

الدرس التاسع

أهلاً بكم في الدرس التاسع من سلسلة دروس تعلم 3D Xna (السلسلة الثانية)، في هذا الدرس سوف نقوم بإكتشاف التصادمات.

في هذا الدرس سوف نقوم بإكتشاف تصادم الطائرة بأحد عناصر المشهد. سوف نقوم بتعريف ثلاث أنواع من التصادمات:

- المباني "Building": عندما يصطدم اللاعب بأحد المباني في المدينة الثلاثية الأبعاد.
- الحدود "Boundary": عندما تخرج الطائرة خارج المدينة، أسفل الأرضية أو عالياً في السماء.
- الهدف "Target": عندما تصطدم بأحد الأهداف، سوف نقوم بإضافتها في الدرس القادم بإذن الله.

لذا قم في البداية بإضافة التعداد التالي في أعلى الكود، على سبيل المثال فوق تعريف المتغيرات:

```
enum CollisionType { None, Building, Boundary, Target }
```

لإكتشاف التصادمات، سوف نعتبر الطائرة ككرة، من باب التبسيط وهي كافية لهذا الهدف.

باستخدامنا الكرة لتمثيل الطائرة، سوف يكون بإمكاننا استخدام خدمة إكتشاف التصادمات المقدمة في الـ Xna. أيضاً سوف نقوم لكل مبنى في المدينة، بإنشاء كائن من النوع مكعب حدود "BoundingBox". بعدها، إذا قمنا بإنشاء كائن من النوع كرة حدود "BoundingSphere" للطائرة، سوف يكون بإمكاننا أن نستخدم الدالة Contains لفحص فيما إذا ما كانا متصادمين أم لا!

لذا أولاً، نحن بحاجة لإنشاء الـ BoundingBox لكل مبنى في المدينة، وبعدها إضافة كل هذه المكعبات معا في مصفوفة واحدة كبيره. لذا إبدأ بتعريف هذه المصفوفة في أعلى الكود:

```
BoundingBox[] buildingBoundingBoxes;  
BoundingBox completeCityBox;
```

المتغير completeCityBox سوف يستخدم لفحص إذا ما كان اللاعب قد خرج من حدود المدينة.

بعدها، قم بإضافة الدالة التالية إلى الكود:

```
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();
}

```

في البداية قمنا بإستخراج عرض و طول المدينة، بعدها قمنا بإنشاء مصفوفة قادرة على تخزين المكعبات.

ثم، قمنا بالمرور على المصفوفة floorPlan و قمنا بفحص نوع المبنى لكل مربع في المدينة. إذا كان هنالك مبنى، نقوم بإنشاء مصفوفة مكونة من متجهين ثلاثي الأبعاد "Vector3": الأول يمثل النقطة السفلى الخلفية اليسرى من المبنى، و الثاني يمثل النقطة العليا الأمامية اليمنى من المبنى. هاتين النقطتان تمثلان المكعب! لذا سنطلب من ال Xna أن تقوم ببناء مكعب الحدود بناء على هاتين النقطتين بإستخدام الدالة BoundingBox.CreateFromPoints. في اللحظة التي نقوم فيها بإنشاء المكعب للمبنى الحالي، نقوم بإضافته إلى القائمة.

في النهاية، قمنا بتحويل القائمة إلى مصفوفة و تخزين هذه المصفوفة في المتغير buildingBoundingBoxes.

ما زلنا بحاجة لإنشاء مكعب الحدود الذي يمثل كامل المدينة، ما يتيح لنا معرفة فيما إذا كانت الطائرة قد خرجت من المدينة. يتم عمل ذلك بإستخدام نفس المنهج بالضبط: نقوم بإنشاء مصفوفة مكونة من نقطتين. الأولى تمثل الزاوية السفلى-الخلفية-اليسرى من المدينة، و النقطة الأخرى تمثل الزاوية العليا-الأمامية-اليمنى من المدينة. من هاتين النقطتين يتم إنشاء المكعب الذي نقوم بتخزينه في المتغير completeCityBox.

قم بإضافة الكود التالي في نهاية الدالة :SetUpBoundingBoxes

```

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

```

نحن بحاجة لتعريف هذه المكعبات مره واحدة فقط، لذا قم بإستدعاء هذه الدالة من داخل الدالة :Initialize

```
SetUpBoundingBoxes ();
```

بعد الإنتهاء من تعريف المكعبات، بإستطاعتنا أن نقوم بإضافة الدالة :CheckCollision

```

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;

    return CollisionType.None;
}
```

هذه الدالة تستقبل كائن من النوع BoundingSphere من الكود المستدعي. الكرة المستقبل هي الكرة التي تمثل الطائرة.

أولاً، نقوم بفحص وجود تصادم بين الطائرة و أحد المكعبات التي تمثل البنايات من خلال إستدعاء الدالة Contains. إذا كانت النتيجة ليست منفصلين "Not Disjoint" ، هذا يعني أن هناك تصادم، لذا نقوم بإرجاع CollisionType.Building.

إذا لم تكن الطائرة متصادمة مع المباني، نقوم بعدها بفحص الدالة Contains بين كرة الطائرة و المكعب المحيط بالمدينة. هذه المرة، يجب أن يحتوي المكعب كامل الطائرة. إذا لم يكن كذلك، ستكون طائرتنا منخفضة جداً، أو مرتفعة جداً، أو خارج حدود المدينة. ما يعني أن علينا إرجاع القيمة CollisionType.Boundary.

إذا لم يكن هنالك تصادم بين الطائرة و المباني، و إذا كانت الطائرة في داخل مكعب المدينة، إذن كل شيء بخير و نقوم بإرجاع القيمة CollisionType.None.

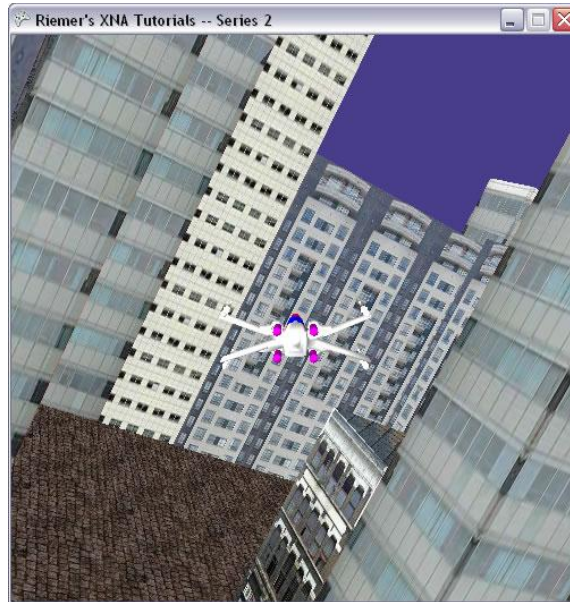
الآن كل ما يلزمنا عمله هو إنشاء كره حدود "BoundingSphere" حول الطائرة، و تمريرها إلى الدالة CheckCollision! لذا قم بإضافة الكود التالي إلى الدالة Update:

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

في السطر الأول قمنا بإنشاء BoundingSphere، مركزها في موقع الطائرة، و قمنا بإعطائها نصف القطر 0.04f، وهي قيمة مقارنة لحجم الطائرة تقريباً.

بعدها، قمنا بتمرير هذه الكرة إلى الدالة CheckCollision. إذا لم يكن نوع التصادم المرجع هو لا شيء "None"، نقوم بإعادة ضبط (إعطائهم القيم الافتراضية) الموقع و الدوران للطائرة و الكاميرا، و نقوم بتقليل سرعة اللعبة لأنه من الواضح أن الأشياء أصبحت سريعة أمام اللاعب.

الآن قم بتشغيل الكود، و قم بالإصطدام بأحد المباني، أو قم بالخروج من حدود المدينة، لتتأكد من نجاح الكود الجديد!



حسنًا هذه الصورة لا تختلف كثيرًا عن الدرس السابق... لكن في هذه المرة يوجد لدينا إكتشاف تصادمات!
ما فائدة محاكاة الطائرة بدون أهداف نقوم بالإطلاق عليها؟ دعنا نقوم بإضافة هذه الأهداف في الدرس القادم بإذن الله.

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
    {
        enum CollisionType { None, Building, Boundary, Target }

        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;
    }
}
```

```

        BoundingBox[] buildingBoundingBoxes;
        BoundingBox completeCityBox;

        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();

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

            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, 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();

        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;
    }

    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;

        return CollisionType.None;
    }
}

```

```

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();
    }
}
}

```