2017

Relatório

1ª e 2ª
etapa de entrega

Introdução à Programação 3D

Fernando Nogueira Camilo Nº13233
Lucas Santos Nº13231

11/15/2017

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Introdução à Programação 3D

Lucas Santos Nº13209
Fernando Camilo Nº13233

Introdução (Review 2ª stage done)

No âmbito da disciplina de Introdução à Programação 3D, da licenciatura de Engenharia de Desenvolvimento de Jogos Digitais, foi proposto à turma, um trabalho pratico a ser entregue em três partes sendo que este relatório é referente às entregas da primeira e segunda etapas de entrega.

Numa primeira entrega foi nos requisitado um programa onde seria possível fazer um mapa virtual de acordo com um "heightmap" (mapa de alturas), que definiria os pontos de declive de alturas do mapa, ainda no contexto do mapa de jogo este teria também uma textura aplicada na sua superfície, através de uma imagem. O segundo ponto a ser entregue seria uma camera, que permite-se a visualização dos objetos de jogo que queremos que sejam renderizados e também fosse capaz de navegar pelo mapa de jogo usando como referencia de altura, as posições no eixo dos Yy dos vértices que constituem o mapa. Por ultimo a camera teria também que ser capaz de ter um movimento de rotação horizontal e vertical, regulado pelo posicionamento do rato do computador.

(Segunda fase de entrega)

Neste segunda etapa era necessário dar iluminação aos objetos de jogo, através das normais que são obtidas a partir do terreno, já "rendered" na primeira fase de entrega, fazer também "render" de dois tanques com o respetivo comportamento dinâmico de acordo com o terreno, através de um método homologo ao que permite a camera de "surface follows", variando apenas no que resulta do calculo do método, visto que no que calculava as alturas eram fornecidos valores para as alturas do terreno e neste novo método são fornecidas as normais que cada vértice constituinte do terreno tem, e o tanque tem também os "bones" que permitem que este seja animado. No que à camera diz respeito foi implementada a "ghost" camera que permite um movimento livre pela área de jogo.

Foi também revisto quase na totalidade o código de forma a retificar os problemas que não foram corrigidos na primeira etapa de entrega, como movimento da camera que acompanha o vetor direção de forma a que a camera siga sempre na direção que aponta e a "surface follows" agora tem um movimento fluido depois de ter sido corrigido o "bug" presente na ultima etapa de entrega.

(Segunda fase de entrega)

A primeira etapa de entrega funcionou como alicerce para a visão final do trabalho pratico, que será um jogo em de combates entre tanques, completamente controlados pelo jogador ou por uma AI ("articial inteligence"), em que os elementos de jogo são completamente dinâmicos e enriquecidos com um sistema de colisões, partículas e iluminação que correspondem as etapas seguintes de entrega do trabalho. Quanto à estrutura do relatório, este foi efetuado de forma a dar todas as

Introdução à Programação 3D

Lucas Santos Nº13209
Fernando Camilo Nº13233

explicações tanto matemáticas como de programação, sobre as técnicas delineadas para se obter o resultado pretendido.

E visto que a plataforma usada foi o monogame, haverá também referencias à plataforma tal como a linguagem de programação (C#) que foi usada na elaboração desta primeira parte do trabalho.

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Introdução à Programação 3D

Lucas Santos Nº13209
Fernando Camilo Nº13233

Programação 3D (Review done)

Todo o trabalho da primeira etapa foi estruturado à volta de duas classes Camera e HeightMap, e para se começar a explicar os procedimentos que compõem as classes, é necessário fazer uma introdução teórica tanto aos conceitos de programação 3D, como da "framework" XNA/MonoGame.

O MonoGame usa uma sistema de coordenadas para padronizar e dar orientação no espaço, em que o valor do eixo dos Xx corresponde a um movimento horizontal, o eixo do Yy vertical e o eixo dos Zz à profundidade, sendo que estes eixos regem-se pela regra da orientação da mão direita, ou seja os valores positivos do eixo dos Zz vai em direção do terceiro dedo da mão direita e o eixo positivo dos Xx vai na direção do dedo indicador .

Este sistema de coordenadas tridimensional não corresponde ao ecrã de visualização que é a duas dimensões portanto é preciso codificar em 2D e isso é feito com projeções. Existem duas possíveis a ortogonal em que os objetos aparecem com o tamanho real no ecrã, e a perspetiva que tenta simular a visão humana. Visto que esta é cónica então existe um efeito de perspetiva em que os objetos mais afastados parecem mais pequenos do que são na realidade e é este efeito que a "perspetive projection" tenta simular.

