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

UNIDADE CURRICULAR: Computação Gráfica

CÓDIGO: 21176

DOCENTE: António Araújo / Pedro Pestana

A preencher pelo estudante

NOME: Pedro Pereira Santos

N.º DE ESTUDANTE: 2000809

CURSO: Licenciatura em Engenharia Informática

DATA DE ENTREGA: 09/11/2023

TRABALHO / RESOLUÇÃO:

Para começar o trabalho, segui a ordem do que foi pedido no relatório de trabalho, começando por implementar o algoritmo do ponto médio.

Após ler os documentos disponibilizados na UC e mais algumas informações na Internet, havia certas fontes que igualavam este algoritmo ao de Bresenham.

Para este trabalho, escolhi aplicar o algoritmo apresentado nos documentos disponibilizados na UC, adaptando o algoritmo dependente do Octante ao qual pertence, obedecendo às especificações impostas para o objeto P e utilizando um vetor (R) para guardar os mesmos. Guardei como variáveis locais os valores importantes a utilizar como variação absoluta das ordenadas e abcissas.

As dificuldades obtidas foram ao definir o octante ao qual a reta pertencia, sendo que praticamente descrevi um por um na condicional. Ao analisar melhor o código e ver onde este se repetia, consegui fazer condicionais para saber para que lado, tanto de x como de y, a reta crescia, ficando apenas com 2 variáveis de crescimento (positivo ou negativo, consoante estas condicionais), uma para x e outra para y.

Usando o facto de que se a linha for vertical o declive dá impossível, pois é obtido por variação de y sobre variação de x, contorna-se este facto atribuindo o valor 0.

Após verificar que a maioria do código estava de facto repetido, consegui separá-lo em 2 casos maioritários, onde a reta crescia maioritariamente em x e onde crescia maioritariamente em y, sendo esta a decisão necessária para aplicar de facto o algoritmo, trocando apenas a ordem das variáveis na definição do valor de decisão, segundo o algoritmo apresentado nas UC. Após verificar novamente as especificações, incorporei um teste, para em caso de haver algum erro, os valores das coordenadas dos pontos não serem inteiros, devolvendo erro.

No inicio tive um pequeno problema, uma vez que não estava a entender o porquê do vetor R ficar com os pontos todos com as mesmas coordenadas. Entretanto, após procurar sobre a linguagem, uma vez que ela é interpretada, a variável que estava a guardar o ponto atual, ficava sempre com as coordenadas do último ponto, sendo por isto necessário utilizar o operador de propagação "…", para criar uma cópia do objeto P ao adicioná-lo ao vetor R.

Na segunda parte do trabalho, uma vez que tenho pouca experiência nestas 3 linguagens, decidi apenas criar o template básico de HTLM, tendo cuidado com as definições da janela, chamando de seguida o ficheiro script.mjs, onde estaria a entrada para a lógica do programa.

De inicio, o primeiro que fiz foi importar as bibliotecas requeridas, de acordo com o explicitado pelo requerido no efólio.

Tendo pouca experiência em JavaScript, mas entendendo os conceitos de linguagens de OOP, fui fazendo as funções necessárias uma a uma, interligando-as quando necessário. Fui começando por uma abordagem Bottom-Up, começando por fazer a base, criando e atribuindo os elementos básicos como a cena, o renderizador e a câmera. Para o fundo da cena preferi atribuir o branco para ser mais fácil identificar os elementos e qualquer erro que possa acontecer.

De seguida, implementei os controlos do módulo de OrbitControls para ser possível manipular e alterar a posição da câmera, dando zoom in e zoom out e rodando sobre o centro da cena (o centro da grid).

Após isto, reparei que podia já definir as cores que vou usar durante a criação das formas necessárias em variáveis globais, tal como criar de uma vez as estruturas e variáveis que iria usar no programa.

Para começar a construir os objetos da cena, comecei por implementar uma função para criar a Grid, que representa um display raster. Este passo foi simples, implementando um ciclo para a criação dos quadrados unitários, representando cada um deles um pixel.

O próximo passo foi verificar como podia receber o input de rato e teclado, descobrindo as funções existentes em JavaScript de window.addEventListener que fica à espera a "ouvir" o ambiente, até "ouvir" o evento pedido, tanto de movimento de rato como pressionar uma tecla, os eventos para os quais criei estes métodos.

