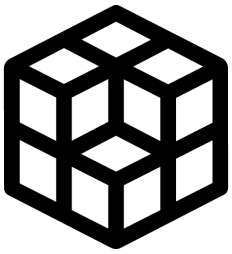
Playing with Meshes

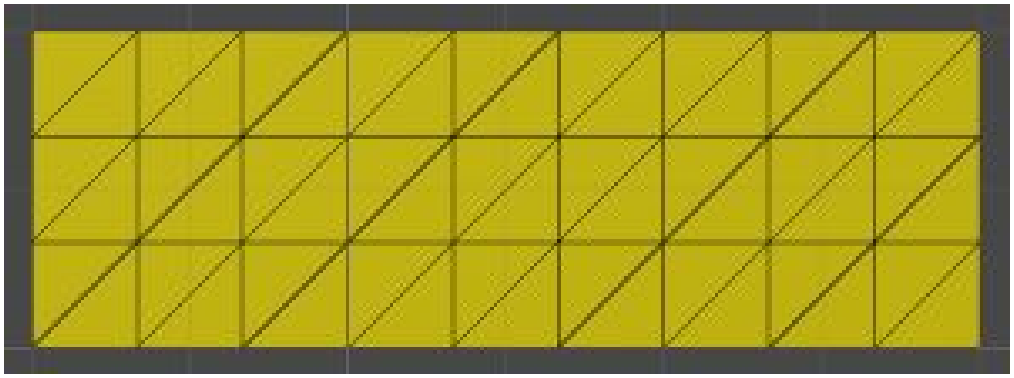
Part I - Vertices and triangles



In this document I'll explain how to create a plane from a script using Unity 3D. During this process I'll teach how vertices and triangles are used to create a Mesh. This is the first tutorial of a series that I will be making on 3D geometry.

Results

This tutorial is a intro to meshes and how to construct them. It will cover, vertices, triangles and mesh density. The result of this tutorial is a script that is able to create plane meshes at runtime. It's possible to define the mesh length and also its density (amount of triangles and vertices).

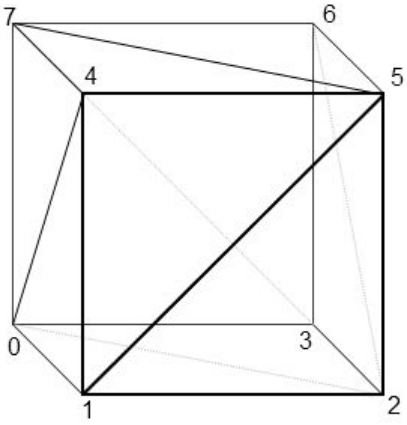


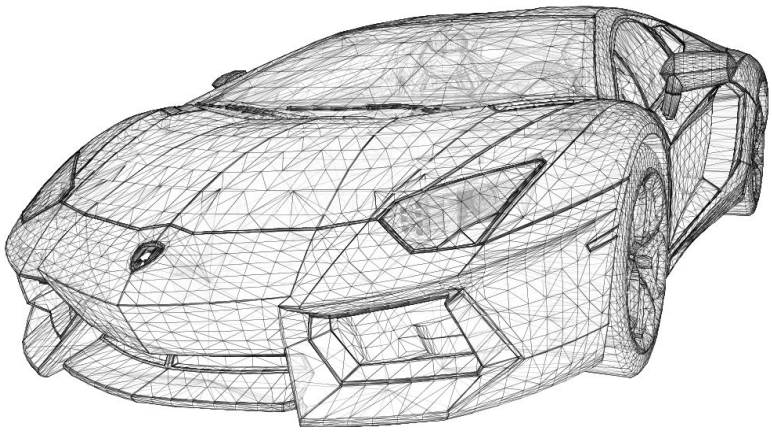
Screenshot of the generated plane.

Index

Brief Intro!

Essentially a mesh is a set of points (called **vertices**) that are arranged to draw things. Those points are linked together by **edges** making a wireframe. They can be as simple as a cube with just 8 vertices (0-7) or as complex as the Ferrari with several vertices.





When 3 edges are combined a triangle is created, it's possible to construct different kinds of shapes through the combination of triangles. Triangles are important because they form a minimum surface, there is no way to create a surface without at least 3 vertices. In that way all surfaces (also called faces) can be broken into several triangles. The objects above are all composed of triangles.

Now that you know the basics let's start creating plane through code in Unity3D. You will need a fresh Unity Project, I'm using for this example Unity 2017.1.0f3.

