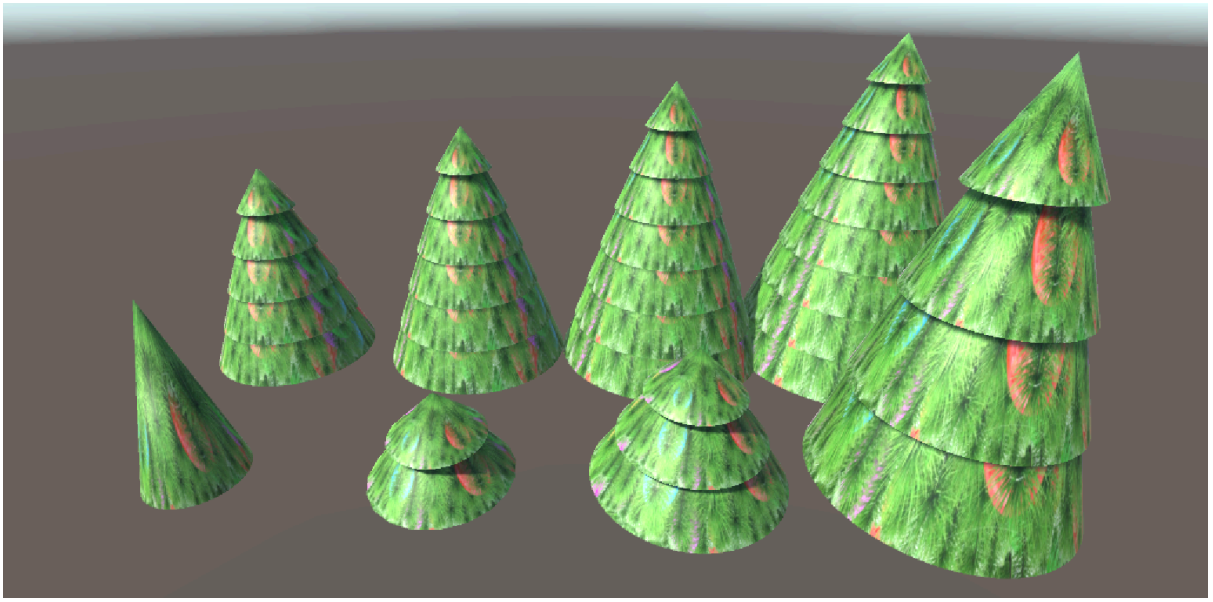
Christmas tree made from vertices and triangles assignment-1



Naam: jiminy de koningh

Nummer: 500828215

What I made:

For this assignment, I have chosen to create a Christmas tree. This tree consists of four important variables and a script that will be explained later. When you place the tree in the scene, you can specify how detailed the tree should be in the number of vertices, how many rings it needs, how thick the tree should be, and how large the tree should be. I believe this assignment meets the medium level, as I can adjust four aspects of the tree in the inspector, and it is composed of cones that increase in size as they move further down the tree. Parts of the code are written by chat GTP but the bulk of it is written by me. I used it for the hard formulas or parts where I could not find my own mistake.

Creating Vertices:

Determining Vertex Count: Figure out how many points you need for your mesh. This depends on the shape you want and how detailed you want it.
Calculating Vertex Positions: Decide where each point should be. For example, if you're making a cone, you might use trigonometry to find the positions in a circular pattern.
Storing Vertex Data: Save the position of each point in a list or array.

```
// Initialize arrays to hold vertices and triangles
Vector3[] vertices = new Vector3[numVertices * rings];
Vector2[] uvs = new Vector2[numVertices * rings];
int[] triangles = new int[(numVertices * rings + 1) * 3];

// Generate vertices in a circular pattern
for (int i = 0; i < rings; i++)
{
    for (int j = 0; j < numVertices; j++)
    {
        // Calculate the angle for each vertex
        float angle = Mathf.PI * 2 * j / (numVertices - 2);

        // Calculate the position of the vertex using trigonometry
        float x = Mathf.Cos(angle) * ((i - rings - 1) / radius);
        float z = Mathf.Sin(angle) * ((i - rings - 1) / radius);
        if (j == 0)
        {
            // Set the middel vertex position
            vertices[j + i * numVertices] = new Vector3(0, rings*hight, 0);
            // Calculate middel UV coordinates based on vertex position
            uvs[j + i * numVertices] = new Vector2(0.5f, 0.5f);
        }
        else
        {
            // Set the vertex position
            vertices[j + i * numVertices] = new Vector3(x, i*hight, z);
            // Calculate UV coordinates based on vertex position
```

Explanation:

Outer Loop (for (int i = 0; i < rings; i++)):
This loop iterates through the rings of the shape you're creating. Each iteration represents a new ring or layer of the shape.

Inner Loop (for (int j = 0; j < numVertices; j++)):
This loop iterates through the vertices within each ring. Each iteration represents a vertex on the outside of the ring.

Calculating Angle (float angle = Mathf.PI * 2 * j / (numVertices - 2)):
This calculates the angle for each vertex around the ring using trigonometry. It evenly spaces the vertices around the circle and the - 2 is there to complete the circle.

