INSTITUTO POLITÉCNICO
DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA

Introdução a Programação 3D

SIMULADOR DE TANQUES – Parte 1

Rui Fernandes nº 11566

Eduardo Alves nº 12561

Índice

Introdução:

Nesta primeira fase de entrega do trabalho implementamos, em MonoGame usando C#, o render do terreno com o mapa de alturas, fornecido pelo professor, e a câmara com surface follow.

Neste relatório iremos apresentar os métodos utilizados para atingir estes objetivos, justificar as nossas escolhas, a nível de código e de formulas matemáticas, e as nossas dificuldades na resolução.

Concluindo, iremos criticar o nosso trabalho discutindo outras formas mais eficientes de concretizar os nossos objetivos.

Análise de Código

Vamos dividir a análise de código em 3 partes:

- Game1.cs

-ClsCâmara.cs

-Mapa.cs

Com esta divisão conseguiremos aprofundar a explicação da função de cada classe.

Game1.cs

Classe criada quando se inicia o projeto pela primeira vez. Nesta são inicializadas outras classes, chamadas quando necessárias (no Update e Draw).

No nosso projeto só inicializamos duas variáveis:

-Mapa;

-Câmara;

```
protected override void Initialize()
{
    // TODO: Add your initialization logic here
    mapa = new Mapa(graphics.GraphicsDevice, Content);
    camera = new ClsCamera(graphics.GraphicsDevice, Vector3.Zero,mapa);
    base.Initialize();
}
```

Fig.01 – Game1 - Método Initialize;

Neste método inicializamos o mapa e a câmara usados no projeto.

Quando se inicializa o mapa é necessário a passagem do GraphicDevice e do Content. O GraphicsDevice por causa da criação do BasicEffect e o content por causa da textura e da sua atribuição à classe.

Na câmara é utilizado o GraphicDevice e um Vector3 inicializado em 0(zero). O GraphicsDevice é necessário para o BasicEffect, o Vector3 é inicializado em 0, mas com a hipótese de ser inicializado em qualquer parte do mapa, e o objeto mapa é necessário para o surface follow, obtendo as alturas dos vértices.

```
protected override void Update(GameTime gameTime)
{
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed || Keyboard.GetState().IsKeyDown(Keys.Escape))
        Exit();

    //Update da câmara a cada gameUpdate
    camera.Update(gameTime);

    base.Update(gameTime);
}

protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);

    //Draw do mapa
    mapa.Draw(graphics.GraphicsDevice,camera);
    base.Draw(gameTime);
}
```

Fig.02 – Game1 – Update e Draw;

Apenas o update da câmara é chamado no método Update. Esta recebe o gameTime devido a sua utilidade na classe.

No método Draw, é chamado o draw do mapa, recebendo este o GraphicsDevice e a câmara.