Getting started

After opening an empty project in Unity, create an empty game object and call it *'Plane'*. On this object, add an empty script named *'Plane'*, paste the code below in it.

```
[RequireComponent(typeof(MeshFilter), typeof(MeshRenderer))]
public class Plane : MonoBehaviour
{
   /*
        # Plane #
        This class represent a plane. This plane is rectangular and made of vertices.
    */

    /* The length on the X direction. */
```

```
    public int Lenght_X;

    /* The length on the Y direction. */
    public int Lenght_Y;

    /* Amount of vertices on the X direction. */
    private int VerticesOn_X;

    /* Amount of vertices on the Y direction. */
    private int VerticesOn_Y;

    /* A array to store vertices. */
    private Vector3[] vertices;

    /* The plane mesh */
    private Mesh mesh;

    /* Generate the Mesh on Awake. */
    void Awake()
    {
        createMesh();
    }

    /* This method will create the mesh. */
    private void createMesh(){}
}
```

This code has all the variables we need. Now we must implement the CreateMesh method. That will be described by parts on the sections: '*The Vertices*', '*The Triangles*' and '*Vertex Density*'.

The Vertices

Let's create an array to store the vertices the plane is going to have, the bigger the amount the more complex the plane is.

The vertical and horizontal length defined on the snipped above, is how big the plane is going to be on the X and Y axis. **For this case we will consider that the vertices are spaced by 1 unit.**

In that way the number of vertices on an axis is given by its *Length + 1*. Once we know how many vertices there are on each axis we can calculate the total amount of vertices on the plane: given by *VerticesOn_X * VerticesOn_Y*. In that way we define the vertices array as:

```
/* Defining the vertices! */
VerticesOn_X = Length_X + 1;
VerticesOn_Y = Length_Y + 1;
vertices = new Vector3[VerticesOn_X * VerticesOn_Y ];
```

Arranging the vertices

Now that we can store vertices let's create and arrange them so they may form a plane. I will start by placing the first on position *[0,0],* and proceed to place each vertice 1 unit to the left until there is a row with *Length_X* size.

Once we have the first row, I start the second row on position *[1,0]* always incrementing in 1 for the next row. That is repeated until we have a matrix of vertices that is *Length_X* by *Length_Y* in size.

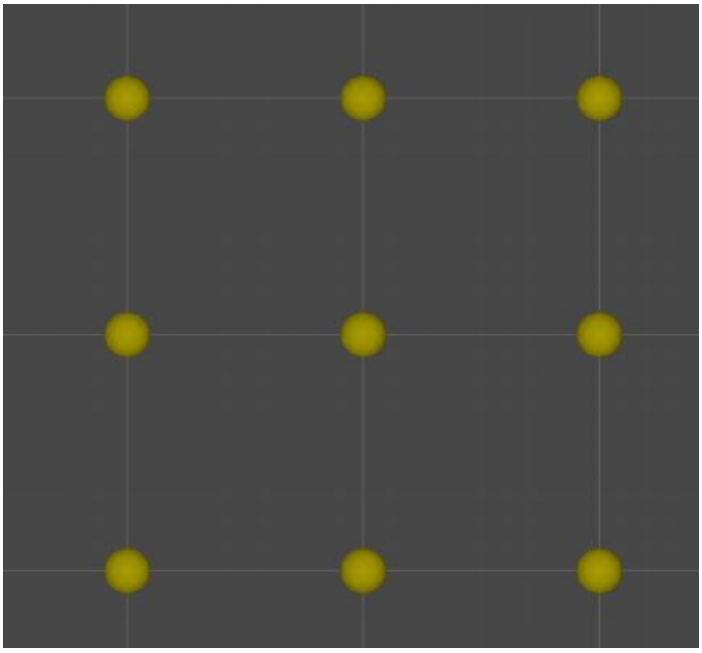
```
/* Arrange the vertices. */
int z = 0;

for (int i = 0; i < Length_X; i++ )
{
   for (int j = 0; j < Length_Y; j++ )
   {
       vertices[z] = new Vector3(j, i, 0);
       z++;
   }
}
```

We have the vertices created and properly placed, it is possible to check them by either placing Unity game objects on their position, like cubes or spheres (You can use the same loop above for that).

Another approach could be to use Unity's gizmo system. On the Plane script add a method called *OnDrawGizmos*

```
 /* Visual debug of the vertices. */
private void OnDrawGizmos()
{
   if (vertices == null)
      return;

   Gizmos.color = Color.yellow;

   /* Draw for each vertice. */
   for (int i = 0; i < vertices.Length; i++)
   {
        Gizmos.DrawSphere(vertices[i], 0.1f);
   }
}
```

Once you run the code, you can see each sphere gizmo drawn on the position of the vertice defined. Note that Gizmos only work on the Editor screen on Unity. This result when *Length_X = 3* and *Length_Y = 3*.



