Although the programming of an algorithm to build a 3D L-system city is beyond the scope of this book, it is possible using Perlin noise and existing building models to generate a city. This is the topic of the next workshop.

On the Web

Procedural City Generator

The before mentioned techniques for procedurally generating a city come from the research work of Müller and Parish. Their approach has been packaged into the *CityEngine* software, a 3D-modeling package purposely built to generate vast city landscapes. A free trial can be downloaded from http://www.procedural.com/.

Unity Hands On

Procedural Cities

Step 1: Create a new Unity project and import the Character Controller package.

Step 2: Add a plane to the scene and an FPC. Position the FPC just above the plane and add a directional light as shown in Figure 7.13. Scale the plane's *x* and *z* sizes to 10.

Step 3: Download *Chapter Seven/Perlin.cs* from the website. Add the file to your Project inside a new folder called *Plugins*.

Step 4: Create a C# file named *MakeCity*, reuse the code from Listing 7.3 changing the class name to *MakeCity*.

Step 5: Play. What you should have is the heightened terrain from the previous example. Now instead of heights, we will add buildings.

Step 6: If you put a print statement inside the for loop in *MakeCity.cs* and print out the height values being assigned to the plane, you will notice that they range between −4 and 4. We will use these values to assign buildings instead of terrain heights. Go to TurboSquid and download eight models of various types of houses and buildings. You will also find eight building models already downloaded from TurboSquid on the website as *Chapter Seven/buildings.zip*.

Step 7: Add the buildings to the Project.

Step 8: Add each building into the Scene and position them on the plane. Because different artists created them, their scaling will be different. Resize and rotate each building as you see fit, such that they have reasonable relative sizes and orientations.

Step 9: Create eight prefabs in the Project—one for each building. Drag each building out of the Hierarchy and onto its own prefab.

Step 10: Modify *MakeCity.cs* to that in Listing 7.4.

Listing 7.4 Using building prefabs to create a city based on Perlin noise height values

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MakeCity : MonoBehaviour {

     public GameObject[] buildings;

     void Start()
     {
         Perlin surface = new Perlin();

         Mesh mesh = this.GetComponent<MeshFilter>().mesh;
         Vector3[] vertices = mesh.vertices;
         float scalex = this.transform.localScale.x;
         float scalez = this.transform.localScale.z;

         for (int v = 0; v <vertices.Length; v++)
         {
             float perlinValue = surface.
                  Noise(vertices[v].x * 2 + 0.1365143f,
                  vertices[v].z * 2 + 1.21688f) * 10;

             perlinValue =
             Mathf.Round((Mathf.
                 Clamp(perlinValue,0,buildings.Length)));
             Instantiate(buildings[(int)perlinValue],
                 new Vector3(vertices[v].x * scalex,
                     vertices[v].y, vertices[v].z * scalez),
                 buildings[(int)perlinValue].transform.
                     rotation);

         }
         mesh.vertices = vertices;
         mesh.RecalculateBounds();
         mesh.RecalculateNormals();

         this.gameObject.AddComponent<MeshCollider>();
     }
}
```

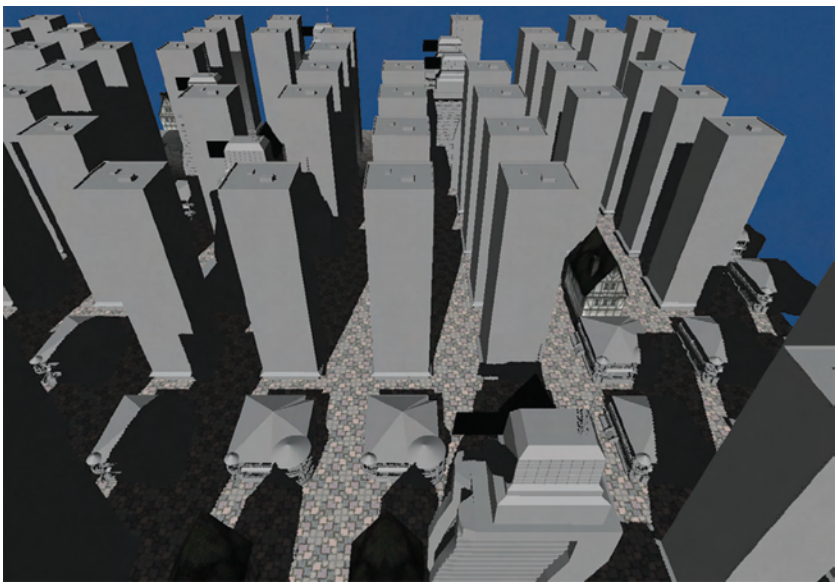

FIG 7.22 A procedurally generated city.

Step 11: Select the plane in the Hierarchy and find the MakeCity script in the Inspector. Set the size of Buildings to 8 and drag and drop each building prefab onto the exposed building array elements in the script. Note that you can have any number of buildings and the code will adjust for it.
Step 12: Play. The result will be a small city as illustrated in Figure 7.22.

Note
As Perlin noise creates smooth regions that when drawn as a grayscale graduate between white and black, you can use these values to determine the city density at these locations. For example, white might represent high density and black low or no density. Where the map is densest, use the skyscraper type building models and where it is lower use small houses. This will create a relatively realistic city.

7.3.4 Infinite Terrain

Infinite terrain or endless terrain is a form of procedurally generated landscape. It uses a mathematical equation and the player's current position to create the landscape as the player moves. Small parts of the map are created as needed based on the player's visible distance. Any map outside the visible range is not generated, and therefore not a load on computer memory.

To create undulation of a terrain, a mathematical formula is used for determining the height based on the *x* and *z* positions on the terrain for which there will always be a *y* value no matter the *x* and *z* values. For