Para estabelecer pontos no espaço são usados grupos de três coordenadas designadas de vértices, que são agrupados geralmente de três em três pelas placas gráficas de forma a construir triângulos (estrutura mais simples que pode ser criada). Para este trabalho usamos o tipo "VertexPositionNormalTexture" que para alem da posição guarda também coordenadas de uma possível textura que se pretenda utilizar e um vetor normal (perpendicular e normal ao vértice) que permite saber a orientação de cada vértice e como a luz ira refletir nesse ponto.

Quanto às primitivas de ligação de vértices, existem quatro possíveis, em que a utilizada foi a "triangle strip" que liga o vértice ao seguinte e o seguinte ao ultimo e penúltimo que antecederam este.

Para transformações são usados produtos de matrizes que por defeito são 4x4 para permitir sempre que haja multiplicação entre elas. Para qualquer "asset" que se queira usar é necessário passar varias matrizes para o "effect" do monogame, em que um "effect" consiste numa estrutura que alberga a geometria do mundo em que pretendemos trabalhar, tal como as matrizes necessárias para colocar todo no mesmo espaço.

A "world matrix" coloca o "asset" no espaço mundo, a "view matrix" recoordena os "assets" do espaço mundo para o da camera, e a "projection matrix" coordena-os com a prespectiva 2D da camera.

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Introdução à Programação 3D

Lucas Santos Nº13209
Fernando Camilo Nº13233

Classe Camera

Variaveis Globais da classe.

Na classe camera começa-se por declarar as variáveis globais de suporte à classe, como o "effect" que é do tipo "BasicEffect", três variáveis que guardam a posição, direção frontal e direção lateral da camera, duas variáveis que guardam os valores do "yaw" e "pitch", que serão em radianos e permitem o movimento de rotação horizontal e vertical sobre o eixo fixo da camera, sendo que o "pitch" é inicializado com o valor 2 de para assim a camera começar com uma ligeira inclinação inferior. Estes valores vão permitir a formação de uma matriz de rotação que os irá aplicar, depois existe uma declaração das matriz de "view" e "projection", são também declaradas duas variáveis que guardam a distancia que é obtida pela diferença entre a posição central e a posição do rato a cada update, por ultimo são declaradas duas variáveis do tipo "float" que irão guardar a altura do vértice do mapa, sobre a qual a camera sobrevoa e a velocidade a que a camera se vai mexer e três variáveis do tipo "bool" que iram ajudar à seleção dos diferentes tipos de cameras.

Construtor da classe

No construtor da classe é inicializado o "effect" que será usado, a variável "aspectRatio" guarda a razão entre a largura e altura da janela.

Quanto à posição, esta toma um valor "default" para a posição inicial de 64 para o eixo dos Xx, 3 para o dos Yy e 64 para os Zz para assim ficar ligeiramente acima do mapa de jogo e no centro deste, já que o mapa tem uma resolução de 128 por 128 .

Depois é criada a matriz "view" usando a função "CreatLookAt" tomando como posição inicial o valor da variável posição, anteriormente definida, a orientação é a soma do vetor posição com o da direção frontal que inicialmente tem o valor zero em todas as suas três coordenadas, e por ultimo a função "CreatLookAt" pede um vetor que funcione como um tripé, sendo que o vetor que lhe é atribuído é o "Up" que apenas tem no eixo dos Yy o valor 1 para um orientação vertical.

A matriz de "projection" é formada pela propriedade "CreatPerspectiveFieldOfView" que pede quatro argumentos, o primeiro define o angulo de abertura da camera que com o auxilio da "MathHelper" e o método "ToRadians", que converte para radianos, definindo o angulo de abertura para 45º, o próximo argumento é o "ratio" da camera que vai tomar o valor do "aspectRatio", previamente calculado, e os dois últimos definem o volume de "render" ("view frustum"), que consiste no volume entre o "near plane" (toma o valor de 1 "float" de distância da camera) e o "far plane" (toma o valor de 1000 "float" de distância da camera), e assim define o que ira aparecer no ecrã.

