الدرس الرابع

أهلا بكم في الدرس الرابع من سلسلة دروس تعلم ال 3D Xna السلسلة الأولى، في هذا الدرس سوف نتحدث عن الكاميرا و غن إحداثيات فضاء العالم.

في الدرس السابق قمنا برسم مثلث، بإستخدام الإحداثيات ال (المحولة) "Pretransformed" . هذه الإحداثيات تتيح لك تحديد موقع الرؤوس مباشرة على الشاشة. مع ذلك، سوف تحتاج عادة إلى إستخدام الإحداثيات الغير محولة"Untransformed" ، و هي ما تسمى بإحداثيات فضاء العالم "World Space Coordinates"، و هي ما نقوم بتحديده في الإحداثيات الثلاثية الأبعاد. حيث تتيح لك إنشاء المشهد كاملا بإستخدام إحداثيات ثلاثية أبعاد بسيطة، و أيضا، من المهم جدا أن يتم تحديد موقع الكاميرا التي سوف يشاهد منها المستخدم المشهد.

دعنا نبدأ بتحسين إحداثيات المثلث في فضاء العالم ثلاثي الأبعاد. قم بإستبدال الكود الموجود في الدالة SetUpVertices بالكود التالي:

```
 private void SetUpVertices()
 {
     vertices = new VertexPositionColor[3];

     vertices[0].Position = new Vector3(0f, 0f, 0f);
     vertices[0].Color = Color.Red;
     vertices[1].Position = new Vector3(10f, 10f, 0f);
     vertices[1].Color = Color.Yellow;
     vertices[2].Position = new Vector3(10f, 0f, -5f);
     vertices[2].Color = Color.Green;

     myVertexDeclaration = new VertexDeclaration(device,
VertexPositionColor.VertexElements);
 }
```

كما ترى، من هنا سوف نبدأ بإستخدام الإحداثي Z.

بما أننا لن نستخدم إحداثيات الشاشة المحولة (حيث X و Y يجب أن تكون بين ال [-1,1])، نحن بحاجة إلى إختيار تقنية أخرى من ملف التأثير. قمت بتسمية التقنية ب Colored، لذا قم بعمل التعديل التالي في الدالة Draw:

```
 effect.CurrentTechnique = effect.Techniques["Colored"];
```

قم بتشغيل الكود.

جميل جدا، قد إختفى المثلث مرة أخرى. لماذا؟ ببساطة، لأنك لم تخبر ال Xna بعد أن يقوم بوضع الكاميرا في العالم الثلاثي الأبعاد، و إلى أين تنظر!

من أجل تحديد موقع الكاميرا، سوف نحتاج لتعريف بعض **المصفوفات**. توقف!! مصفوفات؟!؟

أولا لمحة صغيرة عن المصفوفات. نحن نقوم بتعريف النقاط في العالم الثلاثي الأبعاد. لأن الشاشة لها بعدين فقط، فإنه من المنطقي فقط أن يتم تحويل النقاط الثلاثية الأبعاد إلى الفضاء الثنائي الأبعاد. يتم ذلك من خلال ضرب المواقع الثلاثية الأبعاد بمصفوفة. لذا بإختصار، يجب عليك أن تنظر إلى المصفوفة بإعتبارها عنصر رياضي يحتوي على تحويلات معينة. إذا قمت بضرب نقطة ثلاثية الأبعاد بمثل هذه المصفوفة، سوف تحصل على نقطة محولة. (إذا كنت تريد الحصول على مزيد من المعلومات عن المصفوفات بإمكانك الرجوع إلى درس [المصفوفات و برمجة الألعاب]).

لأن هناك العديد من الخصائص يجب تعريفها عند تحويل النقاط من العالم الثلاثي الأبعاد إلى الشاشة ثنائية الأبعاد، هذا التحويل ينقسم إلى خطوتين إثنتين، حيث نحصل على مصفوفتين. أولا قم بإضافة هذه المتغيرات إلى أعلى كود الصنف الخاص بنا:

```
Matrix viewMatrix;
Matrix projectionMatrix;
```

من أجل تجهيز المتغيرات، قم بإضافة الكود التالي إلى البرنامج:

```
private void SetUpCamera()
{
    viewMatrix = Matrix.CreateLookAt(new Vector3(0, 0, 50), new Vector3(0, 0,
0), new Vector3(0, 1, 0));
    projectionMatrix = Matrix.CreatePerspectiveFieldOfView(MathHelper.PiOver4,
device.Viewport.AspectRatio, 1.0f, 300.0f);
}
```

يبدو أن هذين السطرين معقدين شيئا ما، ولكن كل ما يفعلانه هو تعريف موقع و عدسة الكاميرا.

