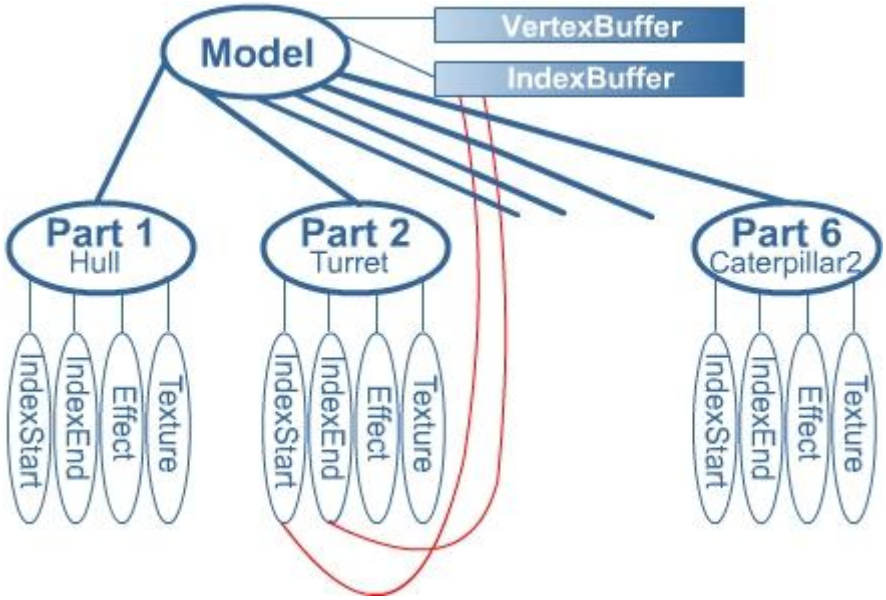
الدرس الخامس

أهلا بكم في الدرس الخامس من سلسلة دروس تعلم 3D Xna (السلسلة الثانية)، في هذا الدرس سوف نقوم بإستيراد نموذج الطائرة من ملف.

المدينة جميلة، ولكن كيف يفترض بنا أن نرسم الطائرة في المشهد؟ هل يجب علينا أن نقوم ببرمجة كل رأس فيها؟ لحسن الحظ، بإمكاننا أن نقوم بتحميل النماذج "models" التي تم تخزينها في ملف. بإختصار: النموذج هو تركيب "structure" يقوم بتخزين كل المعلومات الازمة لرسم عنصر معين. حيث تحتوي على إحداثيات الرؤوس، بالإضافة إلى معلومات العاموديات "Normal"، معلومات الألوان، و إحداثيات الخامات إن لزم. حيث تقوم بتخزين بياناتها في ذاكرة رؤوس"Vertexbuffers" و ذاكرة فهارس "indexbuffers"، حيث يمكننا ببساطة أن نقوم بتحميلهم من الملف.

ولكن هذا ليس كل شيئ. النموذج قد يحتوي على أجزاء متعددة، لوصف عنصر واحد. تخيل أننا نريد تحميل نموذج دبابة من ملف. هذا النموذج قد يحتوي على جزء واحد يصف جسم "Hull" الدبابة، و جزء آخر يقوم بوصف برج الدبابة "Turret" ، و جزء آخر للباب و جزئين آخرين لكل من أحزمة الدبابة "Caterpillars" (الجزء الذي تمشي عليه الدبابة). النموذج يقوم بتخزين كل الرؤوس في ذاكرة رؤوس واحدة كبيرة، وكل جزء من النموذج يحتوي على الفهارس التي تمثله، حيث تشير هذه الفهارس إلى الرؤوس في ذاكرة الرؤوس الكبيرة. النموذج يخزن كل ذواكر الفهارس لكل أجزاء النموذج بعد بعضها البعض في ذاكرة فهارس كبيرة. بإمكانك أن ترى ذاكرة الرؤوس و الفهارس الكبيرة هاتين في الجزء العلوي الأيمن من الرسم فيما يلي:



كل جزء من النموذج ببساطة يحتوي على المعلومات التي تدلنا أي جزء من ذاكرة الفهارس الكبيرة تنتمي إلى الجزء الحالي.

