الدرس الرابع

أهلا بكم في الدرس الرابع من سلسلة دروس تعلم 3D Xna (السلسلة الثانية)، في هذا الدرس سوف نقوم برسم المدينة بشكل فعلي.

للأمانة، لن تتعلم أشياء جديدة كثيرة في هذا الدرس: سوف نقوم ببساطة بتوسيع الكود الموجود في الدرس السابق. حيث سنقوم برسم المباني على الأرضية التي قمنا بتجهيزها، و بعدها سنقوم برسم الأسطح فوق المباني. لهذا السبب، سأقوم بتلخيص الإضافات و التعديلات.

لأن برنامجنا سوف يدعم عدة أشكال من المباني (5 حسب صورة الخامة التي لدينا)، سوف نحتاج إلى مصفوفة تحتوي على إرتفاعات هذه المباني. لذا قم بتعريف هذه المصفوفة في أعلى الكود لديك:

```
int[] buildingHeights = new int[] { 0, 2, 2, 6, 5, 4 };
```

بعدها، سوف نقوم بمعالجة المصفوفة floorPlan، بحيث تعكس محتوياتها نوع المبنى. حيث الصفر "0" تبقى صفر، ولكن الواحد يتم إستبدالها برقم عشوائي بين الواحد "1" و الخمسه "5"، يمثل نوع المبنى.

لعمل ذلك، قم بإضافة الكود التالي في نهاية الدالة LoadFloorPlan:

```
Random random = new Random();
int differentBuildings = buildingHeights.Length - 1;
for (int x = 0; x < floorPlan.GetLength(0); x++)
    for (int y = 0; y < floorPlan.GetLength(1); y++)
        if (floorPlan[x, y] == 1)
            floorPlan[x, y] = random.Next(differentBuildings) + 1;
```

قمنا أولا بتجهيز كائن من النوع Random، حيث سوف نستقبل من هذا الكائن أرقام عشوائية من خلال إستدعاء الدالة Next. هذه الدالة سوف ترجع رقم عشوائي موجب، أقل من قيمة الوسيط المرسل.

بعدها نقوم بالمرور على المصفوفة floorPlan، و كل مره يتم إيجاد واحد في المصفوفة يتم إستبدالها برقم عشوائي موجب بين 1 و 5.

بعدها، سوف نبدأ بتوسيع الدالة SetUpVertices بحيث تقوم بتزويدنا أيضا بالرؤوس الخاصة بجدران الأبنية و الأسطح.

قم بإضافة المتغيرات التالية إلى بداية كود الدالة SetUpVertices:

```
int differentBuildings = buildingHeights.Length - 1;
float imagesInTexture = 1 + differentBuildings * 2;
```

لنبدأ بتعديل الكود بحيث يقوم برسم أسطح البنايات. إستبدل الكود الموجود داخل التكرار (في الدالة SetUpVertices) بالكود التالي:

```
int currentbuilding = floorPlan[x, z];
```

```
 //floor or ceiling
 verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z), new Vector3(0, 1, 0), new
Vector2(currentbuilding * 2 / imagesInTexture, 1)));
 verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z - 1), new Vector3(0, 1, 0), new
Vector2((currentbuilding * 2) / imagesInTexture, 0)));
 verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z), new Vector3(0, 1, 0), new
Vector2((currentbuilding * 2 + 1) / imagesInTexture, 1)));

 verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z - 1), new Vector3(0, 1, 0), new
Vector2((currentbuilding * 2) / imagesInTexture, 0)));
 verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z - 1), new Vector3(0, 1, 0), new
Vector2((currentbuilding * 2 + 1) / imagesInTexture, 0)));
 verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z), new Vector3(0, 1, 0), new
Vector2((currentbuilding * 2 + 1) / imagesInTexture, 1)));
```

هذا الكود هو بالضبط نفس الكود الخاص بالدرس السابق، مع بعض الإضافات المهمة. بداية، هذا الكود يقوم برسم مثلثين، بغض النظر عن ماهية الرقم المخزن في المصفوفة floorPlan. بعدها، يتم أخذ الإرتفاع (إحداثي Y) من المصفوفة buildingHeights التي قمنا بتجهيزها في بداية الدرس. ثم بإمكانك ملاحظة أن إحداثي X للخامة تم حسابه بشكل تلقائي. بعدها بما أن لدينا 5 أنواع مختلفة من المباني، إحداثي X للجانب الأيسر من خامة الأرضية هي صفر "0"، و للجانب الأيسر للسقف الاول هي 2\11، و السقف الثاني هي 4\11، ...، و للسقف الأخير 10\11. حسب المعادلة: المبنى الحالي * 2\ عدد الصور.

```
currentbuilding * 2 / imagesInTexture
```

