HowToUse

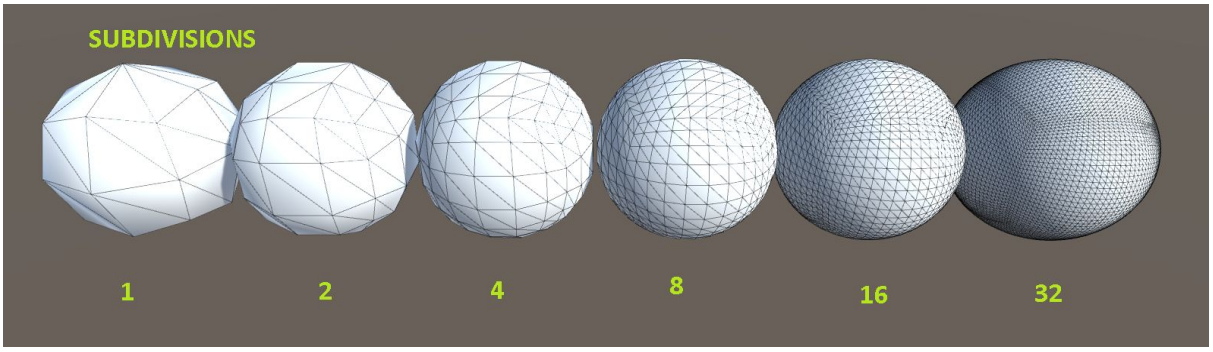
1. Press on the “GameObject” tab, then press “3D Object” → “SphereGenerator” and then one of the spheres of your choosing
2. Change properties to the extend that they will satisfy your needs
3. Done.

The Information that needs to be feeded into the functions/component:

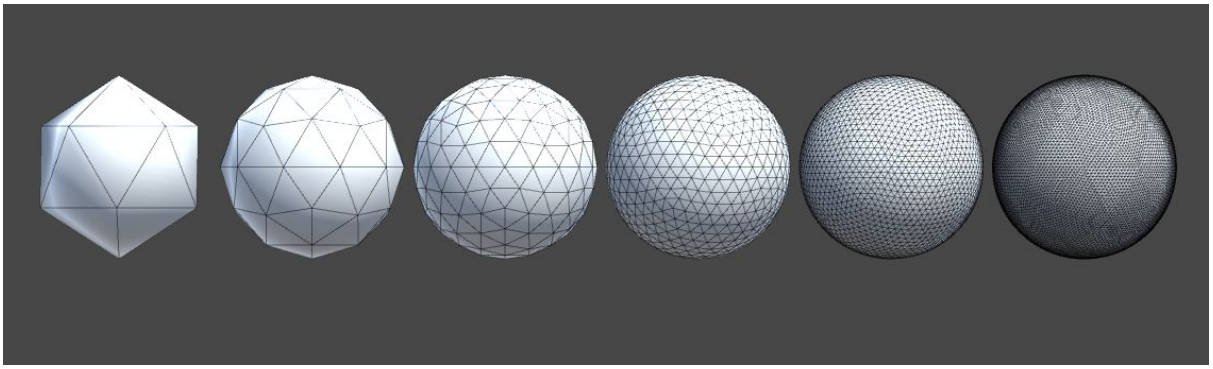
Subdivisions

This is what makes this tool interesting. The IcoSphere needs a subdivision of at least 0 and the NormalizedCubeSphere one of at least 1.

The NormalizedCubeSphere is a cube, whose vertices were normalized which results in a smoother sphere as more subdivisions are added.