كل جزء أيضا يحتوي على التاثيرات الازمة، و صور الخامات (إن تم إستخدامها) التي يجب إستخدامها لهذا الجزء من النموذج. بهذه الطريقة، ستكون لديك إمكانية جعل بطاقة الرسوميات تقوم برسم برج الدبابة بمعدن لامع مع إنعكاس، ورسم أحزمة الدبابة بتأثير آخر بإستخدام خامة مثلا.

يكفي نظريات ☺، دعنا نرى كيف نطبق ذلك بشكل عملي. سوف نقوم بتحميل طائرة فضائية إلى المشهد، يمكنك تنزيل نموذج الطائرة من هنا(أو من الملفات المرفقة في الدرس الأول). قم بتنزيل الملف و بعدها قم بإستيراده إلى مجلد ال Content في ال Solution Explorer، بنفس الطريقة التي قمنا بها مع الصور السابقة! يجب أن يكون لديك الآن عنصر بإسم 'xwing'.

بعدها، سوف نقوم بإسناد النموذج إلى متغير من النوع Model ، لذا قم بإضافة السطر التالي إلى أعلى الكود:

```
Model xwingModel;
```

بعدها، سوف نقوم بإضافة دالة صغيرة، بإسم LoadModel، تقوم بتحميل النموذج في المتغير:

```
private Model LoadModel(string assetName)
{
    Model newModel = Content.Load<Model> (assetName);
    return newModel;
}
```

الدالة تأخذ إسم هذا النموذج. وتقوم بتحميل كل بيانات النموذج من الملف في كائن جديد من النوع Model، بعدها تقوم الدالة بإرجاع هذا الكائن المملوء. حتى الآن، هذه الدالة تقوم بتحميل النموذج تماما بنفس الطريقة التي إستخدمناها في تحميل الخامة.

جميل، ولكن حتى اللحظة، النموذج يحتوي على التأثيرات الإفتراضية فقط. هذا يعني أننا سوف نحتاج أن ننسخ التأثيرات الخاصة بنا في كل جزء من النموذج!

هذا ببساطة يتم من خلال توسيع الدالة LoadModel كالتالي:

```
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;
}
```

لكل جزء من الشبكة "mesh" في النموذج، قمنا بوضع نسخة من التأثير الخاص بنا في الجزء. الآن أصبح النموذج محمل و مجهز بشكل كامل!

بعدها، إستدعي هذه الدالة من داخل الدالة LoadContent من أجل تحميل النموذج في المتغير:

```
xwingModel = LoadModel("xwing");
```

رأينا فيما سبق الكثير من النظريات، ولكن لحسن الحظ ذلك لم ينتج عنه الكثير من الكود الصعب. بما أننا قمنا بتحميل النموذج في المتغير xwingModel، بإمكاننا أن ننتقل إلى الكود الذي يقوم برسم هذا النموذج.

للحفاظ على التنظيم، سنقوم بإنشاء دالة منفصلة بإسم DrawModel:

```
 private void DrawModel()
 {
     Matrix worldMatrix = Matrix.CreateScale(0.0005f, 0.0005f, 0.0005f)
* Matrix.CreateRotationY(MathHelper.Pi) * Matrix.CreateTranslation(new
Vector3(19, 12, -5));

     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(projection
Matrix);
         }
         mesh.Draw();
     }
 }
```

هذا الكود يحتوي على بعض الإختلافات المهمة عن الكود الذي يقوم برسم المدينة. أولا، نحتاج إلى مصفوفة عالم ليست محايدة.وذلك مطلوب، لأنه عندما تريد أن تقوم برسم الطائرة بشكل مباشر، سوف تواجه مشكلتين:

- سوف تكون الطائرة كبيرة جدا.
- سوف يتم تدوير الطائرة 180 درجة، حيث سوف تكون مقدمة الطائرة بإتجاه Z الموجب، بينما ال Xna تأخذ إتجاه Z السالب كإتجاه أمامي.

