الدرس السابع

أهلا بكم في الدرس السابع من سلسلة دروس تعلم ال 3D Xna السلسلة الأولى، في هذا الدرس سوف نتحدث عن اساسيات إنشاء التضاريس.

حتى الآن، قد درسنا عناوين كافيه لكي نبدأ بإنشاء التضاريس الخاصة بنا. دعنا نبدأ بشيئ بسيط، عن طريق ربط 3*4 رؤوس محددة. بكل الأحوال، سوف نقوم بعمل المحرك الخاص بنا بشكل ديناميكي، لذا في الدرس القادم سوف نستطيع ببساطة تحميل عدد اكبر بكثير من النقاط. لعمل ذلك، علينا إنشاء متغيرين جديدين في الصنف:

```
private int terrainWidth = 4;
private int terrainHeight = 3;
```

سوف نفترض أن النقاط ال 4*3 متساوية الأبعاد. لذا الشيئ الوحيد الذي نجهله عن نقاطنا حتى الآن هو إحداثي ال Z. حيث سوف نستخدم مصفوفة من أجل تخزين هذه المعلومات، لذا سوف نقوم بإضافة التالي إلى أعلى كود الصنف:

```
private float[,] heightData;
```

حتى الآن، إستخدم الدالة التالية من أجل تعبئة المصفوفة:

```
private void LoadHeightData()
{
    heightData = new float[4, 3];
    heightData[0, 0] = 0;
    heightData[1, 0] = 0;
    heightData[2, 0] = 0;
    heightData[3, 0] = 0;

    heightData[0, 1] = 0.5f;
    heightData[1, 1] = 0;
    heightData[2, 1] = -1.0f;
    heightData[3, 1] = 0.2f;

    heightData[0, 2] = 1.0f;
    heightData[1, 2] = 1.2f;
    heightData[2, 2] = 0.8f;
    heightData[3, 2] = 0;
}
```

طبعا لا تنسى إستدعائها من داخل الدالة LoadContent. تأكد أنك قمت بإستدعائها قبل الدالة SetUpVertices، لأن هذه الدالة سوف تستخدم محتويات المصفوفة heightData.

```
LoadHeightData();
```

الآن و بعد ملئ مصفوفة الإرتفاعات، بإستطاعتنا إنشاء الرؤوس. بما أن لدينا تضاريس بحجم 3*4، سوف نحتاج ل 12 (=عرض التضاريس* طول التضاريس) رأس. المسافة بين النقاط متساوية (المسافة بينهم متساوية)، إذن يمكننا بسهولة تغيير الدالة SetUpVertices. مبدئيا، لن نستخدم الإحداثي Z بعد، حيث سوف نلاحظ الفرق لاحقا في هذا الدرس.

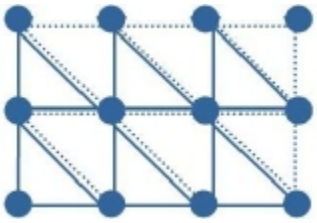
```
private void SetUpVertices()
{
```

```
    vertices = new VertexPositionColor[terrainWidth * terrainHeight];
    for (int x = 0; x < terrainWidth; x++)
    {
        for (int y = 0; y < terrainHeight; y++)
        {
            vertices[x + y * terrainWidth].Position = new Vector3(x, 0, -y);
            vertices[x + y * terrainWidth].Color = Color.White;
        }
    }

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

لا شيئ سحري يحصل هنا، انت ببساطة تعرف 12 نقطة و تعطيهم اللون الأبيض. لاحظ أن التضاريس سوف تنمو بإتجاه محور X الموجب (اليمين) و بإتجاه محور Z السالب (للأمام).

الجزء التالي أصعب قليلا: تعريف الفهارس التي تحدد المثلثات المطلوبه من أجل ربط ال 12 رأس. أفضل طريقة لعمل ذلك هي من خلال عمل مجموعتين من الرؤوس:



سوف نبدأ برسم المثلثات المرسومة بخطوط متواصلة. لعمل ذلك، قم بتغير الدالة SetUpIndices لتبدو بالشكل التالي:

```
 private void SetUpIndices()
 {
     indices = new int[(terrainWidth - 1) * (terrainHeight - 1) * 3];
     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;
         }
     }
 }
```

تذكر أن ال terrainWidth و ال terrainHeight يمثلان عدد النقاط الأفقي و العمودي في التضاريس. سوف نحتاج صفين كل صف مكون من 3 مثلثات، ما يعطينيا 6 مثلثات. ذلك سوف يتطلب منا 3*2*3 =18 فهرس (= (عرض التضاريس-1) * (طول التضاريس -1)*3)، لهذا، السطر الأول يقوم بإنشاء مصفوفة تحتوي على هذا العدد من الأرقام.

بعدها، نقوم بتعبئة المصفوفة بالفهارس. حيث نقوم بالمرور على إحداثيات X و Y سطر بعد سطر، حيث يتم إنشاء المثلثات. في السطر الأول، عندما تكون y=0، نحتاج إلى إنشاء 6 مثلثات من الرؤوس بدأ من 0 (أسفل-يسار) حتى 7 (وسط يمين). بعدها، تصبح قيمة y=1 و 1*عرض التضاريس=4 يتم إضافته إلى كل فهرس: في هذه المرة نقوم بإنشاء المثلثات ال 6 بناء على النقاط من 4 (وسط - ايسر) حتى 11 (يمين - أعلى).

لجعل الأمور أسهل، قمنا أولا بتعريف 4 إختصارات لفهارس أربع زوايا لكل رباعي. من رباعي نقوم بتخزين ثلاثة فهارس، تعرف مثلث واحد. هل تذكر الغربلة؟ تتطلب تعريف الرؤوس بترتيب بإتجاه عقارب الساعة. لذا أولا نقوم بتعريف الرأس العلوي- الأيسر، بعدها الرأس الأسفل- الأيمن ثم الرأس الأسفل- الأيسر.

المتغير counter هو طريقة سهلة من أجل تخزين الرؤوس في مصفوفة، حيث نقوم بزيادته في كل مرة يتم إضافة فهرس إلى المصفوفة. عندما تنتهي الدالة، سوف تكون المصفوفة تحتوي على جميع الفهارس الازمة لرسم جميع المثلثات السفلى اليسرى.

قمنا بعمل دالة الرسم بطريقة تتيح لل Xna رسم عدد من المثلثات تتحدد بناء على طول مصفوفة الفهارسن لذا بإمكانك تشغيل الكود مباشرة!

لابد و أنك قد لاحظت أن المثلثات تبدو رفيعة، لذا حاول أن تضع الكاميرا في الموقع (0,10,0) و إعد تشغيل البرنامج. يجب أن ترى 6 مثلثات في الجزء الأيمن من الشاشة، كل نقطة من كل مثلث هي على نفس الإحداثي في محور Z. الآن قم بتغير إرتفاع النقاط بناء على مصفوفة heightData:

```
vertices[x + y * terrainWidth].Position = new Vector3(x, heightData[x,y], -y);
```

عند تشغيل ذلك، سوف تلاحظ أن المثلثات لا تقع على نفس المستوى.

تذكر أنك مازلت فقط تقوم برسم المثلثات السفلية-اليسرى. لذا عندما تريد رسم المثلثات بإستخدام الوانهم المصمته بدلا من الحدود فقط، سوف تكون نصف الشبكه لديك غير مغطاه. لحل ذلك، دعنا نعرف بعض الفهارس الإضافية من أجل رسم المثلثات العلوية اليمنى. سوف نحتاج نفس الرؤوس، لذا التغيير الوحيد الازم هو الدالة SetUpIndices:

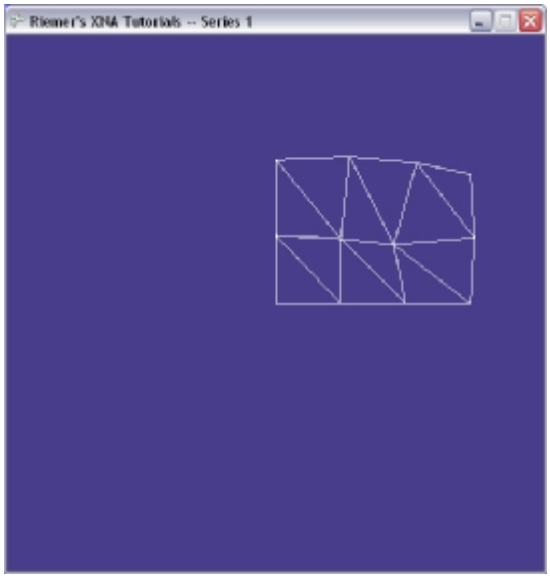
```
private void SetUpIndices()
{
    indices = new int[(terrainWidth - 1) * (terrainHeight - 1) * 6];
    int counter = 0;
    for (int y = 0; y < terrainHeight - 1; y++)
    {
        for (int x = 0; x < terrainWidth - 1; x++)
        {
            int lowerLeft = x + y*terrainWidth;
            int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