السطر الأول يقوم بإنشاء مصفوفة تحتوي على موقع و إتجاه الكاميرا، التي ننظر من خلالها إلى المشهد. الوسيط الأول يحدد موقع الكاميرا. حيث قمنا بوضعها في على بعد 50 وحدة من محور Z الموجب. الوسيط الثاني يحدد النقطة التي تنظر الكاميرا إليها. حيث جعلناها تنظر إلى النقطة (0,0,0) نقطة الأصل الثلاثية الأبعاد. في هذه اللحظة، قمنا بتعريف محور العرض للكاميرا الخاصة بنا، ولكن ما زال بإمكانا تدوير الكاميرا حول هذا المحور. لذا يجب علينا أن نحدد متجه لكي يمثل الإتجاه للأعلى.

السطر الثاني يقوم بإنشاء مصفوفة تخزن الطريقة التي تنظر بها الكاميرا إلى المشهد، بشكل أقرب إلى تعريف العدسة إذا أردت. الوسيط الأول يحدد زاوية الرؤية، 45 درجة في حالتنا. بعدها نقوم بتحديد ال "Aspect Ratio" و هي النسبة بين عرض الشاشة و طولها. في حالتنا حجم الشاشة 500*500 إذن النسبة هي 1، ولكن هذا قد يكون مختلف في حالات و أبعاد أخرى. الوسيطين الأخيرين يحددان مجال العرض. بحيث أن أي عنصر يبتعد عن الكاميرا مسافة أقل من 1f لن يتم عرضه و أي عنصر يبتعد عن الكاميرا مسافة أكبر من 300f سوف لن يتم عرضه أيضا. هذه المسافات تسمى سطوح الإقتصاص "Clipping Planes" القريبة و البعيدة، بما أن كل العناصر التي لن تكون بين هذين السطحين سوف يتم إقتصاصهما (= لن يتم عرضهما).

الآن لدينا هذه المصفوفات، نحن بحاجة إلى تمريرها إلى التقنية "technique" الخاصة بنا، حيث سيتم دمجهما.

هذا يتم بإستخدام السطور التالية من الكود، التي يجب إضافتها إلى الدالة Draw:

```
effect.Parameters["xView"].SetValue(viewMatrix);
effect.Parameters["xProjection"].SetValue(projectionMatrix);
effect.Parameters["xWorld"].SetValue(Matrix.Identity);
```

بالرغم من أن أول سطرين موضحين في الأعلى، سوف يتم مناقشتهما بشكل مفصل اكثر في السلسلة الثالثة.

السطر الثالث يقوم بتحديد وسيط آخر سوف يتم مناقشته في الدرس القادم.

لا تنسى أن تستدعي هذه الدالة من داخل الدالة LoadContent:

```
SetUpCamera();
```

الآن قم بتشغيل الكود. يجب أن ترى الصورة الموجودة في الاسفل: مثلث، زاويته السفلى اليمنى تقع تقريبا اسفل الركن العلوي الأيمن للشاشة. ذلك بسبب أننا اعطينا الركن الأسفل الايمن قيمة سالبة للإحداثي Z، من أجل وضعها على بعد أكبر بقليل عن الكاميرا من الأركان الأخرى.

شيئ مهم يجب عليك ملاحظته قبل أن تبدأ التجربه: سوف ترى أن الركن الأخضر للمثلث هو على الجهة اليمنى من الشاشة، و هو ما يبدو طبيعي لأنك قمت بتعريفه على الجزء الموجب من المحور X. لذا، إذا أردت أن تضع الكاميرا على محور Z السالب:

```
 viewMatrix = Matrix.CreateLookAt(new Vector3(0, 0, -50), new Vector3(0, 0, 0),
new Vector3(0, 1, 0));
```

عليك أن تتوقع رؤية النقطة الخضراء في النصف الأيسر للشاشة. حاول التشغيل الآن.

