Procedural Fog on a Windowpane

TNM084 - Procedural Images

EMMA FORSLING PARBORG

Linköping University
March 16, 2016

Abstract

This report details an interactive web application utilizing WebGL, that enables the user to create fog on a windowpane. For this project the goal was to create the fog utilizing noise functions. For this, Classic Perlin noise and Simplex noise have been utilized. The result is a web application that creates fog on a windowpane, in which hidden features can be found.

1 Introduction

The aim for the project is to create fog on a windowpane, utilizing different noise functions in order to create a realistic fog in real-time. The application should enable user interaction and allow a user to spread fog on the windowpane to look for hidden features on it.

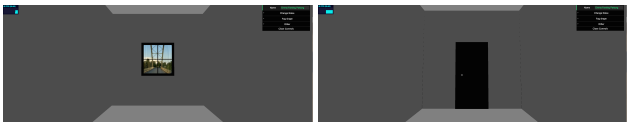
2 Making some Noise

For this project, two procedural three-dimensional noise functions have been utilized, Classic Perlin noise and Simplex noise. Classic Perlin noise is a gradient noise, meaning that it uses a pseudo random gradient, that is associated with regularly spaced points and interpolates a smooth function between those points [Gustavsson 2005].

Simplex noise was presented in 2001 by Ken Perlin, as a replacement for his Classic Perlin noise algorithm. The Simplex noise was developed in order to solve the limitations that the former noise algorithm has. One example of this is that Simplex noise utilizes a simplex grid, meaning that it is based on the simplest possible primitive. In 2D the simplex grid consists of triangles, compared to the regular square grid that Classic Perlin Noise utilizes. This simpler grid lowers the computational complexity.

3 Creating the Scene

When creating the scene, the API `Three.js` was utilized. The scene is fairly plain, consisting of planes that create walls, windowpane, door etc, see Figure 1.



(a) **(b)**

Figure 1: *These figures illustrate the scene created for the project.*

All meshes, except for the windowpane, utilizes a fragment shader that colors the meshes differently depending on a value sent to the shader. The

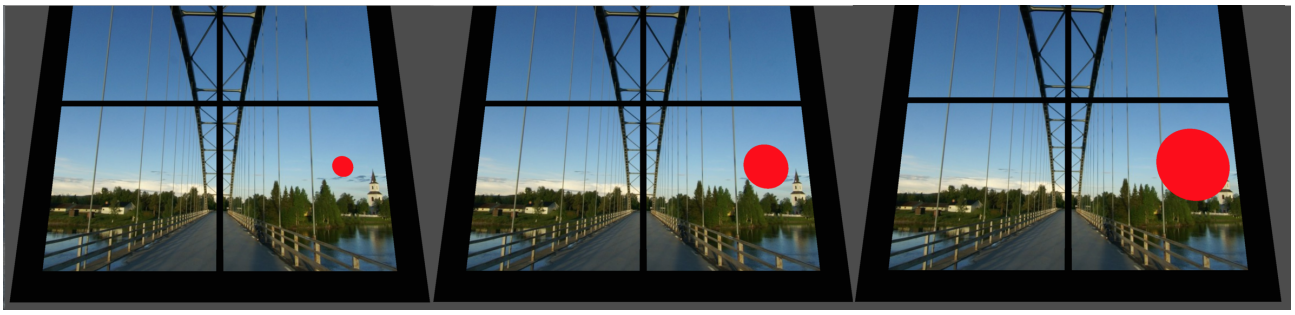


Figure 2: *This figure illstrates how a circle shape is created when a user clicks on the windowpane. From left to right: the radius for the circle increases over time, and thereby the circle expands.*

windowpane utilizes separate shaders since more computations will be made for it, when creating the fog.

In order to view the frame rate `stats.min.js` has been utilized, which creates a window in the top left corner of the canvas. To create an interactive GUI, `dat.min.js` has been used and is displayed in the top right corner of the canvas.

For the scene, `FlyControls.js` has been utilized which enables the user to walk around in the scene using the keys **W**, **A**, **S**, **D**. The user can also move up/down using the keys **R** and **D**, and rotate utilizing the arrow keys ←⇵→.

In order to handle mouse click events for the windowpane, two functions have been written, `onDocumentTouchStart(event)` and `onDocumentMouseDown(event)`. In the latter, the clicked mouse position is retrieved and updated to the shaders, if the user has clicked on the windowpane.

