الدرس الحادي عشر

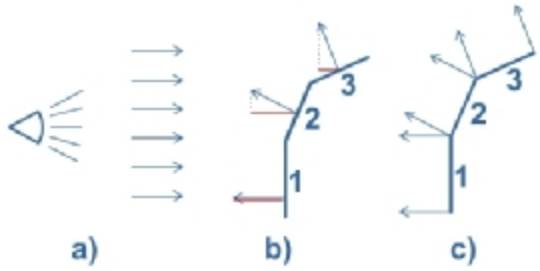
أهلا بكم في الدرس الحادي عشر من سلسلة دروس تعلم ال 3D Xna السلسلة الأولى، في هذا الدرس سوف نتحدث عن أساسيات الإضاءة.

حتى بعد إستخدام الألوان و الذاكرة الخلفية، ما زالت التضاريس ينقصها بعض من تفاصيل العمق خصوصا عندما تقوم بتلوينها بشكل مصمت بإستخدام ال FillMode. بإضافة بعض الإضاءة، سوف تبدو التضاريس بشكل أفضل بكثير.

في هذا الدرس سوف ترى تأثير الإضاءة على مثلثين بسيطين، لذا سوف يكون بإمكاننا الفهم بشكل أفضل عن أسلوب عمل الإضاءة في ال Xna. سوف نستخدم الكود من الدرس الرابع (إحداثيات فضاء العالم)، لذا قم بإسترجاع ذلك الكود الآن.

في هذا الدرس، سوف نقوم بإستخدام ضوء إتجاهي "Directional Light". تخيل ذلك كضوء الشمس: الضوء سوف يسير بإتجاه واحد محدد. من أجل حساب تأثير الضوء المصطدم بمثلث، تحتاج ال Xna إلى مدخل آخر:المتجه العمودي "Normal" في كل رأس.

لاحظ الشكل التالي:



إذا كان مصدر الضوء هو (a)، وقمت بتسليطه بإتجاه السطوح الثلاثة المعروضة، كيف سوف يعرف ال Xna أن السطح رقم 1 يجب أن يكون أكثر إشعاعا من السطح رقم 3؟ إذا نظرت إلى الخطوط الحمراء الرفيعة في الصورة (b) سوف تلاحظ أن طول هذه الخطوط بإمكانه التعبير بشكل جيد عن كمية الإضاءة المعكوسة (و بالتالي المرئية) على كل سطح. لذا كيف نستطيع حساب أطوال هذه الخطوط؟ في الحقيقة، ال Xna تقوم بالعمل من أجلنا. كل ما علينا عمله هو أن نحدد العمودي (الزاوية بينه و بين السطح 90 درجة، الخط الأزرق الرفيع) على كل من السطوح و من ثم ال Xna سوف تقوم بالباقي (عملية إسقاط بسيطة لجيب التمام "Cosine") من أجلنا!

هذا هو سبب إحتياجنا لإضافة العمودي (العمودي على السطوح، الخطوط الزرقاء الرفيعه) إلى معلومات الرؤوس. النوع VertexPositionColor لن يكون بإستطاعته عمل ذلك، ولسوء الحظ ال Xna لا توفر لنا تركيب "Struct" بإمكانه أن يحتوي على الموقع، اللون و العمودي. ولكن تلك ليست بالمشكلة، حيث بإمكاننا عمل تركيب خاص بنا بسهولة. ضع الكود التالي في أعلى كود الصنف، مباشرة فوق تعريف المتغيرات:

```
 public struct VertexPositionNormalColored
 {
     public Vector3 Position;
     public Color Color;
     public Vector3 Normal;

     public static int SizeInBytes = 7 * 4;
     public static VertexElement[] VertexElements = new VertexElement[]
     {
         new VertexElement( 0, 0, VertexElementFormat.Vector3,
VertexElementMethod.Default, VertexElementUsage.Position, 0 ),
         new VertexElement( 0, sizeof(float) * 3, VertexElementFormat.Color,
```

