

الدرس الرابع عشر و الأخير

أهلاً بكم في الدرس الرابع عشر و الأخير من سلسلة دروس تعلم 3D Xna (السلسلة الثانية)، في هذا الدرس سنقوم بإضافة تأخير على حركة الكاميرا التي تتبع الطائرة، "Lerp".

محاكي الطيران الخاص بنا على وشك الإنتهاء. أحد أكثر الأشياء المزعجة التي مازالت موجودة، هو أن الطائرة تستجيب بشكل فوري إلى مدخلات لوحة المفاتيح. في الحقيقة، ليس ذلك ما يسبب الإزعاج، إنما إستجابة الكاميرا للحركة بشكل لحظي هو الذي يسبب ذلك الإزعاج. حيث ينتج عن هذا السلوك الحركة السريعة للكاميرا.

باستخدام التقنيات المتوفرة داخليا في ال Xna، حل هذه المشكلة سهل جدا. الشيء الذي سوف نقوم بعمله هو جعل الكاميرا أبطأ قليلا. للحصول على ذلك، سوف نقوم بتخزين وتتبع دوران الكاميرا بشكل منفصل. الآن عندما يقوم اللاعب بالضغط على أحد الأسهم، سوف يتغير دوران الطائرة بشكل مباشر. و بدلا من تعديل دوران الكاميرا بشكل مباشر بناء على دوران الطائرة. سوف نقوم بجعل الكاميرا تدور بشكل أبطأ بالنسبة لدوران الطائرة. سوف يستغرق الدوران أكثر ببعض الإطارات قبل أن يصبح دوران الكاميرا مساوي لدوران الطائرة.

لنبدأ بإضافة متغير رباعي "Quaternion"، حيث سوف نستخدمه لتخزين دوران الكاميرا:

```
Quaternion cameraRotation = Quaternion.Identity;
```

بعدها، قم بتعديل الكود في الدالة UpdateCamera بحيث تستخدم هذا المتغير للدوران بدلا من المتغير الخاص بالطائرة:

```
Vector3 campos = new Vector3(0, 0.1f, 0.6f);
campos = Vector3.Transform(campos,
Matrix.CreateFromQuaternion(cameraRotation));
campos += xwingPosition;

Vector3 camup = new Vector3(0, 1, 0);
camup = Vector3.Transform(camup,
Matrix.CreateFromQuaternion(cameraRotation));
```

الآن ما زلنا بحاجة لجعل دوران الكاميرا يتبع دوران الطائرة. هذا ما سوف نقوم بعمله: في كل إطار "frame" سوف نجعل الكاميرا تقترب من دوران الطائرة ب 10% من قيمة الدوران الكلي. يبدو ذلك صعب جدا، ولكن باستخدام ال Xna ليست هنالك أي صعوبة: فقط قم بإضافة السطر التالي كأول سطر في الدالة UpdateCamera:

```
cameraRotation = Quaternion.Lerp(cameraRotation, xwingRotation, 0.1f);
```

دالة ال Lerp هي أحد أروع الدوال في ال Xna. حيث بإمكانك تحديد رباعيين "Quaternions" للدالة Lerp، و تحدد أنك تريد معرفة قيمة ال 10% بين القيمتين. هذا ما يفعله السطر السابق.

أكثر من ذلك، في ال Xna الكائنات Vector2، Vector3، Vector4، Matrix، Quaternion، و ربما المزيد من الكائنات تحتوي على الدالة Lerp. لذا كمثال و للتوضيح أكثر: افترض أن لدينا متغيريين من النوع Vector2: V1=(10,50) و V2=(20,100). إذا قمنا باستخدام الأمر Vector2.Lerp(V1,V2,0.2f)، سيعطينا ذلك 20% بين ال V1 و ال V2، وهو في حالتنا هذه (12,60). بالتأكيد سوف تستخدم هذه الدالة بشكل كبير عندما تقوم بإنشاء لعبة.

حسنًا هذا كل شيء، لقد كنت أرغب أن يكون الموضوع أكثر تعقيدًا ولكن الدالة Lerp إهتمت بالجزء الصعب من الموضوع. عندما تقوم بتشغيل هذا الكود، سوف ترى أن الكاميرا تتبع الطائرة بشكل أكثر سلاسة و نعومة.



حاول حل التمرين التالي، لممارسة ما قد تعلمته في هذا الدرس:

- قم بتغيير القيمة 0.1 إلى القيمة 0.01 و القيمة 0.5 لملاحظة الفرق.