The Triangles

Triangles will be responsible for filling up the spaces between the vertices and giving shape to the plane. The triangles are where you will be able to see the textures and the shaders applied to the plane.

In order to add triangles to the plane, we must create an *UnityEngine Mesh.* Set it on the plane's *MeshFilter* component, and add the vertices created to the mesh vertices array.

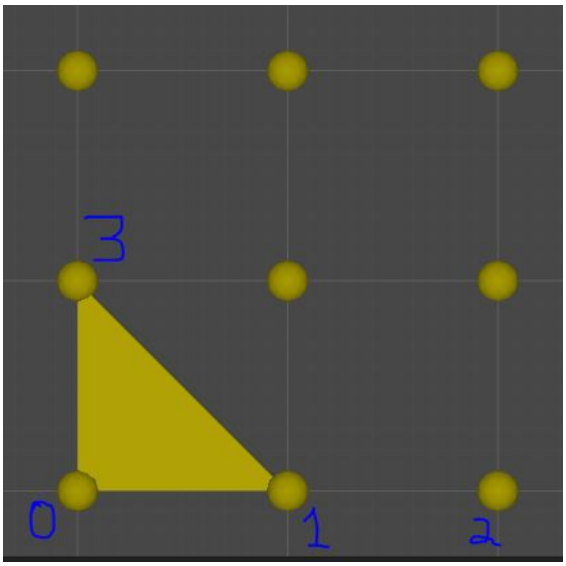
```
/* Let's initialize the mesh. */
mesh = new Mesh();

/* And set the mesh to our object mesh filter. */
GetComponent<MeshFilter>().mesh = mesh;

/* Maybe name it? */
mesh.name = "Custom Plane";

/* Set the vertices of the mesh. */
mesh.vertices = vertices;
```

Creating a triangle

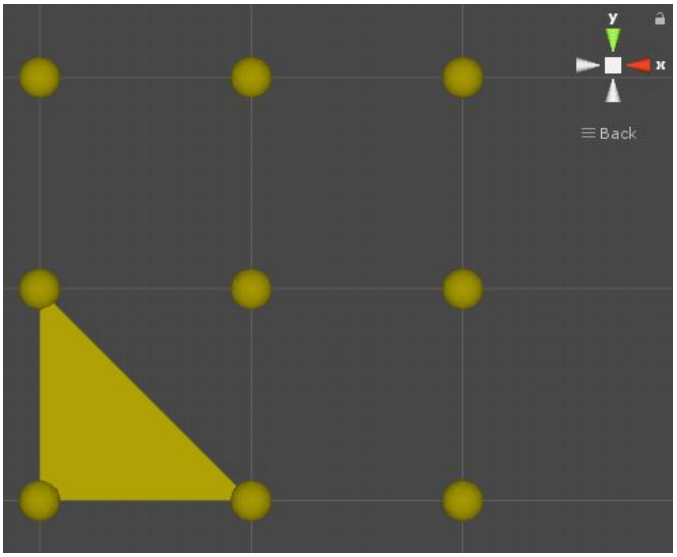
Once the *Mesh* is properly set up, it's time to create an array to store the triangle information. This array will be a sequence of vertex indexes.

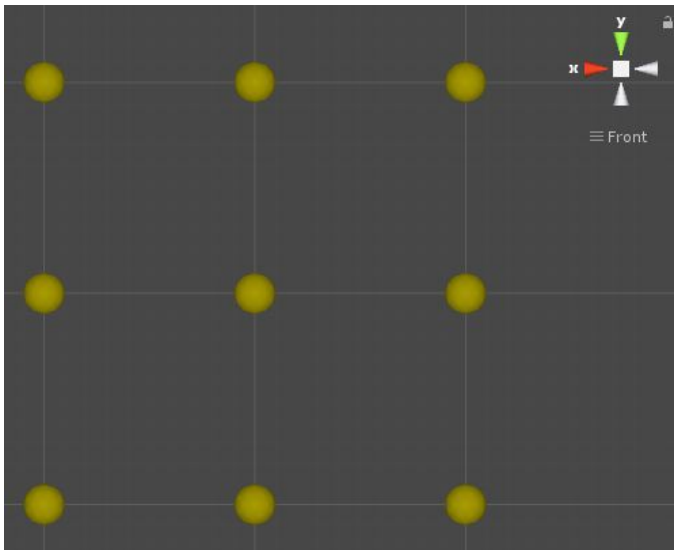
The *MeshRenderer* will read them three at a time, drawing triangles based on it. In the case below, the code is using vertex 0, 3 and 1 to make a triangle.

```
/* Set a new triangle. */
int[] triangles = new int[6];
triangles[0] = 0;
triangles[1] = 3;
triangles[2] = 1;
```



In order to improve performance the triangles are drawn facing a specific direction. Imagine that this plane is going to represent the ground of your game. In this case, there is no need to draw triangles on the surface that is facing down (it will never be seen by the player).



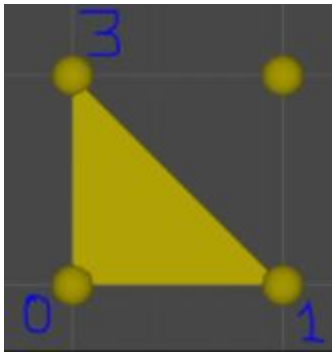


On the picture above it's possible to see that the triangle is only shown, when looking from the Z axis.

So how do we know which direction the triangle is being drawn?

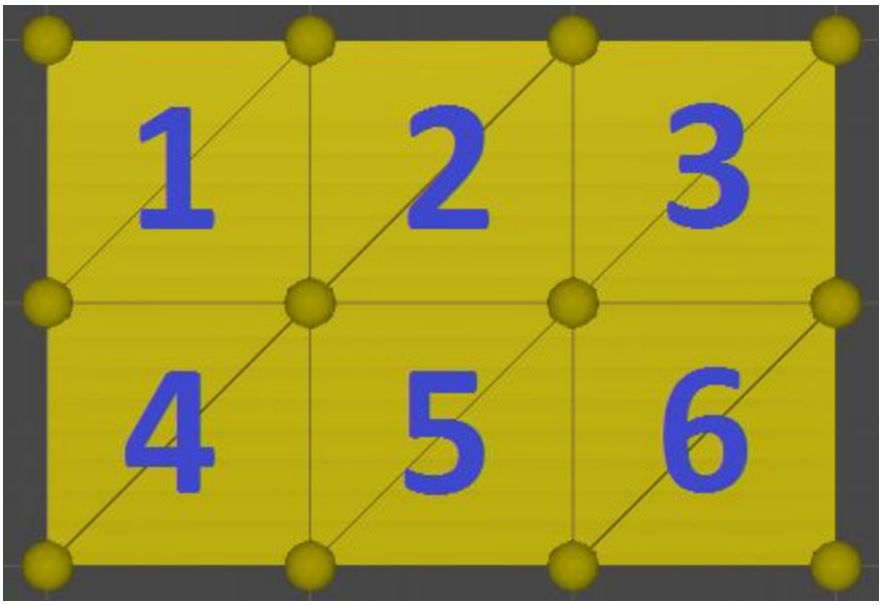
It depends on the order that the vertices are stored in the *triangles* array of the *MeshRenderer.* They must be stored in a clockwise direction in order to be seen from the arrow area.

In the example above we store the vertices in the order **0, 3 and 1** on the *triangles* array (clockwise direction from the spectator perspective).



The size of the triangles array

To instantiate the triangles array with the correct size, it's necessary to understand how many triangles there will be on the plane. It's possible to think in terms of squares inside the plane and then multiply the amount of squares by 2. In that way, on a plane that has 6 squares we have 12 triangles.



In order to count the number of squares in the plane we can follow the rule:

```
SquaresOn_X = VerticesOn_X - 1
```

On the example above, there is 4 vertices on the horizontal axis and 3 squares. The rule verifies for the vertical axis too, 3 vertices and 2 squares. The number of squares give us the number of triangles.

Given the amount of triangles, the size of the *triangles array* is given by: *amount of triangles times 3*. Because each triangle is formed by 3 vertices. In that way we can write:

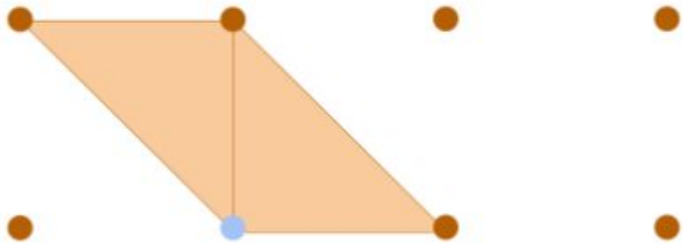
```
/* Size of the triangle array is given by the max number of triangles * 3 */
int maxTriangles = ( (VerticesOn_X - 1) * (VerticesOn_Y - 1) ) * 2;

/* Initialize triangle array. */
int[] triangles = new int[maxTriangles * 3];
```

