الدرس الثاني عشر

أهلا بكم في الدرس الثاني عشر من سلسلة دروس تعلم ال 3D Xna السلسلة الأولى، في هذا الدرس سوف نقوم بإضافة المتجهات العمودية إلى التضاريس الخاصة بنا.

هنا سوف نقوم بإضافة بيانات المتجهات العمودية إلى كل الرؤوس في التضاريس، حيث سوف يصبح بإمكان بطاقة الرسوميات عمل بعض حسابات الإضاءة عليها. كما شاهدنا في الدرس السابق سوف نحتاج إلى إضافة متجهه عمودي لكل من الرؤوس الخاصة بنا.

هذا المتجه يجب أن يكون عمودي على المثلث الموجود عليه الرأس. في الحالة التي يكون فيها الرأس مشترك بين عدة مثلثات (كما في التضاريس الخاصة بنا)، يجب عليك أن تحسب العموديات لكل المثلثات التي تستخدم هذا الرأس، و تخزين معدل هذه العموديات في ذلك الرأس.

أولا علينا إعادة الكود الخاص بدرس إضافة الألوان (الدرس العاشر)، بعدها سوف نكون بحاجة إلى إضافة التركيب "Struct" (الذي يتيح لنا تخزين بيانات المجهات العموديه) إلى أعلى كود الصنف:

```
 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 ),
     };
 }
```

لا تنسى أن تعدل تعريف مصفوفة الرؤوس:

```
 VertexPositionNormalColored[] vertices;
```

كما لا تنسى تعديل ال VertexDeclaration في داخل الدالة SetUpVertices:

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

قم بإضافة السطر الذي يقوم عمليا برسم المثلثات على الشاشة في الدالة Draw:

```
 device.DrawUserIndexedPrimitives(PrimitiveType.TriangleList, vertices, 0,
vertices.Length, indices, 0, indices.Length / 3);
```

الآن نحن جاهزين لحساب المتجهات العمودية، الذي سوف يتم في داخل الدالة الجديدة CalculateNormals. إبدأ بمسح العموديات في كل رأس من الرؤوس:

```
private void CalculateNormals()
{
    for (int i = 0; i < vertices.Length; i++)
        vertices[i].Normal = new Vector3(0, 0, 0);
}
```

بعد ذلك، نحن بحاجة للمرور على كل مثلث من المثلثات. لكل مثلث، سوف نقوم بحساب المتجه العمودي. في النهاية، سوف تقوم بإضافة هذا المتجه إلى المتجه العمودي الموجود في كل من الرؤوس الثلاثة المكونة للمثلث:

```
for (int i = 0; i < indices.Length / 3; i++)
{
    int index1 = indices[i * 3];
    int index2 = indices[i * 3 + 1];
    int index3 = indices[i * 3 + 2];

    Vector3 side1 = vertices[index1].Position - vertices[index3].Position;
    Vector3 side2 = vertices[index1].Position - vertices[index2].Position;
    Vector3 normal = Vector3.Cross(side1, side2);

    vertices[index1].Normal += normal;
    vertices[index2].Normal += normal;
    vertices[index3].Normal += normal;
}
```

هنا نقوم بالنظر إلى فهارس الرؤوس الثلاثه الخاصة بالمثلث. إذا كنت تعرف جانبين من المثلث، سيكون بإمكانك حساب العمودي من خلال حساب الضرب التقاطعي "Cross Product" لهذين الجانبين. في أي متجهين، عملية الضرب التقاطعي لهذين المتجهيين يعطينا متجه يمثل العمودي لكلا المتجهيين.

لذا أولا نقوم بحساب جانبين للمثلث، من خلال طرح موقع أحد الزوايا من موقع زاوية أخرى. في اللحظة التي نعرف فيها العمودي للمثلث، نقوم بإضافته إلى العمودي لكل من الرؤوس الثلاثة.