بهذا نكون قد أنهينا السلسلة الثانية من دروس ال Xna ثلاثية الأبعاد. لقد قطعنا شوطًا طويلاً، من رسم مثلث بسيطة في السلسلة الأولى، إلى إنشاء محاكي طيران حقيقي في هذه السلسلة! بالتأكيد هذه ليست نقطة النهاية، لذا بعد أن نأخذ إجازة، هنالك سلسلة ثالثة بالإنظار. في السلسلة الثالثة سوف نتعرف على ال HLSL، وهي حيث يبدأ المرح فعلياً.

نحمد لله على إنهاء ترجمة هذه السلسلة، و نستعينه على إنهاء ما تبقى من دروس ال Xna في السلاسل القادمة بإذنه تعالى.

مواضيع السلسلة القادمة سوف تكون متقدمة بعض الشيء، إضافة إلى أنها سوف تعطي نتائج متقدمة و أكثر احترافية. حيث سوف تتعرض بشكل أساسي إلى لغة ال HLSL "لغة التظليل العالية المستوى" و بعض المفاهيم و التقنيات المتعلقة بها.

سأبدأ في ترجمة السلسلة القادمة بعد إجازة (لا أدري كم ستطول، ولكن أرجو أن تكون قصيرة). حتى ذلك الحين أرجو أن تحصل هذه السلسلة و سابقتها على أكبر عدد من القراء، حيث يكون بإمكاننا الإنطلاق بشكل جماعي في السلسلة الثالثة.

المترجم : خلدون خالد

الفريق العربي للبرمجة/ <http://arabteam2000-forum.com/> :

تم بحمد الله

الكود النهائي للسلسلة:

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace XNAseries2
{
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        struct Bullet
        {
            public Vector3 position;
            public Quaternion rotation;
        }

        private struct VertexPointSprite
        {
            private Vector3 position;
            private float pointSize;

            public VertexPointSprite(Vector3 position, float pointSize)
            {
                this.position = position;
                this.pointSize = pointSize;
            }

            public static readonly VertexElement[] VertexElements =
            {
                new VertexElement(0, 0, VertexElementFormat.Vector3, VertexElementMethod.Default,
                VertexElementUsage.Position, 0),
                new VertexElement(0, sizeof(float)*3, VertexElementFormat.Single,
                VertexElementMethod.Default, VertexElementUsage.PointSize, 0),
            };
            public static int SizeInBytes = sizeof(float) * (3 + 1);
        }

        enum CollisionType { None, Building, Boundary, Target }

        const int maxTargets = 50;
        int[] buildingHeights = new int[] { 0, 2, 2, 6, 5, 4 };

        GraphicsDeviceManager graphics;
        GraphicsDevice device;

        Effect effect;
        Vector3 lightDirection = new Vector3(3, -2, 5);
        Matrix viewMatrix;
        Matrix projectionMatrix;

        Texture2D sceneryTexture;
        Texture2D bulletTexture;
        Texture2D[] skyboxTextures;
        Model xwingModel;
        Model targetModel;
        Model skyboxModel;

        Vector3 xwingPosition = new Vector3(8, 1, -3);
        Quaternion xwingRotation = Quaternion.Identity;
        Quaternion cameraRotation = Quaternion.Identity;

        int[,] floorPlan;
        float gameSpeed = 1.0f;

        VertexBuffer cityVertexBuffer;
        VertexDeclaration texturedVertexDeclaration;
        VertexDeclaration pointSpriteVertexDeclaration;

        BoundingBox[] buildingBoundingBoxes;
        BoundingBox completeCityBox;
```

```

List<BoundingSphere> targetList = new List<BoundingSphere> ();
List<Bullet> bulletList = new List<Bullet> ();          double lastBulletTime = 0;

public Game1()
{
    graphics = new GraphicsDeviceManager(this);
    Content.RootDirectory = "Content";
}

protected override void Initialize()
{
    graphics.PreferredBackBufferWidth = 500;
    graphics.PreferredBackBufferHeight = 500;
    graphics.IsFullScreen = false;
    graphics.ApplyChanges();
    Window.Title = "Riemer's XNA Tutorials -- Series 2";