Lucas Santos Nº13209
Fernando Camilo Nº13233

Por ultimo no que ao construtor da classe diz respeito as matrizes anteriormente definidas, “view” e “projection”, são atribuídas ao effect., a velocidade toma o valor de 0.5, e a posição do rato é colocada para o centro do ecrã.

```
public Camera(GraphicsDevice device) {
    effect = new BasicEffect(device);

    // Calcula o aspectRatio.
    float aspectRatio = (float)(device.Viewport.Width / device.Viewport.Height);

    //Posição inicial da camera.
    posicao = new Vector3(64.0f, 3.0f, 64.0f);

    //A camera fica na posição anteriormente indicada, com direcção inicial zero, e orientação vertical.
    view = Matrix.CreateLookAt(posicao, posicao + direcaoFrontal, Vector3.Up);

    //O angulo de abertura é de 45º, com o aspect ratio calculado anteriormente, e o "near plane" esta a 1 valor de
    //distancia da origem da camera e o far plane a 1000 de distancia.
    projection = Matrix.CreatePerspectiveFieldOfView(MathHelper.ToRadians(45.0f), aspectRatio, 1.0f, 1000.0f);

    //Apliacar as matrix anteriormente criadas ao effect.
    effect.View = view;
    effect.Projection = projection;
    velocidade = 0.5f;

    //Inicia a posição do rato no centro do ecrã.
    Mouse.SetPosition(device.Viewport.Width / 2, device.Viewport.Height / 2);
}
#endregion
```

Metodo Update

O único método da classe camera é o “update” que está encarregue de receber o “input” do jogador e o mapa tal como fazer a atualização da posição da camera, o método começa por ter duas condicionantes, e uma terceira que não está implementada referente à camera que segue o tanque, estas verificam se foi pressionada a tecla F1 e na outra a F2, isto para tornar o “boleano” referente ao tipo de camera para verdadeiro e os outros para falso, de seguida é formado um vetor 2D para guardar a posição inicial em que é colocado o rato, no centro do ecrã dividindo em dois a largura da janela, para a coordenada do Xx e a mesma operação para altura da janela para ter a coordenada respetiva ao eixo dos Yy, usando o vetor anterior guarda-se dentro da variável “posicaoXrato” a distancia entre a coordenada do Xx da posição inicial e o mesmo tipo de coordenada em que o rato se encontra no momento de chamada do “Update” e repete-se o processo para o movimento vertical (eixo Yy), usa-se esse valor faz-se o produto por 0.7, para reduzir a sua velocidade de transição e converte-se para radianos e assim é calculado o valor do yaw (eixo Xx) e o pitch (eixoYy), somando sempre o novo valor, correspondente à nova posição e o valor da posição que se verificou no ultimo Update. No caso do “yaw” é subtraído sempre o valor novo ao atual para ter o movimento de camera correspondente ao do rato, mas no “pitch” é uma adição para ter um movimento vertical invertido, para emular um “joystick” o movimento vertical de uma camara num tripé.

Introdução à Programação 3D

Lucas Santos Nº13209
Fernando Camilo Nº13233

```
#region Camera Modes
if (key.IsKeyDown(Keys.F1)) // FOLLOW TERRAIN
{
    aFloat = true;
    surfaceFollows = false;
    modelFollows = false;
}
if (key.IsKeyDown(Keys.F2)) // FREE CAMERA
{
    surfaceFollows = true;
    modelFollows = false;
    aFloat = false;
}
```

Por ultimo, no que a "Input" diz respeito selecionou-se quatro teclas do "NumPad", de acordo com o enunciado, para modificar a posição da camera no eixo dos Xx e Zz e assim permitir movimentos no plano XZ. Essa alteração de posição é feita pela aplicação do escalar velocidade anteriormente definido na direção frontal caso se utilize a tecla 8, e contrária à direção frontal no caso da tecla recebida no "input" ser a 5. A mesma metodologia é usada para o movimento lateral mas é subtraída à posição para que a tecla 4 faça a camera mover-se para a esquerda e a 6 para a direita.

```
#region Controls
//Movimento transversal da camera pelos três eixos ortogonais.
//Esquerda
if (key.IsKeyDown(Keys.NumPad4))
{
    posicao -= velocidade * direcaoLateral;
}
//Direita
if (key.IsKeyDown(Keys.NumPad6))
{
    posicao -= velocidade * -direcaoLateral;
}
//Frente
if (key.IsKeyDown(Keys.NumPad8))
{
    posicao += velocidade * direcaoFrontal;
}
//Tras
if (key.IsKeyDown(Keys.NumPad5))
{
    posicao += velocidade * -direcaoFrontal;
}
#endregion
```

O método "CreatFromYawPitchRoll" da classe "Matrix" é usado para criar uma matriz que aplica os valores provenientes das variáveis "yaw" e "pitch", esta matriz vai permitir transformar o vetor "foward" e "right" que advêm da classe Vector3 para assim este vetor transformado passar a ser a nova direção da camera.

