptor lógico entre o primeiro pressionar do X que guarda o ponto em P e o segundo pressionar do X que guarda o ponto em Q.

De seguida, definiu-se os parâmetros da cena, baseados nos tutoriais indicados no manual oficial do *“three.js”* e no livro *“Discover three.js”*.

A posição da câmara é situada uma unidade imediatamente acima do *“sizeRasterDisplay”* para que no arranque do programa o utilizador observe a totalidade do tabuleiro.

A partir deste ponto o código divide-se em 3 seções:

- Função *“start”* e *“animate”* – inicializar e renderizar as cenas
- Eventos – definição dos eventos que capturam as iterações do utilizador.
- Funções auxiliares – processam as entradas e devolvem as saídas solicitadas no enunciado.

A função *“start”* inicializa todo o processo e a função *“animate”* executa a renderização de cenas recursivamente, permitindo assim *“ver”* as alterações que são resultado das entradas do utilizador. No *“start”* é chamada a função *“displayRaster”* que cria o tabuleiro e a *“axesXYHelper”* cria os eixos X e Y.

Os eventos são de dois tipos *“mousemove”* e *“keydown”*. O *“mousemove”* foi adaptado do manual de apoio ao *“raycaster”* e devolve os eixos de X e Y normalizados, isto é, uma representação dos eixos cartesianos situados entre -1 e +1. Existem alguns eventos auxiliares que permitem saber mais dados sobre o pixel onde o rato se encontra providenciando ao utilizador auxílio no momento de escolha dos pontos a calcular comentados linha a linha e com texto explicativo no *“index.html”*. No *“keydown”*, apenas duas teclas de interação, *“backspace”* chama a função *“reset”* que reinicializa o tabuleiro e o *“x”* ou *“X”* é usado para capturar a posição do pixel (plano de 2 dimensões que tem o seu ponto central com coordenadas discretas). No primeiro pressionar X é guardada a posição P e no segundo pressionar X é guardada a posição do Q e, logo de seguida, são chamadas as funções *“drawTilesMP”* e *“drawRealLine”* que desenhavam os ladrilhos e a reta verdadeira, respetivamente.

A função *“axesXYHelper”*, cria duas linhas, ambas a iniciar em (0,0), e terminam no tamanho igual ao tamanho do tabuleiro mais metade do tamanho do pixel, pois o (0,0) encontra-se no meio de uma quadricula, e são apenas os eixos positivos, como indicado no enunciado.

A função *“resetScene”*, remove todos os descendentes da cena e depois chama as funções *“displayRaster”* e *“axesXYHelper”*, e por fim, atribui os valores de origem à câmara.

A função *“createPixel”*, cria uma simulação de um pixel. Para tal, utiliza-se a geometria de um plano com o tamanho de lado unitário, em que a posição do mesmo será uma coordenada discreta (x,y) passada como parâmetro, a cor também será passada por parâmetro, pois varia em função da posição no tabuleiro. Aqui o importante a reter é que a posição dos objetos em *“three.js”* refere-se sempre ao seu centro, logo um plano na posição (0,0) com tamanho lado 1, irá ocupar a área entre -0,5 e 0,5 em ambas as direções dos eixos XY, um objeto na posição (1,0) irá ocupar a área 0,5 a 1,5 no eixo do X e -0,5 a 0,5 no eixo do Y, esta lógica é replicada ao longo da criação do tabuleiro e é o conceito mais importante da aplicação.

A função *“displayRaster”*, adaptada de (Gertis, 2019), define a cor que cada quadricula (pixel) deverá ter no tabuleiro, e a sua posição é definida em acordo com a coluna e a linha do tabuleiro. Recuperando a lógica do raciocínio anterior, os pontos centrais de todos os pixéis (ou *“tiles”*) criados correspondem ao número da linha e da coluna. Exemplo, na linha 5, coluna 4, teremos a posição (5,4) e o plano unitário, será criado na posição (5,4) e a sua área irá estender-se de 4,5 a 5,5 em X e 3,5 a 4,5 em Y. Realça-se que estes elementos serão adicionados à *“layer 1”*, para que possam ser intercetados pelo *“raycaster”*.

A função *“drawRealLine”*, desenha uma linha entre P e Q, o código segue a mesma linha de raciocínio do utilizado na função *“axesXYHelper”* e, como na AF2 para o desenho de linhas.

A função *“createTile”* tem semelhanças com a *“createPixel”*, mas contém as propriedades específicas solicitadas no enunciado, a saber, é uma *“box”* com tamanho unitário, tal como as quadriculas do tabuleiro, e com a altura de  $\frac{1}{4}$  do lado, com transparência a verdadeiro e opacidade a 0.5, a cor é o amarelo (definida na variável a *“colorTile”*).

A função *“drawTilesMP”*, chama a função *“lineMP”* para obter os pontos calculados pelo algoritmo do ponto médio que se encontram entre P e Q, e guarda-os na variável *“pixelsPositions”*. Depois, na lista *“tilesMP”* são guardados os ladrilhos criados com as posições de X e Y que a função *“lineMP”* devolveu e se encontravam na lista *“pixelsPositions”*. Por fim, é feita a adição de ladrilho a ladrilho à cena.

## BIBLIOGRAFIA

Brisson, J., Coelho, A., Ferreira, A., Gomes, M. R., & Pereira, J. M. (2018).

*Introdução à Computação Gráfica*. FCA.

Brisson, L. (2013). *Instituto Superior Técnico*. Obtido em 14 de 10 de 2022, de

Rasterização: <http://disciplinas.ist.utl.pt/leic-cg/textos/livro/Rasterizacao.pdf>

Foley, J. D., Van Dam, A., Feiner, S. K., & Hughes, J. F. (1995). *Computer*

*Graphics: Principles and Practice, Second Edition in C*. Addison-Wesley Publishing Company.

Gertis, E. (01 de 02 de 2019). <https://medium.com/>. Obtido de Loops in javascript make checkerboards!:

<https://physevangertis.medium.com/loops-in-javascript-make-checkerboards-cf9c17308072>