Calculating Vertex Position (float x = Mathf.Cos(angle) * ((i - rings - 1) / radius) and float z = Mathf.Sin(angle) * ((i - rings - 1) / radius)):
This calculates the x and z coordinates of each vertex using the calculated angle and radius. It determines the position of the vertex relative to the center of the shape aka the middle vertex.

Generating Triangles:

Counting Triangles: Think about how many triangles you need to form the surface of your shape. Typically, it's three times the number of points minus two.
Connecting Vertices: Create triangles by connecting the points together. For shapes like cones or pyramids, start from a central point and connect it to successive points around the edge.

```
}
//Debug.Log(vertices[i]);
// Generate triangles to form the circle
for (int j = 0; j < numVertices - 1; j++)
{
    int baseIndex = (j + i * numVertices) * 3; // Base index for the current triangle

    triangles[baseIndex] = 0;  // Central vertex
    triangles[baseIndex + 1] = j + 1 + i * numVertices; // Vertex after the current one
    triangles[baseIndex + 2] = j + i * numVertices; // Current vertex
}
```

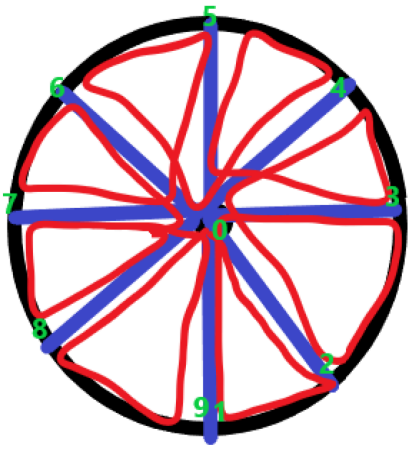
Explanation:

Looping through Vertices (for (int j = 0; j < numVertices - 1; j++)):
This loop iterates through each vertex in the current ring, excluding the last one. We don't include the last one because it will make a diagonal triangle between the rings of the tree.

Base Index Calculation (int baseIndex = (j + i * numVertices) * 3):
It calculates the base index for the current triangle. This index is used to access elements in the triangles array. It works in such a way that every time we need to make a new ring it skips the already used spaces in the triangles array.

Assigning Triangle Vertices:
triangles[baseIndex] = 0: Sets the first vertex of the triangle to the central vertex (index 0).
triangles[baseIndex + 1] = j + 1 + i * numVertices: Sets the second vertex of the triangle to the vertex after the current one.
triangles[baseIndex + 2] = j + i * numVertices: Sets the third vertex of the triangle to the current vertex.

We draw the tree counterclockwise so it looks like we need to draw the triangles clockwise or else the tree would be inside out. The green numbers are the vertices and the black and blue lines are the triangles. If you follow the red line that would be the order the triangles are saved.



if i = 0 and j = 0	if i = 1 and j = 0
triangles[0] = 0;	triangles[15] = 0;
triangles[1] = 2;	triangles[16] = 10;
triangles[2] = 1;	triangles[17] = 9;
if i = 0 and j = 1	if i = 1 and j = 1
triangles[3] = 0;	triangles[18] = 0;
triangles[4] = 3;	triangles[19] = 11;
triangles[5] = 2;	triangles[20] = 10;

Linking Triangles to a UV Map:
Understanding UV Coordinates: UV coordinates determine how textures are applied to the surface of the mesh. Each vertex has a corresponding UV coordinate.
Calculating UV Coordinates: Decide where on the texture each point should map to. This can involve some math based on the shape and texture you're using.
Storing UV Data: Save the UV coordinates in another list or array, making sure they correspond to the vertices in the correct order.

```
uvs[j + i * numVertices] = new Vector2(0.5f, 0.5f);
```

Setting UV Coordinates for Center Vertex (uvs[j + i * numVertices] = new Vector2(0.5f, 0.5f);):
This line sets the UV coordinates for the center vertex of each ring.
UV coordinates (0.5, 0.5) represent the middle of the texture, ensuring that the texture is centered on the vertex. This is because the texture is always scaled from 0 to 1 there for 0.5;0.5 is always the center of the texture.

```
// Calculate UV coordinates based on vertex position
uvs[j + i * numVertices] = new Vector2((x + ((i - rings - 1) / radius)) / (((i - rings - 1) / radius) * 2), (z + ((i - rings - 1) / radius)) / (((i - rings - 1) / radius) * 2));
```

Setting UV Coordinates for Other Vertices (uvs[j + i * numVertices] = new Vector2((x + ((i - rings - 1) / radius)) / (((i - rings - 1) / radius) * 2), (z + ((i - rings - 1) / radius)) / (((i - rings - 1) / radius) * 2));):
This line sets the UV coordinates for vertices other than the center vertex.
It calculates the UV coordinates based on the position of the vertex relative to the center and the radius of the shape.

The formula ensures that UV coordinates are proportionally distributed across the texture, allowing for proper texture mapping onto the shape. It looks very long and complicated but it is just the same equation as for the vertices but again. I could split it up to make it more readable but I don't think that is necessary.

Bronnen:

Dit verslag is geschreven met behulp van.
ChatGPT. (n.d.). https://chat.openai.com/c/a2f57601-6571-454b-ad70-2d6b83bc9c61