# Unit 1 - Lesson 4. Fun Fractal (Mesh, material, shader, enumerator, recursion)

**Aim:**

* What are the major components in Unity game and how do we use mesh, enum and recursion?

**Objectives:** After the lesson, students should be able to:

* Obtain understanding of basic components of a unity game
* Obtain understanding of mesh, material and shader
* Obtain understanding of enumerator and related methods
* Use recursion in unity games

**CLASS PROCEDURE:**

***Do Now:*** What is a fractal? Work with your partner and find the following images of fractals online:

1. A fractal made of triangles
2. A fractal made of cubes

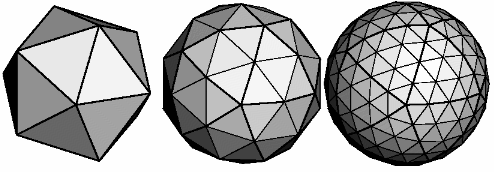
***Class Discussion / Presentation:***

1. How do we create 3-D fractal images in Unity using a C# script?

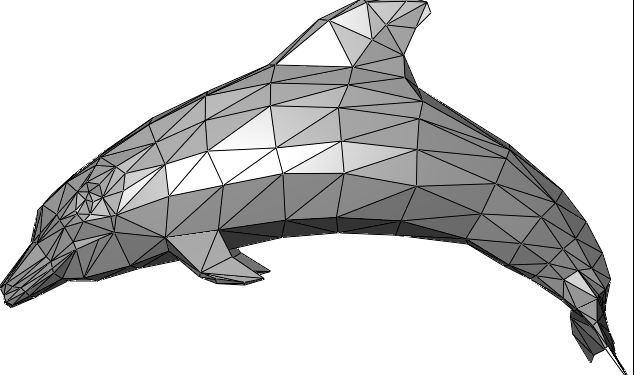
* What is the recursive process?
* How do we design the recursion?

1. What is a mesh?

* Well, how do you think the computer graphic software display a 3-D sphere? It’s through a collection of points in 3-D space plus a set of triangles! Why triangles? Since they are the most basic 2-D shapes!



* A Mesh is a construct used by the graphics hardware to draw complex stuff. It is a 3-D object that contains points and triangles. The triangles constitute the surface of whatever the mesh represents.
* Mesh can be imported into Unity, can be contained in Unity’s default shapes, or be generated by code.



1. What is a material?

* Materials are used to define the visual properties of objects. They can range from very simple, like a constant color, to very complex.
* Materials consist of a shader and whatever data the shader needs. Shaders are basically scripts that tell the graphics card how an object's polygons should be drawn.
* The standard diffuse shader uses a single color and optionally a texture, along with the light sources in the scene, to determine the appearance of polygons.

1. How do we use gameObject.AddComponent?

* The AddComponent method creates a new component of a certain type, attached it to the game object, and returns a reference to it.

[MeshFilter](http://docs.unity3d.com/Documentation/ScriptReference/MeshFilter.html) filter = gameObject.AddComponent<[MeshFilter](http://docs.unity3d.com/Documentation/ScriptReference/MeshFilter.html)>();

filter.mesh = mesh;

1. What is the order of invoking methods in Unity?

Initialize() 🡪 Start() 🡪 Update()

1. What is an enumerator (enum)?

* Enumeration is the concept of going through some collection one item at a time, like looping over all elements in an array. An enumerator – or iterator – is an object that provides an interface for this functionality. System.Collections.[IEnumerator](http://social.msdn.microsoft.com/search/en-us?query=IEnumerator) describes such an interface.

1. What does yield do and how do we use yield?

* The yield statement is used by iterators to make life easy for them. To make enumeration possible, you'd need to keep track of your progress. This involves some boilerplate code that is essentially always the same. What you'd really want is to just write something like return firstItem; return secondItem; until you are done. The yield statement allows you to do exactly that.
* When you're using yield, an enumerator object is created behind the scenes to take care of the tedious bits. That's why our CreateChildren method has [IEnumerator](http://social.msdn.microsoft.com/search/en-us?query=IEnumerator) as its return type.

1. How do coroutines work?

* When you're creating a coroutine in Unity, what you're really doing is creating an iterator. When you pass it to the StartCoroutine method, it will get stored and gets asked for its next item every frame, until it is finished.
* The yield statements produce the items. The statements in between – the stuff that you want to happen – are side-effects of the iterator doing its job

***Pair – sharing Activity:***

Open the PDF document named “P2. Construct a Fractal”. Read the instructions and then discuss with your partner on how to create a Unity project that can generate 3-D fractals.

***Class Activity:***

Create a Unity project to generate 3-D fractals using recursion, coroutines, enum and randomness.

using UnityEngine;

using System.Collections;

public class Fractal : MonoBehaviour {

private static Vector3[] childDirections = {

Vector3.up,

Vector3.right,

Vector3.left,

Vector3.forward,

Vector3.back

};

private static Quaternion[] childOrientations = {

Quaternion.identity,

Quaternion.Euler(0f, 0f, -90f),

Quaternion.Euler(0f, 0f, 90f),

Quaternion.Euler(90f, 0f, 0f),

Quaternion.Euler(-90f, 0f, 0f)

};

public Mesh[] meshes;

public Material material;

public int maxDepth;

public float childScale;

public float spawnProbability;

public float maxRotationSpeed;

public float maxTwist;

private float rotationSpeed;

private int depth;

private Material[,] materials;

private void InitializeMaterials () {

materials = new Material[maxDepth + 1, 2];

for (int i = 0; i <= maxDepth; i++) {

float t = i / (maxDepth - 1f);

t \*= t;

materials[i, 0] = new Material(material);

materials[i, 0].color = Color.Lerp(Color.white, Color.yellow, t);

materials[i, 1] = new Material(material);

materials[i, 1].color = Color.Lerp(Color.white, Color.cyan, t);

}

materials[maxDepth, 0].color = Color.magenta;

materials[maxDepth, 1].color = Color.red;

}

private void Start () {

rotationSpeed = Random.Range(-maxRotationSpeed, maxRotationSpeed);

transform.Rotate(Random.Range(-maxTwist, maxTwist), 0f, 0f);

if (materials == null) {

InitializeMaterials();

}

gameObject.AddComponent<MeshFilter>().mesh =

meshes[Random.Range(0, meshes.Length)];

gameObject.AddComponent<MeshRenderer>().material =

materials[depth, Random.Range(0, 2)];

if (depth < maxDepth) {

StartCoroutine(CreateChildren());

}

}

private IEnumerator CreateChildren () {

for (int i = 0; i < childDirections.Length; i++) {

if (Random.value < spawnProbability) {

yield return new WaitForSeconds(Random.Range(0.1f, 0.5f));

new GameObject("Fractal Child").AddComponent<Fractal>().

Initialize(this, i);

}

}

}

private void Initialize (Fractal parent, int childIndex) {

meshes = parent.meshes;

materials = parent.materials;

maxDepth = parent.maxDepth;

depth = parent.depth + 1;

childScale = parent.childScale;

spawnProbability = parent.spawnProbability;

maxRotationSpeed = parent.maxRotationSpeed;

maxTwist = parent.maxTwist;

transform.parent = parent.transform;

transform.localScale = Vector3.one \* childScale;

transform.localPosition =

childDirections[childIndex] \* (0.5f + 0.5f \* childScale);

transform.localRotation = childOrientations[childIndex];

}

private void Update () {

transform.Rotate(0f, rotationSpeed \* Time.deltaTime, 0f);

}

}