الدرس الثالث

أهلا بكم في الدرس الثالث من سلسلة دروس تعلم 3D Xna (السلسلة الثانية)، في هذا الدرس سوف نقوم بإنشاء أرضية المدينة.

الآن وقد رأينا كيف بإمكاننا إستيراد صور بسيطة إلى مشروع ال Xna الخاص بنا، و كيف تقوم الXna بعرض هذه الصور على المثلثات، لن يكون من الصعب علينا إنشاء كمية كبيرة من الصور. من المهم جدا إيجاد طريقة لجعل الكمبيوتر يقوم بتعريف كل النقاط لنا بشكل تلقائي.

كمثال بسيط، دعنا نقوم بإنشاء شبكة مكونة من 3*3 صور، بدون الصورة في الوسط. هذا يعني 8 صور، إذا 16 مثلث، و 48 رأس. بدلا من تعريف جميع هذه الرؤوس يدويا، دعنا نقوم بإنشاء مصفوفة بإسم "floorPlan"، في أعلى الكود. هذه المصفوفة سوف تحتوي لاحقا الأماكن التي نريد أن يكون فيها مباني في المدينة ثلاثية الأبعاد:

```
int[,] floorPlan;

VertexBuffer cityVertexBuffer;
VertexDeclaration texturedVertexDeclaration;
```

بما أننا سوف نقوم بتعريف الرؤوس مره واحدة فقط، سوف نقوم بتخزين هذه الرؤوس في الذاكرة الخاصة ببطاقة الرسوميات و ذلك من خلال تخزينها في ذاكرة رؤوس "VertexBuffer" (راجع السلسلة الأولى) بدلا من تخزينهم في مصفوفة بسيطة كما فعلنا في الدرس السابق. سنحتاج أيضا إلى "VertexDeclaration" لإرسالها إلى بطاقة الرسوميات، حيث سيكون بإمكانها معرفة طبيعة البيانات المرسلة إليها.

سنقوم أولا بإنشاء دالة بسيطة تقوم بملئ المصفوفة "floorPlan" بالبيانات:

```
private void LoadFloorPlan()
{
    floorPlan = new int[,]
    {
        {0,0,0},
        {0,1,0},
        {0,0,0},
    };
}
```

في هذه البيانات، الصفر "0" تعني 'قم برسم خامة الأرضية' و الواحد "1" تعني اترك هذه المكان فارغا (مفتوحا). لاحقا في هذه السلسلة، الواحد سوف تعني وجود مبنى. بهذه الطريقة سوف نحصل على مرونة عالية في البرنامج، ببساطة قيامنا بتغيير الصفر إلى واحد، يعني أننا سوف نحصل على مبنى إضافي يرسم في مدينتنا ثلاثية الأبعاد! قم بإستدعاء هذه الدالة من داخل الدالة Initialize:

```
LoadFloorPlan();
```

الآن سوف نقوم بتحديث الدالة SetUpVertices، حيث سوف تقوم بقراءة البيانات من داخل المصفوفة و بشكل تلقائي تقوم بإنشاء الرؤوس المقابلة. في الدرس السابق تعلمنا كيف نقوم بتغطية المثلثات بالصور. هذه المرة، سوف

نقوم بتحميل ملف صورة واحدة، تتكون من مجموعة من صور الخامات مرتبة خلف بعضها البعض. الجزء الأيسر الأقصى من الصورة يمثل بلاط (قرميد) الأرضية، يتبعها صورة حائط و سقف لكل نوع مختلف من المباني. بإمكانك تنزيل ملف الخامة من الرابط هنا لرؤية ما أعنيه (أو من الملفات المرفقة في الدرس الأول). ببساطة هي صورة واحدة تحتوي مجموعة من الصور المختلفة.

بإمكانك حذف خامة ال 'riemerstexture' من المشروع، من خلال الضغط بالزر الأيمن عليها في ال Solution Explorer و إختيار Delete. بعدها قم بإضافة ملف الخامات إلى المشروع بنفس الطريقة في الدرس السابق. في النهاية، سوف نقوم بإعادة تسمية متغير الخامة 'texture' إلى 'sceneryTexture'، لأننا سوف نستخدم أكثر من خامة في البرنامج لاحقا، قم بتغيير الاسم في أعلى الكود، ولا تنسى تغييره في داخل LoadContent():

```
sceneryTexture = Content.Load<Texture2D> ("texturemap");
```

