Relatório Tanques – 2ªFase

Bruno Couto A10664

Vítor Gomes A10658

18/11/16

Índice

Introdução

Foi-nos proposto, para a disciplina de Introdução à Programação 3D, a criação de um jogo 3D usando a framework do Monogame. A proposta apresentada foi de desenvolver um jogo de batalha entre tanques, onde o jogador controla um dos tanques e tem como objectivo destruir o adversário.

O trabalho foi dividido em 3 fases. Na primeira foi pedido o render do terreno a partir de um mapa de alturas, e o controlo de pelo menos uma câmara com *surface follow*.

Nesta segunda fase é pedida, além do conteúdo anterior, a iluminação do jogo e também o render e controlo dos tanques e a respectiva interacção correcta dos mesmos com o terreno.

Terreno

Para criar o terreno, foi-nos proporcionado um *height map* que indica os valores das alturas nos diferentes pontos no terreno.

Criamos um *array* que guarda a informação da cor dos pixéis da textura do *height map.* Depois geramos os vértices do terreno de acordo com os valores dentro dessa textura.

```
//Gerar vértices
int x = 0, z = 0;
for (int j = 0; j < altura / 2; j++)
{
    for (int i = 0; i < altura * 2; i++)
    {
        //Calcular coordenadas da textura
        int u, v;
        u = (x % 2 == 0) ? 0 : 1; //Operador Condicional
        v = (z % 2 == 0) ? 0 : 1;

        //Escalas:
        //bigaltura (512 * 512): 0.2f;
        //altura (128 * 128): 0.04f;
        float scale = 0.04f;
        vertexes[(2 * j * altura) + i] = new VertexPositionNormalTexture(new Vector3
            (x, texels[(z * altura + x)].R * scale, z), Vector3.Zero, new Vector2(u, v));
        z++;
        if (z >= altura)
        {
            x++;
            z = 0;
        }
    }
}
```

Imagem 1 Código para gerar os vértices do terreno

Depois aplicamos uma textura ao terreno que é repetida por cada *quad* presente no terreno. O mapa é criado através de um for que desenha cada faixa do terreno.

```
//Desenhar o terreno, uma strip de cada vez
for (int i = 0; i < altura - 1; i++)
{
    graphics.DrawUserIndexedPrimitives(PrimitiveType.TriangleStrip, vertexes, i * altura, altura * 2,
    indice, 0, altura * 2 - 2);
}
```

Imagem 2 Código para desenhar o terreno

Câmara

A câmara foi desenvolvida como pedido no enunciado, ou seja, para mover a câmara usamos as teclas 8, 4, 5, 6 do *Numpad* – Frente, esquerda, baixo e direita, respectivamente. Para controlar a direcção usamos o rato. Também se pode fazer a câmara subir com a tecla 7 e descer com a tecla 1 do *Numpad.*

Para a câmara fazer *surface follow* criamos uma função que calcula a altura do terreno no local onde a câmara se encontra. Depois é só aplicar o resultado da função á câmara, deixando as coordenadas X e Z como se encontram e apenas alterando o valor Y da câmara.

```
static float surfaceFollow()
{
    //A e B - vertices superiores
    //C e D - vertices inferiores
    //A-----------B
    //C-----------D
    int xA, zA, xB, zB, xC, zC, xD, zD;
    float yA = 0, yB = 0, yC = 0, yD = 0;
    xA = (int)position.X;
    zA = (int)position.Z;
    xB = xA + 1;
    zB = zA;
    xC = xA;
    zC = zA + 1;
    xD = xB;
    zD = zC;

    //encontrar valor de Y de cada vertice
    yA = vertices[xA * heightmap + zA].Position.Y;
    yB = vertices[xB * heightmap + zB].Position.Y;
    yC = vertices[xC * heightmap + zC].Position.Y;
    yD = vertices[xD * heightmap + zD].Position.Y;

    //calcular nova altura da camara
    float yAB, yCD, cameraY;
    yAB = (1 - (position.X - xA)) * yA + (position.X - xA) * yB;
    yCD = (1 - (position.X - xC)) * yC + (position.X - xC) * yD;
    cameraY = (1 - (position.Z - zA)) * yAB + (position.Z - zA) * yCD;
    return (cameraY + 1);
}
```

