Computer Graphics

Lecture 11 – Procedural Geometry

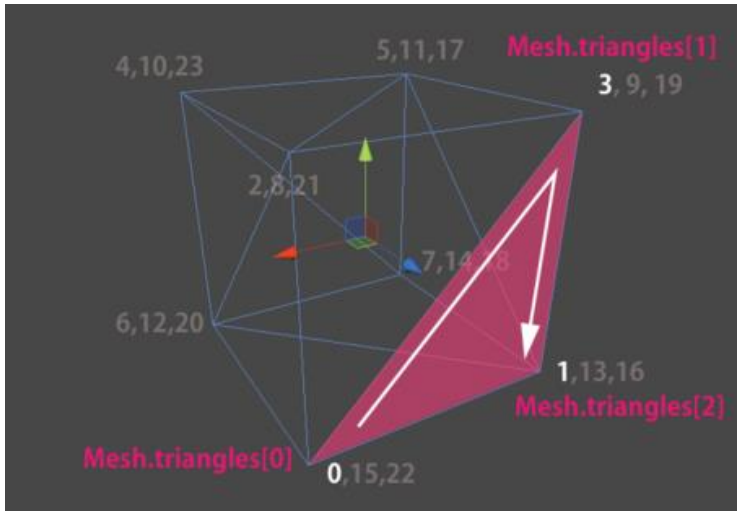
Edirlei Soares de Lima
<edirlei.lima@universidadeeuropeia.pt>

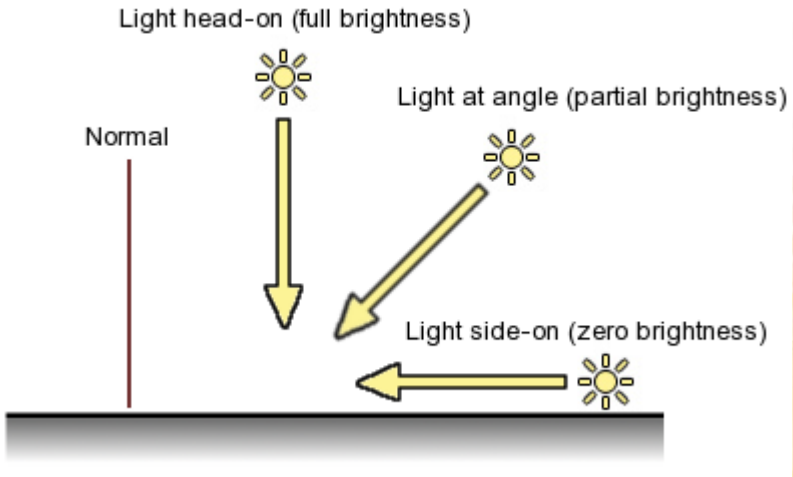
Procedural Content Generation

- Procedural content generation is the method of creating data algorithmically.
  - In computer graphics, it is commonly used to create textures and 3D models.
  - In video games, it is used to automatically create several kinds of content: vegetation, buildings, sound, textures, indoor maps, outdoor maps, ecosystems, road networks, urban environments, puzzles, levels, stories, ...

Procedural Geometry in Unity

- 3D objects are represented by a **set of triangles**. Each triangle is defined by **three vertices**. In addition, each vertex has a **normal vector**, which usually points outward (perpendicular to the mesh surface), and a **UV coordinate** for texture mapping.
  - The vertices and triangles define the spatial structure of the object.
  - The normals define how light affects the surface of the object.
  - The UV coordinates define how textures are mapped to the surface.