من ناحية أخرى إحداثيات X للجانب الأيمن للأرضية هي 1\11، و للسقف الأول هي 3\11،...، للسقف الأخير هي 11\11. حسب المعادلة (المبنى الحالي*2+1)\عدد الصور في الخامة.

```
(currentbuilding * 2 + 1) / imagesInTexture
```

بإختصار: إذا وجد صفر "0"، يتم رسم الأرضية، إذا وجد رقم مبنى، يتم رسم صورة السقف الخاص بالمبنى ذو الإرتفاع المقابل، الذي تم إيجاده في مصفوفة الإرتفاعات buildingHeights. بإمكانك محاولة تشغيل هذا الكود، و في كل مره تقوم فيها بتنفيذ البرنامج سوف ترى صورة سقف جديدة بإرتفاع مختلف، لأن نوع المبنى يتم إختياره بشكل عشوائي.

الأخبار الجيدة: قمنا بتغطية كل الأشياء الجديدة في هذا الدرس. و لكن الأخبار السيئة: أن رسم جدران المباني تتضمن عملية نسخ كمية من نفس الكود، مع بعض الإختلافات في قيم الإحداثيات الخاصة بالرؤوس و بالخامات. لنقوم بالمهمة، في ما يلي الدالة كاملة، ملحقة بنقاش بسيط:

```
 private void SetUpVertices()
```

```
 {
     int differentBuildings = buildingHeights.Length - 1;
     float imagesInTexture = 1 + differentBuildings * 2;

     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 currentbuilding = floorPlan[x, z];

            //floor or ceiling
            verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z), new Vector3(0, 1, 0), new
Vector2(currentbuilding * 2 / imagesInTexture, 1)));
            verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z - 1), new Vector3(0, 1, 0), new
Vector2((currentbuilding * 2) / imagesInTexture, 0)));
            verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z), new Vector3(0, 1, 0), new
Vector2((currentbuilding * 2 + 1) / imagesInTexture, 1)));

            verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z - 1), new Vector3(0, 1, 0), new
Vector2((currentbuilding * 2) / imagesInTexture, 0)));
            verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z - 1), new Vector3(0, 1, 0), new
Vector2((currentbuilding * 2 + 1) / imagesInTexture, 0)));
            verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z), new Vector3(0, 1, 0), new
Vector2((currentbuilding * 2 + 1) / imagesInTexture, 1)));

            if (currentbuilding != 0)
            {
                //front wall
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x +
1, 0, -z - 1), new Vector3(0, 0, -1), new Vector2((currentbuilding * 2) /
imagesInTexture, 1)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z - 1), new Vector3(0, 0, -1), new
Vector2((currentbuilding * 2 - 1) / imagesInTexture, 0)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
0, -z - 1), new Vector3(0, 0, -1), new Vector2((currentbuilding * 2 - 1) /
imagesInTexture, 1)));

                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z - 1), new Vector3(0, 0, -1), new
Vector2((currentbuilding * 2 - 1) / imagesInTexture, 0)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x +
1, 0, -z - 1), new Vector3(0, 0, -1), new Vector2((currentbuilding * 2) /
imagesInTexture, 1)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x +
1, buildingHeights[currentbuilding], -z - 1), new Vector3(0, 0, -1), new
Vector2((currentbuilding * 2) / imagesInTexture, 0)));

                //back wall
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x +
1, 0, -z), new Vector3(0, 0, 1), new Vector2((currentbuilding * 2) /