هذه المشاكل يمكن التغلب عليها بسهولة، من خلال رسم الطائرة بإستخدام مصفوفة عالم تحتوي على عملية تصغير بالإضافة إلى عملية تدوير. ألق نظرة على تعريف هذه المصفوفة، تم تصغير النموذج بمعامل تصغير يساوي 0.0005، مما يجعلها أصغير 2000 مره! إضافة إلى ذلك، تم تدوير النموذج بقيمة Pi دائري (قيمة Pi دائري تساوي 180 درجة). أخيرا، تم تحريك النموذج إلى الموقع (19,12,-5) في مدينتنا ثلاثية الأبعاد.

الإختلاف الثاني هو أنك تحتاج لأخذ مصفوفات العظام "Bones" الخاصة بالنموذج بالحسبان، بحيث يتم رسم الأجزاء المختلفة من النموذج في أماكنها الصحيحة.

المتبقي من الدالة هو عملية إعداد التأثير من أجل رسم النموذج. بما أن الرؤوس الخاصة بالطائرة تحتوي فقط على معلومات الألوان، سوف نقوم بإستخدام التقنية "Colored" من أجل رسمها.

لا تنسى إستدعاء هذه الدالة من داخل الدالة Draw:

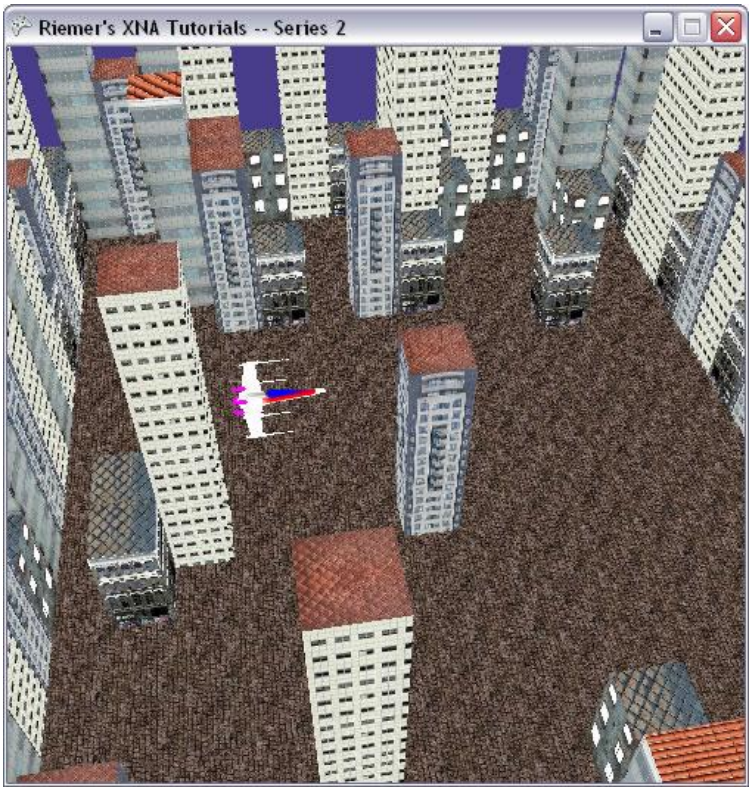
```
 protected override void Draw(GameTime gameTime)
 {
     device.Clear(ClearOptions.Target | ClearOptions.DepthBuffer,
```

```
Color.DarkSlateBlue, 1.0f, 0);

      DrawCity();
      DrawModel();

      base.Draw(gameTime);
 }
```

يجب أن تحصل على النتيجة النهائية التالية:



حاول حل التمارين التالية:

- أنقل الطائرة إلى مكان مختلف.
- قم بتغيير الإستدارة و التحجيم قبل رسم الطائرة.

الكود حتى الآن:

```
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;
        Matrix viewMatrix;
        Matrix projectionMatrix;

        Texture2D sceneryTexture;
        Model xwingModel;

        int[,] floorPlan;

        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();

            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},
                 {1,0,0,1,1,0,0,0,1,1,0,0,0,1,1},
                 {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},
                 {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
                 {1,0,1,1,0,0,0,1,0,0,0,0,0,0,1},
```

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

                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)
        {
            base.Update(gameTime);
        }

        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.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.CreateTranslation(new Vector3(19, 12, -5));

            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);
                }
                mesh.Draw();
            }
        }
    }
}
```