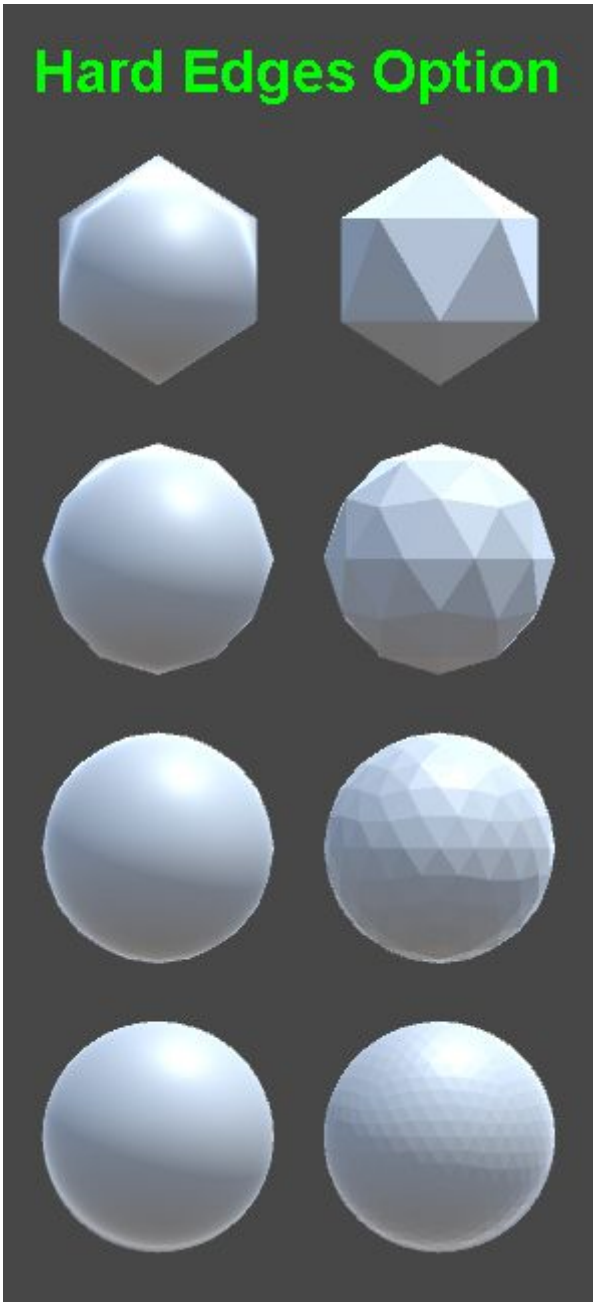
Unlike the NormalizedCubeSphere the number of vertices a lot faster on the IcoSphere with increasing subdivisions so be careful. Because of Unitys MaxVertexCount the maximum subdivision for the Ico-Sphere is 6.



Radius

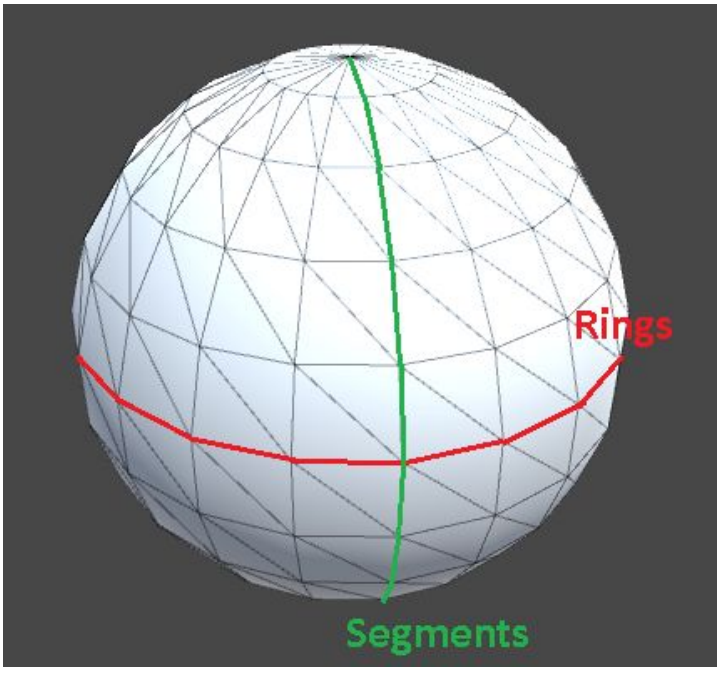
Not really needed, but as a parameter of a sphere I will leave it in case someone might need it.

