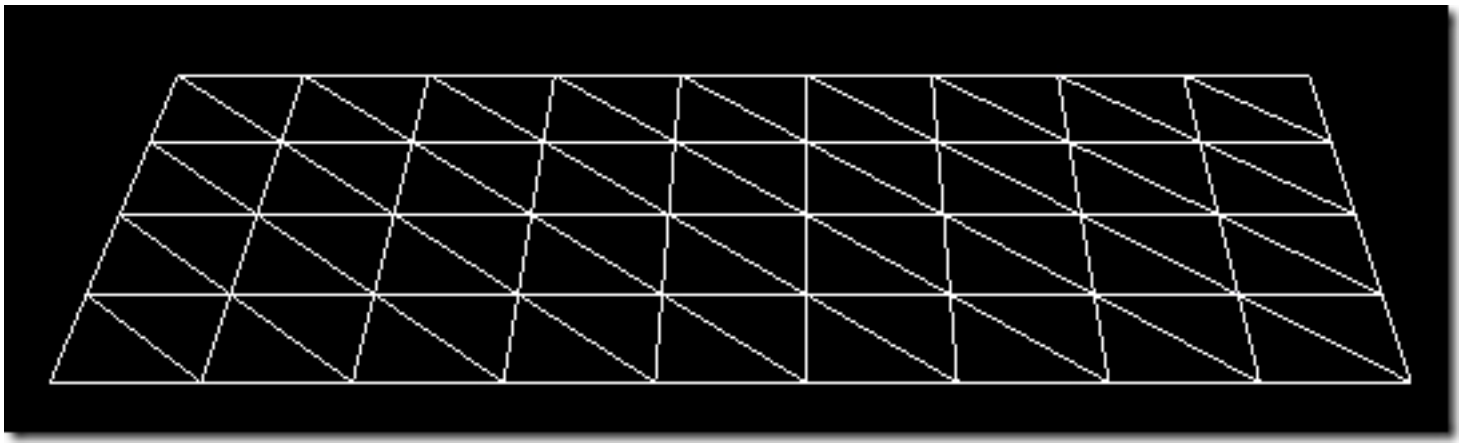
No artigo entitulado “ Um quadrado com Index Buffer ” propusemos um desafio: o de construir uma malha quadriculada para o desenho de um chão. A função de construção da malha deveria receber como parâmetro o número de vértices na profundidade e largura, além de gerar a geometria centralizada no ponto (0,0). Nesse artigo, iremos resolver esse desafio.

Entendendo a geometria

Antes de iniciarmos, vamos ver um *wireframe* esquemático da malha que queremos construir:



Na figura acima, temos uma malha contendo 10 vértices de largura por 5 de profundidade. Observe que o número de quadrados será sempre um a menos, tanto na largura quanto na profundidade. Assim, podemos tirar algumas relações:

1. Para um conjunto de WxD vértices, teremos (W-1)x(D-1) quadrados
2. Como a distância entre os pontos é 1, as dimensões da malha também será (W-1)x(D-1). Portanto, para centraliza-la, teremos que subtrair metade dessa valor das coordenadas de x e z;

Criando o array de vértices

Vamos iniciar com a função createData que fará a criação dos vértices. Nela, iremos criar um objeto chamado data, que conterá dentro dele o array de vértices e índices, respectivamente. Como podemos criar vértices lado-a-lado e um abaixo do outro, podemos simplesmente encadear dois for:

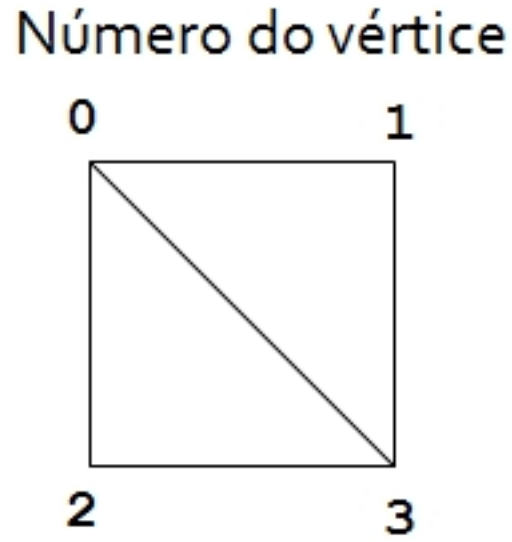
function createData(width, depth) { var data = {} data.vertices = []; //Centralização da malha var hw = (width-1) / 2; var hd = (depth-1) / 2; for (var z = 0; z Observe que, se quiséssemos criar a malha iniciando no ponto (0,0) os valores das coordenadas x e z seriam exatamente iguais ao das variáveis x e z. Porém, para que a malha fosse centralizada aplicamos o que foi descrito na observação 2.

Criando o array de índices

Se você tentou fazer esse desafio, provavelmente este é o local onde você quebrou a cabeça. Quais são os índices corretos?