    LoadFloorPlan();
    lightDirection.Normalize();
    SetUpBoundingBoxes();
    AddTargets();

    base.Initialize();
}

protected override void LoadContent()
{
    device = graphics.GraphicsDevice;

    effect = Content.Load<Effect> ("effects");
    sceneryTexture = Content.Load<Texture2D> ("texturemap");
    bulletTexture = Content.Load<Texture2D> ("bullet");
    xwingModel = LoadModel("xwing");
    targetModel = LoadModel("target");
    skyboxModel = LoadModel("skybox", out skyboxTextures);

    SetUpVertices();
    SetUpCamera();

    pointSpriteVertexDeclaration = new VertexDeclaration(device,
VertexPointSprite.VertexElements);
}

private Model LoadModel(string assetName)
{
    Model newModel = Content.Load<Model> (assetName);          foreach (ModelMesh mesh in
newModel.Meshes)
        foreach (ModelMeshPart meshPart in mesh.MeshParts)
            meshPart.Effect = effect.Clone(device);
    return newModel;
}

private Model LoadModel(string assetName, out Texture2D[] textures)
{
    Model newModel = Content.Load<Model> (assetName);
    textures = new Texture2D[newModel.Meshes.Count];
    int i = 0;
    foreach (ModelMesh mesh in newModel.Meshes)
        foreach (BasicEffect currentEffect in mesh.Effects)
            textures[i++] = currentEffect.Texture;

    foreach (ModelMesh mesh in newModel.Meshes)
        foreach (ModelMeshPart meshPart in mesh.MeshParts)
            meshPart.Effect = effect.Clone(device);

    return newModel;
}

private void LoadFloorPlan()
{
    floorPlan = new int[,]
    {
        {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,1,1,0,0,0,1,1,0,0,1,0,1},
        {1,0,0,1,1,0,0,0,1,0,0,0,1,0,1},
        {1,0,0,0,1,1,0,1,1,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,1,0,0,1},
    }
}

```

```

        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,1,1,0,0,0,1,0,0,0,0,0,0,1},
        {1,0,1,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,1,0,0,0,0,0,0,0,0,1},
        {1,0,0,0,0,1,0,0,0,1,0,0,0,0,1},
        {1,0,1,0,0,0,0,0,0,1,0,0,0,0,1},
        {1,0,1,1,0,0,0,0,1,1,0,0,0,1,1},
        {1,0,0,0,0,0,0,0,1,1,0,0,0,1,1},
        {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
    };

    Random random = new Random();
    int differentBuildings = buildingHeights.Length - 1;
    for (int x = 0; x < floorPlan.GetLength(0); x++)
        for (int y = 0; y < floorPlan.GetLength(1); y++)
            if (floorPlan[x, y] == 1)
                floorPlan[x, y] = random.Next(differentBuildings) + 1;
}

private void SetUpBoundingBoxes()
{
    int cityWidth = floorPlan.GetLength(0);
    int cityLength = floorPlan.GetLength(1);

    List<BoundingBox> bbList = new List<BoundingBox> ();
    for (int x = 0; x <
cityWidth; x++)
    {
        for (int z = 0; z < cityLength; z++)
        {
            int buildingType = floorPlan[x, z];
            if (buildingType != 0)
            {
                int buildingHeight = buildingHeights[buildingType];
                Vector3[] buildingPoints = new Vector3[2];
                buildingPoints[0] = new Vector3(x, 0, -z);
                buildingPoints[1] = new Vector3(x + 1, buildingHeight, -z - 1);
                BoundingBox buildingBox = BoundingBox.CreateFromPoints(buildingPoints);
                bbList.Add(buildingBox);
            }
        }
    }
    buildingBoundingBoxes = bbList.ToArray();

    Vector3[] boundaryPoints = new Vector3[2];
    boundaryPoints[0] = new Vector3(0, 0, 0);
    boundaryPoints[1] = new Vector3(cityWidth, 20, -cityLength);
    completeCityBox = BoundingBox.CreateFromPoints(boundaryPoints);
}

private void AddTargets()
{
    int cityWidth = floorPlan.GetLength(0);
    int cityLength = floorPlan.GetLength(1);

    Random random = new Random();

    while (targetList.Count < maxTargets)
    {
        int x = random.Next(cityWidth);
        int z = -random.Next(cityLength);
        float y = (float)random.Next(2000) / 1000f + 1;
        float radius = (float)random.Next(1000) / 1000f * 0.2f + 0.01f;

