الدرس الثامن

أهلا بكم في الدرس الثامن من سلسلة دروس تعلم 3D Xna (السلسلة الثانية)، في هذا الدرس سوف نقوم بتحريك الطائرة بناء على مدخلات المستخدم.

***ملاحظة**: ما سنقوم بعمله يسمى "Flight kinematics"، أو "kinematics" لوحدها، حيث يعبر هذا المصطلح عن علم الحركة المجردة!

الآن لدينا الكاميرا تتموضع فوق الطائرة، حان الوقت لجعل الطائرة تطير. سوف يتم عمل ذلك من خلال الدالة Update.

في الطائرة الحقيقية، عندما تتحرك إلى اليسار، يتم تعديل لوحات التوجيه "flaps" بحيث تدور الطائرة حول محورها المتجه إلى الأمام. بعدها، عندما تقوم بسحب عصا التحكم بإتجاهك، سوف تكون قد إرتفعت مقدمة الطائرة، بكمية محددة مرتبطة بدوران معين حول المحور الصحيح.

الدالة ProcessKeyboard التالية سوف تقوم بقراءة مدخلات لوحة المفاتيح و بناء عليها تقوم بتغيير قيم الزوايا:

```
 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;
 }
```

كل ما يفعله هذا الكود، هو تخزين القيم التي تلزمنا لتدوير الطائرة حول محورها المتجه للأمام عندما تضغط على زر السهم الأيمن أو الأيسر، كما ناقشنا في الأعلى. الآن، قيمة الدوران هذه يجب إن يتم إضافتها إلى الدوران الحالي للطائرة. أولا سوف نقوم بإنشاء رباعي لتمثيل الدوران الجديد، و بعدها إضافته إلى الدوران الحالي، ويتم تخزين النتيجة النهائية في المتغيرxwingRotation، في الدالة ProcessKeyboard أيضا:

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

قمنا بإنشاء الرباعي الذي يقابل الدوران حول المحور (0,0,-1) المتجه للأمام. بعدها، هذا الدوران يتم إضافته إلى الدوران الفعلي للطائرة.

بإمكانك ملاحظة أن هذه الدالة تستقبل الكائن GameTime، لذا كمية الدوران تعتمد على كمية الوقت التي مضت من اللعبة. نتيجة ذلك هي أن كمية الدوران ستكون نفسها على الأجهزة السريعة و البطيئة.

قم بإستدعاء هذه الدالة من داخل الدالة Update:

```
 ProcessKeyboard(gameTime);
```

من المؤكد أنك لاحظت أيضا أننا إستخدمنا المتغير gameSpeed، الذي يتم زيادة قيمته إذا كان اللاعب يلعب بشكل جيد، و سوف يتم تقليلها عندما يصطدم الاعب بشيئ. ما زلنا بحاجة إلى تعريف المتغير في بداية الكود:

```
 float gameSpeed = 1.0f;
```

الآن قم بتشغيل البرنامج! عندما تضغط على الزر الأيسر أو الأيمن في لوحة المفاتيح، سوف تدور الطائرة حول محورها المتجه للأمام!

دعنا نحاول دفع مقدمة الطائرة إلى الأعلى. لذا قم بإضافة الكود التالي إلى الدالة ProcessKeyboard:

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

بالطبع، سوف نحتاج لتضمين هذه القيم الجديدة إلى الدوران الإضافي:

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

الضغط على الزر الأسفل سوف يؤدي إلى رفع مقدمة الطائرة إلى الأعلى. عند تشغيل هذا الكود يجب أن يكون بإمكانك أن تدير الطائرة في أي إتجاه و زاوية تريد!

ليست مرحة كفاية☹، لأن الطائرة عمليا لم تتحرك بعد. لذا دعنا نقوم بإنشاء دالة أخرى، لنسمها MoveForward، حيث ستقوم بتحديث موقع الطائرة بناء على الدوران الحالي للطائرة:

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

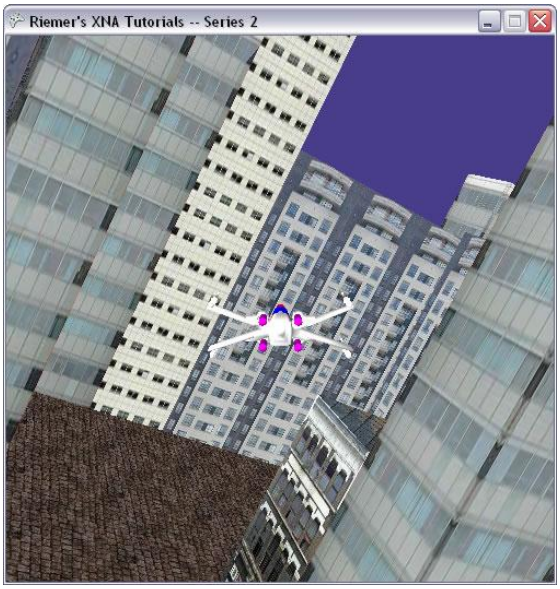
أولا، قمنا بحساب إتجاه الحركة بناء على الزاوية. حيث قمنا بذلك من خلال أخذ المتجه (0,0,-1) المتجه للأمام، و بعدها قمنا بتحويله بإستخدام دوران الطائرة الموجود لدينا. بهذه الطريقة، حصلنا على متجه يمثل المحور الأمامي الحالي للطائرة.

بعدها قمنا بضرب هذا المتجه بمتغير السرعة، و من ثم إضافة النتيجة إلى الموقع الحالي. بما أن متغير الموقع تم تمريره بطريقة المناداة بالمرجع (لم أجد ترجمة أفضل!☺) "Call by Reference": إذن هذه التغييرات سوف يتم تخزينها في الكود المستدعي.

طبعا نحن بحاجة لإستدعاء هذه الدالة من داخل الدالة Update. كما نحتاج إلى تمرير كمية الحركة المطلوبة:

```
 float moveSpeed = gameTime.ElapsedGameTime.Milliseconds / 500.0f *
gameSpeed;
 MoveForward(ref xwingPosition, xwingRotation, moveSpeed);
```

و بهذا نكون قد حصلنا على المطلوب☺! عندما تقوم بتشغيل الكود، يجب أن يكون لديك الإمكانية للإنطلاق بالطائرة في أرجاء المدينة الثلاثية الأبعاد، كما هو ظاهر في الصورة التالية:



تمرين هذا الدرس هو تمرين على سياقة الطائرة☺، حاول الطيران حول المدينة، قم بعمل بعض الدورات، لاحظ أثناء ذلك إضاءة المباني و الطائرة أثناء و كيفية تغييرها.

الآن بعد أن أصبح بإمكاننا الطيران، ربما أصبح الوقت ملائما لعمل بعض إكتشاف التصادمات، لأنه ليس من الطبيعي الطيران من خلال الجدران.

الكود :

```
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
    {
        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;
        Model xwingModel;

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

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

        VertexBuffer cityVertexBuffer;
        VertexDeclaration texturedVertexDeclaration;

        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();

            base.Initialize();
        }

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


           effect = Content.Load<Effect> ("effects");
           sceneryTexture = Content.Load<Texture2D> ("texturemap");
           xwingModel = LoadModel("xwing");

           SetUpVertices();
           SetUpCamera();
       }

       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 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},
                {1,0,0,0,0,0,0,0,0,0,0,0,1,0,1},
                {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},
                {1,0,0,0,0,1,0,0,0,0,0,0,0,0,1},
                {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 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);
             UpdateCamera();

             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;
        }

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


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

            DrawCity();
            DrawModel();

            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();
            }
        }
    }
 }
```