Hard Edges (only applicable on the IcoSphere)
Hard Edges on the right vs smooth edges on the left



This Option splits the faces of the smoothed option into its vertices and recalculates the normals. This results in an increase in Vertex Count however, thus not allowing for very high subdivision levels due to the ~65 k vertex limit.

Rings And Segments (Only applies to the Pole Sphere)
Pretty much self explanatory...



The Script

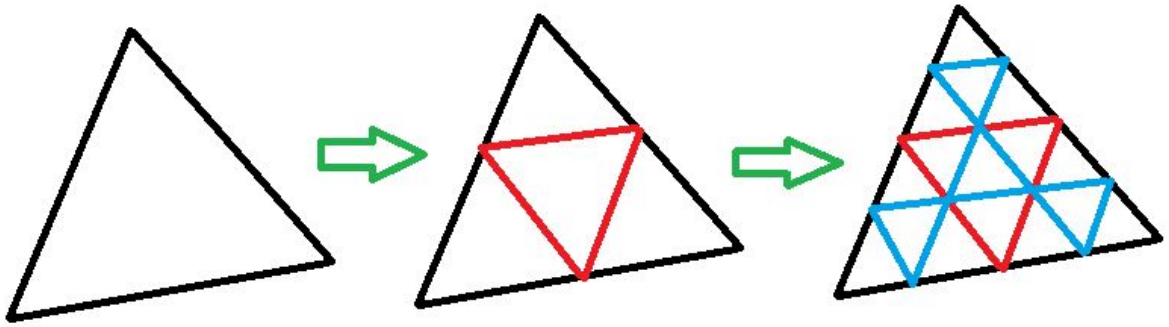
This is for users who want to mess with the code:
I will focus on “generateMesh()” as that is the only interesting function for each Sphere.

IcoSphere

Here I create the base Mesh which will be subdivided. You can see it if you set the subdivisions to 0.

```
//base starting mesh
List<Vector3> vertices = new List<Vector3>(){
    new Vector3(0.000000f, -1.000000f, 0.000000f),
    new Vector3(0.723600f, -0.447215f, 0.525720f),
    new Vector3(-0.276385f, -0.447215f, 0.850640f),
    new Vector3(-0.894425f, -0.447215f, 0.000000f),
    new Vector3(-0.276385f, -0.447215f, -0.850640f),
    new Vector3(0.723600f, -0.447215f, -0.525720f),
    new Vector3(0.276385f, 0.447215f, 0.850640f),
    new Vector3(-0.723600f, 0.447215f, 0.525720f),
    new Vector3(-0.723600f, 0.447215f, -0.525720f),
    new Vector3(0.276385f, 0.447215f, -0.850640f),
    new Vector3(0.894425f, 0.447215f, 0.000000f),
    new Vector3(0.000000f, 1.000000f, 0.000000f)
};
int[] triangles = {
    0, 1, 2,
    1, 0, 5,
    0, 2, 3,
    0, 3, 4,
    0, 4, 5,
    1, 5, 10,
    2, 1, 6,
    3, 2, 7,
    4, 3, 8,
    5, 4, 9,
    1, 10, 6,
    2, 6, 7,
    3, 7, 8,
    4, 8, 9,
    5, 9, 10,
    6, 10, 11,
    7, 6, 11,
    8, 7, 11,
    9, 8, 11,
    10, 9, 11
};
```

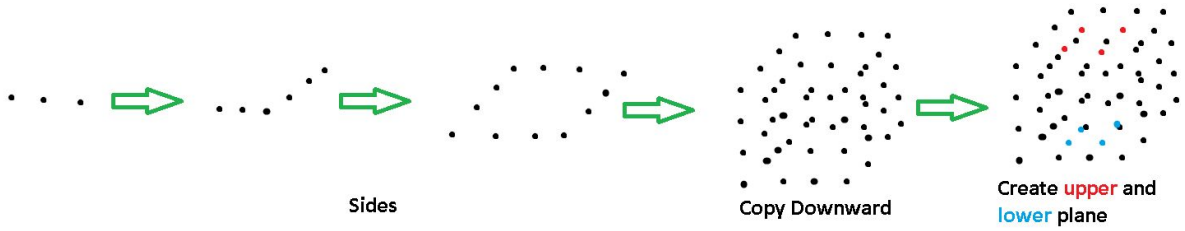
For each subdivision each triangle on the base Mesh will be subdivided, while preventing double/overlapping vertices.



