TA 7

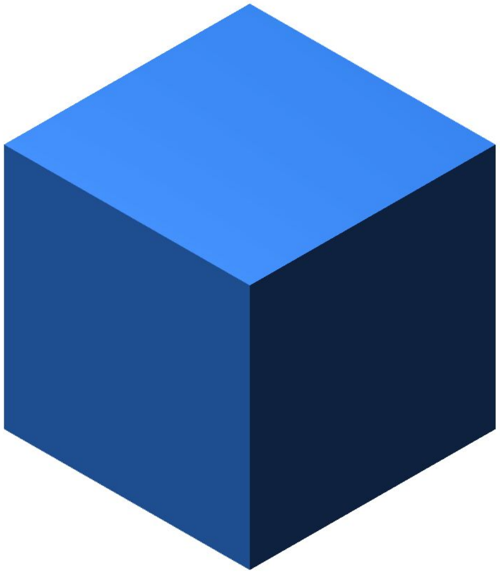
- Catmull-Clark Subdivision
- EX3
- C# Collections and Data Structures

Subdivision Surfaces

Computer Graphics 2020

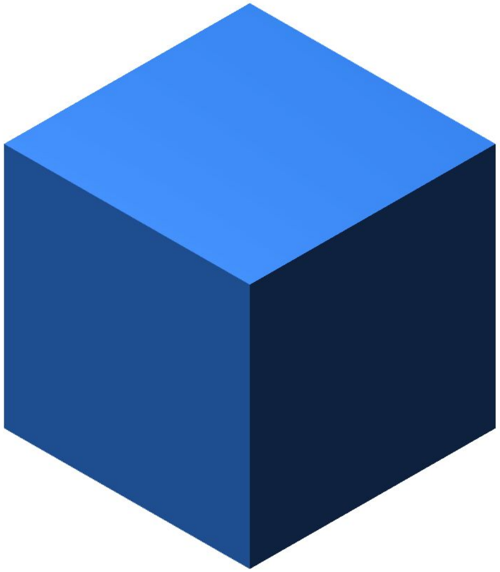
Catmull-Clark Subdivision

- A technique used in 3D computer graphics to create smooth surfaces by using a type of *subdivision surface* modeling



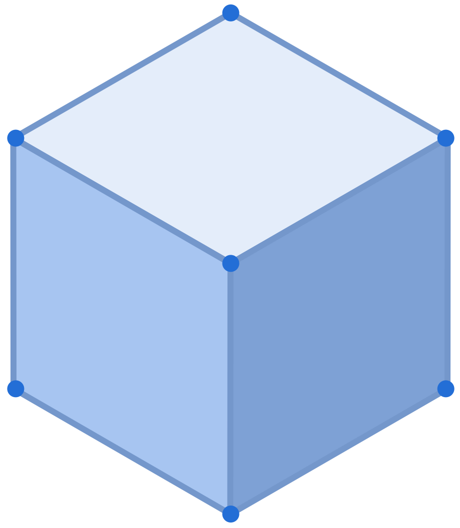
Catmull-Clark Subdivision

- Devised by Edwin Catmull and Jim Clark in 1978
- Edwin Catmull later went on to become president of Pixar and Walt Disney Animation Studios



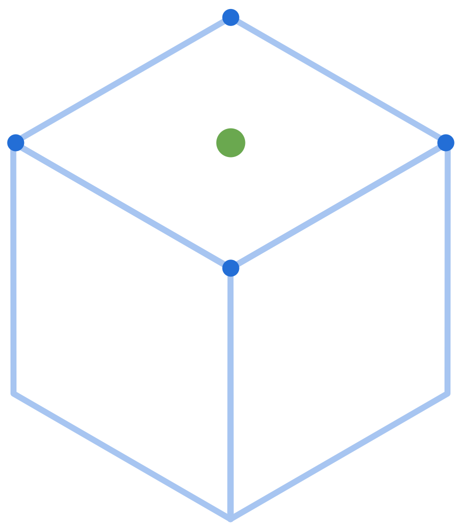
Catmull-Clark Subdivision

- Start with a mesh of an arbitrary polyhedron. All the vertices in this mesh shall be called ***original points***