Introdução à Programação 3D

Lucas Santos Nº13209
Fernando Camilo Nº13233

A ultima alteração que é efetuada à posição da camera depende do tipo selecionado, se for uma camera do tipo “surface follows” a coordenada do eixo dos Yy é dada através de um método da classe “HeightMap” designado de “CalculateInterpolation”, em que este método recebe a posição da Cameira no plano XZ, verifica o valor dos seus quatro vértices mais próximos a esta posição, sendo que estas posições são calculadas por arredondamentos superiores e inferiores aos valores da posição da camera, é calculado um peso, que consiste na distancia desse vértice à posição da camera, multiplicado pelo valor da altura do vértice vizinho mas num mesmo eixo e com este produto obtém-se a percentagem de proximidade desse ponto à camera (peso), repete-se pelos quatro vértices “vizinhos” e por ultimo calcula-se o peso dos vértices à origem no eixo perpendicular (normalmente faz-se a interpolação horizontal e depois vertical) assim conseguimos obter por interpolação dos pontos vizinhos o valor da que a camera deve ter naquele momento. Este método é usado no update para assim termos a posição correta da camera a cada frame, sendo que a esse valor é aplicado um “off-set” de 2 para a camera ficar acima do mapa. No caso do tipo de camera ser “ghost” (livre) então a altura da camera é manipulada pelas teclas 7 e 1 do “NumPad”.

```
//Alteração da posição Y da camera com o calculo da altura anteriormente feito.
if (surfaceFollows == true && aFloat == false && modelFollows == false)
{
    altura = mapa.CalculateInterpolation(posicao.X, posicao.Z) + 2.0f;
    posicao.Y = altura;
}
if (aFloat == true)
{
    if (key.IsKeyDown(Keys.NumPad7))
        posicao.Y += 1.0f;
    else if (key.IsKeyDown(Keys.NumPad1))
        posicao.Y -= 1.0f;
}
```

Por ultimo a “view” é actualizada com a nova posição da camera, nova direção que é somada à posição e o vector “Up” da camera passa a ser definido pelo produto interno entre o vetor que representa a direção lateral pelo que guarda a direção frontal e obtém-se um vetor perpendicular aos dois que é o vector Up e assim sendo temos os três eixos capazes de definir sempre a camera no espaço. E volta-se a colocar o rato no centro do ecrã no fim de cada Update através do método “setPosition” da classe “Mouse”.

```
//Update da view e assim consequentemente da camera.
view = Matrix.CreateLookAt(posicao, (posicao + direcaoFrontal), Vector3.Cross(direcaoLateral, direcaoFrontal));

//faz o reset da posição para o centro do ecra.
Mouse.SetPosition(device.Viewport.Width / 2, device.Viewport.Height / 2);
```

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Introdução à Programação 3D

Lucas Santos Nº13209
Fernando Camilo Nº13233

Classe HeightMap

Na classe "HeightMap" começa-mos por declarar as variáveis "terreno" que e do tipo "Color" que vai ser um array de cores, "mHeight" e "mWidth" que vão guardar as medidas do mapa, duas variáveis "Texture2D", sendo estas "map" e "terrain", em que a variável map vai ser usada para a criação de relevo, e a variável terrain usada para "cobrir" o relevo com uma textura mais apelativa. Declaramos por fim um array de "VertexPositionNormalTexture" que vai armazenar os vários vértices, e um array de "short" que vai armazenar os vários índices. No construtor, tal como na class camara, é inicializado o "effect", a variavel "aspectRatio", as variáveis "mHeight" e "mWidth" com as suas devidas medidas, o array de cores terreno com o seu devido tamanho e a variável scale, que vai definir a altitude do relevo; De seguida temos o método "CreateVertexAndIndex" onde começamos por definir o numero de vértices que vamos usar na variável "vertexCount", e de seguidas criamos um array de "VertexPositionNormalTexture" com o devido tamanho definido anteriormente. Começamos por definir os vértices num for duplo, organizando estes de forma vertical e ao mesmo tempo obtendo a cor da imagem guardando esta na variável colorY. Depois começamos por definir os índices, começamos por definir o numero de vértices, e criando um array de shorts, com o devido tamanho. Declaramos os índices também por organização na vertical. Por fim temos a função Draw onde tivemos que criar um for especificamente para conseguirmos desenhar o mapa Strip a Strip em vez de desenhar um todo de uma vez, usando a função DrawIndexedPrimitives, que recebe o numero de vértices, o "startIndex" que será onde os índices vão começar a cada ciclo, e o numPrimitives que será o numero de primitivas a desenhar.