Imagem 3 Código do cálculo da altura do terreno numa posição

Iluminação

Foi-nos pedido nos objectivos do projecto, aplicarmos a iluminação ao terreno, e para isso ser possível tivemos de calcular e implementar as normais do terreno, que foram feitas através de cada vértice de cada triângulo.

```
static private void CalcularNormais()
{
    for (int x = 0; x <= altura - 1; x++)
    {
        for (int z = 0; z <= altura - 1; z++)
        {
            VertexPositionNormalTexture vertice = vertexes[x * altura + z];

            // centro
            if (x > 0 && x < altura - 1 && z > 0 && z < altura - 1)
            {

                //com 8 cross's
                VertexPositionNormalTexture verticeCima = vertexes[(x * altura + z) - 1];
                VertexPositionNormalTexture verticeCimaDireita = vertexes[(x * altura + z) + altura - 1];
                VertexPositionNormalTexture verticeDireita = vertexes[(x * altura + z) + altura];
                VertexPositionNormalTexture verticeBaixoDireita = vertexes[(x * altura + z) + altura + 1];
                VertexPositio
                VertexPositio                                                    z) + 1 - altura];
                VertexPositionNormalTexture verticeEsquerda = vertexes[(x * altura + z) - altura];
                VertexPositionNormalTexture verticeCimaEsquerda = vertexes[(x * altura + z) - altura - 1];


                Vector3 vectorCima = verticeCima.Position - vertice.Position;
                Vector3 vectorEsquerda = verticeEsquerda.Position - vertice.Position;
                Vector3 vectorBaixo = verticeBaixo.Position - vertice.Position;
                Vector3 vectorDireita = verticeDireita.Position - vertice.Position;
```

struct Microsoft.Xna.Framework.Graphics.VertexPositionNormalTexture

Imagem 4 Código do cálculo das normais

Depois de calculado as normais, implementámos o código da iluminação no projecto com os valores demonstrados abaixo.

```
//luzes
efeitoTerreno.LightingEnabled = true;
efeitoTerreno.DirectionalLight0.Enabled = true;
efeitoTerreno.DirectionalLight0.Direction = new Vector3(1, -1, 1);
efeitoTerreno.DirectionalLight0.SpecularColor = new Vector3(0.5f, 0.5f, 0.5f);
efeitoTerreno.SpecularPower = 1000f;
efeitoTerreno.AmbientLightColor = new Vector3(0.4f, 0.4f, 0.4f);

efeitoTerreno.SpecularColor = new Vector3(1, 1, 1);
efeitoTerreno.DirectionalLight1.Enabled = false;
efeitoTerreno.DirectionalLight2.Enabled = true;

efeitoTerreno.FogEnabled = true;
efeitoTerreno.FogColor = Color.CornflowerBlue.ToVector3(); // For best results,
efeitoTerreno.FogStart = 15;
efeitoTerreno.FogEnd = 50.0f;
```

Imagem 5 Código dos efeitos da iluminação

Tanques

Para desenharmos o tanque fizemos a classe Tank, no seu construtor temos parâmetros como *graphicsdevice* e *content manager*, mas também uma posição. Também temos a leitura dos bones do modelo 3D e os seus valores iniciais.

```
moving = false;
alive = true;
vetorBase = new Vector3(0, 0, 1);
direcao = vetorBase;
direcaoAnterior = vetorBase;
positionAnterior = position;
this.posicao = position;
target = position + direcao;
rotacaoY = 0;
rotacao = Matrix.CreateRotationY(rotacaoY);
velocidade = 0.05f;
scale = 0.00125f;

device = graphicsDevice;
world = rotacao
    * Matrix.CreateScale(scale)
    * Matrix.CreateTranslation(position);
```

Imagem 6 Código de valores iniciais no construtor

```
// Load the tank model from the ContentManager.
tankModel = content.Load<Model>("tank");

// Look up shortcut references to the bones we are going to animate.
leftBackWheelBone = tankModel.Bones["l_back_wheel_geo"];
rightBackWheelBone = tankModel.Bones["r_back_wheel_geo"];
leftFrontWheelBone = tankModel.Bones["l_front_wheel_geo"];
rightFrontWheelBone = tankModel.Bones["r_front_wheel_geo"];
leftSteerBone = tankModel.Bones["l_steer_geo"];
rightSteerBone = tankModel.Bones["r_steer_geo"];
turretBone = tankModel.Bones["turret_geo"];
cannonBone = tankModel.Bones["canon_geo"];
hatchBone = tankModel.Bones["hatch_geo"];

// Store the original transform matrix for each animating bone.
leftBackWheelTransform = leftBackWheelBone.Transform;
rightBackWheelTransform = rightBackWheelBone.Transform;
leftFrontWheelTransform = leftFrontWheelBone.Transform;
rightFrontWheelTransform = rightFrontWheelBone.Transform;
leftSteerTransform = leftSteerBone.Transform;
rightSteerTransform = rightSteerBone.Transform;
turretTransform = turretBone.Transform;
cannonTransform = cannonBone.Transform;
hatchTransform = hatchBone.Transform;

// Allocate the transform matrix array.
boneTransforms = new Matrix[tankModel.Bones.Count];
```