الآن سوف يتم رسم ضعف عدد الرؤوس، هذا هو السبب وراء إستخدام 6* بلا من 3*. كما ترى فإن المجموعة الثانية من المثلثات تم رسمها بترتيب بإتجاه عقارب الساعة بالنسبة للكاميرا: أولا الزاوية العلوية اليسرى، بعدها العلوية اليمنى و في النهاية السفلى اليمنى.

تشغيل هذا الكود سوف يعطيك مشهد ثلاثي الأبعاد أفضل. لقد أخذنا بعين الإعتبار إستخدام المتغيرين terrainWidth و terrainHeight فقط، لذا هذين هما المتغيرين اللآزم تغيرهما من أجل زيادة حجم الخارطة، معا بالإضافة إلى المصفوفة heightData. من الجميل إيجاد طريقة لملئ هذه المصفوفة بشكل تلقائي، و هو ما سوف نفعله في الدرس القادم، إن شاء الله.



إذا لم ترى أي مثلث مرسوم عند قيامك بتشغيل الكود، إرجع إلى الملاحظة في نهاية الدرس السابق. المشكلة لديك يجب أن تحل عن طريق إنشاء مصفوفة تقوم بتخزين قيم من نوع "short" بدلا من الأعداد الصحيحة "Integers" و بتعبئتها بالطريقة التالية:

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

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

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

بإمكانك محاولة حل التمارين التالية من أجل ممارسة ما تعلمته في هذا الدرس:

- قم بتغيير القيم الخاصة بمصفوفة heightData ولاحظ الفروق.
- حاول إضافة صف إضافي إلى الشبكة.

الكود 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
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;
        GraphicsDevice device;
      Effect effect;
        VertexPositionColor[] vertices;
        VertexDeclaration myVertexDeclaration;
        int[] indices;

        Matrix viewMatrix;
        Matrix projectionMatrix;

        private int terrainWidth = 4;
        private int terrainHeight = 3;
        private float[,] heightData;

        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");

            LoadHeightData();
            SetUpVertices();
            SetUpCamera();
            SetUpIndices();
        }

        private void SetUpVertices()
        {
            vertices = new VertexPositionColor[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);
                    vertices[x + y * terrainWidth].Color = Color.White;
                }
            }

            myVertexDeclaration = new VertexDeclaration(device, VertexPositionColor.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 SetUpCamera()
        {
            viewMatrix = Matrix.CreateLookAt(new Vector3(0, 10, 0), new Vector3(0, 0, 0), new Vector3(0, 0, -1));
            projectionMatrix = Matrix.CreatePerspectiveFieldOfView(MathHelper.PiOver4, device.Viewport.AspectRatio,
1.0f, 300.0f);
        }

        private void LoadHeightData()
        {
            heightData = new float[4, 3];
            heightData[0, 0] = 0;
            heightData[1, 0] = 0;
            heightData[2, 0] = 0;
            heightData[3, 0] = 0;

            heightData[0, 1] = 0.5f;
            heightData[1, 1] = 0;
            heightData[2, 1] = -1.0f;
            heightData[3, 1] = 0.2f;

            heightData[0, 2] = 1.0f;
            heightData[1, 2] = 1.2f;
            heightData[2, 2] = 0.8f;
            heightData[3, 2] = 0;
        }

        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;
            device.RenderState.FillMode = FillMode.WireFrame;

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

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

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