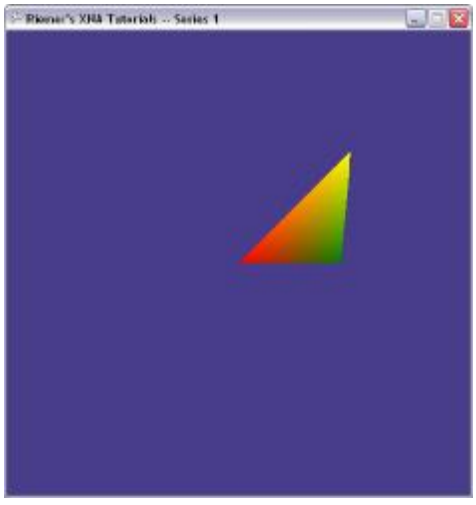
قد يكون هذا ما لا تتوقعه بالضبط. ال Xna ترسم فقط المثلثات المواجهة للكاميرا. ال Xna تحدد أن المثلثات المواجة للكاميرا يجب أن تكون معرفة بإتجاه مع عقارب الساعة بالنسبة للكاميرا. إذا قمت بوضع الكاميرا على محور Z السالب، سوف يتم إعتبار نقاط الزوايا للمثلث الخاص بنا -الموجودة في مصفوفة النقاط- معرفة بإتجاه عكس عقارب الساعة بالنسبة للكاميرا، وبهذا لن يتم رسمها!

الغربلة "Culling" تساعد بشكل كبير على زيادة الأداء، كما أنه بإمكانها أن تقلل عدد المثلثات التي سوف يتم رسمها.

بكل الأحوال، عند تصميم تطبيق معين، الأفضل أن يتم إيقاف الغربلة عن طريق وضع السطر التالي في دالة Draw:

```
device.RenderState.CullMode = CullMode.None;
```

هذا ببساطة سوف يؤدي إلى رسم جميع المثلثات، حتى المثلثات التي لا تواجهه الكاميرا. يجب عليك أن تعرف أن هذا لا يجب أن يتم أبدا في مشروع نهائي، لأن ذلك سوف يؤدي إلى إبطاء عملية الرسم، لأن كل المثلثات سوف يتم رسمها، حتى التي لا تواجهه الكاميرا! الآن قم بإرجاع الكاميرا إلى الجزء الموجب من محور Z.



بإمكانك أن تجرب التمرين التالي لممارسة ما تعلمته في هذا الدرس:

- قم بتجربة تغيير مواقع النقاط الثلاثية الأبعاد، و موقع الكاميرا.

كود هذا الدرس كاملا:

```
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 XNAtutorial
{
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;
        GraphicsDevice device;
        Effect effect;
        VertexPositionColor[] vertices;
        VertexDeclaration myVertexDeclaration;

        Matrix viewMatrix;
        Matrix projectionMatrix;

        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 1";

            base.Initialize();
        }

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

           effect = Content.Load<Effect> ("effects");
           SetUpVertices();

            SetUpCamera();
        }

        private void SetUpVertices()
        {
            vertices = new VertexPositionColor[3];

            vertices[0].Position = new Vector3(0f, 0f, 0f);
            vertices[0].Color = Color.Red;
            vertices[1].Position = new Vector3(10f, 10f, 0f);
            vertices[1].Color = Color.Yellow;
            vertices[2].Position = new Vector3(10f, 0f, -5f);
            vertices[2].Color = Color.Green;

            myVertexDeclaration = new VertexDeclaration(device,
```

```
VertexPositionColor.VertexElements);
        }

        private void SetUpCamera()
        {
            viewMatrix = Matrix.CreateLookAt(new Vector3(0, 0, 50), new Vector3(0, 0,
0), new Vector3(0, 1, 0));
            projectionMatrix = Matrix.CreatePerspectiveFieldOfView(MathHelper.PiOver4,
device.Viewport.AspectRatio, 1.0f, 300.0f);
        }

        protected override void UnloadContent()
        {
        }

        protected override void Update(GameTime gameTime)
        {
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
                this.Exit();

            base.Update(gameTime);
        }

        protected override void Draw(GameTime gameTime)
        {
            device.Clear(Color.DarkSlateBlue);

            device.RenderState.CullMode = CullMode.None;

            effect.CurrentTechnique = effect.Techniques["Colored"];
            effect.Parameters["xView"].SetValue(viewMatrix);
            effect.Parameters["xProjection"].SetValue(projectionMatrix);
            effect.Parameters["xWorld"].SetValue(Matrix.Identity);

            effect.Begin();
            foreach (EffectPass pass in effect.CurrentTechnique.Passes)
            {
                pass.Begin();

                device.VertexDeclaration = myVertexDeclaration;
                device.DrawUserPrimitives(PrimitiveType.TriangleList, vertices, 0, 1);

                pass.End();
            }
            effect.End();

            base.Draw(gameTime);
        }
    }
 }
```