Filling the triangles array

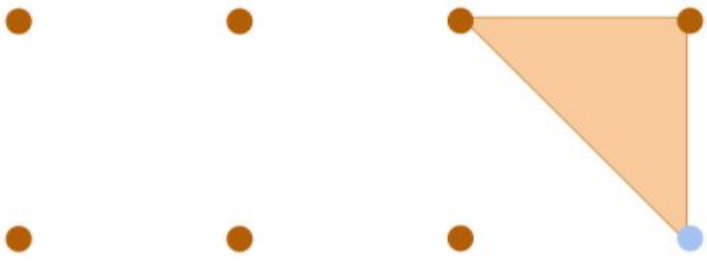
At this point the triangles array is initialized but empty. The vertices should be stored with the correct order to form triangles. There are several ways to implement this, for this tutorial I iterate through each vertex with exemption of the top row. Each vertex fall into one of the cases below:

Case I - The vertex is the first of a row : If the vertex is the first on a row (blue), it will draw one triangle using its top and right vertex.

Case II - The vertex is in the middle : In this case we will draw 2 triangles, on to the left and the other to the right of the pivot vertex.



Case III - The vertex is in the end of a row : This case we draw only the top triangle.



In that way, it's possible to fill a row with triangles by going vertex by vertex. Notice that doesn't make sense to run this loop with the last row of vertices. The code below will do exactly as shown above, identifying the cases and creating the triangle on the proper way.

```
z = 0; // Index of the triangles array.

foreach (Vector3 verticy in mesh.vertices)
{
   /* Find the vertex index */
   int index = Array.IndexOf(mesh.vertices, verticy);

   /* We don't need to create triangles for the last row o vertices.
   since we started from bottom to top. In that way we quit.
   */
   if (index + VerticesOn_X >= mesh.vertices.Length)
      break;

   /* Is this vertex in the beginning of a row? */
   bool beginningOfRow = false;
   if (index == 0)
      beginningOfRow = true;
   else if (index % VerticesOn_X == 0)
      beginningOfRow = true;
   else
      beginningOfRow = false;

   /* Is this vertex in the end of a row? */
   bool endOfRow = false;
   if ( (index+1) % VerticesOn_X == 0)
      endOfRow = true;
   else
      endOfRow = false;

   /* Draw the triangle(s) for the current vertex.*/
   if (beginningOfRow)
   {
      /* Draw to the right. */
      triangles[z] = index;
      triangles[z + 1] = index + VerticesOn_X;
      triangles[z + 2] = index + 1;
      z += 3;
   }
   else if (endOfRow)
   {
      /* Draw to the left. */
      triangles[z] = index;
      triangles[z + 1] = index + VerticesOn_X - 1;
      triangles[z + 2] = index + VerticesOn_X;
```

```
      z += 3;
   }
   else
   {
      /* Case the vertex is in the middle. Draw left and right. */
      triangles[z] = index;
      triangles[z + 1] = index + VerticesOn_X;
      triangles[z + 2] = index + 1;
      z += 3;
      triangles[z] = index;
      triangles[z + 1] = index + VerticesOn_X - 1;
      triangles[z + 2] = index + VerticesOn_X;
      z += 3;
      }
    }

/* Add the triangles to the mesh. */
mesh.triangles = triangles;
```

At this point the *createMesh* method is fully developed. You can run the code developed so far and will be able to generate a plane at runtime.

Vertex density

In order to create a plane of any size it's only needed four vertices and two triangles. However there are situations where a plane with more vertices is needed. As an example we can think of a rugged surface. It could be created by a plane with multiple vertices, where there is an uneven position displacement between them.

The amount of vertices in a plane is called **vertex density**. In this part of the tutorial we will go over two forms of increasing vertex density: '*Density with uneven edge size'* and '*Density with equal edge size'.*

In order to increase the amount of vertices while keeping the same plane area, we must reduce the distance between vertices. So far we have created a plane that has its vertices distant by 1 unit. This time we will need two variables to hold the edge size on each axis.

```
/* The size of an edge */
private float edgeSize_X;
private float edgeSize_Y;
```