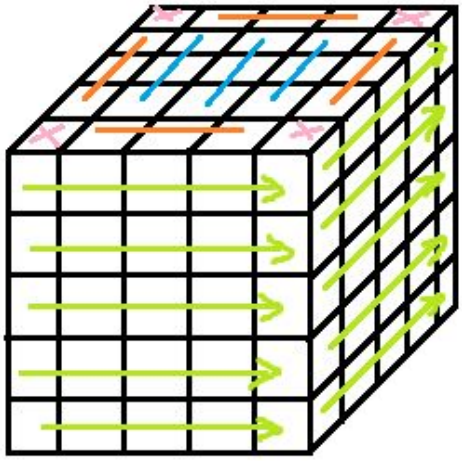
After each subdivision the vertices will be normalized.
This process repeats for each subdivision.

NormalizedCubeSphere:
Unlike the Ico-Sphere the algorithm isn't as smooth here, because it is not that easy to create subdivisions while preventing double vertices.

The first step is creating the vertices on the sides. Then those will be copied downward and then the cube lids will be filled.



After that the triangles will be created. Because of the structure it looks ugly in code, but you won't have to look at it if you have the functions.
The algorithm will start looping around the sides of the cube and going down whenever it finished a loop. Then faces at the edges will be created. The corners and then the center filled. Done.