```

```
imagesInTexture, 1)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
0, -z), new Vector3(0, 0, 1), new Vector2((currentbuilding * 2 - 1) /
imagesInTexture, 1)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z), new Vector3(0, 0, 1), new
Vector2((currentbuilding * 2 - 1) / imagesInTexture, 0)));

                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z), new Vector3(0, 0, 1), new
Vector2((currentbuilding * 2 - 1) / imagesInTexture, 0)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x +
1, buildingHeights[currentbuilding], -z), new Vector3(0, 0, 1), new
Vector2((currentbuilding * 2) / imagesInTexture, 0)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x +
1, 0, -z), new Vector3(0, 0, 1), new Vector2((currentbuilding * 2) /
imagesInTexture, 1)));

                //left wall
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
0, -z), new Vector3(-1, 0, 0), new Vector2((currentbuilding * 2) /
imagesInTexture, 1)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
0, -z - 1), new Vector3(-1, 0, 0), new Vector2((currentbuilding * 2 - 1) /
imagesInTexture, 1)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z - 1), new Vector3(-1, 0, 0), new
Vector2((currentbuilding * 2 - 1) / imagesInTexture, 0)));

                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z - 1), new Vector3(-1, 0, 0), new
Vector2((currentbuilding * 2 - 1) / imagesInTexture, 0)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z), new Vector3(-1, 0, 0), new
Vector2((currentbuilding * 2) / imagesInTexture, 0)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
0, -z), new Vector3(-1, 0, 0), new Vector2((currentbuilding * 2) /
imagesInTexture, 1)));

                //right wall
                        verticesList.Add(new VertexPositionNormalTexture(new
Vector3(x + 1, 0, -z), new Vector3(1, 0, 0), new Vector2((currentbuilding * 2) /
imagesInTexture, 1)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x +
1, buildingHeights[currentbuilding], -z - 1), new Vector3(1, 0, 0), new
Vector2((currentbuilding * 2 - 1) / imagesInTexture, 0)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x +
1, 0, -z - 1), new Vector3(1, 0, 0), new Vector2((currentbuilding * 2 - 1) /
imagesInTexture, 1)));

                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x +
1, buildingHeights[currentbuilding], -z - 1), new Vector3(1, 0, 0), new
Vector2((currentbuilding * 2 - 1) / imagesInTexture,
0)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x +
1, 0, -z), new Vector3(1, 0, 0), new Vector2((currentbuilding * 2) /
imagesInTexture, 1)));
                verticesList.Add(new VertexPositionNormalTexture(new Vector3(x +
1, buildingHeights[currentbuilding], -z), new Vector3(1, 0, 0), new
Vector2((currentbuilding * 2) / imagesInTexture, 0)));
            }
        }
    }
```

```
    cityVertexBuffer = new VertexBuffer(device, verticesList.Count *
VertexPositionNormalTexture.SizeInBytes,
BufferUsage.WriteOnly);          cityVertexBuffer.SetData<VertexPositionNormalTex
ture> (verticesList.ToArray());
    texturedVertexDeclaration = new VertexDeclaration(device,
VertexPositionNormalTexture.VertexElements);
}
```

☺ هذا ما يمكن تسميته دالة! جميل جدا، ولكنها لا تحتوي شيئ جديد بالنسبة لنا. أولا يتم رسم الأرضيات أو الأسقف. بعدها يتم إضافة 6 رؤوس لكل من الجدران الأربعة. لاحظ أن كل رأس يجب أن يخزن الإحداثيات ثلاثية الأبعاد الصحيحة للموقع، و إحداثيات الخامة بحيث يتم رسم الجزء الصحيح من الخامة على الحائط، و قيم العمودي على الرؤوس الصحيحة بحيث يمكننا إضافة إضاءة في الدرس التالي.

بإمكانك أيضا ملاحظة مدى فائدة الدالة Add الموجودة ضمن ال List.

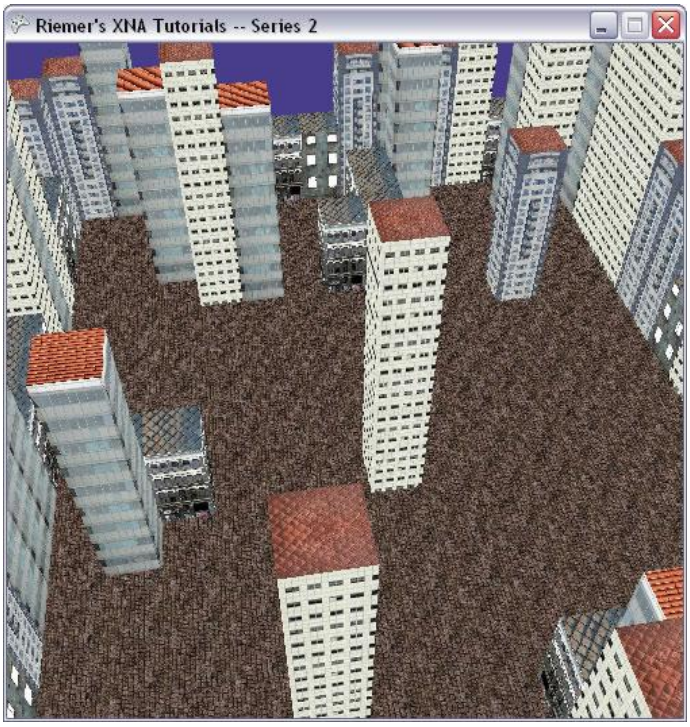
لجعل العمل أكثر جاذبية، دعنا نغير أبعاد و محتويات المصفوفة floorPlan:

```
 floorPlan = new int[,]
 {
  {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
  {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
  {1,0,0,1,1,0,0,0,1,1,0,0,1,0,0,1},
  {1,0,0,1,1,0,0,0,0,1,0,0,0,1,0,1},
  {1,0,0,0,1,1,0,0,1,1,0,0,0,0,0,1},
  {1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1},
  {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
  {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
  {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
  {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
  {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
  {1,0,1,1,0,0,0,1,0,0,0,0,0,0,0,1},
  {1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1},
  {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
  {1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
  {1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1},
  {1,0,0,0,0,1,0,0,0,1,0,0,0,0,1,1},
  {1,0,1,0,0,0,0,0,0,1,0,0,0,0,1,1},
  {1,0,1,1,0,0,0,0,0,1,1,0,0,0,1,1},
  {1,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1},
  {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
 };
```

يجب أن يكون الكود الآن قابل للتشغيل، بما أن الكود في الدالة DrawCity يتغير بناء على محتويات ال VertexBuffer. لكن دعنا أولا نقوم بتغيير موضع الكاميرا، و إلا لن ترى إلا جزء بسيط من المدينة:

```
 viewMatrix = Matrix.CreateLookAt(new Vector3(20, 13, -5), new
Vector3(8, 0, -7), new Vector3(0, 1, 0));
```

الآن قم بتشغيل الكود! هذا ما يجب أن ترى ☺



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
    {
        int[] buildingHeights = new int[] { 0, 2, 2, 6, 5, 4 };

        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[,]
              {
                  {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
                  {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
                  {1,0,0,1,1,0,0,0,1,1,0,0,1,0,1},
                  {1,0,0,1,1,0,0,0,1,0,0,0,1,0,1},
                  {1,0,0,0,1,1,0,1,1,0,0,0,0,0,1},
                  {1,0,0,0,0,0,0,0,0,0,0,0,1,0,1},
                  {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
                  {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
                  {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
                  {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
                  {1,0,1,1,0,0,0,1,0,0,0,0,0,0,1},
                  {1,0,1,0,0,0,0,0,0,0,0,0,0,0,1},
                  {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
                  {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
                  {1,0,0,0,0,1,0,0,0,0,0,0,0,0,1},
                  {1,0,0,0,0,1,0,0,0,1,0,0,0,0,1},
                  {1,0,1,0,0,0,0,0,0,1,0,0,0,0,1},
                  {1,0,1,1,0,0,0,0,1,1,0,0,0,1,1},
                  {1,0,0,0,0,0,0,0,1,1,0,0,0,1,1},
                  {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
              };

             Random random = new Random();
             int differentBuildings = buildingHeights.Length - 1;
             for (int x = 0; x < floorPlan.GetLength(0); x++)
                 for (int y = 0; y < floorPlan.GetLength(1); y++)
                     if (floorPlan[x, y] == 1)
                         floorPlan[x, y] = random.Next(differentBuildings) + 1;
         }

         private void SetUpCamera()
         {
             viewMatrix = Matrix.CreateLookAt(new Vector3(20, 13, -5), new Vector3(8, 0, -7), new
Vector3(0, 1, 0));
             projectionMatrix = Matrix.CreatePerspectiveFieldOfView(MathHelper.PiOver4,
device.Viewport.AspectRatio, 0.2f, 500.0f);
         }

         private void SetUpVertices()
         {
             int differentBuildings = buildingHeights.Length - 1;
             float imagesInTexture = 1 + differentBuildings * 2;

             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 currentbuilding = floorPlan[x, z];