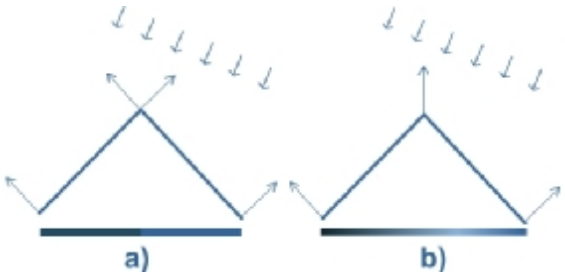
```
VertexElementMethod.Default, VertexElementUsage.Color, 0 ),
         new VertexElement( 0, sizeof(float) * 4, VertexElementFormat.Vector3,
VertexElementMethod.Default, VertexElementUsage.Normal, 0 ),
     };
 }
```

قد يبدو ذلك معقدا، لكني متأكد أنك قد فهمت اول ثلاث سطور على الأقل: هذا التركيب الجديد بإمكانه أن يحتوي على الموقع، اللون و العمودي؛ وهو ما نحتاج إليه بالضبط! الباقي من التركيب اعقد قليلا، وسوف نقوم بمناقشته بالتفصيل في السلسلة الثالثة من الدروس.

الآن بإمكاننا تغيير تعريف مصفوفة الرؤوس الخاصة بنا:

```
 VertexPositionNormalColored[] vertices;
```

الآن بإمكاننا تعريف السطوح مع العمودي عليها. ولكن أولا دعنا نلقي نظره على الصورة التالية، حيث تمثل الأسهم العلوية إتجاه الضوء و شريط الألوان اسفل الرسمه يمثل لون كل بكسل على طول السطح:



إذا قمنا ببساطة بتعريف المتجهات العمودية، من السهل أن نرى أنه قد أصبح لدينا حافة "edge" في الإضاءة (لاحظ شريط اللون الموجود تحت الصورة (a)). سبب ذلك أن السطح الايمن مضاء بشكل أكبر من السطح الأيسر. لذا سوف يكون من السهل ملاحظة أن السطح مكون من مثلثات منفصلة. مع ذلك، إذا قمنا بوضع عمودي واحد على الرأس العلوي المشترك كما يظهر في الصورة (b)، سوف تقوم ال Xna بشكل تلقائي بتعديل الإضاءة في كل بكسل على السطح! ذلك سوف يعطينا تأثير أنعم بكثير، كما ترى في شريط اللون في الصورة(b) . هذا المتجه هو المعدل "Average" للمتجهين في اعلى الصورة (a).

كما العادة، معدل المتجهين يمكن إيجاده من خلال جمع المتجهين و قسمتهم على 2.

لإستعراض هذا المثال سوف نقوم أولا بتغيير موقع الكاميرا:

```
 viewMatrix = Matrix.CreateLookAt(new Vector3(0, -40, 100), new Vector3(0, 50,
0), new Vector3(0, 1, 0));
```

بعدها سوف نقوم بتعديل الدالة SetUpVertices بحيث يتم تعريف 6 رؤوس تمثل المثلثين في المثال السابق:

```
 private void SetUpVertices()
 {
     vertices = new VertexPositionNormalColored[6];

     vertices[0].Position = new Vector3(0f, 0f, 50f);
     vertices[0].Color = Color.Blue;
     vertices[0].Normal = new Vector3(1, 0, 1);

     vertices[1].Position = new Vector3(50f, 0f, 0f);
```

```
      vertices[1].Color = Color.Blue;
      vertices[1].Normal = new Vector3(1, 0, 1);

      vertices[2].Position = new Vector3(0f, 50f, 50f);
      vertices[2].Color = Color.Blue;
      vertices[2].Normal = new Vector3(1, 0, 1);

      vertices[3].Position = new Vector3(-50f, 0f, 0f);
      vertices[3].Color = Color.Blue;
      vertices[3].Normal = new Vector3(-1, 0, 1);

      vertices[4].Position = new Vector3(0f, 0f, 50f);
      vertices[4].Color = Color.Blue;
      vertices[4].Normal = new Vector3(-1, 0, 1);

      vertices[5].Position = new Vector3(0f, 50f, 50f);
      vertices[5].Color = Color.Blue;
      vertices[5].Normal = new Vector3(-1, 0, 1);

      for (int i = 0; i < vertices.Length; i++)
          vertices[i].Normal.Normalize();

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

هذا يعرف السطحين في الصورة السابقة. بإضافة إحداثي Z (غير ال 0)، المثلثات الآن ثلاثية الأبعاد. بإمكانك أن تلاحظ أنني قمت بتعريف المتجهات العمودية الخاصة بالمثلثات، كما تظهر في المثال في الصورة (a).

في النهاية قمنا بتسوية المتجهات العمودية. معنى ذلك أننا قمنا بتحجيم المتجهات العمودية بحيث يصبح طولهم 1 تماما. ذلك مطلوب من أجل الوصول إلى إضاءة صحيحة، بما أن طول العمودي له تأثير على كمية الإضاءة.

لاحظ أيضا أننا قمنا بتغير تعريف ال VertexDeclaration، لأن الرؤوس أصبحت تحتوي على بيانات إضافية يجب تمريرها لبطاقة الرسوميات.

كل ما تبقى علينا تعديله في الدالة Draw هو القليل:

```
  device.DrawUserPrimitives(PrimitiveType.TriangleList, vertices, 0, 2);
```

الذي يقوم برسم المثلثين المعرفين في مصفوفة الرؤوس. عندما تقوم بتشغيل الكود، يجب أن ترى سهم (المثلثين الذين قمنا برسمهما)، ولكنك لا ترى أي عمق لأننا لم نقم بتعريف الضوء حتى الآن! بإمكاننا تعريف ذلك من خلال وضع هذه الوسائط الإضافية إلى التأثير الخاص بنا:

```
  effect.Parameters["xEnableLighting"].SetValue(true);
  Vector3 lightDirection = new Vector3(0.5f, 0, -1.0f);
  lightDirection.Normalize();
  effect.Parameters["xLightDirection"].SetValue(lightDirection);
```

هذا يطلب من التقنية "Technique" الخاصة بنا أن تقوم بتفعيل حسابات الإضاءة (الآن التقنية سوف تحتاج للعموديات)، و كما نحتاج لتحديد الإتجاه الخاص بالضوء. لاحظ مرة أخرى أننا بحاجة لتسوية هذا المتجه، حيث يصبح طوله 1. ربما تريد أيضا أن تقوم بتغيير لون الخلفية إلى أسود، حيث ستحصل على مشهد أفضل.

الآن قم بتشغيل الكود سوف ترى ما اعنيه بكلمة 'حواف الإضاءة': الضوء مشع بشكل جيد على الصفيحة اليسرى و أقل إشعاعا على الصفيحة اليمنى. بإمكانك ملاحظة الفرق بين المثلثين بشكل واضح جدا! وهو ما عرضته في الجزء الأيسر من الصورة في الأعلى.

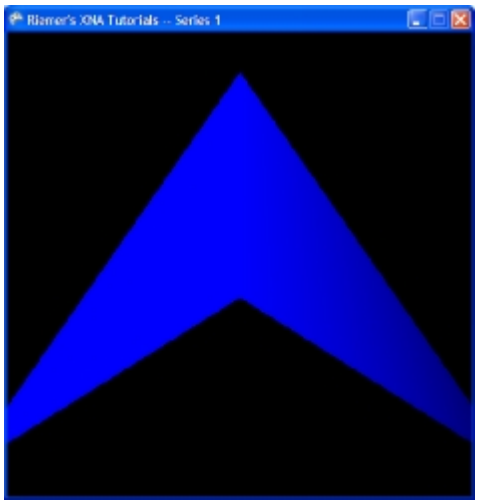
الآن لقد حان الوقت لدمج المتجهات الموجودة على الحافة المشتركة بين المثلثين من (1,0,1-) و (1,0,1) إلى

(0,0,1) = (-1+1,0,1+1)/2:

```
vertices[0].Position = new Vector3(0f, 0f, 50f);
vertices[0].Color = Color.Blue;
vertices[0].Normal = new Vector3(0, 0, 1);
vertices[1].Position = new Vector3(50f, 0f, 00f);
vertices[1].Color = Color.Blue;
vertices[1].Normal = new Vector3(1, 0, 1);
vertices[2].Position = new Vector3(0f, 50f, 50f);
vertices[2].Color = Color.Blue;
vertices[2].Normal = new Vector3(0, 0, 1);