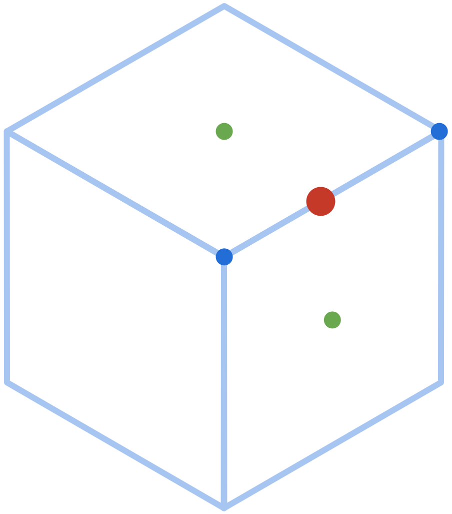
Catmull-Clark Subdivision

- For each face, add a ***face point*** at the average position of all original points of the respective face



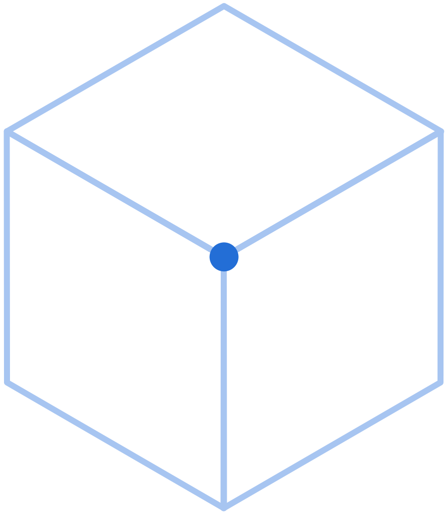
Catmull-Clark Subdivision

- For each edge, add an ***edge point*** at the average of the two neighbouring face points and its two original endpoints



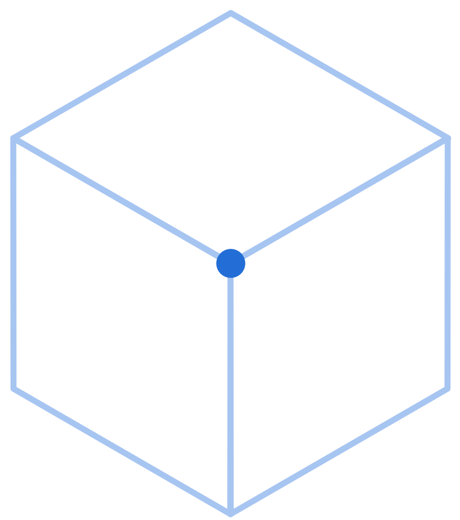
Catmull-Clark Subdivision

- To calculate the new position of each original point, we need to define some averages



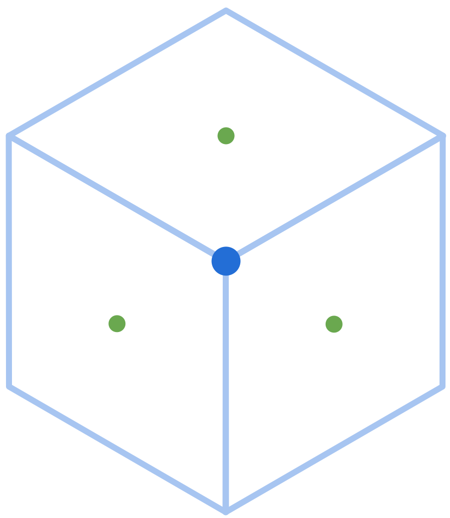
Catmull-Clark Subdivision

- For a vertex p, the number of edges neighboring p is also the number of adjacent faces
- Denote this number n



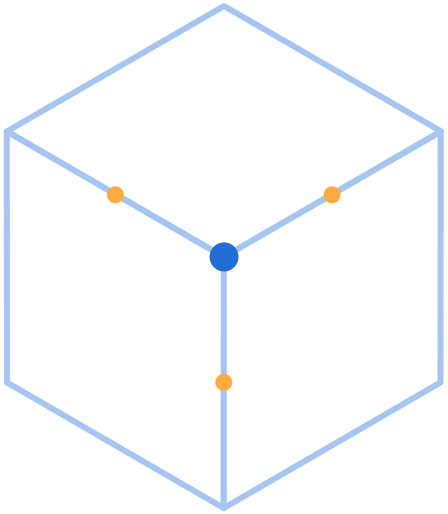
Catmull-Clark Subdivision

- Let f be the average of all n (recently created) face points for faces touching p