Imagem 7 Código da leitura dos bones

O tanque é movimentado pelas teclas W, A, S, D para o primeiro jogador e I, J, K, L para o segundo, também é possível controlar o canhão do primeiro com as teclas das setas.

```
//  Abre e fecha a Comporta
if (currentKeyboardState.IsKeyDown(Keys.PageUp))
    this.HatchRotation = -1;
if (currentKeyboardState.IsKeyDown(Keys.PageDown))
    this.HatchRotation = 0;

if (currentKeyboardState.IsKeyDown(Keys.W))
{
    //Mover para a frente
    this.wheelBackLeftRotationValue = (float)gameTime.TotalGameTime.TotalSeconds * 5;
    this.wheelBackRightRotationValue = (float)gameTime.TotalGameTime.TotalSeconds * 5;
    this.wheelFrontLeftRotationValue = (float)gameTime.TotalGameTime.TotalSeconds * 5;
    this.wheelFrontRightRotationValue = (float)gameTime.TotalGameTime.TotalSeconds * 5;

    posicao += Vector3.Normalize(direcao) * velocidade;
    moving = true;
}

if (currentKeyboardState.IsKeyDown(Keys.S))
{
    //Mover para trás
    this.wheelBackLeftRotationValue = -(float)gameTime.TotalGameTime.TotalSeconds * 10;
    this.wheelBackRightRotationValue = -(float)gameTime.TotalGameTime.TotalSeconds * 10;
    this.wheelFrontLeftRotationValue = -(float)gameTime.TotalGameTime.TotalSeconds * 10;
    this.wheelFrontRightRotationValue = -(float)gameTime.TotalGameTime.TotalSeconds * 10;

    posicao -= Vector3.Normalize(direcao) * velocidade;
    moving = true;
```

Imagem 8 Código do controlo do tanque

Para o comportamento correcto do tanque dependendo do terreno em que se encontra, usamos o *surface follow* da câmara e também calculamos a normal do tanque na posição que se encontra do mapa.

```
2 references | vpj20142015, 1 day ago | 1 author, 1 change
static public Vector3 NormalHeighmap(Vector3 posicao)
{
    //Posição arredondada para baixo da camara
    int xTank, zTank;
    xTank = (int)posicao.X;
    zTank = (int)posicao.Z;

    //Os 4 vértices que rodeiam a posição da camara
    Vector2 pontoA, pontoB, pontoC, pontoD;
    pontoA = new Vector2(xTank, zTank);
    pontoB = new Vector2(xTank + 1, zTank);
    pontoC = new Vector2(xTank, zTank + 1);
    pontoD = new Vector2(xTank + 1, zTank + 1);


    Vector3 Ya, Yb, Yc, Yd;
    Ya = Terreno.vertexes[(int)pontoA.X * Terreno.altura + (int)pontoA.Y].Normal;
    Yb = Terreno.vertexes[(int)pontoB.X * Terreno.altura + (int)pontoB.Y].Normal;
    Yc = Terreno.vertexes[(int)pontoC.X * Terreno.altura + (int)pontoC.Y].Normal;
    Yd = Terreno.vertexes[(int)pontoD.X * Terreno.altura + (int)pontoD.Y].Normal;

    //Interpolação bilenear (dada nas aulas)
    Vector3 Yab = (1 - (posicao.X - pontoA.X)) * Ya + (posicao.X - pontoA.X) * Yb;
    Vector3 Ycd = (1 - (posicao.X - pontoC.X)) * Yc + (posicao.X - pontoC.X) * Yd;
    Vector3 Y = (1 - (posicao.Z - pontoA.Y)) * Yab + (posicao.Z - pontoA.Y) * Ycd;

    //Devolver normal
    return Y;
}
```

Imagem 9 Código do cálculo da normal do tanque

Conclusão

Na primeira fase do trabalho sucedemos em desenvolver competências básicas para a programação 3D, sendo elas, a criação de um algoritmo que seja capaz de gerar um terreno em 3D através da textura proporcionada. Também nos permitiu tomar conhecimento do funcionamento de uma câmara em 3D e o controlo da mesma.

Nesta segunda fase houve dificuldades sentidas na execução de cada um dos objectivos propostos, mais evidentes na resolução das normais do terreno, mas foram desenvolvidos com sucesso e permitiu-nos ganhar mais conhecimento em como criar um jogo em 3D em Monogame.