Mapa.cs

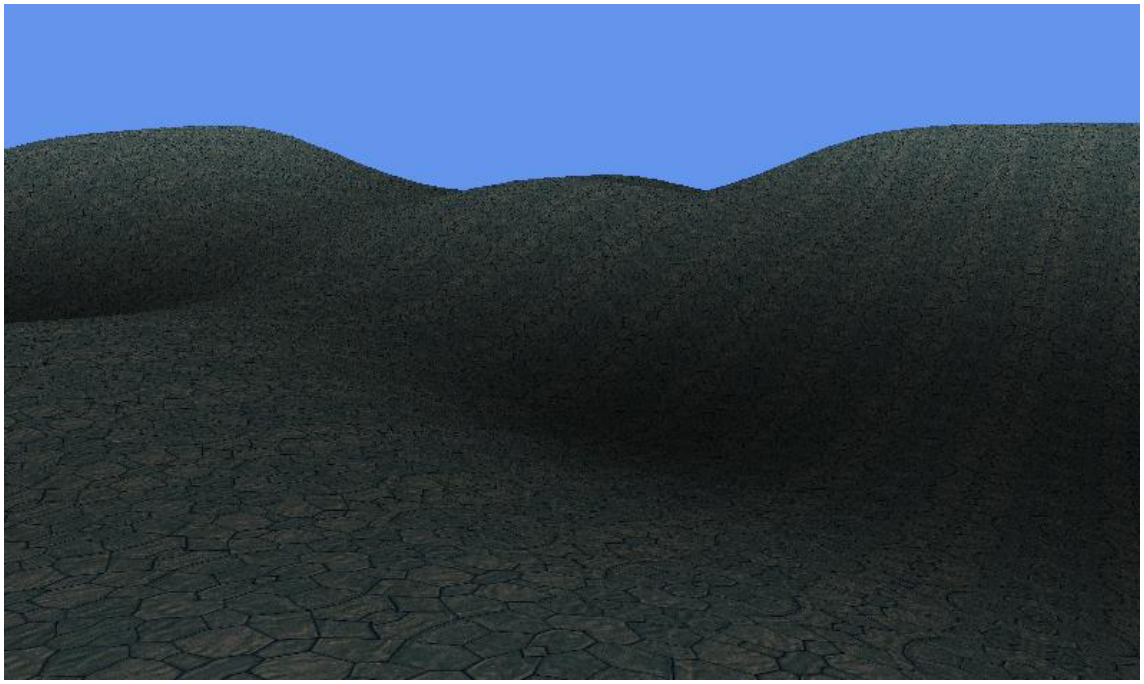

Fig.03 – Terreno

As técnicas utilizadas para o desenvolvimento do terreno foram:

- Utilizando a imagem fornecida pelo professor passar a cor de cada pixel para uma array de cores;
- Gerar vértices, tendo cada um a altura respetiva na posição do array de cores;
- Gerar os índices, que definem as primitivas do tipo TriangleStrip a desenhar;

- Passar a informação para o GPU, utilizando o vertexBuffer e inderBuffer;

Técnicas

A classe mapa possui as seguintes variáveis globais

```
namespace TrabalhoFinal
{
    class Mapa
    {
        BasicEffect effect;
        Texture2D mapaImagem,texture;
        Matrix worldMatrix,viewMatrix;
        Color[] pixeis;

        VertexPositionColorTexture[] vertices;
        short[] verIndex;
        VertexBuffer vertexBuffer;
        IndexBuffer indexBuffer;
        float maxHeight, maxWidht;
```

Fig.04 –Variáveis globais da classe mapa;

Guardar num array de cores a informação da textura

```
private void ReadPixeis()
{
    pixeis = new Color[mapaImagem.Width * mapaImagem.Height];
    mapaImagem.GetData<Color>(pixeis);
}
```

Fig.05 – Método ReadPixeis;

Geração de mapa

```
//Cria o mapa atráves de indices e de triangle list
private void CreateMap()
{
    verIndex = new short[6 * (mapaImagem.Width - 1) * (mapaImagem.Height - 1)];
    vertices = new VertexPositionColorTexture[mapaImagem.Width * mapaImagem.Height];

    maxHeight = mapaImagem.Height;
    maxWidht = mapaImagem.Width;

    for(int z = 0;z<mapaImagem.Width;z++)
    {
        for(int x = 0;x<mapaImagem.Height;x++)
        {
            vertices[x+z*mapaImagem.Width] = new VertexPositionColorTexture(
                new Vector3((float)x,(float)pixeis[x+z*mapaImagem.Width].R/255*10f, (float)z),
                pixeis[x + z * mapaImagem.Width], new Vector2(x%2,z%2));
        }
    }

    int contador = 0;

    //Indices calculados 6 a 6 de modo que cada ciclo seja um "quadrado" da textura
    for(int y = 0;y<mapaImagem.Height-1;y++)
    {
        for(int x = 0;x<mapaImagem.Width-1;x++)
        {
            verIndex[contador] = (short)(x + y * mapaImagem.Width);
            verIndex[contador + 1] = (short)(x + y * mapaImagem.Width + 1);
            verIndex[contador+2] = (short)(x + (y + 1) * mapaImagem.Width);
            verIndex[contador + 3] = (short)(x + y * mapaImagem.Width + 1);
            verIndex[contador + 4] = (short)(x + (y + 1) * mapaImagem.Width + 1);
            verIndex[contador + 5] = (short)(x + (y + 1) * mapaImagem.Width);
            contador += 6;
        }
    }
}
```

Fig.06 – Método de geração de mapa a partir de vértices;

São iniciados dois arrays, um para indexar os vértices e outro para guardar os vértices do terreno. O tamanho do array para o número de vértices é calculado multiplicando a altura e o comprimento da imagem fornecida pelo professor.

São usados dois ciclos *for* para percorrer o array de vértices e preenchendo-o com o *Vector3*, *Color* e um *Vector2*, sendo esta a textura.

Para o array de índices, como estamos a fazer por triangle list, temos e criar cada índice 6 a 6 já que pra construir um quadrado do mapa é preciso 2 triângulos. Sendo este usado no indexBuffer para fornecer a ordem para desenho, dos vértices, no Draw.

Construtor da classe Mapa

```
public Mapa(GraphicsDevice device, ContentManager Content)
{
    effect = new BasicEffect(device);
    worldMatrix = Matrix.Identity;
    mapaImagem = Content.Load<Texture2D>("lh3d1");
    texture = Content.Load<Texture2D>("ground1");

    ReadPixeis();

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

    effect.Projection = Matrix.CreatePerspectiveFieldOfView(
        MathHelper.ToRadians(45.0f),
        aspectRatio, 1.0f, 10.0f);

    effect.LightingEnabled = false;
    effect.VertexColorEnabled = true;

    effect.TextureEnabled = true;
    effect.Texture = texture;

    CreateMap();

    vertexBuffer = new VertexBuffer(device,
        typeof(VertexPositionColorTexture),
        vertices.Length,
        BufferUsage.None);

    vertexBuffer.SetData<VertexPositionColorTexture>(vertices);

    indexBuffer = new IndexBuffer(device,
        typeof(short),
        verIndex.Length,
        BufferUsage.None);

    indexBuffer.SetData(verIndex);
}
```

Fig.06 – Construtor;

No construtor são chamados os métodos ReadPixeis e CreatMap, explicados em cima, e o vertexBuffer e o indexBuffer.

Draw

```
//Metodo draw com triangle list
public void Draw(GraphicsDevice device,ClsCamera camera)
{
    effect.World = worldMatrix;
    effect.View = camera.ViewMatrixCamera;
    effect.Projection = camera.ProjectionMatrixCamera;

    effect.CurrentTechnique.Passes[0].Apply();

    device.SetVertexBuffer(vertexBuffer);
    device.Indices = indexBuffer;

    device.DrawIndexedPrimitives(PrimitiveType.TriangleList, 0, 0, verIndex.Length / 3);
}
```

Fig.07 – Método Draw

World Matrix valor estático, a camara é que se mexe, Buffers, TriangleList

Propriedades

```
public float MapBoundariesHeight
{
    get
    {
        return maxHeight;
    }
}

public float MapBoundariesWidth
{
    get
    {
        return maxWidht;
    }
}

public VertexPositionColorTexture[] mapVertices
{
    get
    {
        return vertices;
    }
}
```

Fig.08 – Funções

Funções que retornam dados para serem utilizados na classe camara.cs.

Câmara.cs

A câmara implementa as seguintes funcionalidades:

- Surface follow, com limites do terreno;
- Movimento através do numpad do teclado;
- Direção (yaw e pitch) através do rato

ClsCamera

```
public ClsCamera(GraphicsDevice device, Vector3 startPos,Mapa map)
{
    width = device.Viewport.Width;
    height = device.Viewport.Height;
    yaw = 0;
    pitch = 0;
    position = new Vector3(8.0f,5.0f, 8.0f);
    dir = Vector3.Zero - position;
    dir.Normalize();
    effect = new BasicEffect(device);
    float aspectRatio = (float)(width /
        height);
    viewMatrix = Matrix.CreateLookAt(
        position,
        dir,
        Vector3.Up);
    projectionMatrix = Matrix.CreatePerspectiveFieldOfView(
        MathHelper.ToRadians(45.0f),
        aspectRatio, 0.5f, 50.0f);

    effect.View = viewMatrix;
    effect.Projection = projectionMatrix;
    oldState = Mouse.GetState();

    this.map = map;
}
```

Fig.09 – ClsCamera Construtor

Inicia a posição da câmara no mundo e a direção para onde este está virado, o BasicEffect é utilizado para termos acesso a viewMatrix e a projectionMatrix.