Catmull-Clark Subdivision

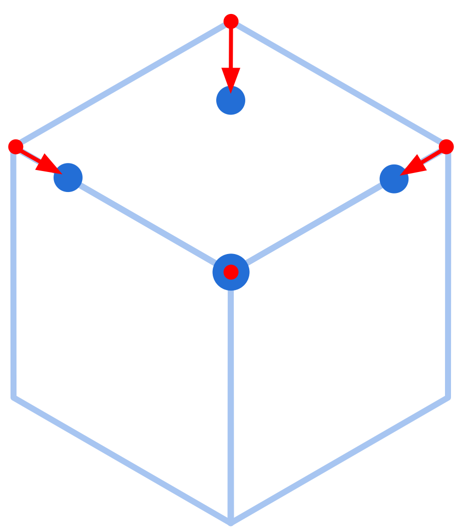
- Let r be the average of all n edge midpoints for (original) edges touching p
- Not to be confused with new edge points!



Catmull-Clark Subdivision

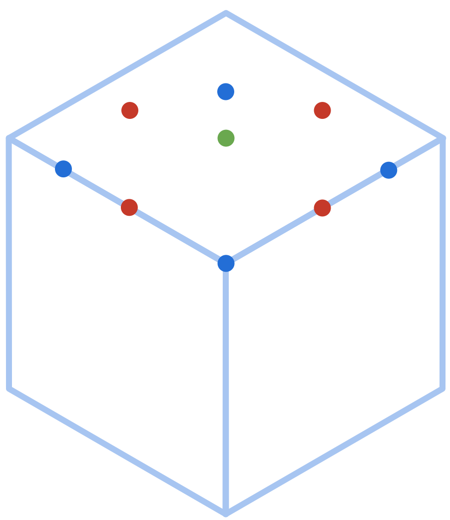
- The new position is given by:

$$\frac{f + 2r + (n-3)p}{n}$$



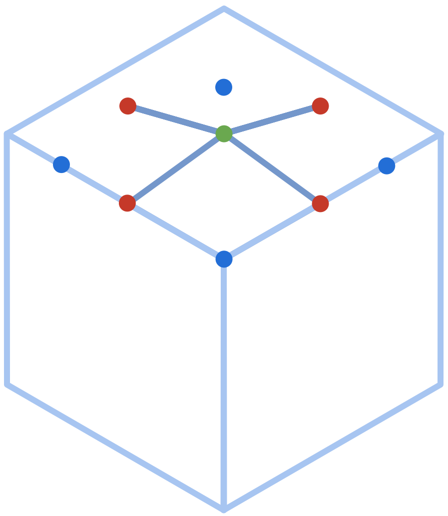
Catmull-Clark Subdivision

- We now have all new vertices in their final positions!



