الدرس الثالث عشر

أهلا بكم في الدرس الثالث عشر من سلسلة دروس تعلم ال 3D Xna السلسلة الأولى، في هذا الدرس سوف نقوم بتحسين الأداء من خلال إستخدام ذاكرة الرؤوس و ذاكرة الفهارس.

في هذه اللحظة التضاريس تعمل بشكل جيد، مرة أخرى، في كل إطار "Frame" للعبة يتم إرسال جميع الرؤوس و الفهارس إلى بطاقة الرسوميات. هذا يعني أننا في كل إطار نقوم بإرسال نفس البيانات. من الواضح جدا أنه من الممكن تحسين ذلك.

نحن بحاجة إلى إرسال البيانات إلى بطاقة الرسوميات مرة واحدة فقط، بعدها يجب على بطاقة الرسوميات تخزين هذه البيانات في الذاكرة الخاصة به (التي تتميز بالسرعة العالية). يمكن عمل ذلك من خلال تخزين الرؤوس في ذاكرة للرؤوس "VertexBuffer"، و تخزين الفهارس في ذاكرة للفهارس "IndexBuffer".

إبدأ بتعريف المتغييرن التاليين في أعلى الكود:

```
 VertexBuffer myVertexBuffer;
 IndexBuffer myIndexBuffer;
```

سوف نقوم بتجهيز و تعبئة المتغيرات في الدالة جديدة، CopyToBuffers. الكود التالي يؤدي المهمة بالنسبة لذاكرة الرؤوس:

```
 private void CopyToBuffers()
 {
     myVertexBuffer = new VertexBuffer(device, vertices.Length *
VertexPositionNormalColored.SizeInBytes, BufferUsage.WriteOnly);
     myVertexBuffer.SetData(vertices);
 }
```

السطر الأول يقوم بإنشاء ال VertexBuffer، الذي يعمل على حجز جزء كافي من الذاكرة في بطاقة الرسوميات بحيث تستوعب جميع الرؤوس الخاصة بنا. لذلك يجب عليك أن تحدد كم عدد البايتات التي نحتاجها. حيث يتم إيجاده من خلال ضرب عدد الرؤوس بعدد البايتات التي تستخدم في الرأس الواحد.

السطر الثاني يقوم بنسخ البيانات من مصفوفة الرؤوس إلى الذاكرة في بطاقة الرسوميات.

نحتاج أيضا لعمل نفس الشي للفهارس، لذا ضع الكود التالي في نهاية الدالة:

```
 myIndexBuffer = new IndexBuffer(device, typeof(int), indices.Length,
BufferUsage.WriteOnly);
 myIndexBuffer.SetData(indices);
```

لإيجاد كم بايت يلزمنا حجزه، قمنا بتمرير نوع كل فهرس، إضافة إلى عدد الفهارس التي سوف نقوم بتخزينها. السطر الثاني يقوم عمليا بنسخ الفهارس إلى ذاكرة بطاقة الرسوميات.

لا تنسى أن تستدعي هذه الدالة من داخل الدالة LoadContent:

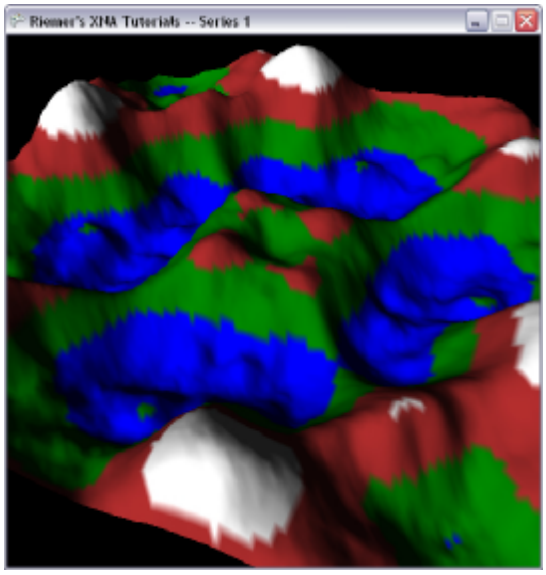
```
 CopyToBuffers();
```

بعد إنهاء ذلك، كل ما عليك عمليه هو أن ترشد بطاقة الرسوميات أن تقوم بالرسم من داخل الذاكرة الخاصة بها بإستخدام الدالة DrawIndexedPrimitive بدلا من الدالة DrawUserIndexedPrimitive. قبل إستدعائنا لهذه الدالة، نحتاج أن نطلب من بطاقة الرسوميات أن تقرأ البيانات من الذاكرة الخاصة بها، في داخل الدالة Draw:

```
 device.VertexDeclaration = myVertexDeclaration;
 device.Indices = myIndexBuffer;
 device.Vertices[0].SetSource(myVertexBuffer, 0,
VertexPositionNormalColored.SizeInBytes);
 device.DrawIndexedPrimitives(PrimitiveType.TriangleList, 0, 0,
vertices.Length, 0, indices.Length / 3);
```

قمنا بتحديد المكان الذي سوف تقوم بطاقة الرسوميات بقراءة الرؤوس و الفهارس منه، ثم طلبنا منها رسم المثلثات.

تشغيل هذا الكود سوف يعطيك نفس النتيجة في الدرس السابق، ولكن في هذه المرة لا يوجد هناك نقل بيانات على بطاقة الرسوميات!



بإمكانك محاولة حل التمرين التالي، لممارسة ما قد تعلمته:

- في هذا الكود، ما زلنا نحتفظ بمصفوفات الرؤوس و الفهارس كمتغيرات عامة "Global"، فقط لأننا بحاجة لأن نعرف أطوالها في إستدعاء الدالة DrawIndexesPrimitives. لذا إذا كنت تتذكر أطوالها بإمكانك أن تتخلص منها و تحرر بعض من الذاكرة.

الحمد لله الذي أعاننا على إنهاء هذه السلسلة بنجاح إن شاء الله. و نرجو من الله أن يوفقنا في ترجمة السلاسل المتبقية

السلسلة التالية ستكون مهمة و شيقة جدا، لمن يرغب في متابعتها. سأبدأ بترجمتها في القريب العاجل إن شاء الله، و سأبدأ بنشرها بعد فتره ليست بالطويلة، لكي يتمكن أكبر عدد من قراءة هذه السلسلة و السلسلة السابقه الخاصة بالبرمجة ثنائية الأبعاد.

المترجم : خلدون خالد

الفريق العربي للبرمجة : http://arabteam2000-forum.com/

الكود النهائي للسلسلة:

```
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;
        VertexBuffer myVertexBuffer;

        VertexDeclaration myVertexDeclaration;
        int[] indices;
        IndexBuffer myIndexBuffer;

        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);

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

            CopyToBuffers();
        }

        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 CopyToBuffers()
        {
            myVertexBuffer = new VertexBuffer(device, vertices.Length *
VertexPositionNormalColored.SizeInBytes, BufferUsage.WriteOnly);
            myVertexBuffer.SetData(vertices);

            myIndexBuffer = new IndexBuffer(device, typeof(int), indices.Length,
BufferUsage.WriteOnly);
            myIndexBuffer.SetData(indices);
        }

        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.Indices = myIndexBuffer;
                device.Vertices[0].SetSource(myVertexBuffer, 0,
VertexPositionNormalColored.SizeInBytes);
                device.DrawIndexedPrimitives(PrimitiveType.TriangleList, 0, 0, vertices.Length, 0,
indices.Length / 3);

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

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