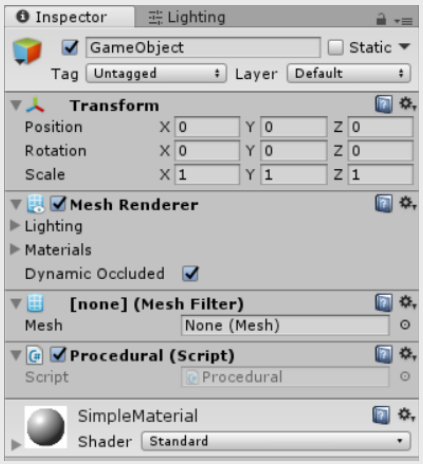


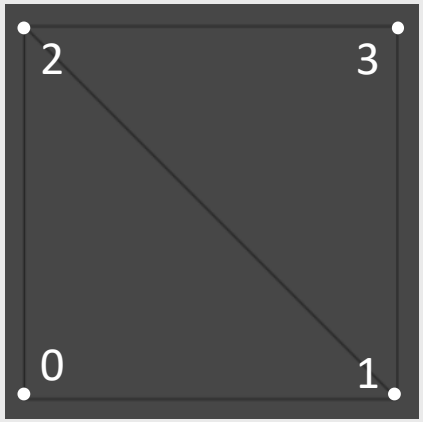


Procedural Geometry in Unity

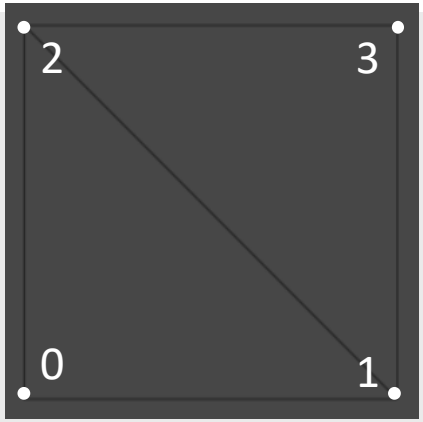
- Creating a procedural quad:

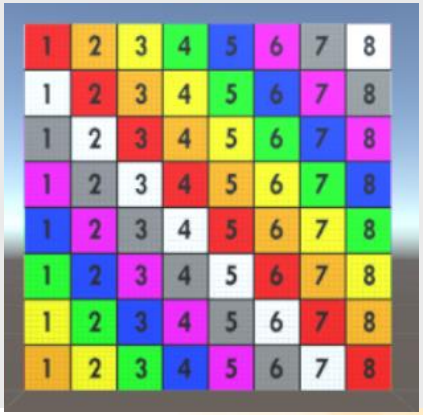
```
public class ProceduralQuad : MonoBehaviour {

  private Mesh mesh;

  void Start()
  {
    mesh = new Mesh();
    mesh.name = "MyQuad";
    GetComponent<MeshFilter>().mesh = mesh;

    Vector3[] vertices = new Vector3[4];
    vertices[0] = new Vector3(0, 0, 0);
    vertices[1] = new Vector3(1, 0, 0);
    vertices[2] = new Vector3(0, 1, 0);
    vertices[3] = new Vector3(1, 1, 0);
    mesh.vertices = vertices;

    ...
```





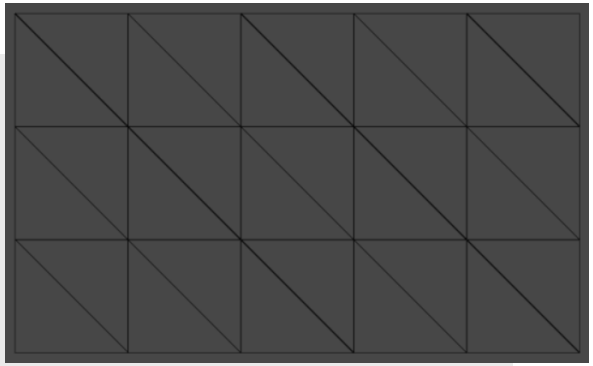
```
    int[] triangles = new int[6];
    triangles[0] = 0;
    triangles[1] = 2;
    triangles[2] = 1;
    triangles[3] = 2;
    triangles[4] = 3;
    triangles[5] = 1;
    mesh.triangles = triangles;

    Vector3[] normals = new Vector3[4];
    normals[0] = -Vector3.forward;
    normals[1] = -Vector3.forward;
    normals[2] = -Vector3.forward;
    normals[3] = -Vector3.forward;
    mesh.normals = normals;

    Vector2[] uv = new Vector2[4];
    uv[0] = new Vector2(0, 0);
    uv[1] = new Vector2(1, 0);
    uv[2] = new Vector2(0, 1);
    uv[3] = new Vector2(1, 1);
    mesh.uv = uv;
  }
}
```





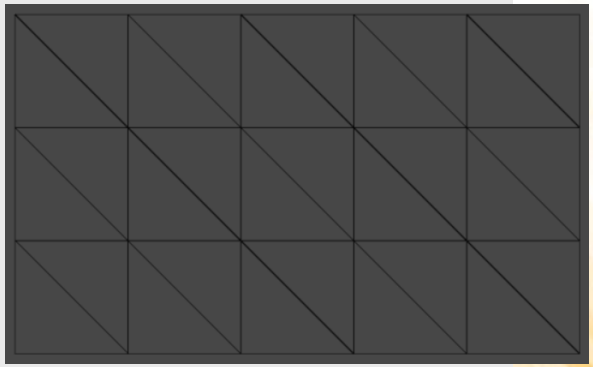
Procedural Geometry in Unity

- Creating a procedural grid:



```
public class ProceduralGrid : MonoBehaviour {
  private Mesh mesh;
  public int xSize = 5;
  public int ySize = 3;

  void Start(){
    mesh = new Mesh();
    GetComponent<MeshFilter>().mesh = mesh;
    mesh.name = "MyGrid";
    Vector3[] vertices = new Vector3[(xSize + 1) * (ySize + 1)];
    int vetCount = 0;
    for (int y = 0; y <= ySize; y++){
      for (int x = 0; x <= xSize; x++){
        vertices[vetCount] = new Vector3(x, y, 0);
        vetCount++;
      }
    }
    mesh.vertices = vertices;
```

```
int[] triangles = new int[(xSize * ySize) * 6];
int triCont = 0;
int vertIndex = 0;
for (int y = 0; y < ySize; y++){
  for (int x = 0; x < xSize; x++){
    triangles[triCont] = vertIndex;
    triangles[triCont + 1] = vertIndex + xSize + 1;
    triangles[triCont + 2] = vertIndex + 1;
    triangles[triCont + 3] = vertIndex + 1;
    triangles[triCont + 4] = vertIndex + xSize + 1;
    triangles[triCont + 5] = vertIndex + xSize + 2;
    triCont += 6;
    vertIndex++;
  }
  vertIndex++;
}
mesh.triangles = triangles;
```



```
    Vector3[] normals = new Vector3[vertices.Length];

    for (int i = 0; i < vetCount; i++){
      normals[i] = -Vector3.forward;
    }

    mesh.normals = normals;

    Vector2[] uv = new Vector2[vertices.Length];
    vetCount = 0;
    for (int y = 0; y <= ySize; y++){
      for (int x = 0; x <= xSize; x++){
        uv[vetCount] = new Vector2((float)x/xSize, (float)y/ySize);
        vetCount++;
      }
    }
    mesh.uv = uv;
  }
}
```

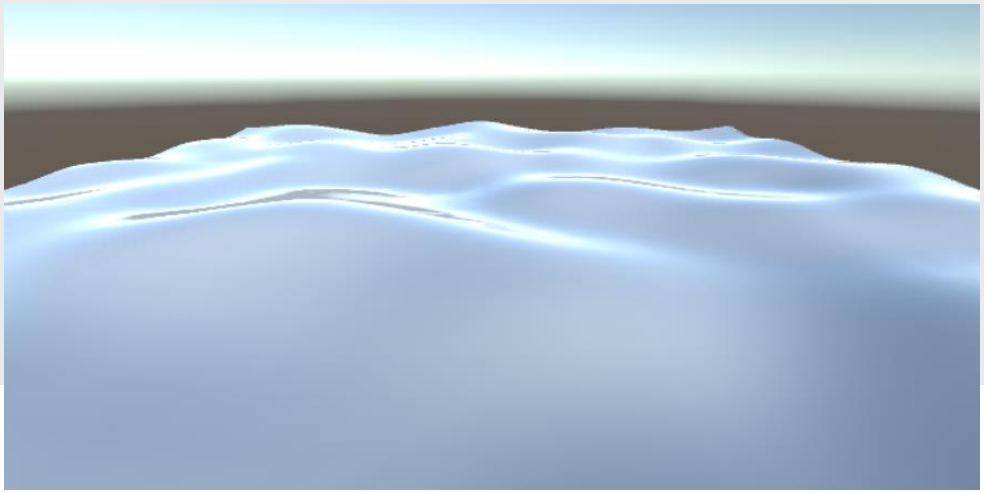
Procedural Terrain Using Perlin Noise

- Procedural Terrain:

```
public class ProceduralTerrain : MonoBehaviour {
  public int size = 50;
  public int resolution = 512;
  public float refinement = 0.01f;
  public float heightScale = 0.04f;
  public float sampleModifier = 0f;

  void Start() {
    float[,] heightData = new float[resolution, resolution];
    Terrain terrain = GetComponent<Terrain>();
    for (int x = 0; x < resolution; x++){
      for (int y = 0; y < resolution; y++){
        float noise = Mathf.PerlinNoise(
                              sampleModifier + (x * refinement),
                              sampleModifier + (y * refinement));
        heightData[x, y] = noise * heightScale;
      }
    }
```