        BoundingSphere newTarget = new BoundingSphere(new Vector3(x, y, z), radius);

        if (CheckCollision(newTarget) == CollisionType.None)
            targetList.Add(newTarget);
    }
}

private void SetUpCamera()
{
    viewMatrix = Matrix.CreateLookAt(new Vector3(20, 13, -5), new Vector3(8, 0, -7), new
Vector3(0, 1, 0));
    projectionMatrix = Matrix.CreatePerspectiveFieldOfView(MathHelper.PiOver4,

```

```
device.Viewport.AspectRatio, 0.2f, 500.0f);
}

private void SetUpVertices()
{
    int differentBuildings = buildingHeights.Length - 1;
    float imagesInTexture = 1 + differentBuildings * 2;

    int cityWidth = floorPlan.GetLength(0);
    int cityLength = floorPlan.GetLength(1);

    List<VertexPositionNormalTexture> verticesList = new List<VertexPositionNormalTexture> ();
    for (int x = 0; x < cityWidth; x++)
    {
        for (int z = 0; z < cityLength; z++)
        {
            int currentbuilding = floorPlan[x, z];

            //floor or ceiling
            verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
            buildingHeights[currentbuilding], -z), new Vector3(0, 1, 0), new Vector2(currentbuilding * 2 /
            imagesInTexture, 1)));
            verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
            buildingHeights[currentbuilding], -z - 1), new Vector3(0, 1, 0), new Vector2((currentbuilding * 2) /
            imagesInTexture, 0)));
            verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
            buildingHeights[currentbuilding], -z), new Vector3(0, 1, 0), new Vector2((currentbuilding * 2 + 1) /
            imagesInTexture, 1)));
            verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
            buildingHeights[currentbuilding], -z - 1), new Vector3(0, 1, 0), new Vector2((currentbuilding * 2) /
            imagesInTexture, 0)));
            verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
            buildingHeights[currentbuilding], -z - 1), new Vector3(0, 1, 0), new Vector2((currentbuilding * 2 + 1)
            / imagesInTexture, 0)));
            verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
            buildingHeights[currentbuilding], -z), new Vector3(0, 1, 0), new Vector2((currentbuilding * 2 + 1) /
            imagesInTexture, 1)));

            if (currentbuilding != 0)
            {
                //front wall
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1, 0, -z - 1),
                new Vector3(0, 0, -1), new Vector2((currentbuilding * 2) / imagesInTexture, 1)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
                buildingHeights[currentbuilding], -z - 1), new Vector3(0, 0, -1), new Vector2((currentbuilding * 2 - 1)
                / imagesInTexture, 0)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x, 0, -z - 1), new
                Vector3(0, 0, -1), new Vector2((currentbuilding * 2 - 1) / imagesInTexture, 1)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
                buildingHeights[currentbuilding], -z - 1), new Vector3(0, 0, -1), new Vector2((currentbuilding * 2 - 1)
                / imagesInTexture, 0)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1, 0, -z - 1),
                new Vector3(0, 0, -1), new Vector2((currentbuilding * 2) / imagesInTexture, 1)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
                buildingHeights[currentbuilding], -z - 1), new Vector3(0, 0, -1), new Vector2((currentbuilding * 2) /
                imagesInTexture, 0)));

                //back wall
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1, 0, -z), new
                Vector3(0, 0, 1), new Vector2((currentbuilding * 2) / imagesInTexture, 1)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x, 0, -z), new
                Vector3(0, 0, 1), new Vector2((currentbuilding * 2 - 1) / imagesInTexture, 1)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
                buildingHeights[currentbuilding], -z), new Vector3(0, 0, 1), new Vector2((currentbuilding * 2 - 1) /
                imagesInTexture, 0)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
                buildingHeights[currentbuilding], -z), new Vector3(0, 0, 1), new Vector2((currentbuilding * 2 - 1) /
                imagesInTexture, 0)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1, 0, -z), new
                Vector3(0, 0, 1), new Vector2((currentbuilding * 2) / imagesInTexture, 1)));

                //left wall
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x, 0, -z), new
                Vector3(-1, 0, 0), new Vector2((currentbuilding * 2) / imagesInTexture, 1)));
            }
        }
    }
}
```