Update

```
public void Update(GameTime gametime)
{
    Vector3 oldPos = position;
    float speed = 0.5f;

    yawPitchCalc();

    HeightY();

    Move(speed);

    //verificação da camera se passa limites do campo e caso passe atriu a posição antiga
    if ((position.X < 0 || position.Z < 0))
        position = oldPos;
    if ((position.X > 127 || position.Z > 127))
        position = oldPos;

    //calculo do vetor direção
    dir.X = (float)Math.Cos(yaw) * (float)Math.Cos(pitch) + position.X;
    dir.Z = -(float)Math.Sin(yaw) * (float)Math.Cos(pitch) + position.Z;
    dir.Y = (float)Math.Sin(pitch) + position.Y;

    //actualizar viewMatriz da camera
    viewMatrix = Matrix.CreateLookAt(position, dir, Vector3.Up);
    Mouse.SetPosition((int)(width / 2), (int)(height / 2));
    oldState = mouse;
}
```

Fig.10 – Update;

Método que calcula o vetor direção, usando a posição, em cada gameUpdate e se o jogador premiu alguma tecla de controlo. E a rotação?

Guarda sempre a posição do frame anterior para que caso a nova posição saia do mapa este obtem a posição antiga, evitando alguns bugs.

A posição anterior é sempre guardada e comparada com a nova, sendo esta uma posição fora do mapa, esta obtém a posição anterior invés.

No final do update, atualiza-se o estado do rato para o centro do viewport.

YawPitchCalc

```
//Cálculo do yaw e pitch atráves da deslocação do rato do meio do ecra
// ate a posição final
public void yawPitchCalc()
{
    mouse = Mouse.GetState();
    Vector2 mousePos;

    mousePos.X = mouse.X;
    mousePos.Y = mouse.Y;

    mousePos.X -= width / 2;
    mousePos.Y -= height / 2;

    yaw -= mousePos.X * scale;

    pitch = MathHelper.Clamp(pitch + mousePos.Y * scale, -1.5f, 1.5f);
}
```

Fig.10 – Método yawPitchCalc;

Método que calcula o yaw e o pitch através da deslocação do rato desde o centro do viewport.

Move

```
//função com o movimento
public void Move(float speed)
{
    KeyboardState keys = Keyboard.GetState();
    if (keys.IsKeyDown(Keys.NumPad5))
    {
        position.X -= (dir.X - position.X) * speed;
        position.Z -= (dir.Z - position.Z) * speed;

    }
    if (keys.IsKeyDown(Keys.NumPad8))
    {
        position.X += (dir.X - position.X) * speed;
        position.Z += (dir.Z - position.Z) * speed;
    }

    //Cálculo das normais para andar paralelo à direçao
    if (keys.IsKeyDown(Keys.NumPad4))
    {
        position -= speed * Vector3.Cross(dir - position, Vector3.Up);
    }
    if (keys.IsKeyDown(Keys.NumPad6))
    {
        position += speed * Vector3.Cross(dir - position, Vector3.Up);
    }
}
```

Fig.11 – Método Move;

Usando o input das teclas do numPad é calculada a posição da câmara com uma velocidade determinada.

Este movimento é calculado de acordo com a sua direção.