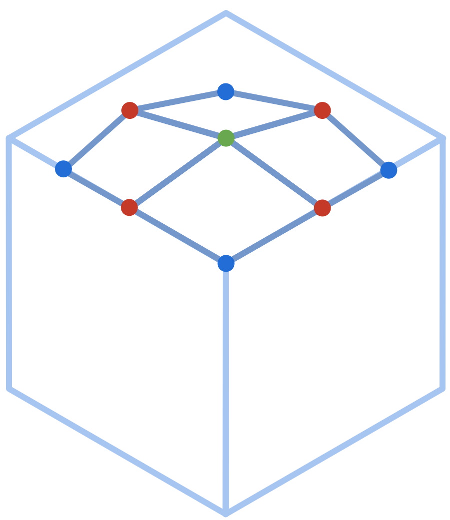
Catmull-Clark Subdivision

- Connect each face point to its corresponding edge points



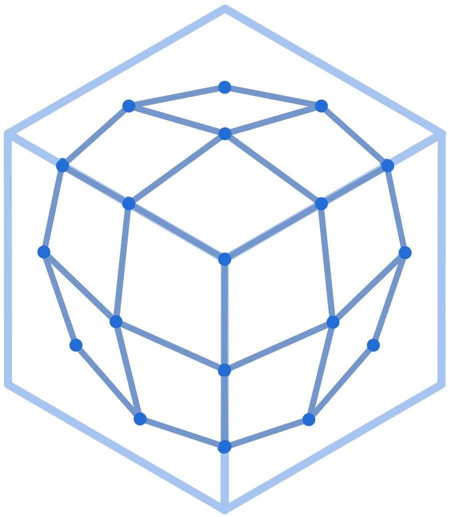
Catmull-Clark Subdivision

- Connect newly positioned original point to the edge points neighboring it



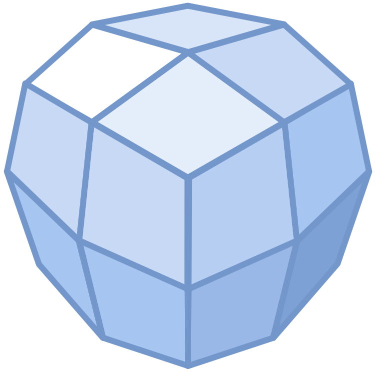
Catmull-Clark Subdivision

- Define new faces as enclosed by edges



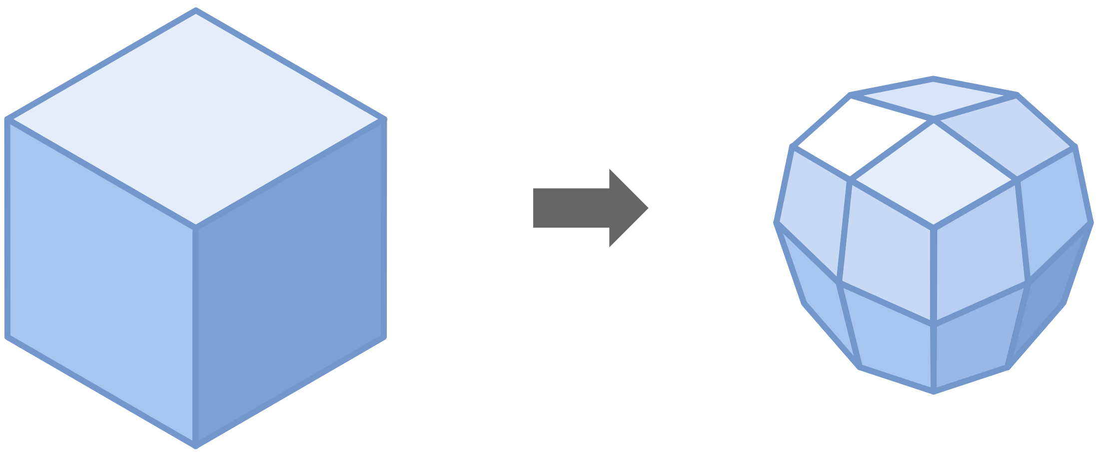
Catmull-Clark Subdivision

- We completed one iteration of Catmull-Clark subdivision!



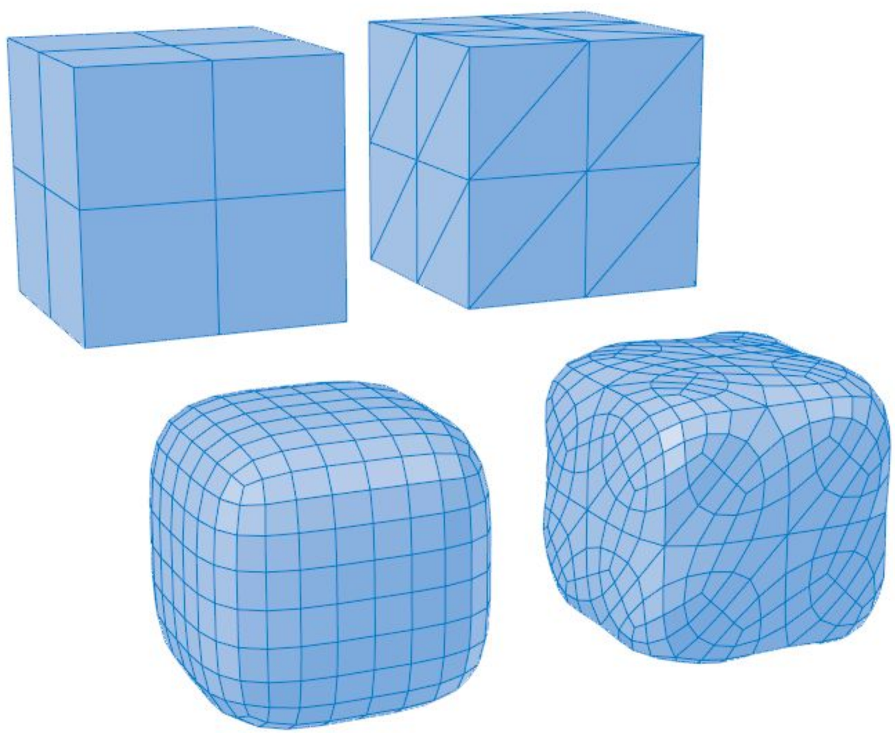
Catmull-Clark Subdivision

- The new mesh will generally look smoother
- The new mesh will consist only of quads, which in general **will not be planar**



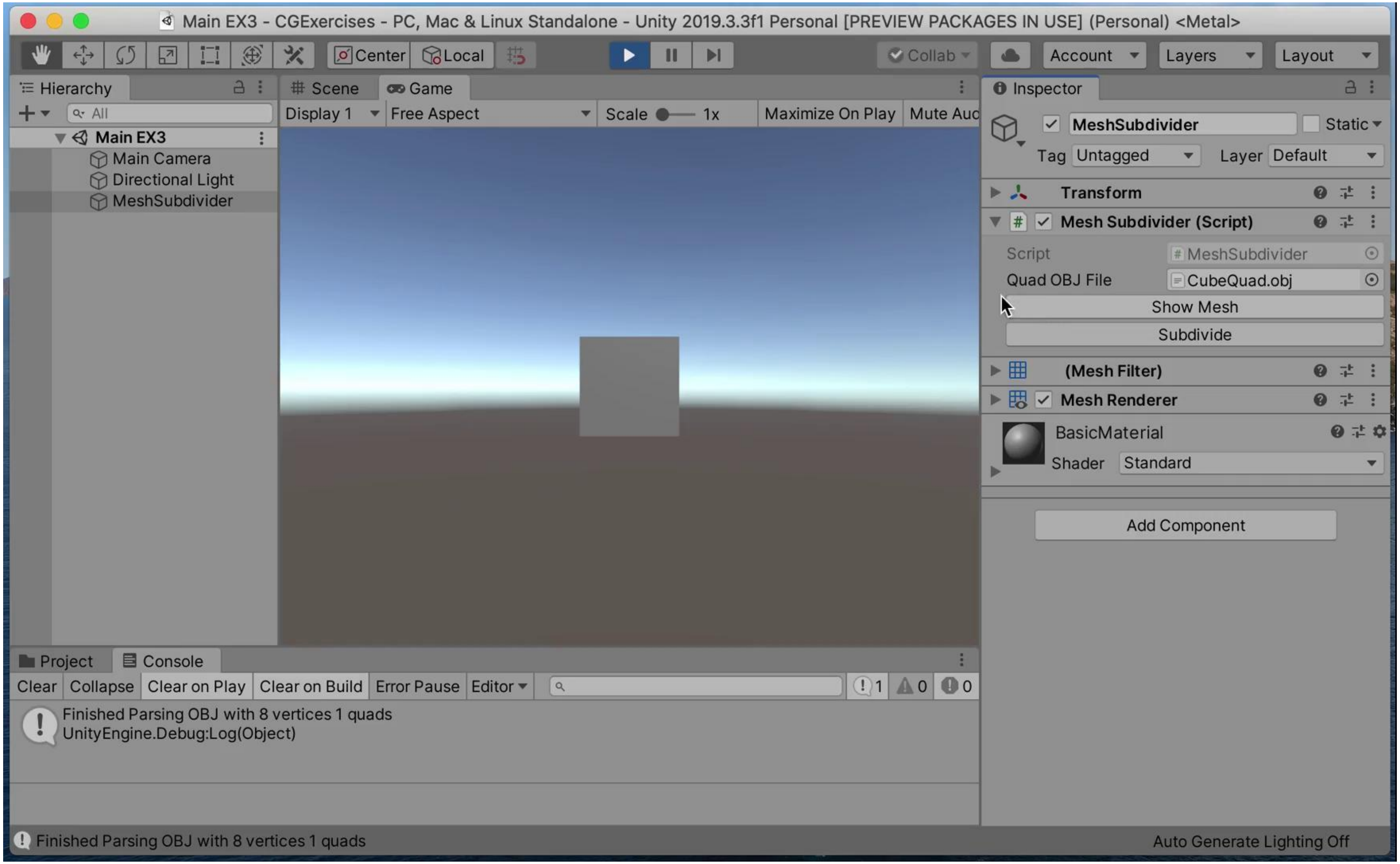
Catmull-Clark Subdivision

- For example, if we split a cube into triangles then subdivide:



EX3

- In this exercise you will implement Catmull-Clark subdivision algorithm
- You will be working only with quad meshes
- You **must** submit this exercise in pairs