```
verticesList.Add(new VertexPositionNormalTexture(new Vector3(x, 0, -z - 1), new
Vector3(-1, 0, 0), new Vector2((currentbuilding * 2 - 1) / imagesInTexture, 1)));
verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z - 1), new Vector3(-1, 0, 0), new Vector2((currentbuilding * 2 - 1)
/ imagesInTexture, 0)));

verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z - 1), new Vector3(-1, 0, 0), new Vector2((currentbuilding * 2 - 1)
/ imagesInTexture, 0)));
verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z), new Vector3(-1, 0, 0), new Vector2((currentbuilding * 2) /
imagesInTexture, 0)));
verticesList.Add(new VertexPositionNormalTexture(new Vector3(x, 0, -z), new
Vector3(-1, 0, 0), new Vector2((currentbuilding * 2) / imagesInTexture, 1)));

//right wall
verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1, 0, -z), new
Vector3(1, 0, 0), new Vector2((currentbuilding * 2) / imagesInTexture, 1)));
verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z - 1), new Vector3(1, 0, 0), new Vector2((currentbuilding * 2 - 1)
/ imagesInTexture, 0)));
verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1, 0, -z - 1),
new Vector3(1, 0, 0), new Vector2((currentbuilding * 2 - 1) / imagesInTexture, 1)));

verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z - 1), new Vector3(1, 0, 0), new Vector2((currentbuilding * 2 - 1)
/ imagesInTexture, 0)));
verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1, 0, -z), new
Vector3(1, 0, 0), new Vector2((currentbuilding * 2) / imagesInTexture, 1)));
verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z), new Vector3(1, 0, 0), new Vector2((currentbuilding * 2) /
imagesInTexture, 0)));
    }
}

cityVertexBuffer = new VertexBuffer(device, verticesList.Count *
VertexPositionNormalTexture.SizeInBytes, BufferUsage.WriteOnly);

cityVertexBuffer.SetData<VertexPositionNormalTexture> (verticesList.ToArray());
texturedVertexDeclaration = new VertexDeclaration(device,
VertexPositionNormalTexture.VertexElements);
}

protected override void UnloadContent()
{
}

protected override void Update(GameTime gameTime)
{
    ProcessKeyboard(gameTime);
    float moveSpeed = gameTime.ElapsedGameTime.Milliseconds / 500.0f * gameSpeed;
    MoveForward(ref xwingPosition, xwingRotation, moveSpeed);

    BoundingSphere xwingSpere = new BoundingSphere(xwingPosition, 0.04f);
    if (CheckCollision(xwingSpere) != CollisionType.None)
    {
        xwingPosition = new Vector3(8, 1, -3);
        xwingRotation = Quaternion.Identity;
        gameSpeed /= 1.1f;
    }

    UpdateCamera();
    UpdateBulletPositions(moveSpeed);

    base.Update(gameTime);
}

private void UpdateCamera()
{
    cameraRotation = Quaternion.Lerp(cameraRotation, xwingRotation, 0.1f);

    Vector3 campos = new Vector3(0, 0.1f, 0.6f);
    campos = Vector3.Transform(campos, Matrix.CreateFromQuaternion(cameraRotation));
    campos += xwingPosition;

    Vector3 camup = new Vector3(0, 1, 0);
    camup = Vector3.Transform(camup, Matrix.CreateFromQuaternion(cameraRotation));

    viewMatrix = Matrix.CreateLookAt(campos, xwingPosition, camup);
```

```

        projectionMatrix = Matrix.CreatePerspectiveFieldOfView(MathHelper.PiOver4,
device.Viewport.AspectRatio, 0.2f, 500.0f);
    }

    private void ProcessKeyboard(GameTime gameTime)
    {
        float leftRightRot = 0;

        float turningSpeed = (float)gameTime.ElapsedGameTime.TotalMilliseconds / 1000.0f;
        turningSpeed *= 1.6f * gameSpeed;
        KeyboardState keys = Keyboard.GetState();
        if (keys.IsKeyDown(Keys.Right))
            leftRightRot += turningSpeed;
        if (keys.IsKeyDown(Keys.Left))
            leftRightRot -= turningSpeed;

        float upDownRot = 0;
        if (keys.IsKeyDown(Keys.Down))
            upDownRot += turningSpeed;
        if (keys.IsKeyDown(Keys.Up))
            upDownRot -= turningSpeed;