بإمكانك حذف محتويات الدالة SetUpVertices. و البدء بالكود التالي، هذا الكود مبني على الدرس الأخير في السلسلة الأولى:

```
 private void SetUpVertices()
 {
     int cityWidth = floorPlan.GetLength(0);
     int cityLength = floorPlan.GetLength(1);


    List<VertexPositionNormalTexture> verticesList = new
List<VertexPositionNormalTexture> ();
    for (int x = 0; x < cityWidth; x++)
    {
        for (int z = 0; z < cityLength; z++)
        {
          //if floorPlan contains a 0 for this tile, add 2
triangles
        }
    }

    cityVertexBuffer = new VertexBuffer(device, verticesList.Count *
VertexPositionNormalTexture.SizeInBytes, BufferUsage.WriteOnly);

    cityVertexBuffer.SetData<VertexPositionNormalTexture>
(verticesList.ToArray());
    texturedVertexDeclaration = new VertexDeclaration(device,
VertexPositionNormalTexture.VertexElements);
}
```

يقوم هذا الكود أولا بإستخراج عرض و طول المدينة المستقبلية، و التي سوف تكون 3*3 في حالتنا الحالية بناء على ال floorMap. بعدها نقوم بإنشاء قائمة "List" قادرة على تخزين VertexPositionNormalTextures. الميزة التي تتميز بها القائمة على المصفوفة هو أنه لا يلزم تحديد عدد العناصر التي تريد إضافتها. لإنك تقوم فقط بإضافة العناصر، و القائمة تقوم بالنمو بشكل تلقائي.

بعدها، نقوم بالمرور على محتويات المصفوفة floorMap. حيث إذا وجد صفر "0" في المصفوفة، نريد أن نضيف 6 رؤوس إلى القائمة، لتمثيل مثلثين. سنقوم بهذا لاحقا، حتى الآن تخيل أنه عندما ينتهي التكرار ستكون القائمة تحتوي على 6 رؤوس لكل صفر يجده في المصفوفة floorMap.

لتخزين الرؤوس في الذاكرة الخاصة ببطاقة الرسوميات، قمنا بإنشاء ذاكرة رؤوس "VertexBuffer" كافية تماما لإستقبال الرؤوس (راجع السلسلة الأولى). بعدها قمنا بتحويل القائمة إلى مصفوفة، حيث بإمكاننا نسخ جميع الرؤوس إلى الذاكرة في بطاقة الرسوميات بإستخدام الدالة SetData.

في النهاية، قمنا بإنشاء VertexDeclaration مرتبط بنوع الرؤوس المستخدم.

لإنهاء هذه الدالة، سوف نحتاج لإضافة الكود التالي داخل التكرار:

```
 int imagesInTexture = 11;
 if (floorPlan[x,z] == 0)
 {
     verticesList.Add(new VertexPositionNormalTexture(new Vector3(x, 0,
-z), new Vector3(0, 1, 0), new Vector2(0, 1)));
     verticesList.Add(new VertexPositionNormalTexture(new Vector3(x, 0,
-z-1),new Vector3(0, 1, 0), new Vector2(0, 0)));
     verticesList.Add(new VertexPositionNormalTexture(new Vector3(x +
1,0,-z), new Vector3(0, 1, 0), new Vector2(1.0f / imagesInTexture, 1)));

     verticesList.Add(new VertexPositionNormalTexture(new Vector3(x, 0,-
z-1), new Vector3(0, 1, 0), new Vector2(0, 0)));
     verticesList.Add(new VertexPositionNormalTexture(new Vector3(x +
1,0,-z-1), new Vector3(0, 1, 0), new Vector2(1.0f / imagesInTexture,
0)));
     verticesList.Add(new VertexPositionNormalTexture(new Vector3(x +
1,0,-z), new Vector3(0, 1, 0), new Vector2(1.0f / imagesInTexture, 1)));
 }
```

في كل مره يتم فيها إيجاد صفر "0"، يتم تعريف مثلثين. المتجهات العمودية لكل رأس "Normal" تشير إلى الأعلى (0،1،0) بإتجاه السماء، و الجزء الصحيح من صورة الخامة يتم وضعها فوق المثلثات: المربع بين [0،0] و [1\عدد الصور في الخامة، 1]. في صورة الخامات لدينا، قمت بتخزين 11 صورة. لذا إحداثيات ال X للصورة الأولى تمتد من 0 إلى 1\11. ألق نظرة أخرى على ملف الخامه لتصل المعلومة بشكل أفضل.

حتى الآن قمنا بتعريف الكثير من الرؤوس، تمثل مثلثين لكل صفر يتم إيجادة في المصفوفة floorMap. ماذا أيضا، قمنا بتخزين هذه الرؤوس في الذاكرة في بطاقة الرسوميات.