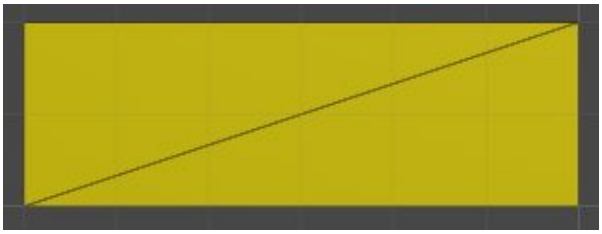
Since we have different edge sizes we must also change the original loop that arrange the vertices on the vertex array. The new loop can be found below, you should replace it in the *createMesh* method.

```
/* Arrange the vertices. */
int z = 0;
float posX = 0;
float posY = 0;
for (int i = 0; i < VerticesOn_Y; i++ )
{
    for (int j = 0; j < VerticesOn_X; j++ )
    {
        vertices[z] = new Vector3(posY, posX, 0);
        posX += edgeSize_X;
        z++;
    }
    posX = 0;
    posY += edgeSize_Y;
}
```

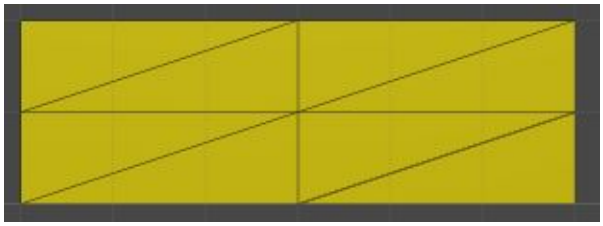
Density with uneven edge size

Simply increasing the number of vertices is easy, and there are several implementations. On this tutorial I will simply multiply the **number of edges** by 2, on each axis of the plane.

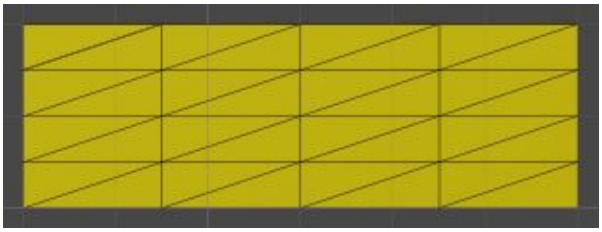
Density 1 is the minimum value, therefore we will have the minimum amount of vertices to create the plane. Furthermore for density 2, there will be 2 edges on each side, resulting in 9 vertices. Density can be increased as much as desired.



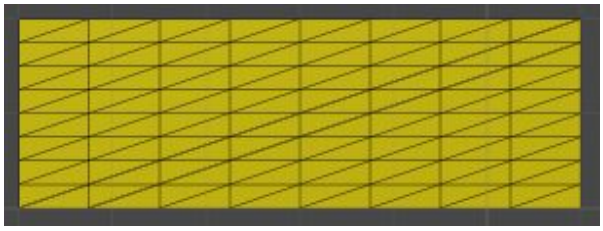
Density: 1 - 4 vertices



Density: 2 - 9 vertices



Density: 3 - 25 vertices



Density: 4 - 81 vertices

In order to implement this in our current code we must go back to the definition of the vector array. It's necessary to calculate the number of vertices on each side of the plane this is given by:

```
(int)Mathf.Pow(Vertex_Density + 1, 2)/2;
```

So the lines:

```
/* Defining the vertices! */
VerticesOn_X = Length_X + 1;
VerticesOn_Y = Length_Y + 1;
```

Will be replaced by:

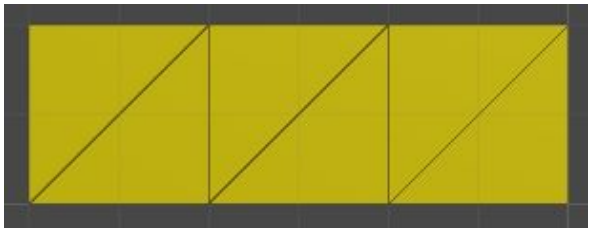
```
/* Defining the vertices! */
int numberOfVertices = (int)Mathf.Pow(Vertex_Density + 1, 2)/2;
VerticesOn_X = numberOfVertices;
VerticesOn_Y = numberOfVertices;
```

We must also calculate the edge size on x and y axis, fortunately that's pretty simple too.

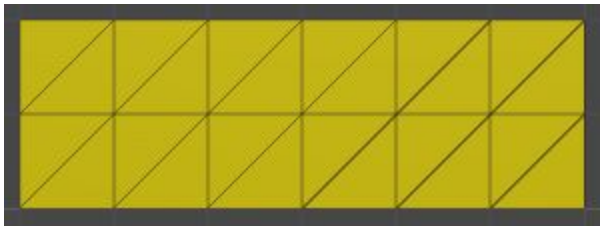
```
/* Defining the vertices! */
edgeSize_X = (float)Length_X / (float)(VerticesOn_X - 1);
edgeSize_Y = (float)Length_Y / (float)(VerticesOn_Y - 1);
```