Height

```
//Função que cálcula a altura do mapa e assim atribui essa altura a posição da camera
public void HeightY()
{
    Vector3 verticeA,verticeB,verticeC,verticeD;

    //Obtem os vertices adjacentes a camera
    if ((int)(position.X) + (int)(position.Z + 1) * (int)map.MapBoundariesHeight < map.MapBoundariesHeight * map.MapBoundariesWidth && (int)(position
    {
        verticeA = map.mapVertices[(int)(position.X) + (int)position.Z * (int)map.MapBoundariesHeight].Position;
        verticeB = map.mapVertices[(int)(position.X + 1) + (int)position.Z * (int)map.MapBoundariesHeight].Position;
        verticeC = map.mapVertices[(int)(position.X) + (int)(position.Z + 1) * (int)map.MapBoundariesHeight].Position;
        verticeD = map.mapVertices[(int)(position.X+1) + (int)(position.Z + 1) * (int)map.MapBoundariesHeight].Position;
    }
    else
    {
        verticeA = map.mapVertices[(int)map.MapBoundariesWidth * (int)map.MapBoundariesHeight-1].Position;
        verticeB = map.mapVertices[(int)map.MapBoundariesWidth * (int)map.MapBoundariesHeight - 1].Position;
        verticeC = map.mapVertices[(int)map.MapBoundariesWidth * (int)map.MapBoundariesHeight - 1].Position;
        verticeD = map.mapVertices[(int)map.MapBoundariesWidth * (int)map.MapBoundariesHeight - 1].Position;
    }

    //interpolação das alturas para à medida que se anda com a camera o movimento ser fluido
    //interpolação feita com o peso que cada vertice da à camera
    float Ya, Yb, Yc, Yd;
    Ya = verticeA.Y;
    Yb = verticeB.Y;
    Yc = verticeC.Y;
    Yd = verticeD.Y;

    float Yab = (1 - (position.X - verticeA.X)) * Ya + (position.X - verticeA.X) * Yb;
    float Ycd = (1 - (position.X - verticeC.X)) * Yc + (position.X - verticeC.X) * Yd;
    float Y = (1 - (position.Z - verticeA.Z)) * Yab + (position.Z - verticeA.Z) * Ycd;

    position.Y = Y + 2;
}
```

Fig.12 – Método Height;

Usando o objeto mapa, obtêm-se os 4 vértices mais próximos da câmara e faz a interpolação destes para o movimento da câmara ser suave a descer e a subir o terreno

Dificuldades e Análise Crítica

AS grandes dificuldades sentidas neste trabalho foi os controlos da câmara usando um vetor3 como direção e a sua orientação, a indexação dos vértices e obter a altura do terreno como a sua interpolação.

Na nossa perspetiva, numa forma de melhorar este código, usaríamos o triangleStrip na indexação e no Draw, como uma melhor estruturação do código tal como a sua organização.

Conclusão

O desenvolvimento deste trabalho permitiu-nos aprofundar os conhecimentos adquiridos nas aulas. Embora ainda não estejamos completamente a vontade com as técnicas aplicadas, sentimo-nos confiantes com o trabalho apresentado.

