الدرس العاشر

أهلا بكم في الدرس العاشر من سلسلة دروس تعلم ال 3D Xna السلسلة الأولى، في هذا الدرس سوف نقوم بإعطاء التضاريس بعض من الألوان.

حتى الآن لديك تضاريس من الممكن تدويرها، ولكن بالتأكيد من الافضل أن تبدو هذه التضاريس معبئة ببعض الألوان بدلا من الخطوط البيضاء. أحد الطرق لعمل ذلك هو من خلال إستخدام الألوان الطبيعية، مثل الألوان التي نراها في الجبال. في الأسفل يكون لدينا بحيرات زرقاء، و بعدها أشجار خضراء، ثم الجبل البني، و في النهاية تكون الثلوج متلبدة على قمة الجبل. هذا يعني أننا بحاجة إلى عمل تحسين على كود الدالة SetUpVertices قليلا، بحيث يصبح بإمكانها تخزين الألوان الصحيحة لكل رأس.

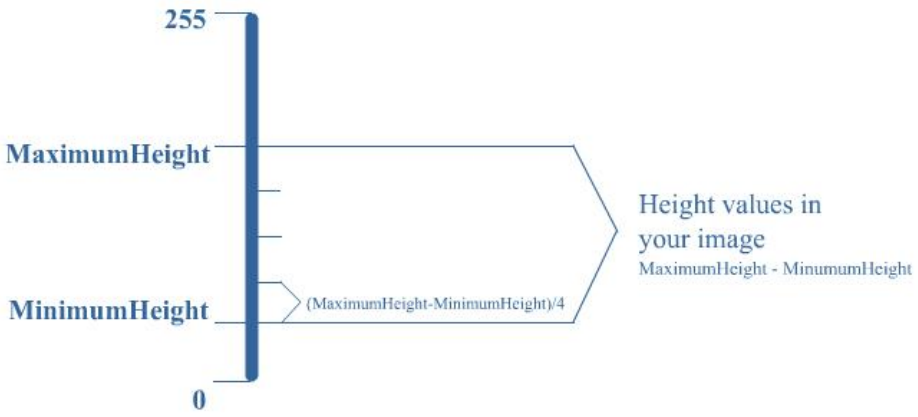
ليس بإستطاعتك في كل الأحول أن تتوقع أن كل صورة يكون فيها بحيرة في الإرتفاع صفر، و قمة الجبل في الإرتفاع 255 (القيمة القصوى في ملف ال bmp للبكسل). تخيل صورة فيها قيم كبيره بين ال 50 و 200، هذه الصورة سوف تنتج بالأغلب تضاريس بدون أي بحيرات أو قمم جبلية في الأعلى!

لكي يبقى الموضوع عام قدر الإمكان، سوف نقوم أولا بإكتشاف اقل و أقصى قيمة في الصورة. ثم تخزين هذه القيم في المتغيرات minHeight و maxHeight، حيث نقوم بإيجادهما من خلال وضع الكود التالي في أعلى الدالة 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];
    }
}
```

حيث يقوم بفحص كل نقطة في الشبكه إذا كان إرتفاعها أقل من قيمة ال minHeight الحالية، أو أعلى من قيمة ال maxHeight. إذا كانت كذلك، يقوم بتخزين قيمة الإرتفاع الحالي في المتغير المناظر.

بعد ملئ هذه المتغيرات، بإستطاعتك ملئ 4 مناطق من الألوان:



الآن من أجل تعريف الرؤوس و الوانها، يجب عليك تحديد اللون إعتمادا على مناطق الإرتفاع الصحيحه، كالتالي:

```
 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;
```

عند تشغيل هذا الكود، سوف ترى شبكه جميلة من الخطوط الملونة. عندما نريد روؤية التضاريس الملونة كامله، كل ما علينا هو حذف هذا السطر:

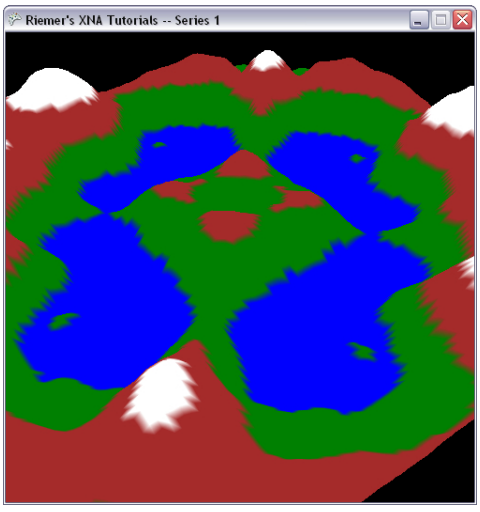
```
 device.RenderState.FillMode = FillMode.WireFrame;
```

عند تشغيل الكود، إقض بعض الوقت في تدوير التضاريس. في بعض الحواسيب، سوف تلاحظ أنه يتم رسم بحيرات غير مرئية خلف القمم المتوسطة. هذا لأننا لم نقوم بتعريف الذاكرة الخلفية "Z-Buffer" بعد! هذه الذاكرة الخلفية ليست إلا مصفوفة حيث تقوم بطاقة الرسوميات بمتابعه إحداثيات العمق لكل بكسل يجب أن يتم رسمه على الشاشة (إذن في حالتنا هي مصفوفة بحجم 500*500). في كل مرة تستقبل فيها بطاقة الرسوميات مثلث من أجل رسمه، تقوم بفحص فيما إذا كانت بكسلات هذا المثلث أقرب إلى الشاشة أكثر من البكسلات المتواجدة في الذاكرة الخلفية. إذا كانت كذلك، فإن محتويات الذاكرة الخلفية يتم تغييرها بهذه البكسلات لهذه المنطقة.

بالتأكيد، إن كل هذه العملية تعمل بشكل تلقائي. كل ما علينا عمله، هو تجهيز الذاكرة الخلفية الخاصة بنا بأكبر مسافة متحتملة للبدء بها. لذا في الحقيقة، علينا في البداية ملئ الذاكرة بالواحدات 1. لعمل ذلك بشكل تلقائي في كل تحديث للشاشة، قم بإضافة السطر التالي إلى أعلى الدالة Draw:

```
 device.Clear(ClearOptions.Target|ClearOptions.DepthBuffer, Color.Black, 1.0f,
0);
```

(الإشارة | هي عملية Bitwise OR، في هذه الحالة تعني أن كلا من الهدف (الألوان) بالإضافة إلى ذاكرة العمق DepthBuffer يجب أن يتم مسحهما). الآن الجميع يجب أن يرى التضاريس تدور كما هو متوقع.



بالرغم من أننا حصلنا على الألوان للتضاريس، ولكنها في الحقيقة لا تبدو بشكل أفضل مما كانت عليه في الدرس السابق.

بإمكانك محاولة حل التمرين التالي، من أجل ممارسة ما تعلمته:

- الإضائة لازمة من أجل الحصول على تأثير ثلاثي الأبعاد. في الصورة السابقة، قد حصلنا على شعور ثلاثي أبعاد بسيط، وذلك بسبب إستخدام عدة ألوان. قم بتغير الألوان إلى لون واحد، وسوف تلاحظ إختفاء الشعور الثلاثي الأبعاد تماما!

الكود حتى اللحظة:

```
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;
         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();
        }
```

```
        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 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);

                    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, 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(60, 80, -80), 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.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);
        }
    }
}
```