

الدرس الثاني عشر

أهلاً بكم في الدرس الثاني عشر من سلسلة دروس تعلم 3D Xna (السلسلة الثانية)، سنقوم في هذا الدرس باستخدام تقنية مزج ألفا "Alpha Blending"، إضافة إلى عمل إكتشاف تصادم للرصاص.

كما أتوقع، فإن نتيجة الدرس السابق لم تعجبك بالتأكيد. حسب توقعي فإن الحدود السوداء حول الكره النارية لا تبدو طبيعية. و أنت على تماماً على حق، يجب أن يتم مزج الحدود السوداء مع أي شيء يكون خلفها. إذا أهلاً بكم إلى هذا الدرس في مزج ألفا.

حتى اللحظة، إذا كان هنالك عنصرين يتشاركان في نفس البكسل، فإن ذلك البكسل يأخذ لون العنصر الأقرب للكاميرا. في أبسط أشكال مزج ألفا، فإن الألوان من العنصرين سوف يتم إضافتهما إلى بعضهما البعض، و البكسل يأخذ 'مجموع' ألوانهما. يطلق على ذلك اسم مزج ألفا الإضافي (الجمعي) "Additive Alpha Blending".

على سبيل المثال. تصور أن لدينا خلفية زرقاء بشكل كامل. عند وضع مثلث أحمر أمامها باستخدام الإعدادات الحالية لدينا، سوف يؤدي إلى ظهور المثلث بلون أحمر ببساطة. بينما عند تشغيل مزج ألفا الإضافي، سوف يكون لون بكسلات المثلث تحتوي على مجموع اللونين الأحمر + الأزرق، لذا سوف يظهر كامل المثلث باللون الأرجواني (البنفسجي).

اللون الأسود هو حالة خاصة. في مصطلحات ال Xna فإن الأسود ليس لونا. إنه لا شيء. لذا أسود + أزرق سوف يعطينا أزرق ببساطة. بشكل عام أيضاً، أسود + أي لون سوف ينتج عنه نفس اللون. و هذا هو السبب الذي قمنا لأجله برسم حدود الكره النارية باللون الأسود، لأنه كلما زاد اللون الأسود في الصورة، فإننا سوف نرى من خلال مساحة أكبر من الخلفية.

متى نريد أن نقوم بتشغيل مزج ألفا؟ نحن بحاجة أن نقوم بمزج رسومات الرصاصات مع الخلفية، ولكن باقي أجزاء المشهد لا يجب أن يتغير. لذا نحن بحاجة لتشغيل مزج ألفا قبل رسم الرصاصات، و من ثم إيقافها بعد ذلك.

قم بإضافة الكود التالي إلى الدالة DrawBullets، قبل عملية استدعاء الدالة effect.Begin:

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

باختصار، عندما يتنافس عنصرين على بكسل واحد و يكون مزج ألفا مشغل، يتم حساب اللون النهائي بالشكل التالي:

$$\text{FinalColor} = \text{Color1} * \text{SourceBlend} + \text{Color2} * \text{DestinationBlend}$$

حيث سنحصل في حالتنا على:

$$\begin{aligned} \text{FinalColor} &= \text{Color1} * 1 + \text{Color2} * 1 \\ \text{FinalColor} &= \text{Color1} + \text{Color2} \end{aligned}$$

كما قلت لك، الموضوع ببساطة هو عملية جمع. من الضروري جداً أن تقوم بإيقاف مزج ألفا بعد أن تنتهي من رسم الرصاصات (و إلا، فإنه في الإطار التالي سوف يتم رسم كل المشهد باستخدام مزج ألفا الإضافي!)، لذا قم بإضافة السطر التالي في نهاية الدالة:

```
device.RenderState.AlphaBlendEnable = false;
```

تشغيل هذا الكود يجب أن يعطي نتيجة أفضل بكثير للرصاص.

لدينا رصاصات تبدو بشكل جميل، ولكن ما هو الشيء الجيد في الرصاص إذا لم يكن باستطاعته تدمير الأهداف؟! لحسن الحظ، لدينا دالة تفحص تصادم عنصرين معا. لذا توجه إلى الدالة `UpdateBulletPositions`، و قم بإضافة الكود التالي في نهاية التكرار "for-loop" الذي يقوم بالمرور على عناصر القائمة `bulletList`:

```
BoundingBox bulletSphere = new BoundingBox(currentBullet.position,
0.05f);
CollisionType colType = CheckCollision(bulletSphere);
if (colType != CollisionType.None)
{
    bulletList.RemoveAt(i);
    i--;

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

يبدأ هذا الكود بإنشاء كائن من النوع كرة حدود "Bounding Sphere" لكل رصاصة. بعدها، نقوم بتمرير هذه الكرة إلى الدالة `CheckCollision`، حيث تقوم بفحص وجود تصادم بين هذه الكرة و المباني أو الأهداف في المدينة.

إذا كان هنالك تصادم، يتم حذف الرصاصة من القائمة `bulletList`، حيث نتأكد أن ال `Xna` لن تقوم برسمها مرة أخرى. و الأهم، إذا كان هنالك تصادم بين الرصاصة و أحد الأهداف، نقوم بزيادة سرعة اللعبة قليلا. الدالة `CheckCollision` تهتم بشكل تلقائي بعملية حذف الهدف.



- هذا كل شيء! بالنسبة للمشاهد الثلاثي الأبعاد، لدينا الطائرة و بعض الأهداف التي بإمكاننا الإطلاق عليها و تدميرها، لقد بدأ العمل يظهر كلعبة 😊. دعنا الآن نقوم بعمل شيء ما بخصوص الخلفية.

حاول حل التمرين التالي:

- قم بحذف السطر الذي يقوم بإيقاف مزج ألفا، و لاحظ كيف سوف تكون النتيجة!!

كود المشروع:

```
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;
        Model xwingModel;
        Model targetModel;

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

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



```

        {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,
buildingHeights[currentbuilding], -z), new Vector3(0, 0, 1), new Vector2((currentbuilding * 2) /
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);

    BoundingBox xwingSpere = new BoundingBox(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()
{
    Vector3 campos = new Vector3(0, 0.1f, 0.6f);
    campos = Vector3.Transform(campos, Matrix.CreateFromQuaternion(xwingRotation));
    campos += xwingPosition;

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

    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);

    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.I
ndex] * 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;
    }
}
}
}

```