Relatório E-fólio-A de Computação Gráfica - Miguel Gonçalves - 1901337

Algoritmo do Ponto Médio

Para o desenvolvimento do algoritmo, obtive uma boa explicação neste video de Abdul Bari, e então sabemos que temos a fórmula do segmento de recta, $y=mx+c$, que podemos utilizar. Tendo em conta que inclinação da recta (m) pode ser 1 quando temos um ângulo de 45º, $m < 1$ quando é menor que 45º e $m > 1$ quando é maior que 45º, podemos dividir o algoritmo em 2, sendo uma parte em que resolvemos em ordem a X (o que incrementamos por cada iteração) e a outra parte em ordem a Y.

O cálculo é feito imaginando um ponto central entre os dois pixels e verificando a distância entre o pixel de cima e o pixel de baixo. O que estiver mais perto é o pixel escolhido.

Para o ponto inicial, foi desenvolvida a equação para P1, em que $P1 = 2 * dy - dx$ ou $P1 = 2 * dx - dy$ (dependendo do eixo que é iterado). Inclui também uma variável direção, de maneira a obter as simetrias necessárias para preencher o resto das octantes. Por cada iteração, verificamos se o ponto calculado é maior, igual ou inferior a 0.

No algoritmo para o X, caso $P <= 0$ (o y atual é mais próximo do que o próximo ponto y), usamos a equação $P = P + 2 * dy$, de modo a incrementar apenas o x e não o y. Caso contrário, usamos a equação $P = P + 2 * (dx -dy)$ e incrementa-se / decrementa-se o y (dependendo da direção).

Para Y, as equações mudam ligeiramente, o $P1 = 2 * dx - dy$, para P <= 0 temos $P= P + 2 * dx$ e para P > 0, temos $P = P + 2 * (dx -dy)$, incrementando ou decrementando o x com o valor da direcção (1 ou -1).

Implementação no ThreeJS

Para este projecto, decidi optar por implementar um padrão Singleton que controla todos os outros componentes (câmera, renderer, grid, etc..) necessários para a criação da cena. Neste componente, inicializo todos os outros componentes para termos apenas uma instância de cada, garantindo o acesso a partir do singleton.

Queria uma estrutura de classes em que pudesse ter uma função de update e de resize:

- Update - Corria a cada frame
- Resize - Corria quando a janela era resized

Havia um problema, para concretizar isto teria de adicionar um requestAnimationFrame() em cada função update, o que pode impactar a performance ou até não funcionar corretamente.

A solução que seria ideal era criar um evento em que o singleton pudesse escutar. Sempre que esse evento é ativado, o singleton chama as funções update dos componentes que controla. Encontrei um pacote chamado event-emitter que me permite emitir eventos custom. Criei uma classe Tempo para armazenar o tempo entre frames e apenas fazer update a cada 16ms.

Para inicializar uma cena, podemos criar uma instância de uma classe THREE.Scene() e armazenar no singleton. Para adicionar/remover objetos a uma cena, usamos as funções scene.add() e scene.remove() respectivamente.

Precisamos também de uma câmera e de um renderer. A câmera, como referido no enunciado, é uma perspective camera que é apontada para o centro da cena (onde a grelha vai ser construída). É também nesta classe que adicionamos os Orbit Controls, que nos permitem movimentar a câmera na cena. O renderer utiliza o WebGL Renderer que engloba o canvas. É utilizada uma classe Tamanhos para guardar o tamanho da aplicação e poder emitir o evento de resize.

Para a construção da grelha, foram desenvolvidas duas classes: Grid e GridTile. A Grid gera o comportamento da grelha, enquanto a GridTile gere a sua própria construção como uma BoxGeometry do ThreeJS e armazena os dados relativos ao tile. A classe Grid tem um método de criação em que preenche um array de GridTile's, mudando-lhes as cores alternadamente (usei o método de verificar se é par ou ímpar dependendo das posições x e y).

Depois da grelha construída, precisava de uma maneira de escutar os inputs do utilizador (neste caso, pelas teclas X e Backspace) e também a posição do ponteiro do rato a cada momento. Para resolver isto, criei mais duas classes, a classe Mouse e Keyboard.

A classe Mouse, apenas vai escutar o evento “pointermove” e guardar as coordenadas de acordo com a posição do ponteiro em relação à janela do browser num Vector2.

A classe Keyboard vai escutar dois eventos, “keydown” e “keyup”. Vai também ter uma estrutura em que armazena as teclas que estão a ser premidas. Quando o evento “keydown” é emitido, verificamos se a tecla premida não está na estrutura (o que quer dizer que pode ser premida), então:

- Backspace - chamamos a função reset da grid
- X - Utilizamos um raycaster do three que “dispara” um raio do ponto central da câmera até o ponteiro do rato, guardando os objetos interceptados. Como este objeto só pode ser um tile, chamamos a função de selecionar o tile na grid. Adicionamos por fim, a tecla premida à estrutura.
- Outra tecla - Não acontece nada.

Quando o evento “keyup” é emitido, removemos a tecla premida da estrutura, permitindo assim voltar a utilizar essa tecla.

Quando a função de selecionar o tile é chamada, esta irá receber o tile que foi premido e é feita as seguintes verificações:

- Verifica se já foram clicados selecionados 2 tiles, se sim, então reseta a grid e retorna.
- Seleciona o tile, removendo o antigo da cena, mudando a cor e adicionando de novo.
- Verifica se este tile é o segundo selecionado, se sim, corre o algoritmo de Bresenham entre os dois pontos selecionados, resultando num array de posições para o raster. Após isto, é desenhado os raster tiles de acordo com os pontos calculados. Aqui é usada uma nova classe RasterTile que é uma cópia do GridTile (apenas com algumas alterações nos estilos e posição no eixo dos Z).

Adicionei também à classe Grid, um event listener para que fique à escuta do click do botão reset em que vai atualizar o tamanho da grelha e fazer reset.

Por fim, criei uma classe Information que irá atualizar os dados no painel criado no html para termos uma melhor visualização dos dados que estão a ser clicados e calculados.

Bibiografia

https://www.youtube.com/watch?v=RGB-wlatStc - Bresenham's Line Drawing Algorithm, Abdul Bari
https://github.com/dekkai-data/event-emitter - Event Emitter package, Dekkai