بما أننا أنهينا هذه الدالة، سنقوم البدء بكتابة الكود الذي سوف يقوم برسم المثلثات. للمحافظة على ترتيب الدالة Draw، سنقوم بتعريف دالة جديدة:

```
private void DrawCity()
{
    effect.CurrentTechnique = effect.Techniques["Textured"];
    effect.Parameters["xWorld"].SetValue(Matrix.Identity);
    effect.Parameters["xView"].SetValue(viewMatrix);
    effect.Parameters["xProjection"].SetValue(projectionMatrix);
```

```
        effect.Parameters["xTexture"].SetValue(sceneryTexture);
        effect.Begin();
        foreach (EffectPass pass in effect.CurrentTechnique.Passes)
        {
            pass.Begin();
            device.VertexDeclaration = texturedVertexDeclaration;
            device.Vertices[0].SetSource(cityVertexBuffer, 0,
VertexPositionNormalTexture.SizeInBytes);
            device.DrawPrimitives(PrimitiveType.TriangleList, 0,
cityVertexBuffer.SizeInBytes / VertexPositionNormalTexture.SizeInBytes /
3);
            pass.End();
        }
        effect.End();
    }
```

مازالنا نستخدم التقنية Textured لرسم المثلثات من الرؤوس. كما العادة، نحتاج لتحديد مصفوفات العالم و العرض و الإسقاط. بما أننا نريد أن تقوم بطاقة الرسوميات بأخذ الألوان من الخامة، سنحتاج لتمرير هذه الخامة إليها.

المثلثات يتم رسمها من ذاكرة الرؤوس "VertexBuffer"، بإستخدام الكود من الدرس الأخير في السلسلة الأولى. الوسيط الأخير للدالة DrawPrimitives يقوم بشكل تلقائي بتحديد عدد المثلثات التي سوف يتم رسمها من ذاكرة الرؤوس. إذا قمت بقسمة الحجم الكلي بالبايتات لذاكرة الرؤوس على عدد البايتات المستخدمة من رأس واحد، سوف تحصل على عدد الرؤوس المخزنة في تلك الذاكرة. و بما أن 3 رؤوس تعرف مثلث واحد، فأنت تعرف عدد المثلثات التي يتم سيتم رسمها!

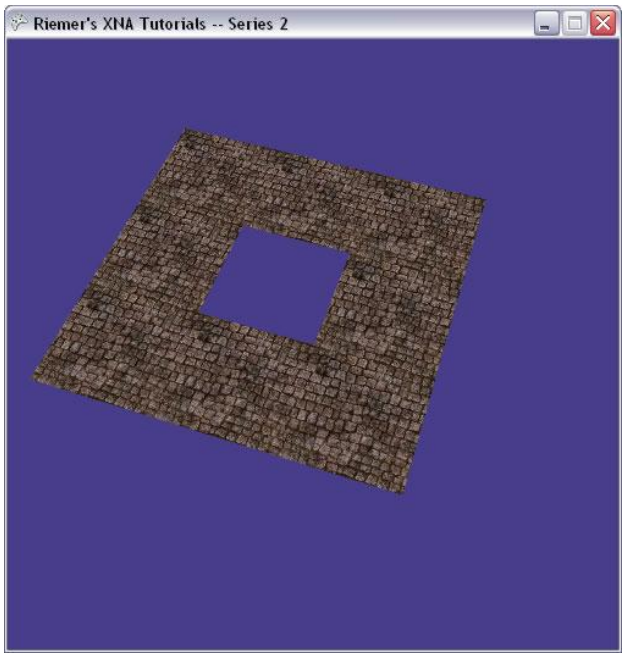
لا تنسى إستدعاء هذه الدالة من داخل الدالة Draw:

```
    protected override void Draw(GameTime gameTime)
    {
        ClearOptions.DepthBuffer, Color.DarkSlateBlue, 1.0f, 0);

        DrawCity();

        base.Draw(gameTime);
    }
```

هذا الكود يجب أن يكون قابل للتشغيل! يجب أن ترى مربع صغير تتوسطه فتحة في المنتصف، مثلما قمنا بالتعريف داخل الدالة LoadFloorPlan. ربما تكون فكرة جيدة أن نقوم بتغيير موقع الكاميرا قليلا:

```
    viewMatrix = Matrix.CreateLookAt(new Vector3(3, 5, 2), new Vector3(2,
0, -1), new Vector3(0, 1, 0));
```

الآن يجب أن تحصل على الشكل في الصورة التالية:



حاول حل التمارين التالية، لممارسة ما قد تعلمناه:

- قم بتغيير محتويات المصفوفة floorPlan.
- قم بتغير حجم المصفوفة floorPlan.
- قم بتغيير إحداثيات الخامة للرؤوس، بحيث تقوم بطاقة الرسوميات بإستخدام صورة أخرى من الخامات الموجودة لتغطية الأرضية.