Anexo

ClsCamera.cs

```
namespace TrabalhoFinal
{
    class ClsCamera
    {
        private Matrix viewMatrix, projectionMatrix;
        private BasicEffect effect;
        Vector3 position, dir;
        float yaw, pitch;
        float width, height;
        float scale = MathHelper.ToRadians(10) /500;
        MouseState oldState,mouse;
        Mapa map;

        public ClsCamera(GraphicsDevice device, Vector3 startPos,Mapa map)
        {
            width = device.Viewport.Width;
            height = device.Viewport.Height;
            yaw = 0;
            pitch = 0;
            position = new Vector3(8.0f,5.0f, 8.0f);
            dir = Vector3.Zero - position;
            dir.Normalize();
            effect = new BasicEffect(device);
            float aspectRatio = (float)(width /
                height);
            viewMatrix = Matrix.CreateLookAt(
                position,
                dir,
                Vector3.Up);
            projectionMatrix = Matrix.CreatePerspectiveFieldOfView(
                MathHelper.ToRadians(45.0f),
                aspectRatio, 0.5f, 50.0f);

            effect.View = viewMatrix;
            effect.Projection = projectionMatrix;
            oldState = Mouse.GetState();

            this.map = map;
        }


        public void Update(GameTime gametime)
        {
            Vector3 oldPos = position;
            float speed = 0.5f;

            yawPitchCalc();

            HeightY();

            Move(speed);

            //verificação da camera se passa limites do campo e caso passe atriu
a posição antiga
            if ((position.X < 0 || position.Z < 0))
                position = oldPos;
            if ((position.X > 127 || position.Z > 127))
```

```
            position = oldPos;

        //calculo do vetor direção
        dir.X = (float)Math.Cos(yaw) * (float)Math.Cos(pitch) + position.X;
        dir.Z = -(float)Math.Sin(yaw) * (float)Math.Cos(pitch) + position.Z;
        dir.Y = (float)Math.Sin(pitch) + position.Y;

        //actualizar viewMatriz da camera
        viewMatrix = Matrix.CreateLookAt(position, dir, Vector3.Up);
        Mouse.SetPosition((int)(width / 2), (int)(height / 2));
        oldState = mouse;
    }

    public Matrix ViewMatrixCamera
    {
        get
        {
            return viewMatrix;
        }
    }

    public Matrix ProjectionMatrixCamera
    {
        get
        {
            return projectionMatrix;
        }
    }
    //Cálculo do yaw e pitch atráves da deslocação do rato do meio do ecra
    // ate a posição final
    public void yawPitchCalc()
    {
        mouse = Mouse.GetState();
        Vector2 mousePos;

        mousePos.X = mouse.X;
        mousePos.Y = mouse.Y;

        mousePos.X -= width / 2;
        mousePos.Y -= height / 2;

        yaw -= mousePos.X * scale;

        pitch = MathHelper.Clamp(pitch + mousePos.Y * scale, -1.5f, 1.5f);
    }
    //função com o movimento
    public void Move(float speed)
    {
        KeyboardState keys = Keyboard.GetState();
        if (keys.IsKeyDown(Keys.NumPad5))
        {
            position.X -= (dir.X - position.X) * speed;
            position.Z -= (dir.Z - position.Z) * speed;

        }
        if (keys.IsKeyDown(Keys.NumPad8))
        {
            position.X += (dir.X - position.X) * speed;
            position.Z += (dir.Z - position.Z) * speed;
        }

        //Cálculo das normais para andar paralelo à direçao
```

```
            if (keys.IsKeyDown(Keys.NumPad4))
            {
                position -= speed * Vector3.Cross(dir - position, Vector3.Up);
            }
            if (keys.IsKeyDown(Keys.NumPad6))
            {
                position += speed * Vector3.Cross(dir - position, Vector3.Up);
            }
        }
        //Função que cálcula a altura do mapa e assim atribui essa altura a
posição da camera
        public void HeightY()
        {
            Vector3 verticeA,verticeB,verticeC,verticeD;

            //Obtem os vertices adjacentes a camera
            if ((int)(position.X) + (int)(position.Z + 1) *
(int)map.MapBoundariesHeight < map.MapBoundariesHeight * map.MapBoundariesWidth
&& (int)(position.X) + (int)(position.Z + 1) * (int)map.MapBoundariesHeight >0)
            {
                verticeA = map.mapVertices[(int)(position.X) + (int)position.Z *
(int)map.MapBoundariesHeight].Position;
                verticeB = map.mapVertices[(int)(position.X + 1) +
(int)position.Z * (int)map.MapBoundariesHeight].Position;
                verticeC = map.mapVertices[(int)(position.X) + (int)(position.Z +
1) * (int)map.MapBoundariesHeight].Position;
                verticeD = map.mapVertices[(int)(position.X+1) + (int)(position.Z
+ 1) * (int)map.MapBoundariesHeight].Position;
            }
            else
            {
                verticeA = map.mapVertices[(int)map.MapBoundariesWidth *
(int)map.MapBoundariesHeight-1].Position;
                verticeB = map.mapVertices[(int)map.MapBoundariesWidth *
(int)map.MapBoundariesHeight - 1].Position;
                verticeC = map.mapVertices[(int)map.MapBoundariesWidth *
(int)map.MapBoundariesHeight - 1].Position;
                verticeD = map.mapVertices[(int)map.MapBoundariesWidth *
(int)map.MapBoundariesHeight - 1].Position;
            }
            //interpolação das alturas para à medida que se anda com a camera o
movimento ser fluido
            //interpolação feita com o peso que cada vertice da à camera
            float Ya, Yb, Yc, Yd;
            Ya = verticeA.Y;
            Yb = verticeB.Y;
            Yc = verticeC.Y;
            Yd = verticeD.Y;

            float Yab = (1 - (position.X - verticeA.X)) * Ya + (position.X -
verticeA.X) * Yb;
            float Ycd = (1 - (position.X - verticeC.X)) * Yc + (position.X -
verticeC.X) * Yd;
            float Y = (1 - (position.Z - verticeA.Z)) * Yab + (position.Z -
verticeA.Z) * Ycd;

            position.Y = Y + 2;
        }
    }
}
```