vertices[3].Position = new Vector3(-50f, 0f, 0f);
vertices[3].Color = Color.Blue;
vertices[3].Normal = new Vector3(-1, 0, 1);
vertices[4].Position = new Vector3(0f, 0f, 50f);
vertices[4].Color = Color.Blue;
vertices[4].Normal = new Vector3(0, 0, 1);
vertices[5].Position = new Vector3(0f, 50f, 50f);
vertices[5].Color = Color.Blue;
vertices[5].Normal = new Vector3(0, 0, 1);
```

عندما تقوم بتشغيل هذا الكود، سوف ترى أن الإنعكاس موزع بشكل جيد من المنطقة اليمنى المظلمة إلى المنطقة المشرقة اليسرى. ليس من الصعب تخيل أن هذا التأثير سوف يعطي نعومة أفضل و أجمل بكثير على عدد اكبر من المثلثات، مثل التضاريس الخاصة بنا.



أيضا، يمكنك أن تلاحظ أنه بما ان النقاط الوسطى تستخدم **نفس** المتجهات العمودية، بإمكاننا دمج الرؤوس المشتركه ال 2*2 إلى 2*1 رأس بإستخدام فهرس للذاكره. لاحظ أيضا، أنه و في الحالة التي ستكون فيها **بحاجة** إلى إنشاء الحواف، سوف تحتاج لأن تحدد مجهين عموديين منفصلين.

بإمكانك محاولة حل التمرين التالي، لممارسة ما قد تعلمته:

- حاول تحريك المتجهات العمودية قليلا إلأى اليسار و إلى اليمين في كلا الحالتين المنفصلات و المشتركات.

كود الدرس:

```
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
    {
        public struct VertexPositionNormalColored
        {
          public Vector3 Position;
            public Color Color;
            public Vector3 Normal;

            public static int SizeInBytes = 7 * 4;
            public static VertexElement[] VertexElements = new VertexElement[]
            {
                new VertexElement( 0, 0, VertexElementFormat.Vector3, VertexElementMethod.Default,
VertexElementUsage.Position, 0 ),
                new VertexElement( 0, sizeof(float) * 3, VertexElementFormat.Color,
VertexElementMethod.Default, VertexElementUsage.Color, 0 ),
                new VertexElement( 0, sizeof(float) * 4, VertexElementFormat.Vector3,
VertexElementMethod.Default, VertexElementUsage.Normal, 0 ),
            };
        }

        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;
        GraphicsDevice device;
        Effect effect;
        VertexPositionNormalColored[] 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 VertexPositionNormalColored[6];

            vertices[0].Position = new Vector3(0f, 0f, 50f);
            vertices[0].Color = Color.Blue;
            vertices[0].Normal = new Vector3(0, 0, 1);
            vertices[1].Position = new Vector3(50f, 0f, 00f);
            vertices[1].Color = Color.Blue;
            vertices[1].Normal = new Vector3(1, 0, 1);
            vertices[2].Position = new Vector3(0f, 50f, 50f);
            vertices[2].Color = Color.Blue;
            vertices[2].Normal = new Vector3(0, 0, 1);

            vertices[3].Position = new Vector3(-50f, 0f, 0f);
            vertices[3].Color = Color.Blue;
            vertices[3].Normal = new Vector3(-1, 0, 1);
            vertices[4].Position = new Vector3(0f, 0f, 50f);
            vertices[4].Color = Color.Blue;
            vertices[4].Normal = new Vector3(0, 0, 1);
            vertices[5].Position = new Vector3(0f, 50f, 50f);
            vertices[5].Color = Color.Blue;
            vertices[5].Normal = new Vector3(0, 0, 1);

            for (int i = 0; i < vertices.Length; i++)
                vertices[i].Normal.Normalize();

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

        private void SetUpCamera()
        {
            viewMatrix = Matrix.CreateLookAt(new Vector3(0, -40, 100), new Vector3(0, 50, 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.Black);

            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.Parameters["xEnableLighting"].SetValue(true);
            Vector3 lightDirection = new Vector3(0.5f, 0, -1.0f);
            lightDirection.Normalize();
            effect.Parameters["xLightDirection"].SetValue(lightDirection);

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

                device.VertexDeclaration = myVertexDeclaration;
                device.DrawUserPrimitives(PrimitiveType.TriangleList, vertices, 0, 2);

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

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