Density with equal edge size

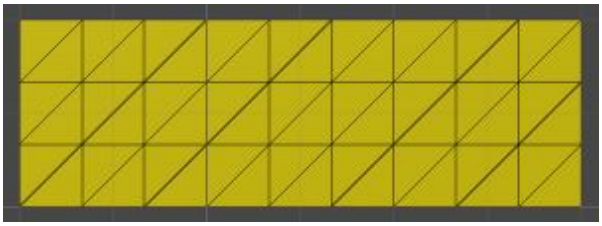
In this case we want to increase the number of vertices, but keep the distance between vertices the same. When the *Length_X* is equal to *Length_Y* it's straightforward, however for different lengths it's a bit more tricky. Note that the minimum amount of vertices (**density 1**) is no longer 4 vertices for all cases. The case below we have a 6 by 2 plane.



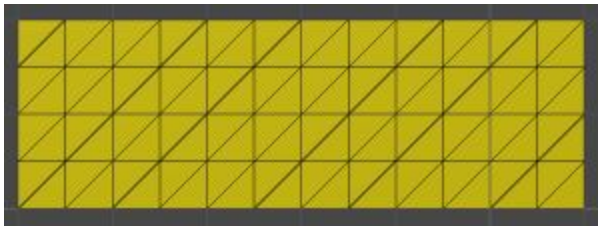
Density: 1 - 8 vertices



Density: 2 - 21 vertices



Density: 3 - 40 vertices



Density: 4 - 65 vertices

The maximum edge size is given by the *Greater Common Divisor (GCD)* of *Length_X* and *Length_Y.* We can implement a simple method that calculates *GCD* as follows:

```
private int GCD(int a, int b)
{
    /* Case a < b */
    if ( a < b )
    {
        int k = a;
        a = b;
        b = k;
    }
    /* Base Case*/
    if ((a % b) == 0)
        return b;
    else
    {
        int r = a % b;
        return GCD(b, r);
    }
}
```

Once we know the maximum edge size, we can divide it by the density we want to get different densities.

```
/* The size of the edges are given by the GCD between the two lengths. */
edgeSize_X = GCD(Length_X, Length_Y);

/* For different densities. */
edgeSize_X = (float)edgeSize_X / (float)Vertex_Density;
edgeSize_Y = edgeSize_X;

VerticesOn_X = Convert.ToInt32((Length_X / edgeSize_X) + 1);
VerticesOn_Y = Convert.ToInt32((Length_Y / edgeSize_X) + 1);
```

The amount of vertices on each axis is given by the equation above. We have the edge size and the amount of vertices, it's all set to be drawn by the *createMesh* method.

Conclusions

Thanks for reading this tutorial, I hope it was able to clarify and nourish ideas on how to work with meshes on your projects. There will be more related to 3D Geometry soon!

It's possible to find the source files and the Unity project on my webpage (http://www.yetanothergamer.com/2017/08/05/tutorial-mixing-mobile-camera-and-3d-geometry/).

