الدرس الثالث عشر

أهلا بكم في الدرس الثالث عشر من سلسلة دروس تعلم 3D Xna (السلسلة الثانية)، في هذا الدرس سوف نقوم بعمل شيئ حيال الخلفية الزرقاء المحيطة بالمدينة. سوف نقوم بإستعمال مكعب السماء "Sky Box" لذلك.

ما هو بالضبط صندوق السماء "Sky Box"؟ في الحقيقة، هو ليس أكثر من مكعب يحتوي على 6 صور لمناظر طبيعية، يتم رسمها حول الكاميرا. هذا يعطي اللاعب الإحساس بأنه موجود في بيئة خارجية، بينما في الحقيقة، هو محصور داخل مكعب!

بإستخدام الدوال التي قمنا بإنشاءها حتى الآن، بإمكاننا أن نتوقع أن يكون عمل ذلك سهل جدا، حيث أن مكعب السماء هو ليس أكثر من شبكه "Mesh". فقط هذه المرة، نحن بحاجة إلى تحميل بعض ملفات الخامات لكي نربطها مع الشبكة.

بإمكانك تنزيل الشبكة بشكل منفصل من هذا الرابط، أيضا بإمكانك الحصول على ملفات الخامات من هذا الرابط، إذا واجهتك مشكلة في فتح الملفات بإمكانك إيجاد الملفات في هذا الرابط. (بإمكانك الحصول على الملفات أيضا من المرفقات في الدرس الأول). ملف الشبكة بحد ذاته مصمم بشكل مدعوم من قبل حزمة تطوير ال DirectX "DirectX SDK". بإمكانك إيجاد العديد من ملفات خامات مكعبات السماء في الإنترنت.بعد تنزيل الملفات قم بإضافة الملف Skybox.x إلى المشروع، أيضا قم بإضافة جميع الصور الخاصة بالخامات.

في هذا الوقت، نحن بحاجة إلى متغيرين إثنين: الأول لتخزين النموذج "model"، و الثاني لتخزين الخامات (لأن كل جزء من الخامة سيكون له خامة مختلفة). لذا قم بإنشاء هذه المتغيرات في أعلى الكود:

```
Texture2D[] skyboxTextures;
Model skyboxModel;
```

في الوضع الطبيعي، سوف نستخدم الدالة LoadModel لتحميل نموذج مكعب السماء، ولكننا لا نستطيع عمل ذلك بما أن هذه الدالة لا تقوم بتحميل الخامات. إذن سوف نقوم بحل المشكلة من خلال كتابة نسخة أخرى من هذه الدالة "Overloading": حيث سوف نقوم بكتابة نسخة جديده من هذه الدالة بنفس الإسم، ولكن بوسائط "Arguments" مختلفة!

```
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;
}
```

لاحظ أن توقيع "signature" هذه الدالة يختلف عن الدالة LoadModel الأولى، حيث في هذه الدالة نقوم بتمرير متغيرين إثنين كوسائط.

السطر الأول و الأسطر الثلاثة الأخيرة مأخوذه بشكل مباشرة من الدالة LoadModel الأولى. فقط الجزء الأوسط هو الجديد.

هل تذكر أن كل جزء من النموذج بإمكانه تخزين تأثير "effect" مختلف عن الأجزاء الأخرى؟ هذه التأثيرات أيضا تحتوي على أسماء الخامات، كل خامة مرتبطه بالجزء الخاص بها من النموذج. لذا نقوم بالمرور على كل تأثير في النموذج، و نقوم بتخزين كل الخامات في مصفوفة الخامات!

قم بإستدعاء هذه الدالة من داخل الدالة LoadContent:

```
skyboxModel = LoadModel("skybox", out skyboxTextures);
```

هذا كل شيئ لتحميل مكعب السماء. كل ما علينا عمله الآن، هو رسم هذا المكعب. مالذي يجعل مكعب السماء شيئ خاص؟ عندما تتحرك الطائرة، يجب أن تتحرك المدينة بالنسبة إليها. هذا ما لا ينطبق على مكعب السماء، حيث يجب أن يبقى على مسافة ثابته من الطائرة. سؤدي هذا إلى جعل مكعب السماء يبدو و كأنه بعيد إلى المالانهاية! لذا عندما تتحرك الطائرة، سنقوم بتحريك مكعب السماء معها، إذن سوف تبقى الطائرة في وسط المكعب.

قم بإضافة الكود التالي إلى نهاية الكود:

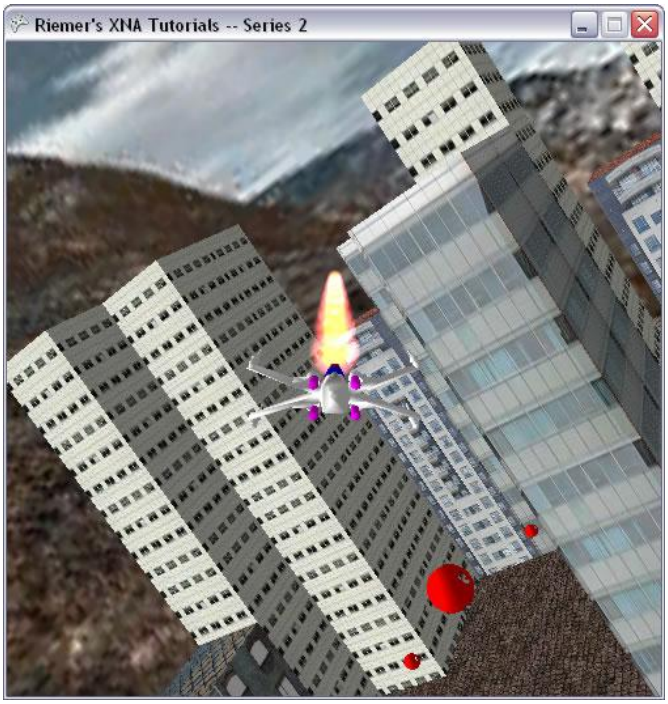
```
private void DrawSkybox()
{
    device.SamplerStates[0].AddressU = TextureAddressMode.Clamp;
    device.SamplerStates[0].AddressV = TextureAddressMode.Clamp;

    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(projection
Matrix);
            currentEffect.Parameters["xTexture"].SetValue(skyboxTexture
s[i++]);
        }
        mesh.Draw();
    }
    device.RenderState.DepthBufferWriteEnable = true;
}
```

هذه الدالة مشروحة بالتفصيل في كتاب Game Programming Recipes 8-2، ولكن بإختصار. قمنا بإعطاء القيمة Clamp لل Texture Addressing mode، و ذلك للتخلص من الحواف التي سوف تظهر على حواف المكعب. نريد أيضا إيقاف ال ذاكرة العمق "ZBuffer" بحيث لن يتوجب علينا تحديد حجم للمكعب. يتم رسم المكعب حول الطائرة بما أن موقعها يستخدم لإنشاء مصفوفة العالم للمكعب.

نحن بحاجة لإستدعاء هذه الدالة في أول سطر في الدالة ()Draw:

```
DrawSkybox();
```

هذا كل شيئ! تشغيل هذا الكود يجب أن يعطيك نتيجة مثل الظاهرة في الصورة:



الكود حتى اللحظة:

```
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;

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

            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()
        {
            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);


             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.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;
            }
        }

        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;
        }
    }
}
```