                    //floor or ceiling
                    verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z), new Vector3(0, 1, 0), new Vector2(currentbuilding * 2 /
imagesInTexture, 1)));
                    verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z - 1), new Vector3(0, 1, 0), new Vector2((currentbuilding * 2) /
imagesInTexture, 0)));
                    verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z), new Vector3(0, 1, 0), new Vector2((currentbuilding * 2 + 1) /
imagesInTexture, 1)));

                    verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z - 1), new Vector3(0, 1, 0), new Vector2((currentbuilding * 2) /
imagesInTexture, 0)));
                    verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z - 1), new Vector3(0, 1, 0), new Vector2((currentbuilding * 2 + 1)
/ imagesInTexture, 0)));
                    verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z), new Vector3(0, 1, 0), new Vector2((currentbuilding * 2 + 1) /
imagesInTexture, 1)));

                    if (currentbuilding != 0)
                    {
                        //front wall
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1, 0, -z -
1), new Vector3(0, 0, -1), new Vector2((currentbuilding * 2) / imagesInTexture, 1)));
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z - 1), new Vector3(0, 0, -1), new Vector2((currentbuilding * 2 - 1)
/ imagesInTexture, 0)));
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x, 0, -z - 1),
new Vector3(0, 0, -1), new Vector2((currentbuilding * 2 - 1) / imagesInTexture, 1)));

                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z - 1), new Vector3(0, 0, -1), new Vector2((currentbuilding * 2 - 1)
/ imagesInTexture, 0)));
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1, 0, -z -
1), new Vector3(0, 0, -1), new Vector2((currentbuilding * 2) / imagesInTexture, 1)));
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z - 1), new Vector3(0, 0, -1), new Vector2((currentbuilding * 2) /
imagesInTexture, 0)));

                        //back wall
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1, 0, -z),
new Vector3(0, 0, 1), new Vector2((currentbuilding * 2) / imagesInTexture, 1)));
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x, 0, -z), new
Vector3(0, 0, 1), new Vector2((currentbuilding * 2 - 1) / imagesInTexture, 1)));
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z), new Vector3(0, 0, 1), new Vector2((currentbuilding * 2 - 1) /
imagesInTexture, 0)));

                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z), new Vector3(0, 0, 1), new Vector2((currentbuilding * 2 - 1) /
imagesInTexture, 0)));
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z), new Vector3(0, 0, 1), new Vector2((currentbuilding * 2) /
imagesInTexture, 0)));
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1, 0, -z),
new Vector3(0, 0, 1), new Vector2((currentbuilding * 2) / imagesInTexture, 1)));

                        //left wall
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x, 0, -z), new
Vector3(-1, 0, 0), new Vector2((currentbuilding * 2) / imagesInTexture, 1)));
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x, 0, -z - 1),
new Vector3(-1, 0, 0), new Vector2((currentbuilding * 2 - 1) / imagesInTexture, 1)));
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z - 1), new Vector3(-1, 0, 0), new Vector2((currentbuilding * 2 - 1)
/ imagesInTexture, 0)));

                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z - 1), new Vector3(-1, 0, 0), new Vector2((currentbuilding * 2 - 1)
/ imagesInTexture, 0)));
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x,
buildingHeights[currentbuilding], -z), new Vector3(-1, 0, 0), new Vector2((currentbuilding * 2) /
imagesInTexture, 0)));
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x, 0, -z), new
Vector3(-1, 0, 0), new Vector2((currentbuilding * 2) / imagesInTexture, 1)));

                        //right wall
```

```
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1, 0, -z),
new Vector3(1, 0, 0), new Vector2((currentbuilding * 2) / imagesInTexture, 1)));
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z - 1), new Vector3(1, 0, 0), new Vector2((currentbuilding * 2 - 1)
/ imagesInTexture, 0)));
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1, 0, -z -
1), new Vector3(1, 0, 0), new Vector2((currentbuilding * 2 - 1) / imagesInTexture, 1)));

                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z - 1), new Vector3(1, 0, 0), new Vector2((currentbuilding * 2 - 1)
/ imagesInTexture, 0)));
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1, 0, -z),
new Vector3(1, 0, 0), new Vector2((currentbuilding * 2) / imagesInTexture, 1)));
                        verticesList.Add(new VertexPositionNormalTexture(new Vector3(x + 1,
buildingHeights[currentbuilding], -z), new Vector3(1, 0, 0), new Vector2((currentbuilding * 2) /
imagesInTexture, 0)));
                    }
                }
            }

             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();
        }
    }
}
```