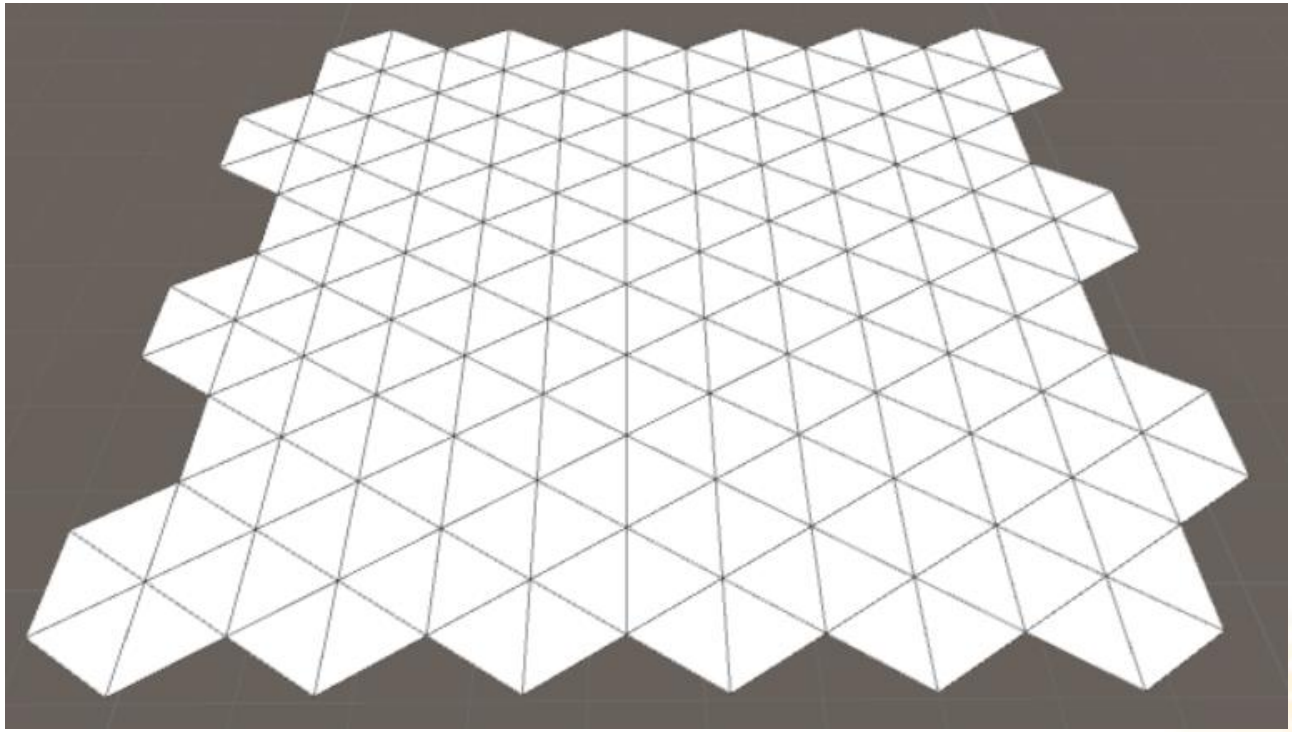
Procedural Terrain Using Perlin Noise

```
    TerrainData terrainData = new TerrainData();
    terrainData.heightmapResolution = resolution + 1;
    terrainData.size = new Vector3(size, size, size);
    terrainData.baseMapResolution = 1024;
    terrainData.SetDetailResolution(1024, 8);
    terrainData.SetHeights(0, 0, heightData);
    terrain.terrainData = terrainData;

    TerrainCollider terrainCollider = GetComponent
                                          <TerrainCollider>();
    terrainCollider.terrainData = terrainData;
  }
}
```



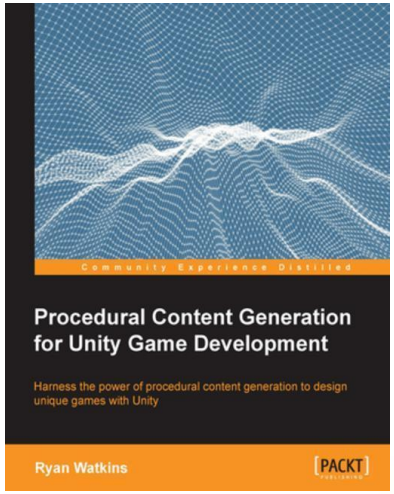
Exercise 1

1) Create a script to generate hexagonal grids.
   - The size of the grid must be defined by a parameter.
   - Example of a 6 x 6 grid:



Further Reading

- Watkins R. (2016). **Procedural Content Generation for Unity Game Development.** Packt Publishing. ISBN: 978-1785287473.



- **Web:**
  - http://catlikecoding.com/unity/tutorials/procedural-grid/
  - http://catlikecoding.com/unity/tutorials/rounded-cube/
  - http://catlikecoding.com/unity/tutorials/ (All Hex Map Tutorials)