# Displacing Vertices on a Mesh at Runtime

For this tutorial we will read each point on a plane and displace it reading from Perlin noise

Over time, we need to scroll our generated noise over time to make it look like its moving.

To do this we will multiply the change in time per frame by a set speed chosen by the user.

```
xOffset += Time.deltaTime * RippleRate;
yOffset += Time.deltaTime * RippleRate;
```

For each vertex in our mesh, we will call a function to displace each vertex along the y axis using its position on the x and z axis and relate that to a sheet of Perlin noise.

We will then multiply the result given by the Perlin noise by RipplePower to increase the max amplitude of our waves.

```
for (int i = 0; i < vertices.Length; i++)
{
    vertices[i].y = GenerateNoise(vertices[i].x, vertices[i].z) * RipplePower;
}
```

In our noise function, we will be reading from a 2D texture so we can call our input x and y rather than x and z.

First we need a point to read from the generated noise calculated from xCoord and yCoord. Then we can read the value of the pixel at that coordinate which will range between 1 and 0.

```
1 reference
float GenerateNoise(float x,float y)
{
    float xCoord = x * NoiseScale + xOffset;
    float yCoord = y * NoiseScale + yOffset;

    return Mathf.PerlinNoise(xCoord, yCoord);

}
```

Finally we add a delay of one frame between updating the mesh.

```
yield return new WaitForSeconds(Time.deltaTime);
if (Repeat)
    UpdateMesh();
```

Like before, updating the mesh clears the data about the mesh and recalculates it to get the new shape of the mesh.

```
2 references
void UpdateMesh()
{
    mesh.Clear();
    mesh.vertices = vertices;
    mesh.triangles = triangles;
    mesh.RecalculateNormals();

    StartCoroutine(PlaneNoise(true));
}
```

Your new mesh should look something like this: