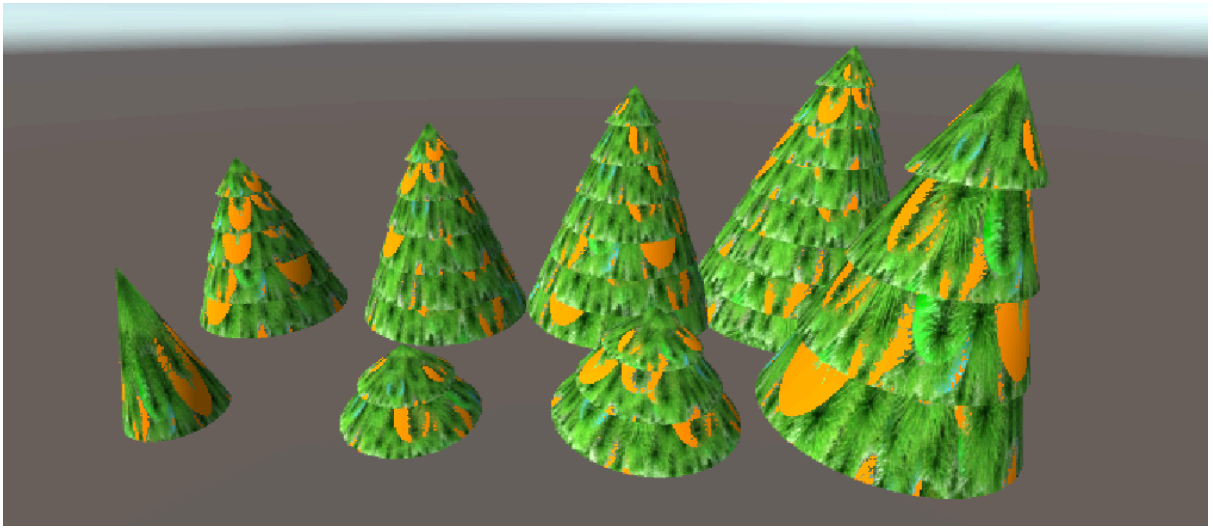
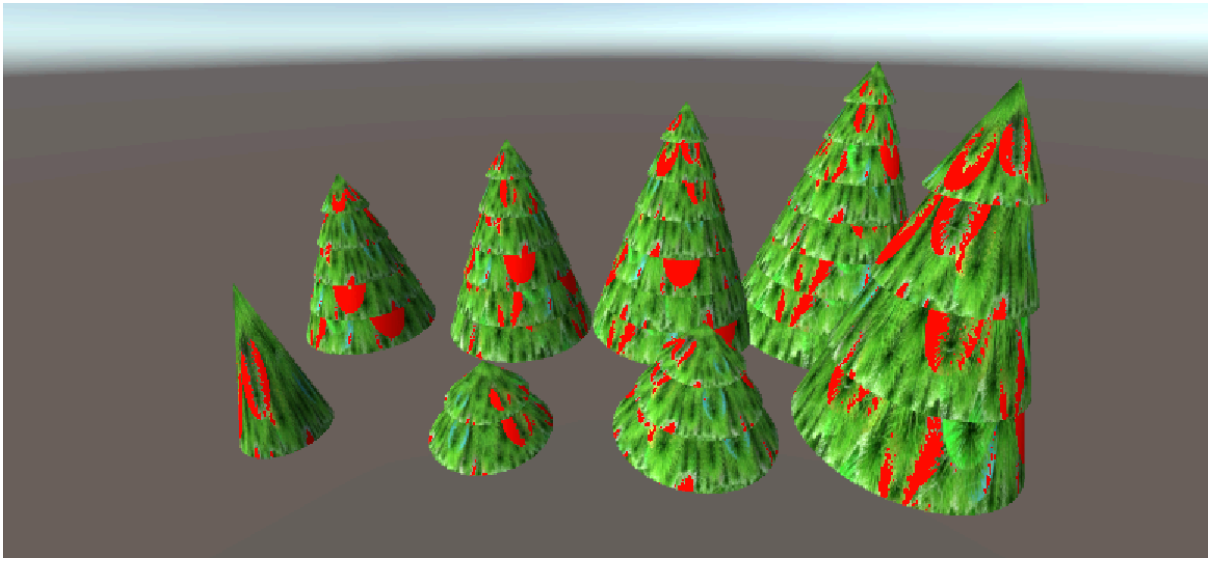
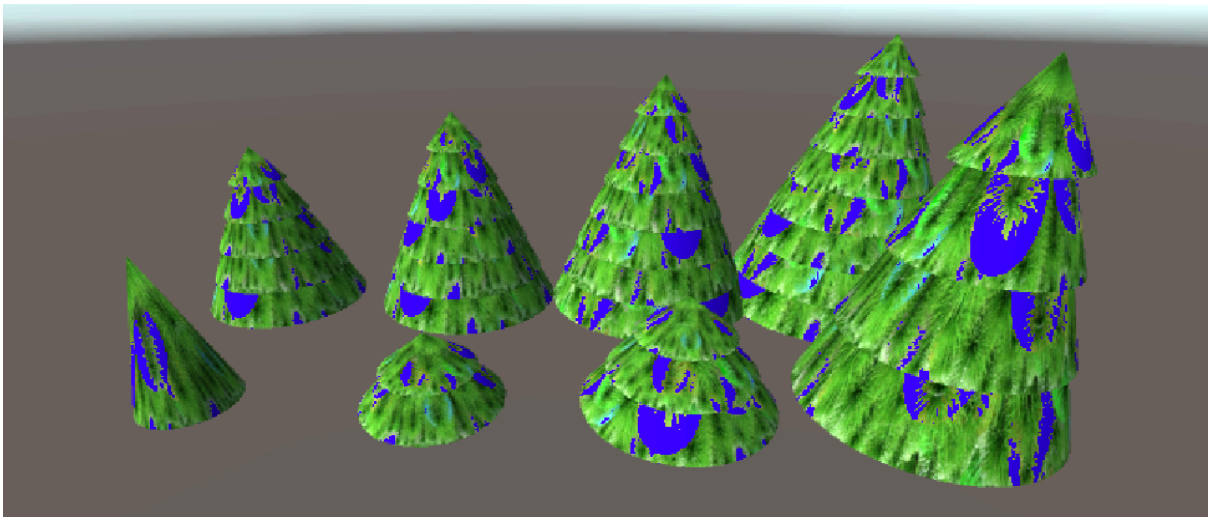
Christmas tree in the wind with magic balls







name student: Jiminy de Koningh.
studenten number: 500828215.

My goal. What did you want to achieve?

My goal is to make a wind shader in Unity, aiming for wind movement on my tree while integrating a rainbow color effect.

I successfully implemented a shader that creates wind effect by moving vertices along a specified direction, with dynamic variation in strength and speed over time. Additionally, I incorporated a rainbow color effect for non-green areas of the object, smoothly transitioning through the full spectrum of colors except green.

Explain your choices and techniques

Basic Wind Shader with Movement:

- Technique: The shader displaced vertices based on a sine wave function to simulate wind effect. The displacement was applied along a specified wind direction.
- Outcome: This iteration provided a simple wind effect, but lacked visual diversity.

Addition of Rainbow Color Effect:

- Technique: A rainbow color effect was added to non-green areas of the object. Colors transitioned between red and yellow using linear interpolation (lerp) based on a sine wave function.
- Outcome: The shader gained visual interest and diversity, but the color transition was limited to red and yellow.

Expanding Rainbow Color Range:

- Technique: The shader code was modified to smoothly transition through the full spectrum of colors for the rainbow effect. This was achieved by converting hue values to RGB using the HSV color space.
- Outcome: The shader now smoothly transitioned through all colors of the rainbow, providing a more visually appealing result.

Dynamic Wind Strength and Speed:

- Technique: Dynamic variation was introduced in both wind strength and speed over time by incorporating sine wave functions into the shader code.
- Outcome: The wind effect varied dynamically, creating a more realistic simulation of wind movement.

Optimization and Adjustment:

- Technique: Parameters such as wind strength, speed, and displacement factor were adjusted to fine-tune the wind effect. Code optimization was performed to ensure better performance.
- Outcome: The shader achieved the desired visual effect with improved performance and customization options.

The code

```
Shader "Unlit/WindShader"
{
    Properties
    {
        _MainTex("Texture", 2D) = "white" {}
        _WindDirection("Wind Direction", Vector) = (0, 0, 1)
        _RainbowSpeed("Rainbow Speed", Float) = 1.0 // Speed of rainbow interpolation
        _WindSpeedMultiplier("Wind Speed Multiplier", Float) = 2.0 // Multiplier for wind speed variation
    }
```

MainTex represents the texture applied to the shader, with the option to change it in the Unity editor. WindDirection determines the direction of the wind effect applied to the shader, while RainbowSpeed controls the speed at which the rainbow color effect transitions. WindSpeedMultiplier adjusts the multiplier for varying the speed of the wind effect over time. I can change these parameters to make the movement and the colors different.

```
void vert(inout appdata_full v)
{
    // Calculate wind effect using a sine wave
    float windFactor = sin(_Time.y * _WindSpeedMultiplier);

    // Move vertex along the wind direction with reduced displacement
    v.vertex.xyz += _WindDirection * windFactor * 0.05;
}
```

In the vert function, which operates on each vertex, the wind is calculated using a sine wave. The _Time.y variable is multiplied by WindSpeedMultiplier to control the speed of the wind over time. This results in a value that swings between -1 and 1, creating a smooth variation in wind strength.

Next, the vertex position (v.vertex.xyz) is adjusted along the specified wind direction (WindDirection) by a factor of windFactor multiplied by 0.05. This value determines the magnitude of the movement applied to the vertex due to the win. By scaling the wind factor, we control the intensity of the wind, causing vertices to move more or less depending on the wind strength.

Overall, this function simulates the movement of vertices in response to the wind, producing an animation effect on the object. Adjusting the WindSpeedMultiplier parameter will alter the speed of the wind effect, while modifying WindDirection will change the direction in which vertices move.

```
void surf(Input IN, inout SurfaceOutput o)
{
    // Sample the main texture
    fixed4 texColor = tex2D(_MainTex, IN.uv_MainTex);

    // Check if the pixel is green
    if (texColor.g > texColor.r && texColor.g > texColor.b)
    {
        // If the pixel is green, keep its original color
        o.Albedo = texColor.rgb;
    }
    else
    {
        // If the pixel is not green, lerp between colors of the rainbow
        float rainbowFactor = (sin(_Time.y * _RainbowSpeed) + 1.0) * 0.5; // Map sin wave to [0, 1] range
        fixed3 rainbowColor = lerp(fixed3(1, 0, 0), fixed3(1, 1, 0), rainbowFactor); // Lerp between red and yellow
        o.Albedo = rainbowColor;
    }

    // Set alpha to 1 for opaque objects
    o.Alpha = 1.0;
}
ENDCG
```

In the `surf` function, the shader determines the final color of each pixel on the object's surface.

1. It samples the main texture (`MainTex`) at the current UV coordinates (`IN.uv_MainTex`) to obtain the base color of the pixel.

2. It checks if the pixel is green by comparing its green component (`texColor.g`) to its red and blue components (`texColor.r` and `texColor.b`). If the green component is greater than both the red and blue components, it means the pixel is green, and the original color is not being altered.

3. If the pixel is not green, the shader switches between colors of the rainbow. It calculates a `rainbowFactor` by mapping a sine wave to the range [0, 1] using `Time.y` and `RainbowSpeed`. This factor determines the position along the rainbow spectrum.

4. The shader then lerps between fixed colors representing red (`fixed3(1, 0, 0)`) and yellow (`fixed3(0, 0, 1)`) based on the `rainbowFactor`. This produces a smooth transition between red and blue hues, resulting in a rainbow effect for non-green pixels.

5. The resulting color is assigned to the `o.Albedo` property, which determines the surface color of the object.

By combining these steps, the shader achieves the desired effect of preserving green pixels while applying a dynamic rainbow color effect to the rest of the surface. Adjusting the `RainbowSpeed` parameter will alter the speed at which the rainbow effect transitions. I wanted to use more colors but in the end it would give me more of a headache so I left it with red and blue for now. Because when I tried to use green as well when the color was green it stayed green.