Na classe HeightMap adicionamos o método CreateNormais(), que faz a interpolação das normais de 2 em 2, em todos os vértices a volta do ponto, fazendo o cross das distancias dos vértices que vai dar a inclinação dos triângulos.

```
#region LADO ESQUERDO
if (x == 0 && z != 0 && z != mHeight - 1)
{
    Vector3 normal = Vector3.Zero;
    normal += Vector3.Cross(Vector3.Normalize(vertices[(z + 1) * mWidth + (x + 0)].Position - vertices[z * mWidth + x].Position), Vector3.Normalize(vertices[(z + 1) * mWidth + (x + 1)].Position - vertices[z * mWidth + x].Position));
    normal += Vector3.Cross(Vector3.Normalize(vertices[(z + 1) * mWidth + (x + 1)].Position - vertices[z * mWidth + x].Position), Vector3.Normalize(vertices[(z + 0) * mWidth + (x + 1)].Position - vertices[z * mWidth + x].Position));
    normal += Vector3.Cross(Vector3.Normalize(vertices[(z - 0) * mWidth + (x + 1)].Position - vertices[z * mWidth + x].Position), Vector3.Normalize(vertices[(z - 1) * mWidth + (x + 1)].Position - vertices[z * mWidth + x].Position));
    normal += Vector3.Cross(Vector3.Normalize(vertices[(z - 1) * mWidth + (x + 1)].Position - vertices[z * mWidth + x].Position), Vector3.Normalize(vertices[(z - 1) * mWidth + (x + 0)].Position - vertices[z * mWidth + x].Position));
    normal /= 4;
    normal = Vector3.Normalize(normal);
    vertices[z * mWidth + x].Normal = normal;
}
```

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Introdução à Programação 3D

Lucas Santos Nº13209
Fernando Camilo Nº13233

Classe Tank

Variáveis Globais da classe.

A classe "Tank" vai modelar os tanques que estarão persentes no jogo, tem varias variáveis globais, em que as primeiras são as três matrizes em que duas serão dadas pela classe camera, a "view" e a "projection", para colocar o tanque no mesmo sentido espacial da camera, enquanto a "world" vai ser usada para aplicar movimentos ao tanque, tanto de rotação como de translação. Depois é criado um "BasicEffect" para albergar todas estas matrizes. Para criar o tanque é preciso uma variável do tipo "Model" que guarda as estruturas que caracterizam o modelo, são depois criadas oito variáveis que iram guardar os "Bones" do tanque e assim permitir que este seja articulado, podendo cada elemento ter ações autónomas. Nas oito variáveis estão incluídas estruturas como o canhão do tanque, as quatro rodas, e também as variáveis "(right)leftSteerBone" que permitem que exista rotação as rodas de acordo com o movimento do tanque. De seguida são criadas oito matrizes que iram aplicar transformações às articulações anteriormente mencionadas e todas estas transformações são guardadas num array de matrizes de forma a facilitar a sua utilização no método "Draw" desta classe. Passando agora para as variáveis do tipo float, começando pela scale que se responsabiliza pela escala do tamanho do tanque, depois duas variáveis que guardam os valores dos angulos de rotação tanto do "turret" como do canhão, o "yaw" tal como na camera é usado para reter o valor do angulo rotação do tanque, as ultimas duas controlam a velocidade de rotação das rodas e o valor máximo que as rodas podem rodar. Por ultimo é criado um vetor 3D para guardar a posição do tanque.