في هذه اللحظة، كل الرؤوس تحتوي متجهات عمودية ضخمة، بينما نحتاج لأن يكون طولهم موحد. لذا أنهي الدالة السابقة بعمل تسوية "Normalization" لجميع المتجهات العمودية:

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

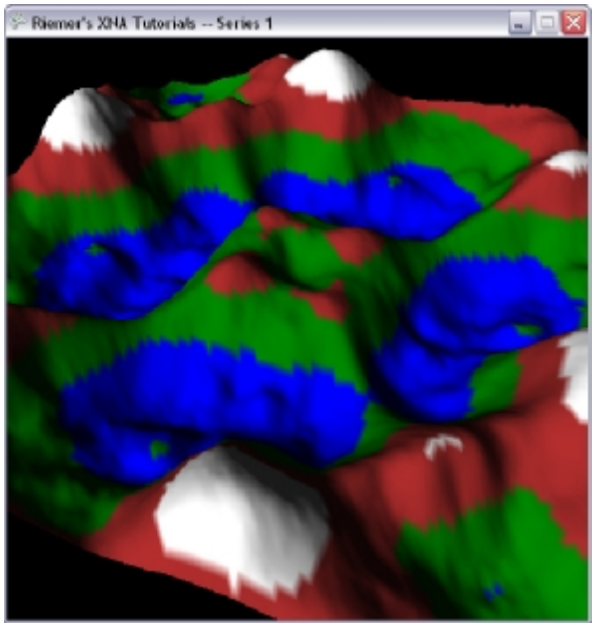
هذا كل شيئ بالنسبة للمتجهات العمودية! لا تنسى أن تستدعي هذه الدالة في نهاية الدالة LoadContent:

```
CalculateNormals();
```

في داخل الدالة Draw، قم بتشغيل الضوء:

```
Vector3 lightDirection = new Vector3(1.0f, -1.0f, -1.0f);
lightDirection.Normalize();
effect.Parameters["xLightDirection"].SetValue(lightDirection);
effect.Parameters["xEnableLighting"].SetValue(true);
effect.Parameters["xAmbient"].SetValue(0.1f);
```

السطر الأخير يقوم بتشغيل الإضاءة المحيطة "Ambient Lighting"، لذا حتى الأجزاء من التضاريس التي لا تصل إليها الإضاءة المتجهة "Directional Light" ما زالت تستقبل بعض الضوء. عندما تقوم بتشغيل الكود، يجب أن تشاهد الصورة التالية:



كما ترى، إستعمال الإضاءة الصحيحة تقوم بإضافة الشعور الثلاثي الأبعاد إلى المشهد بشكل دراماتيكي.