        Quaternion additionalRot = Quaternion.CreateFromAxisAngle(new Vector3(0, 0, -1),
leftRightRot) * Quaternion.CreateFromAxisAngle(new Vector3(1, 0, 0), upDownRot);
        xwingRotation *= additionalRot;

        if (keys.IsKeyDown(Keys.Space))
        {
            double currentTime = gameTime.TotalGameTime.TotalMilliseconds;
            if (currentTime - lastBulletTime > 100)
            {
                Bullet newBullet = new Bullet();
                newBullet.position = xwingPosition;
                newBullet.rotation = xwingRotation;
                bulletList.Add(newBullet);

                lastBulletTime = currentTime;
            }
        }
    }

    private void MoveForward(ref Vector3 position, Quaternion rotationQuat, float speed)
    {
        Vector3 addVector = Vector3.Transform(new Vector3(0, 0, -1), rotationQuat);
        position += addVector * speed;
    }

    private CollisionType CheckCollision(BoundingSphere sphere)
    {
        for (int i = 0; i < buildingBoundingBoxes.Length; i++)
            if (buildingBoundingBoxes[i].Contains(sphere) != ContainmentType.Disjoint)
                return CollisionType.Building;

        if (completeCityBox.Contains(sphere) != ContainmentType.Contains)
            return CollisionType.Boundary;

        for (int i = 0; i < targetList.Count; i++)
        {
            if (targetList[i].Contains(sphere) != ContainmentType.Disjoint)
            {
                targetList.RemoveAt(i);
                i--;
                AddTargets();

                return CollisionType.Target;
            }
        }

        return CollisionType.None;
    }

    private void UpdateBulletPositions(float moveSpeed)
    {
        for (int i = 0; i < bulletList.Count; i++)
        {
            Bullet currentBullet = bulletList[i];
            MoveForward(ref currentBullet.position, currentBullet.rotation, moveSpeed * 2.0f);
            bulletList[i] = currentBullet;

            BoundingSphere bulletSphere = new BoundingSphere(currentBullet.position, 0.05f);
            CollisionType colType = CheckCollision(bulletSphere);

```

```

        if (colType != CollisionType.None)
        {
            bulletList.RemoveAt(i);
            i--;

            if (colType == CollisionType.Target)
                gameSpeed *= 1.05f;
        }
    }

protected override void Draw(GameTime gameTime)
{
    device.Clear(ClearOptions.Target | ClearOptions.DepthBuffer, Color.DarkSlateBlue, 1.0f,
0);

    DrawSkybox();
    DrawCity();
    DrawModel();
    DrawTargets();
    DrawBullets();

    base.Draw(gameTime);
}

private void DrawCity()
{
    effect.CurrentTechnique = effect.Techniques["Textured"];
    effect.Parameters["xWorld"].SetValue(Matrix.Identity);
    effect.Parameters["xView"].SetValue(viewMatrix);
    effect.Parameters["xProjection"].SetValue(projectionMatrix);
    effect.Parameters["xTexture"].SetValue(sceneryTexture);
    effect.Parameters["xEnableLighting"].SetValue(true);
    effect.Parameters["xLightDirection"].SetValue(lightDirection);
    effect.Parameters["xAmbient"].SetValue(0.5f);
    effect.Begin();
    foreach (EffectPass pass in effect.CurrentTechnique.Passes)
    {
        pass.Begin();
        device.VertexDeclaration = texturedVertexDeclaration;
        device.Vertices[0].SetSource(cityVertexBuffer, 0,
VertexPositionNormalTexture.SizeInBytes);
        device.DrawPrimitives(PrimitiveType.TriangleList, 0, cityVertexBuffer.SizeInBytes /
VertexPositionNormalTexture.SizeInBytes / 3);
        pass.End();
    }
    effect.End();
}

private void DrawModel()
{
    Matrix worldMatrix = Matrix.CreateScale(0.0005f, 0.0005f, 0.0005f) *
Matrix.CreateRotationY(MathHelper.Pi) * Matrix.CreateFromQuaternion(xwingRotation) *
Matrix.CreateTranslation(xwingPosition);