Main EX3 - CGExercises - PC, Mac & Linux Standalone - Unity 2019.3.3f1 Personal [PREVIEW PACKAGES IN USE] (Personal) <Metal>
Center
Local
Collab
Account
Layers
Layout
Hierarchy
All
Main EX3
Main Camera
Directional Light
MeshSubdivider
Scene
Game
Display 1
Free Aspect
Scale
1x
Maximize On Play
Inspector
MeshSubdivider
Static
Tag
Untagged
Layer
Default
Transform
Mesh Subdivider (Script)
Script
MeshSubdivider
Quad OBJ File
CubeQuad.obj
Show Mesh
Subdivide
(Mesh Filter)
Mesh Renderer
BasicMaterial
Shader
Standard
Add Component
Project
Console
Clear
Collapse
Clear on Play
Clear on Build
Error Pause
Editor
1
0
0
Finished Parsing OBJ with 8 vertices 1 quads
UnityEngine.Debug:Log(Object)
Finished Parsing OBJ with 8 vertices 1 quads
Auto Generate Lighting Off

EX3

- There are all kinds of ways to implement the algorithm exactly
- Efficiency is important - you may lose points for inefficient code!
- Meshes are data structures - C# provides classes that can help when working with meshes

C# Collections and Data Structures

- C# provides various useful data structures in its `System.Collections.Generic` namespace
- All collections provide methods for adding, removing, or finding items in the collection
- C# Collections docs:

  docs.microsoft.com/en-us/dotnet/standard/collections/

C# Collections and Data Structures

- A few common C# data structures:
  - `List<T>`
  - `Queue<T>`
  - `Stack<T>`
  - `Dictionary<TKey,TValue>`
  - `SortedList<TKey,TValue>`

C# Comparisons and Sorts

- The `System.Collections` classes perform many comparisons when managing collections

- Methods such as `Contains`, `IndexOf` and `Remove` use an ***equality comparer***

- Methods such as `BinarySearch` and `Sort` use an ***ordering comparer***

Equality Comparer

- Subclass of `EqualityComparer<T>`
- Must implement 2 methods:
  - `Equals(T o1, T o2)`
    Returns `true` if `o1` is equal to `o2`, `false` otherwise
  - `GetHashCode(T o)`
    Returns a 32-bit integer representation of the object `o`

Equality Comparer Usage

```
public class Vec3Comparer : EqualityComparer<Vector3>
{
    private static readonly float EPSILON = 0.001f;

    public override bool Equals(Vector3 v1, Vector3 v2) {
        if (Vector3.Distance(v1, v2) < EPSILON) {
            return true;
        }
        return false;
    }

    public override int GetHashCode(Vector3 v) {
        return 0;
    }
}
```

Equality Comparer Usage

```
Vec3Comparer c = new Vec3Comparer();
Dictionary<Vector3, int> d = new Dictionary<Vector3, int>(c);

Vector3 v1 = new Vector3(1f,      1f, 1f);
Vector3 v2 = new Vector3(1.0001f, 1f, 1f);
Vector3 v3 = new Vector3(2f,      2f, 2f);

d.Add(v1, 1);

print(d[v1]); // prints “1”
print(d[v2]); // prints “1”
print(d[v3]); // KeyNotFoundException: key was not present
```

Equality Comparer

- When comparing vectors in Unity *approximately*, the convention is to use `EPSILON = 1e-5f` i.e. `EPSILON = 0.00001f`
- Any vectors of distance $\leqslant$ `EPSILON` from each other can be considered identical
- C# Docs: [EqualityComparer<T> Class]

Ordering Comparer

- For ordering, we just need to implement a comparison method with the signature:

```
int Compare(T o1, T o2)
```

- The method returns:

```
- o1 < o2    ⇒  -1
- o1 == o2   ⇒   0
- o1 > o2    ⇒   1
```

Ordering Comparer Usage

```
public static int Vec3CompareCoordZ(Vector3 v1, Vector3 v2)
{
    if (v1.z < v2.z)
        return -1;
    else if (v1.z == v2.z)
        return 0;
    else
        return 1;
}

// ...
List<Vector3> l = new List<Vector3>();
l.Add(new Vector3(1, 1, 1));
l.Add(new Vector3(1, 1, 2));
l.Add(new Vector3(1, 1, 0)); // [(1,1,1), (1,1,2), (1,1,0)]

l.Sort(Vec3CompareCoordZ);   // [(1,1,0), (1,1,1), (1,1,2)]
```

C# Comparisons and Sorts

- C# ordering comparer documentation:

  docs.microsoft.com/en-us/dotnet/api/system.comparison

- C# general documentation about comparisons and sorts:

  docs.microsoft.com/en-us/dotnet/standard/collections/comparisons-and-sorts-within-collections

Good luck!