Construtor da classe

O construtor da classe recebe como argumentos o "GraphicsDevice", o "content", para carregar os modelos, e a camera. Começa por instanciar o "effect" já criado para recerber o "GraphicsDevice", a matriz "world" é modificada através da função "CreateScale" para aumentar o tamanho do tanque e depois é feito o "load" de todas as articulações do tanque para o "myModel" e são guardadas nas variáveis que foram criadas com esse objetivo, são também guardadas os valores das propriedades "Transform" de cada "bone" dentro das variáveis com esse objetivo e todas essas matrizes de transformação são depois depositadas no array de matrizes para serem utilizadas posteriormente.

Por ultimo é inicializado vetor posição do tanque para as coordenadas zero e a escala do tanque a 0.003.

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Introdução à Programação 3D

Lucas Santos Nº13209
Fernando Camilo Nº13233

```
//<<BONES>>
turretBone = myModel.Bones["turret_geo"];                  //TURRET
cannonBone = myModel.Bones["canon_geo"];                   //CANNON
rightFrontWheelBone = myModel.Bones["r_front_wheel_geo"]; //FRONT RIGHT
leftFrontWheelBone  = myModel.Bones["l_front_wheel_geo"]; //FRONT LEFT
rightBackWheelBone  = myModel.Bones["r_back_wheel_geo" ]; //BACK RIGHT
leftBackWheelBone   = myModel.Bones["l_back_wheel_geo" ]; //BACK LEFT
rightSteerBone  = myModel.Bones["r_steer_geo"];            //RIGHT STEER
leftSteerBone   = myModel.Bones["l_steer_geo"];            //LEFT STEER
//<<TRANSFORM>>
turretTransform = turretBone.Transform;                    //TURRET
cannonTransform  = cannonBone.Transform;                   //CANNON
rightFrontWheelTransform = rightFrontWheelBone.Transform; //FRONT RIGHT
leftFrontWheelTransform  = leftFrontWheelBone.Transform;  //FRONT LEFT
rightBackWheelTransform  = rightBackWheelBone.Transform;  //BACK RIGHT
leftBackWheelTransform   = leftBackWheelBone.Transform;   //BACK LEFT
rightSteerTransform = rightSteerBone.Transform;            //RIGHT STEER
leftSteerTransform  = leftSteerBone.Transform;             //LEFT STEER
```

Método CalculateInclination

Este método é uma copia do que foi criado para a camera de “surface follows” de forma a obter a altura do terreno no ponto em que a camera se encontra, e neste caso foi reutilizado para se obter a normal necessária para dar uma inclinação ao tanque de acordo com o terreno.

Metodo Update

O método “Update” permite que se atualize para novas posições o tanque, começando pela criação de uma matriz rotação através do valor do “yaw”, sendo que esta matriz vai alterar a direção da camera, de maneira homologa ao que foi feito para aplicar rotação na camera. A altura é dada pelo método “CalculateInterpolation” da classe “HeightMap”, também como é feito na classe camera, depois é criada uma matriz de translação que utiliza a posição do frame anterior do tanque para criar o movimento para essa posição. Por ultimo são criados três vetores 3D que funcionam como eixos do tanque em que o “tankNormal” é normalizado e dado pelo método “CalculateInclination” e passa a ser a normal do tanque, o “tankRight” é dado pelo produto interno da direção, calculada anteriormente com a rotação do “yaw”, com a normal do tanque e assim obtém-se um vetor perpendicular aos dois que é a direção transversal, o mesmo método de calculo é usado para de obter o vetor direcional frontal do tanque que define a direção longitudinal do tanque. Estes tres vetores são atribuídos as propriedades “Up”, “Right” e”Forward” da matriz de rotação depois destas ter sido “reset” ao atribuir o valor de matriz identidade.

Introdução à Programação 3D

Lucas Santos Nº13209
Fernando Camilo Nº13233

Agora quanto aos controlos são definidos tal como foi feito na classe Camera, ou seja, ao pressionar as teclas esquerda e direita o "turret" move-se nessas direções e as teclas cima e baixo controlam a subida e descida do canhão, ao passo que as teclas D e A rodam o tanque para a direita e esquerda respetivamente.

```
if (key.IsKeyDown(Keys.A))
{
    this.yaw += MathHelper.ToRadians(1.0f);
    this.steerRange += 0.07f;
    if (steerRange > 0.7f) steerRange = 0.7f;
}
if (key.IsKeyDown(Keys.D))
{
    this.yaw -= MathHelper.ToRadians(1.0f);
    this.steerRange -= 0.07f;
    if (steerRange < -0.7f) steerRange = -0.7f;
}
if (key.IsKeyDown(Keys.W))
{
    this.position -= 0.1f * tankDir;
    this.wheelSpeed += 0.08f;
}
if (key.IsKeyDown(Keys.S))
{
    this.position -= 0.1f * -tankDir;
    this.wheelSpeed -= 0.08f;
}
if (key.IsKeyUp(Keys.A) && key.IsKeyUp(Keys.D)) steerRange = 0;
```