Mapa.cs

```
namespace TrabalhoFinal
{
    class Mapa
    {
        BasicEffect effect;
        Texture2D mapaImagem,texture;
        Matrix worldMatrix,viewMatrix;
        Color[] pixeis;

        VertexPositionColorTexture[] vertices;
        short[] verIndex;
        VertexBuffer vertexBuffer;
        IndexBuffer indexBuffer;
        float maxHeight, maxWidht;

        public Mapa(GraphicsDevice device, ContentManager Content)
        {
            effect = new BasicEffect(device);
            worldMatrix = Matrix.Identity;
            mapaImagem = Content.Load<Texture2D>("lh3d1");
            texture = Content.Load<Texture2D>("ground1");

            ReadPixeis();

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

            effect.Projection = Matrix.CreatePerspectiveFieldOfView(
                MathHelper.ToRadians(45.0f),
                aspectRatio, 1.0f, 10.0f);

            effect.LightingEnabled = false;
            effect.VertexColorEnabled = true;

            effect.TextureEnabled = true;
            effect.Texture = texture;

            CreateMap();

            vertexBuffer = new VertexBuffer(device,
                typeof(VertexPositionColorTexture),
                vertices.Length,
                BufferUsage.None);

            vertexBuffer.SetData<VertexPositionColorTexture>(vertices);

            indexBuffer = new IndexBuffer(device,
                typeof(short),
                verIndex.Length,
                BufferUsage.None);

            indexBuffer.SetData(verIndex);
        }

        private void ReadPixeis()
        {
            pixeis = new Color[mapaImagem.Width * mapaImagem.Height];
            mapaImagem.GetData<Color>(pixeis);
        }
```

```
        //Cria o mapa atráves de indices e de triangle list
        private void CreateMap()
        {
            verIndex = new short[6 * (mapaImagem.Width - 1) * (mapaImagem.Height
- 1)];
            vertices = new VertexPositionColorTexture[mapaImagem.Width *
mapaImagem.Height];

            maxHeight = mapaImagem.Height;
            maxWidht = mapaImagem.Width;

            for(int z = 0;z<mapaImagem.Width;z++)
            {
                for(int x = 0;x<mapaImagem.Height;x++)
                {
                    vertices[x+z*mapaImagem.Width] = new
VertexPositionColorTexture(new
Vector3((float)x,(float)pixeis[x+z*mapaImagem.Width].R/255*10f, (float)z),
pixeis[x + z * mapaImagem.Width], new Vector2(x%2,z%2));
                }
            }

            int contador = 0;

            //Indices calculados 6 a 6 de modo que cada ciclo seja um "quadrado"
da textura
            for(int y = 0;y<mapaImagem.Height-1;y++)
            {
                for(int x = 0;x<mapaImagem.Width-1;x++)
                {
                    verIndex[contador] = (short)(x + y * mapaImagem.Width);
                    verIndex[contador + 1] = (short)(x + y * mapaImagem.Width +
1);
                    verIndex[contador+2] = (short)(x + (y + 1) *
mapaImagem.Width);
                    verIndex[contador + 3] = (short)(x + y * mapaImagem.Width +
1);
                    verIndex[contador + 4] = (short)(x + (y + 1) *
mapaImagem.Width + 1);
                    verIndex[contador + 5] = (short)(x + (y + 1) *
mapaImagem.Width);
                    contador += 6;
                }
            }
        }

        //Metodo draw com triangle list
        public void Draw(GraphicsDevice device,ClsCamera camera)
        {
            effect.World = worldMatrix;
            effect.View = camera.ViewMatrixCamera;
            effect.Projection = camera.ProjectionMatrixCamera;

            effect.CurrentTechnique.Passes[0].Apply();

            device.SetVertexBuffer(vertexBuffer);
            device.Indices = indexBuffer;

            device.DrawIndexedPrimitives(PrimitiveType.TriangleList, 0, 0,
verIndex.Length / 3);
        }
```

```
        public float MapBoundariesHeight
        {
            get
            {
                return maxHeight;
            }
        }

        public float MapBoundariesWidth
        {
            get
            {
                return maxWidht;
            }
        }

        public VertexPositionColorTexture[] mapVertices
        {
            get
            {
                return vertices;
            }
        }
    }
}
```