Appendix - The code

The final code *Plane.cs* that you should have by the end of this tutorial can be found below.

```
using System;
using UnityEngine;

[RequireComponent(typeof(MeshFilter), typeof(MeshRenderer))]
public class Plane : MonoBehaviour
{
    /*
        ---------------------------------------------------------------------------

        # Plane #

        This class represent a plane. This plane is rectangular and made of vertices.

        ---------------------------------------------------------------------------
    */

    /* The length on the X direction. */
    public int Length_X;

    /* The length on the Y direction. */
    public int Length_Y;

    /* 1 is the minimum, each increment doubles the amount of vertices. */
    public int Vertex_Density = 1;

    /* If this is true, the plane will adjust it's side density so the
       distance between vertices is the same.
    */
    public bool EdgesWithSameSize;

    /* Amount of vertices on the X direction. */
    private int VerticesOn_X;

    /* Amount of vertices on the Y direction. */
    private int VerticesOn_Y;

    /* A array to store vertices. */
    private Vector3[] vertices;

    /* The plane mesh */
    private Mesh mesh;

    /* The size of an edge */
    private float edgeSize_X;
```

```
    private float edgeSize_Y;

    /* Generate the Mesh on Awake. */
    void Awake()
    {
        /* Define edge size and number of vertices. */

        if (EdgesWithSameSize)
        {
            /* The size of the edges are given by the GCD between the two lengths. */
            edgeSize_X = GCD(Length_X, Length_Y);

            /* For different densities. */
            edgeSize_X = (float)edgeSize_X / (float)Vertex_Density;
            edgeSize_Y = edgeSize_X;
            VerticesOn_X = Convert.ToInt32((Length_X / edgeSize_X) + 1);
            VerticesOn_Y = Convert.ToInt32((Length_Y / edgeSize_X) + 1);

        }
        else
        {
            /* In this case edges don't have the same size on X and Y.
               In that way we can keep the same shape with different amounts of vertices.*/
            int numberOfVertices = (int)Mathf.Pow(Vertex_Density + 1, 2) / 2;

            VerticesOn_X = numberOfVertices;
            VerticesOn_Y = numberOfVertices;

            edgeSize_X = (float)Length_X / (float)(VerticesOn_X - 1);
            edgeSize_Y = (float)Length_Y / (float)(VerticesOn_Y - 1);
        }

        createMesh();
    }

    /* This method will create the mesh. */
    private void createMesh()
    {
        /* Defining the vertices! */
        vertices = new Vector3[VerticesOn_X * VerticesOn_Y];

        /* Arrange the vertices. */
        int z = 0;
        float posX = 0;
        float posY = 0;
        for (int i = 0; i < VerticesOn_Y; i++ )
        {
            for (int j = 0; j < VerticesOn_X; j++ )
            {
```

```
            vertices[z] = new Vector3(posY, posX, 0);
            posX += edgeSize_X;
            z++;
        }
        posX = 0;
        posY += edgeSize_Y;
    }

    /* Let's initialize the mesh. */
    mesh = new Mesh();

    /* And set the mesh to our object mesh filter. */
    GetComponent<MeshFilter>().mesh = mesh;

    /* Maybe name it? */
    mesh.name = "Custom Plane";

    /* Set the vertices of the mesh. */
    mesh.vertices = vertices;

    /* Size of the triangle array is given by the max number of triangles * 3 */
    int maxTriangles = ( (VerticesOn_X - 1) * (VerticesOn_Y - 1) ) * 2;

    /* We assume at least one triangle */
    if (maxTriangles < 1)
        maxTriangles = 1;

    /* Initialize triangle array. */
    int[] triangles = new int[maxTriangles * 3];

    z = 0; // Index of the triangles array.

    foreach (Vector3 vertex in mesh.vertices)
    {
        /* Find the vertex index */
        int index = Array.IndexOf(mesh.vertices, vertex);

        /* We don't need to create triangles for the last row o vertices.
           since we started from bottom to top. In that way we quit.
        */
        if (index + VerticesOn_X >= mesh.vertices.Length)
            break;

        /* Is this vertex in the beginning of a row? */
        bool beginningOfRow = false;
        if (index == 0)
            beginningOfRow = true;
        else if (index % VerticesOn_X == 0)
            beginningOfRow = true;
```

```
            else
                beginningOfRow = false;

            /* Is this vertex in the end of a row? */
            bool endOfRow = false;
            if ( (index+1) % VerticesOn_X == 0)
                endOfRow = true;
            else
                endOfRow = false;

            /* Draw the triangle(s) for the current vertex.*/
            if (beginningOfRow)
            {
                /* Draw to the right. */
                triangles[z] = index;
                triangles[z + 1] = index + VerticesOn_X;
                triangles[z + 2] = index + 1;
                z += 3;
            }
            else if (endOfRow)
            {
                /* Draw to the left. */
                triangles[z] = index;
                triangles[z + 1] = index + VerticesOn_X - 1;
                triangles[z + 2] = index + VerticesOn_X;
                z += 3;
            }
            else
            {
                /* Case the vertex is in the middle. Draw left and right. */
                triangles[z] = index;
                triangles[z + 1] = index + VerticesOn_X;
                triangles[z + 2] = index + 1;
                z += 3;
                triangles[z] = index;
                triangles[z + 1] = index + VerticesOn_X - 1;
                triangles[z + 2] = index + VerticesOn_X;
                z += 3;
            }

        }

        /* Add the triangles to the mesh. */
        mesh.triangles = triangles;
    }

    /* Visual debug of the vertices. */
    private void OnDrawGizmos()
    {
```

```
        if (vertices == null)
            return;

        Gizmos.color = Color.yellow;
        /* Draw for each vertice. */
        for (int i = 0; i < vertices.Length; i++)
        {
            Gizmos.DrawSphere(vertices[i], 0.1f);
        }
    }

    /* Returns the Greatest Common Divisor between two numbers.*/
    private int GCD(int a, int b)
    {
        /* Case a < b */
        if ( a < b )
        {
            int k = a;
            a = b;
            b = k;
        }
        /* Base Case*/
        if ((a % b) == 0)
            return b;
        else
        {
            int r = a % b;
            return GCD(b, r);
        }
    }
}
```