Com a variável "yaw", controlamos a rotação do tanque sobre o eixo Y, a variável "steerRange" é usada para controlar a rotação de viragem das rodas frontais do tanque, tendo esta um valor máximo controlado de 0.7f. Alteramos também a variável "position", onde acrescentamos um valor e multiplicamos por uma vetor de direção, por fim temos a variável "wheelSpeed", que vai controlar a velocidade de rotação das 4 rodas to tanque. No final verificamos se as teclas de rodar o tanque estão a ser pressionadas, se não estiverem, a rotação de viragem das rodas da frente, fica a 0.

Apos verificarmos as teclas premidas e atualizarmos os valores, acrescentamos esses valores aos "Bones" do tanque.

Por ultimo são guardadas as transformações de cada elemento constituinte do tanque, à "Root" é aplicada a matriz de de escala, rotação e translação, ao "turret" é aplicada uma matriz de rotação no eixo do Yy com o escalar do angulo e a matriz de transformação do mesmo, de maneira análoga aplica-se o mesmo processo ao canhão mas a matriz de rotação é do eixo dos Xx e por ultimo copia-se todas para o "myModel" através do array de matrizes.

Introdução à Programação 3D

Lucas Santos Nº13209
Fernando Camilo Nº13233

```
//TANK
myModel.Root.Transform = Matrix.CreateScale(scale) * rotation * translacao;
//TURRET
turretBone.Transform  = Matrix.CreateRotationY(turretAngle) * turretTransform;
//CANNON
cannonBone.Transform   = Matrix.CreateRotationX(cannonAngle)  * cannonTransform;
//WHEELS
rightFrontWheelBone.Transform = Matrix.CreateRotationX(wheelSpeed) * rightFrontWheelTransform;
leftFrontWheelBone.Transform  = Matrix.CreateRotationX(wheelSpeed) * leftFrontWheelTransform;
rightBackWheelBone.Transform  = Matrix.CreateRotationX(wheelSpeed) * rightBackWheelTransform;
leftBackWheelBone.Transform   = Matrix.CreateRotationX(wheelSpeed) * leftBackWheelTransform;
//STEER
rightSteerBone.Transform = Matrix.CreateRotationY(steerRange) * rightSteerTransform;
leftSteerBone.Transform  = Matrix.CreateRotationY(steerRange) * leftSteerTransform;
```

Metodo Draw

No metodo “Draw” apenas são usados dois foreach para percorrer todas as “mesh” do “myModel” de forma a se puder aplicar as transformações do array de matizes a cada elemento e também dar a cada elemento as matrizes “view”, e “projection” da camera.

ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Introdução à Programação 3D

Lucas Santos Nº13209
Fernando Camilo Nº13233

Conclusão

Por fim é possível verificar que todas as tarefas propostas foram compridas, e também as dificuldades sentidas na ultima entrega foram ultrapassadas, tal como os elementos que careciam correção formam alterados e melhorados, ao ponto de a camera que agora foi entregue esta completamente funcional e fluida.

Foi implementada uma classe que modela o tanque, elemento de interatividade do projeto, sendo que no terreno estão presentes dois tanques idênticos mas com controlos diferentes que mais tarde ira corresponder ao inimigo dentro do jogo. Estes dois elementos tem um comportamento dinâmico em resposta ao terreno e na próxima fase de entrega o tanque inimigo ira mesmo ter um comportamento autónomo.

Quanto ao terreno foram calculadas as normais deste de forma a adicionarmos iluminação ao terreno e na próxima fase vão ser implementadas balas com sistema de colisões e também um sistema de partículas de forma a dinamizar o especto visual do projeto.

Sendo assim esta etapa de entrega já faz jus ao tempo despendido no completo geral das duas entregas, aproximando-se mais de uma composição de excelência relativamente aos objetivos que foram propostos para as duas entregas.

IPCA INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE
Licenciatura em Engenharia de desenvolvimento de jogos de digitais
2017/2018