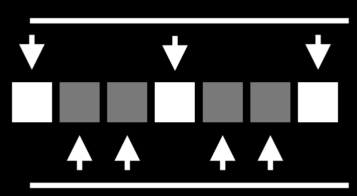
// ProceduralTerrain

Lower frequency = more distance



Sampled

Not sampled

```
float height11 = 0f;

float amplitude = 1f;
float frequency = 1f;

for (int i = Octaves; i > 0; i--) {
  float octave_x0 =  x        / Scale * frequency;
  float octave_z0 =  z        / Scale * frequency;
  float octave_x1 = (x + 1f) / Scale * frequency;
  float octave_z1 = (z + 1f) / Scale * frequency;

  height00 += Mathf.PerlinNoise(octave_x0, octave_z0) * amplitude;
  height01 += Mathf.PerlinNoise(octave_x0, octave_z1) * amplitude;
  height10 += Mathf.PerlinNoise(octave_x1, octave_z0) * amplitude;
  height11 += Mathf.PerlinNoise(octave_x1, octave_z1) * amplitude;

  amplitude *= Persistance;
  frequency *= Lacunarity;
}

int x0 =  x       * CellSize;
```

// ProceduralTerrain

```
float height11 = 0f;

float amplitude = 1f;
float frequency = 1f;

for (int i = Octaves; i > 0; i--) {
  float octave_x0 =  x       / Scale * frequency;
  float octave_z0 =  z       / Scale * frequency;
  float octave_x1 = (x + 1f) / Scale * frequency;
  float octave_z1 = (z + 1f) / Scale * frequency;

  height00 += Mathf.PerlinNoise(octave_x0, octave_z0) * amplitude;
  height01 += Mathf.PerlinNoise(octave_x0, octave_z1) * amplitude;
  height10 += Mathf.PerlinNoise(octave_x1, octave_z0) * amplitude;
  height11 += Mathf.PerlinNoise(octave_x1, octave_z1) * amplitude;

  amplitude *= Persistance;
  frequency *= Lacunarity;
}

int x0 =  x       * CellSize;
```

We process each of our octaves in turn