

الدرس السابع

أهلاً بكم في الدرس السابع من سلسلة دروس تعلم 3D Xna (السلسلة الثانية)، في هذا الدرس سوف نقوم بتحديد موضع الكاميرا بشكل تلقائي (خلف الطائرة)، كما سنقوم باستخدام الرباعيات "Quaternions" من أجل الإستدارة.

تخيل أننا نريد أن نجعل طائرتنا تطير في أنحاء المدينة الثلاثية الأبعاد. حينها سيكون أفضل مكان لموضع الكاميرا فيه هو خلف الطائرة، بغض النظر عن المكان و الدوران الذين تحملهما الكاميرا.

أولاً سوف نقوم بتجهيز متغيرين جديدين، `xwingPosition` و `xwingRotation`. هذه المتغيرات سوف تقوم بتحديد موضع الكاميرا، حيث يجب أن يتغير موضع الكاميرا بنفس طريقة تغير موقع و دوران الطائرة.

بإمكاننا تخزين دوران الطائرة كقيم دوران منفصلة حول محاور X ، Y و Z .

بكل الأحوال، في التطبيق العملي هذا معقد جداً. المشكلة بشكل أساسي هي كما في المصفوفات: أن الدوران حول محور X متبوعاً بدوران حول محور Y لا يساوي الدوران حول محور Y متبوعاً بالدوران حول محور X .

بالإضافة إلى ذلك، فإن إستقاق الدوارن من ثلاث قيم منفصلة يحتاج بعض المعادلات المثلثية (الجيب، جيب التمام، الظل...) (" $\sin, \cos, \tan...$ "). في بعض الأحيان، تقوم هذه الدوال بقلب الإشارات، مما يؤدي إلى جعل الطائرة تطير إلى الخلف ☹ !!

لحل كلا من المشكلتين، سوف نقوم باستخدام الرباعيات "Quaternions" من أجل تخزين دوران الطائرة. الرباعيات شبيهة بالمصفوفة، لكنها تقوم بتخزين الدوران فقط. لا تدع إسم الرباعيات يخيفك: لأنه و بالرغم من صعوبة فهم المنطق الرياضي، فإن استخدام الرباعيات سهل جداً. إذن إبدأ بتعريف المتغيرين التاليين في أعلى الكود:

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

كما ترى، قمنا بتجهيز الموقع الأولي و الدوران للطائرة. في الدرس القادم سوف نرى كيف سوف نقوم بتعديل الرباعي بناء على مدخلات المستخدم، حتى الآن إستخدمنا القيمة المحايدة "Identity".

لتحقيق هدف هذا الدرس، سوف نقوم برسم المدينة ثلاثية الأبعاد، بعدها رسم الطائرة في الموقع و الدوران الصحيحين، و بعدها سنقوم بتغيير موضع الكاميرا بحيث تكون خلف الطائرة مباشرة.

في داخل الدالة `DrawModel`، بإمكاننا فعلياً تعديل مصفوفة العالم الخاصة بالطائرة، بحيث يتم رسمها في المكان الصحيح، و بالدوران الصحيح:

```
Matrix worldMatrix = Matrix.CreateScale(0.0005f, 0.0005f, 0.0005f) *
Matrix.CreateRotationY(MathHelper.Pi) *
Matrix.CreateFromQuaternion(xwingRotation) *
Matrix.CreateTranslation(xwingPosition);
```

يبدو هذا السطر معقد، ولكنه سهل جداً: أولاً يتم تصغير النموذج بحيث يتلائم بشكل مناسب مع المشهد، بعدها يتم تدوير الطائرة 180 درجة بالإتجاه الصحيح للبدء به.

الجديد في هذا الدرس، أن الطائرة يتم تدويرها بناء على قيم الدوران المخزنة في الرباعي `xwingRotation`. بإمكانك هنا ملاحظة مدى سهولة اشتقاق مصفوفة الدوران المرتبطة برباعي. أخيراً، يتم نقل (تحريك) الشبكة "mesh" إلى الموقع الصحيح.

الآن لدينا الطائرة في المكان الصحيح و بالدوران الصحيح، سوف نقوم بتحديد موضع الكاميرا خلف الطائرة. حيث سوف نقوم بإنشاء دالة لعمل ذلك. حيث سوف تقوم هذه الدالة بإنشاء مصفوفة عرض `viewMatrix` جديدة و مصفوفة إسقاط `projectionMatrix` جديدة، تعتمدان على الموقع الصحيح و الدوران الصحيح للطائرة. دعنا نبدأ بالتالي:

```
private void UpdateCamera()
{
    Vector3 campos = new Vector3(0, 0.1f, 0.6f);
}
```

هذا المتجه سوف يحدد الموقع الذي نريد أن تكون فيه الكاميرا، بالنسبة لموقع الطائرة نحن نريد أن تكون الكاميرا للخلف قليلاً ($0.6+Z$) و للأعلى قليلاً ($0.1+Y$) من الطائرة، لذا المتجه السابق يحمل قيم في ال Y و ال Z فقط. هذا المتجه مازال بحاجة إلى عملية تحويل "Transformation": نحتاج إلى عملية نقل "Translation" إلى موقع الطائرة، و بعدها بحاجة إلى عملية تدوير بنفس قيم الدوران الخاصة بالطائرة، حيث تنتهي بشكل جميل خلف و فوق الطائرة.

دعنا أولاً نقوم بمعالجة الدوران:

```
campos = Vector3.Transform(campos,
Matrix.CreateFromQuaternion(xwingRotation));
```

الآن المتغير `campos` يحتوي على المتجه الذي سيكون دائماً خلف (و أعلى قليلاً) الطائرة بغض النظر عن دورانها، إذا كان موقع الطائرة، في الموقع $(0,0,0)$. لأن موقع الطائرة سوف يتغير بشكل ثابت، سوف نحتاج لأن ننقل المتجه `campos` إلى الموقع الذي تتواجد فيه الطائرة، و هو ما يقوم الكود التالي به:

```
campos += xwingPosition;
```

حسناً، إذن الآن لدينا المتجه الذي سوف يكون للخلف قليلاً و للأعلى قليلاً من طائرتنا، بغض النظر أن أي دوران وأو أي إزاحة لديها!

تذكر، عندما ننشئ مصفوفة عرض `viewMatrix`، لن نحتاج فقط إلى موقع الكاميرا و موقع الهدف (و هما ما نعرفهما في هذه اللحظة)، ولكننا نحتاج أيضاً إلى المتجه الذي يمثل المتجه الأعلى للطائرة "Up Vector". حيث يتم إيجاد نفس الطريقة: أولاً نبدأ بمتجه يشير إلى الأعلى، بعدها نقوم بتدوير المتجه بناء على مصفوفة الدوران الخاصة بالطائرة:

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

الآن لدينا كل شيء لإنشاء مصفوفة الكاميرا: الموقع، موقع الهدف و المتجه للأعلى، لذا دعنا ننشئ مصفوفة جديدة:

```
viewMatrix = Matrix.CreateLookAt(campos, xwingPosition, camup);
projectionMatrix =
Matrix.CreatePerspectiveFieldOfView(MathHelper.PiOver4,
device.Viewport.AspectRatio, 0.2f, 500.0f);
```

السطر الأول فقط يحتوي على المتجهات المنشئه حديثا، السطر الثاني بقي كما كان.

بعد إنهاء الدالة، دعنا نتأكد من إستدعائها من داخل الدالة Update:

```
protected override void Update(GameTime gameTime)
{
    UpdateCamera();

    base.Update(gameTime);
}
```

هذا كل شيء! عندما نقوم بتشغيل هذا الكود، سوف ترى أن موضع الكاميرا خلف و أعلى الطائرة، كما في الصورة التالية:



الآن لدينا الكود جاهز بحيث يقوم بتعديل الطائرة و الكاميرا بناء على المتغيرات `xwingPosition` و `xwingRotation`، إذن حان الوقت لقراءة مدخلات لوحة المفاتيح. بكل الأحوال، لدينا الآن كاميرا ديناميكية تلتحق بالطائرة، بغض النظر عن موقعها و دورانها.

حاول حل التمرين التالي، لممارسة ما قد تعلمته:

- غير موقع الطائرة، سوف تلاحظ أن الكاميرا قد لحقت بها!

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;

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

```

```
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)
    {
        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);
    }

    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();
}
}
}
```