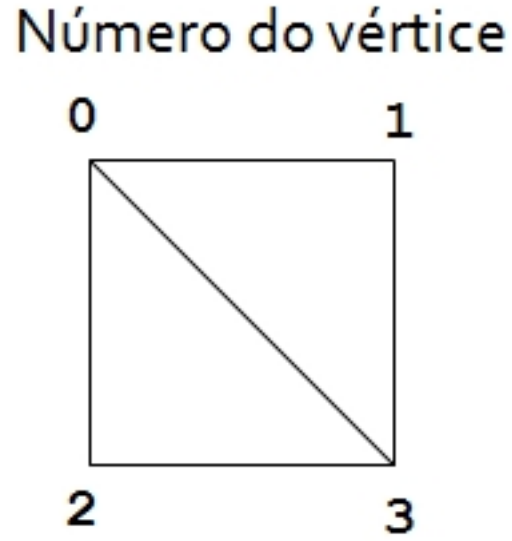
Criaremos os índices quadrado-a-quadrado. Como podemos observar na figura abaixo, cada quadrado é formado por 2 triângulos. Esses dois triângulos, compõe 6 índices sendo dois deles compartilhados.

Por exemplo o quadrado abaixo:



Índice = coluna + linha * largura

Vamos colocar isso numa tabela?



Índice	Par coordenado
0	(x,z)
3	(x+1,z+1)
1	(x+1,z)
0	(x,z)
2	(x,z+1)
3	(x+1,z+1)

De posse dessas informações, agora basta gerar os fors que preenche os índices de cada quadrado:

for (z = 0; z Desenhando a malha

Vamos agora alterar nosso programa principal para desenhar a malha criada. Vamos alterar o programa de modo a reforçar o conceito de malha poligonal. Entretanto, que conceito é esse?

A definição clássica da computação gráfica para malha poligonal é “o conjunto de vértices e índices que compõe as arestas e faces de um poliedro”. Observe que essa definição não leva em consideração aspectos como iluminação, textura ou a posição exata da malha no mundo. Muitas ferramentas de modelagem e até engines de games incluem esse tipo de informação associada ao conceito de malha e, embora isso não seja 100% correto, não deixa de ser prático. É o que faremos em nossos exemplos também.

Vamos começar criando um objeto chamado mesh, na parte “global” do arquivo de exemplo. Em seguida, iremos criar a função initMesh, em substituição a função initBuffers. Essa função:

1. Chamará a função createDate para criar a malha de 256x256;
2. Copiará os dados para os vertex buffers e index buffers;
3. Associará a malha a matriz model.

function initMesh() { var data = createData(256,256); mesh.vertexPosition = glc.createBuffer(gl, gl.ARRAY_BUFFER, 3, data.vertices); mesh.indices = glc.createBuffer(gl, gl.ELEMENT_ARRAY_BUFFER, 1, data.indices); //Configura a matriz model mesh.transform = mat4.create(); }

Atualizamos a função update para levar em consideração a matriz model da malha:

function update(secs) { var speed = 72 * secs; if (Key.isDown(Key.SHIFT)) {

speed *= 3; } if (Key.isDown(Key.LEFT)) { angle += speed; } else if (Key.isDown(Key.RIGHT)) { angle -= speed; } mat4.rotateY(mesh.transform, mat4.create(), glc.toRadians(angle)); }

E, por último, atualizamos a função drawScene para utilizar os dados vindos da malha. Para que a cena apareça bem, precisamos também alterar a posição de nossa câmera.

Observe novas coordenadas nas matrizes model e projection.

function drawScene() { gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT); //Desenhamos apenas se o shader e a malha já estiverem prontos if (!shaderProgram || !mesh) { return; } //Configura a matriz de projeção var projection = mat4.perspective(mat4.create(), glc.toRadians(45), gl.width / gl.height, 0.1, 1000.0); //Configura a matriz view var view = mat4.lookAt(mat4.create(), vec3.fromValues(0.0, 100.0, 400.0), //Onde está vec3.fromValues(0.0, 80.0, 0.0), //Para onde olha vec3.fromValues(0.0, 1.0, 0.0)); //Onde o céu está //Atualiza os valores do shader gl.uniformMatrix4fv(shaderProgram.model, false, mesh.transform); gl.uniformMatrix4fv(shaderProgram.view, false, view); gl.uniformMatrix4fv(shaderProgram.projection, false, projection); //Copia dados do buffer gl.bindBuffer(gl.ARRAY_BUFFER, mesh.vertexPosition); gl.vertexAttribPointer(shaderProgram.vertexPosition, mesh.vertexPosition.itemSize, gl.FLOAT, false, 0, 0); gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, mesh.indices); //Comanda o desenho gl.drawElements(gl.TRIANGLES, mesh.indices.numItems, gl.UNSIGNED_SHORT, 0); }

Por fim, como nosso shader não tem mais informação de cor, iremos criar shaders chamado white.vs e white.fs sem as cores. O fragmente shader simplesmente retornará a cor branca:

white.vs

attribute vec3 aVertexPosition; uniform mat4 uModel; uniform mat4 uView; uniform mat4 uProjection; void main(void) { gl_Position = uProjection * uView * uModel * vec4(aVertexPosition, 1.0); }

white.fs

precision mediump float; void main(void) { gl_FragColor = vec4(1,1,1,1); }

Concluindo

Nesse artigo vimos como desenhar uma malha mais complexa de quadrados com diversos

vértices e índices. Apesar de parecer “bobinha” essa malha é a base para DIVERSOS efeitos gráficos: ondas, terrenos, scanners 3D, etc.

No próximo artigo, veremos como usar essa malha e uma imagem em tons de cinza para carregar um terreno.

Como sempre, é possível ver uma demonstração funcional do exemplo clicando aqui . Ou baixar todos os fontes clicando na figura abaixo:

Até lá!