Triangles at

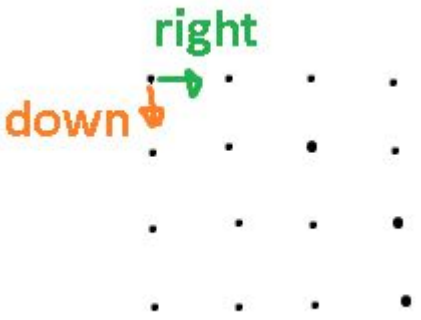
1. sides

2. edges

3.Corners

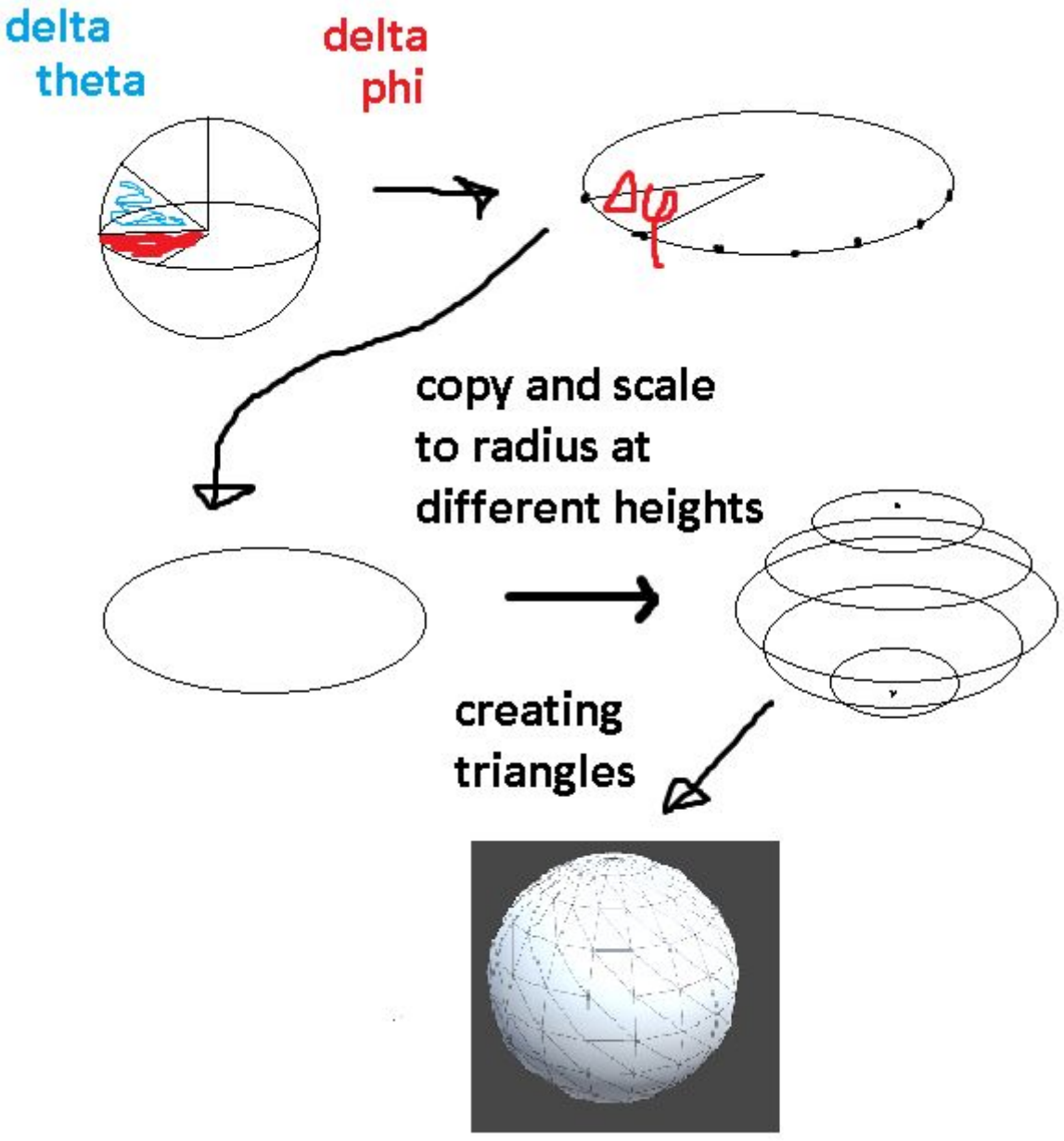
4. lid centers

UVNormalizedCubeSphere

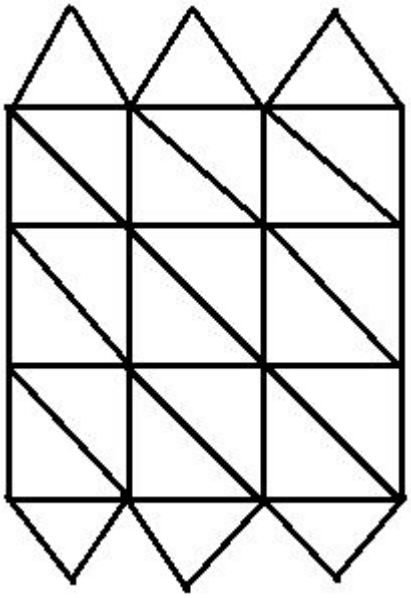


Unlike the NormalizedCubeSphere (which has merged vertices) this one is so simple that I can't really say much about it. Its the same procedure for every face of the cube. and the uvs are the same for each face. Its plausible enough from just looking at the code i think.

PoleSphere



First only an Equator is created. It is then copied along the y Axis, forming a lot of Rings. Connecting them all yields the Sphere.
And this should be the UVs created



SubdividedCubeSphere
Use this one when you need uvs for each face. It also holds the possibility of adding some small uv padding.
The Algorithm: Take a cube, subdivide, normalize, subdivide, normalize …..done;
Unfortunately only up to 6 times, because it reaches > max vertex count at 7; You can friggle with the code and split up the mesh into individual GameObjects if you really need to.

Those are all the Algorithms!

By the time someone of you ask why a certain variable is at that specific point, i will probably have forgotten. The rough idea on how the algorithm works should be enough to work with it though.

If you have questions or something to say, write me to grasbock@gmail.com