Comecei por me focar no pedido de imprimir as coordenadas, relativas à Grid, do rato na consola, verificando se existe interceção, utilizando os métodos existentes em raycaster, atualizando apenas quando o rato sai do "pixel" atual, o que acontece quando existe uma nova interceção com um novo "pixel", imprimindo as novas coordenadas.

Após ter avançado até ao ponto de conseguir identificar qual o ladrilho onde o rato se encontra, comecei pela implementação do sucedido ao pressionar a tecla “x”.

Comecei por criar uma função para “iluminar” o ladrilho atual, notando que necessitava das coordenadas guardadas do rato, em relação à Grid, criando, assim, uma variável global para tal, para utilizá-las na definição da posição do ladrilho a “iluminar”. Dentro desta função, após “iluminar” o ladrilho atual, guardo as coordenadas num ponto, verificando se é o primeiro pronto selecionado. Ao sair da função, testamos se foi o primeiro ou segundo ponto a ser selecionado, e em caso da última opção, chama o algoritmo implementado em lineMP.mjs, guardando o vetor resultante num vetor R, que contém as posições que pertencem à reta definida pelo algoritmo, que será utilizado numa função que cria ladrilhos, com as características pedidas, que formam o caminho do primeiro ponto ao segundo ponto, desenhando-os no ecrã, com a ajuda de um ciclo que percorre o vetor R.

De seguida desenha-se uma linha exata entre esses pontos, com as características pedidas, e limpa-se as coordenadas guardadas nos pontos, preparando-os para uma nova ação idêntica.

De seguida, ao criar as funções para as outras duas teclas pedidas, “retrocesso” e” c,” foi onde encontrei maiores dificuldades.

Para a tecla de “retrocesso” decidi, por fim, criar uma função, de maneira a não ter de guardar todos os objetos criados e adicionados à cena, poupando assim memória, uma vez que, as funções para desenhar a Grid e os eixos, são relativamente rápidas, que elimina todos os conteúdos da cena e do vetor de ladrilhos e, de seguida, chama as funções de desenho da Grid e dos Eixos, facilitando assim o processo.

Para a tecla “c”, foi onde tive a maior dificuldade, uma vez que estava a assimilar criar uma função que chamaria o método de “position.lerp” do objeto da câmera, utilizando “requestAnimationFrame” desta função e desativando os controlos de câmera do módulo de OrbitControls, ativando quando a câmera ficasse na posição inicial, que aponta de cima a Grid.

Após verificar os bugs causados, como de nunca chegar ao ponto de os ativar outra vez, e ficando o programa lento, tentei utilizar um ciclo para tal ,verificando a posição da câmera com a função de camera “position.distanceTo”, não resultando.

Para resolver este problema, após pensando um bocado no assunto, resolvi criar uma variável de controlo “CenterCameraCheck”, que ficaria ativa após clicar na tecla “c”. Após este clique na tecla “c”, é também desativado o acesso aos controlos de OrbitControls. A variável “CenterCameraCheck”, é verificada na função “animate”, a função criada para animar a cena, chamando o método tentado acima, “position.lerp”, no caso de ela ter sido ativada. Uma vez verificada a correta posição da câmera, esta variável é desativada e volta-se a ativar os controlos de OrbitControls.

Ao ter o programa completo, verifiquei que ao clicar “x” quando o rato estava fora da Grid o último ladrilho onde estivera era selecionado, vi a necessidade de adicionar uma condicional, verificando se a posição do rato estava em algum ladrilho da Grid no momento do clique, verificando se o método de raycaster tinha detetado alguma interseção, tendo assim o comportamento esperado.

Com isto, e após testar, limpar e organizar, de melhor maneira, o código, verifiquei que tinha o programa pronto segundo os requisitos impostos.

Web Grafia:

https://elearning.uab.pt/pluginfile.php/3100267/mod_resource/content/1/Introdu%C3%A7%C3%A3o%20ao%20Three%20js.pdf

https://threejs.org/ (Documentação e utilização da biblioteca three.js)

https://www.geeksforgeeks.org/mid-point-line-generation-algorithm/

https://elearning.uab.pt/pluginfile.php/3100281/mod_resource/content/1/Foley%20e%20Van%20Dam%20on%20Bresenham.pdf

https://elearning.uab.pt/pluginfile.php/3100282/mod_resource/content/1/bresenham1965.pdf

https://www.geeksforgeeks.org/javascript/ (Funcionamento da linguagem JavaScript)