    Matrix[] xwingTransforms = new Matrix[xwingModel.Bones.Count];
    xwingModel.CopyAbsoluteBoneTransformsTo(xwingTransforms);
    foreach (ModelMesh mesh in xwingModel.Meshes)
    {
        foreach (Effect currentEffect in mesh.Effects)
        {
            currentEffect.CurrentTechnique = currentEffect.Techniques["Colored"];
            currentEffect.Parameters["xWorld"].SetValue(xwingTransforms[mesh.ParentBone.Index]
* worldMatrix);
            currentEffect.Parameters["xView"].SetValue(viewMatrix);
            currentEffect.Parameters["xProjection"].SetValue(projectionMatrix);
            currentEffect.Parameters["xEnableLighting"].SetValue(true);
            currentEffect.Parameters["xLightDirection"].SetValue(lightDirection);
            currentEffect.Parameters["xAmbient"].SetValue(0.5f);
        }
        mesh.Draw();
    }
}

private void DrawTargets()
{
    for (int i = 0; i < targetList.Count; i++)
    {
        Matrix worldMatrix = Matrix.CreateScale(targetList[i].Radius) *
Matrix.CreateTranslation(targetList[i].Center);
    }
}

```

```
Matrix[] targetTransforms = new Matrix[targetModel.Bones.Count];
targetModel.CopyAbsoluteBoneTransformsTo(targetTransforms);
foreach (ModelMesh mesh in targetModel.Meshes)
{
    foreach (Effect currentEffect in mesh.Effects)
    {
        currentEffect.CurrentTechnique = currentEffect.Techniques["Colored"];
        currentEffect.Parameters["xWorld"].SetValue(targetTransforms[mesh.ParentBone.Index] * worldMatrix);
        currentEffect.Parameters["xView"].SetValue(viewMatrix);
        currentEffect.Parameters["xProjection"].SetValue(projectionMatrix);
        currentEffect.Parameters["xEnableLighting"].SetValue(true);
        currentEffect.Parameters["xLightDirection"].SetValue(lightDirection);
        currentEffect.Parameters["xAmbient"].SetValue(0.5f);
    }
    mesh.Draw();
}
}

private void DrawBullets()
{
    if (bulletList.Count > 0)
    {
        VertexPointSprite[] spriteArray = new VertexPointSprite[bulletList.Count];
        for (int i = 0; i < bulletList.Count; i++)
            spriteArray[i] = new VertexPointSprite(bulletList[i].position, 50);

        effect.CurrentTechnique = effect.Techniques["PointSprites"];
        Matrix worldMatrix = Matrix.Identity;
        effect.Parameters["xWorld"].SetValue(worldMatrix);
        effect.Parameters["xView"].SetValue(viewMatrix);
        effect.Parameters["xProjection"].SetValue(projectionMatrix);
        effect.Parameters["xTexture"].SetValue(bulletTexture);

        device.RenderState.PointSpriteEnable = true;
        device.RenderState.AlphaBlendEnable = true;
        device.RenderState.SourceBlend = Blend.One;
        device.RenderState.DestinationBlend = Blend.One;

        effect.Begin();
        foreach (EffectPass pass in effect.CurrentTechnique.Passes)
        {
            pass.Begin();
            device.VertexDeclaration = pointSpriteVertexDeclaration;
            device.DrawUserPrimitives(PrimitiveType.PointList, spriteArray, 0,
spriteArray.Length);
            pass.End();
        }
        effect.End();

        device.RenderState.PointSpriteEnable = false;
        device.RenderState.AlphaBlendEnable = false;
    }
}

private void DrawSkybox()
{
    device.SamplerStates[0].AddressU = TextureAddressMode.Clamp;
    device.SamplerStates[0].AddressV = TextureAddressMode.Clamp;

    device.RenderState.DepthBufferWriteEnable = false;
    Matrix[] skyboxTransforms = new Matrix[skyboxModel.Bones.Count];
    skyboxModel.CopyAbsoluteBoneTransformsTo(skyboxTransforms);
    int i = 0;
    foreach (ModelMesh mesh in skyboxModel.Meshes)
    {
        foreach (Effect currentEffect in mesh.Effects)
        {
            Matrix worldMatrix = skyboxTransforms[mesh.ParentBone.Index] *
Matrix.CreateTranslation(xwingPosition);
            currentEffect.CurrentTechnique = currentEffect.Techniques["Textured"];
            currentEffect.Parameters["xWorld"].SetValue(worldMatrix);
            currentEffect.Parameters["xView"].SetValue(viewMatrix);
            currentEffect.Parameters["xProjection"].SetValue(projectionMatrix);
            currentEffect.Parameters["xTexture"].SetValue(skyboxTextures[i++]);
        }
        mesh.Draw();
    }
    device.RenderState.DepthBufferWriteEnable = true;
}
```

```
}  
}
```