كود المشروع حتى الآن:

```
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;
        int[] indices;

        Matrix viewMatrix;
        Matrix projectionMatrix;

        private int terrainWidth;
        private int terrainHeight;
        private float[,] heightData;

        float angle = 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 1";

            base.Initialize();
        }

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

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

           Texture2D heightMap = Content.Load<Texture2D>
("heightmap");            LoadHeightData(heightMap);

            SetUpVertices();
            SetUpCamera();
            SetUpIndices();

            CalculateNormals();
        }

        private void SetUpVertices()
        {
            float minHeight = float.MaxValue;
            float maxHeight = float.MinValue;
            for (int x = 0; x < terrainWidth; x++)
            {
                for (int y = 0; y < terrainHeight; y++)
                {
                    if (heightData[x, y] < minHeight)
                        minHeight = heightData[x, y];
                    if (heightData[x, y] > maxHeight)
                        maxHeight = heightData[x, y];
                }
            }

            vertices = new VertexPositionNormalColored[terrainWidth * terrainHeight];
            for (int x = 0; x < terrainWidth; x++)
            {
                for (int y = 0; y < terrainHeight; y++)
                {
                    vertices[x + y * terrainWidth].Position = new Vector3(x, heightData[x, y], -y);

                    if (heightData[x, y] < minHeight + (maxHeight - minHeight) / 4)
                        vertices[x + y * terrainWidth].Color = Color.Blue;
                    else if (heightData[x, y] < minHeight + (maxHeight - minHeight) * 2 / 4)
                        vertices[x + y * terrainWidth].Color = Color.Green;
                    else if (heightData[x, y] < minHeight + (maxHeight - minHeight) * 3 / 4)
                        vertices[x + y * terrainWidth].Color = Color.Brown;
                    else
                        vertices[x + y * terrainWidth].Color = Color.White;
                }
```

```
            }

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

        private void SetUpIndices()
        {
            indices = new int[(terrainWidth - 1) * (terrainHeight - 1) * 6];
            int counter = 0;
            for (int y = 0; y < terrainHeight - 1; y++)
            {
                for (int x = 0; x < terrainWidth - 1; x++)
                {
                    int lowerLeft = x + y*terrainWidth;
                    int lowerRight = (x + 1) + y*terrainWidth;
                    int topLeft = x + (y + 1) * terrainWidth;
                    int topRight = (x + 1) + (y + 1) * terrainWidth;

                    indices[counter++] = topLeft;
                    indices[counter++] = lowerRight;
                    indices[counter++] = lowerLeft;

                    indices[counter++] = topLeft;
                    indices[counter++] = topRight;
                    indices[counter++] = lowerRight;
                }
            }
        }

        private void CalculateNormals()
        {
            for (int i = 0; i < vertices.Length; i++)
                vertices[i].Normal = new Vector3(0, 0, 0);

            for (int i = 0; i < indices.Length / 3; i++)
            {
                int index1 = indices[i * 3];
                int index2 = indices[i * 3 + 1];
                int index3 = indices[i * 3 + 2];

                Vector3 side1 = vertices[index1].Position - vertices[index3].Position;
                Vector3 side2 = vertices[index1].Position - vertices[index2].Position;
                Vector3 normal = Vector3.Cross(side1, side2);

                vertices[index1].Normal += normal;
                vertices[index2].Normal += normal;
                vertices[index3].Normal += normal;
            }

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

        private void SetUpCamera()
        {
            viewMatrix = Matrix.CreateLookAt(new Vector3(0, 100, 100), new Vector3(0, 0, 0), new
Vector3(0, 1, 0));
            projectionMatrix = Matrix.CreatePerspectiveFieldOfView(MathHelper.PiOver4,
device.Viewport.AspectRatio, 1.0f, 300.0f);
        }

        private void LoadHeightData(Texture2D heightMap)
        {
            terrainWidth = heightMap.Width;
            terrainHeight = heightMap.Height;

            Color[] heightMapColors = new Color[terrainWidth * terrainHeight];
            heightMap.GetData(heightMapColors);

            heightData = new float[terrainWidth, terrainHeight];
            for (int x = 0; x < terrainWidth; x++)
                for (int y = 0; y < terrainHeight; y++)
                    heightData[x, y] = heightMapColors[x + y * terrainWidth].R / 5.0f;
        }

        protected override void UnloadContent()
        {
        }

        protected override void Update(GameTime gameTime)
```

```
        {
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
                this.Exit();

            KeyboardState keyState = Keyboard.GetState();
            if (keyState.IsKeyDown(Keys.Delete))
                angle += 0.05f;
            if (keyState.IsKeyDown(Keys.PageDown))
                angle -= 0.05f;

            base.Update(gameTime);
        }

        protected override void Draw(GameTime gameTime)
        {
            device.Clear(ClearOptions.Target | ClearOptions.DepthBuffer, Color.Black, 1.0f, 0);
            device.RenderState.CullMode = CullMode.None;

            Matrix worldMatrix = Matrix.CreateTranslation(-terrainWidth / 2.0f, 0, terrainHeight /
2.0f) * Matrix.CreateRotationY(angle);
            effect.CurrentTechnique = effect.Techniques["Colored"];
            effect.Parameters["xView"].SetValue(viewMatrix);
            effect.Parameters["xProjection"].SetValue(projectionMatrix);
            effect.Parameters["xWorld"].SetValue(worldMatrix);
            effect.Parameters["xEnableLighting"].SetValue(true);
            Vector3 lightDirection = new Vector3(1.0f, -1.0f, -1.0f);
            lightDirection.Normalize();
            effect.Parameters["xLightDirection"].SetValue(lightDirection);
            effect.Parameters["xAmbient"].SetValue(0.1f);

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

                device.VertexDeclaration = myVertexDeclaration;
                device.DrawUserIndexedPrimitives(PrimitiveType.TriangleList, vertices, 0,
vertices.Length, indices, 0, indices.Length / 3);

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

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