في هذا الدرس تعلمنا كيف نستخدم الخامات "texturemap" و كيف نستخدم القوائم "List"، وهي ميزات مفيدة في ال C# و ال Xna.

كود المشروع 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 XNAseries2
{
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        GraphicsDevice device;

        Effect effect;
        Matrix viewMatrix;
        Matrix projectionMatrix;

        Texture2D sceneryTexture;
```

```
        int[,] floorPlan;

        VertexBuffer cityVertexBuffer;
        VertexDeclaration texturedVertexDeclaration;

        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 2";

            LoadFloorPlan();

            base.Initialize();
        }

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


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

           sceneryTexture = Content.Load<Texture2D> ("texturemap");
           SetUpVertices();

            SetUpCamera();
        }

        private void LoadFloorPlan()
        {
            floorPlan = new int[,]
            {
                {0,0,0},
                {0,1,0},
                {0,0,0},
            };
        }

        private void SetUpCamera()
        {
            viewMatrix = Matrix.CreateLookAt(new Vector3(3, 5, 2), new Vector3(2, 0, -1),
new Vector3(0, 1, 0));
            projectionMatrix = Matrix.CreatePerspectiveFieldOfView(MathHelper.PiOver4,
device.Viewport.AspectRatio, 0.2f, 500.0f);
        }

        private void SetUpVertices()
        {
            int cityWidth = floorPlan.GetLength(0);
            int cityLength = floorPlan.GetLength(1);


           List<VertexPositionNormalTexture> verticesList = new
List<VertexPositionNormalTexture> ();
            for (int x = 0; x < cityWidth; x++)
            {
                for (int z = 0; z < cityLength; z++)
                {
                    int imagesInTexture = 11;
                    if (floorPlan[x,z] == 0)
                    {
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x, 0,
```

```
-z), new Vector3(0, 1, 0), new Vector2(0, 1)));
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x, 0,
-z-1),new Vector3(0, 1, 0), new Vector2(0, 0)));
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x +
1, 0, -z), new Vector3(0, 1, 0), new Vector2(1.0f / imagesInTexture, 1)));

                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
0,-z-1), new Vector3(0, 1, 0), new Vector2(0, 0)));
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x +
1, 0, -z-1), new Vector3(0, 1, 0), new Vector2(1.0f / imagesInTexture, 0)));
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x +
1, 0, -z), new Vector3(0, 1, 0), new Vector2(1.0f / imagesInTexture, 1)));
                    }
                }
            }

            cityVertexBuffer = new VertexBuffer(device, verticesList.Count *
VertexPositionNormalTexture.SizeInBytes, BufferUsage.WriteOnly);

            cityVertexBuffer.SetData<VertexPositionNormalTexture>
(verticesList.ToArray());
            texturedVertexDeclaration = new VertexDeclaration(device,
VertexPositionNormalTexture.VertexElements);
        }


         protected override void UnloadContent()
         {
         }

         protected override void Update(GameTime gameTime)
         {
             base.Update(gameTime);
         }

         protected override void Draw(GameTime gameTime)
         {
             device.Clear(ClearOptions.Target | ClearOptions.DepthBuffer,
Color.DarkSlateBlue, 1.0f, 0);

             DrawCity();

             base.Draw(gameTime);
         }

         private void DrawCity()
         {
             effect.CurrentTechnique = effect.Techniques["Textured"];
             effect.Parameters["xWorld"].SetValue(Matrix.Identity);
             effect.Parameters["xView"].SetValue(viewMatrix);
             effect.Parameters["xProjection"].SetValue(projectionMatrix);
             effect.Parameters["xTexture"].SetValue(sceneryTexture);
             effect.Begin();
             foreach (EffectPass pass in effect.CurrentTechnique.Passes)
             {
                 pass.Begin();
                 device.VertexDeclaration = texturedVertexDeclaration;
                 device.Vertices[0].SetSource(cityVertexBuffer, 0,
VertexPositionNormalTexture.SizeInBytes);
                 device.DrawPrimitives(PrimitiveType.TriangleList, 0,
cityVertexBuffer.SizeInBytes / VertexPositionNormalTexture.SizeInBytes / 3);
                 pass.End();
             }
             effect.End();
         }
     }
 }
```