4 Creating the Fog Shape

To create a fog shape, the clicked mouse position and a radius that varies over time is sent to the shader, in order to get a circle shape that increases/decreases, see Figure 2.

In the shader, a radius between the current fragment position and the clicked mouse position is determined, see Equation 1. This radius is compared to the varying radius received in the shader, in order to know which fragments that shall be colored red or have the background image color.

$$r_{shader} = \sqrt{(x_f - x_m)^2 + (y_f - y_m)^2} \tag{1}$$

Here r_{shader} represents the radius calculated in the shader, x_f, y_f, x_m and y_m represent the x- and y-coordinates for the fragment position and the clicked mouse position, respectively.

To determine the radius, that varies over time, two functions where implemented. The first function implements a linear approach, in which the radius increases for the first five seconds and then decreases until it reaches zero. For this the increasing rate for the radius is set to a larger value than the decreasing rate, in order to mimic how real fog spreads faster than it fades away .

The second function that was developed to describe the radius is displayed in Equation 2.

$$r = \frac{-sin(\frac{x}{2.0} - 3.141)}{e^{(\frac{x}{2.0} - 3.141)}} \tag{2}$$

Here, r represents the radius and x represents a value starting from 0 and increasing with 0.01 for each frame. A representation of the equation is displayed in Figure 3.

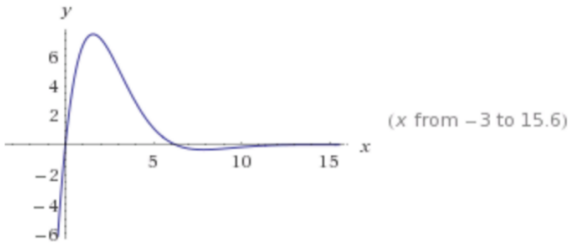


Figure 3: *This figure illustrates what Equation 2 looks like when x goes from -3 to 15.6. Starting with $x = 0$, the radius increases/decreases as shown in the figure. If the radius becomes negative (the slope between $x = 5$ and $x = 15$) the radius is set to zero.*

This second function is set to the default function, when determining the radius, whilst the first function can be activated in the GUI.

After the circular shape has been created, a sharp edge exists between the fog and the background. To remedy this, `smoothstep` is used, which performs a Hermite interpolation between 0 and 1, see Equation 3 and Figure 4.

$$smoothstep(x) = 3x^2 - 2x^3$$

where

$$x = clamp(\frac{x - minValue}{maxValue - minValue}, 0.0, 1.0) \tag{3}$$

Here the $minValue$ and $maxValue$ is set to 0.0 and the outline of the fog, respectively. In Figure 4 this outline is set to the varying radius of the fog.

Figure 4: *This figure illstrates how the circle shape looks after smoothstep have been performed.*

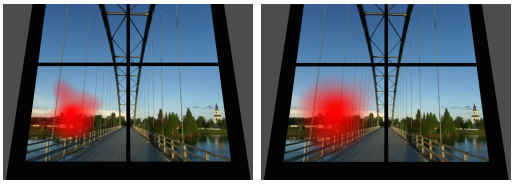
At this stage the shape of the fog is perfectly circular, and to remedy this a noise is added to its outline, see Equation 4 and Figure 5.

$$fog_o = r * noise(p * f_n) + 1.5 \tag{4}$$

Here fog_o and r represents the calculated outline for the fog and the radius, respectively. $noise()$ represents a function call to either Classic Perlin noise or Simplex noise, p represents the position for the current fragment given in world coordinates and f_n is a value ranging between $[0.0, 0.1]$.

5 Creating the Fog Color

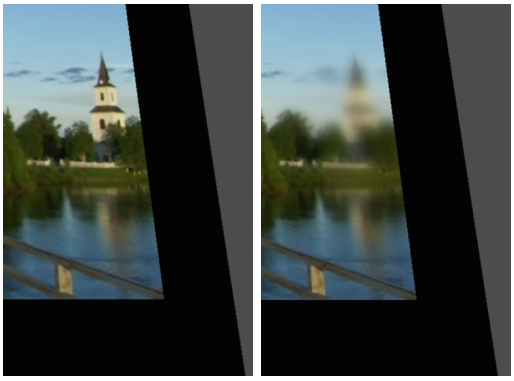
After the shape of the fog has been created, the next step is to determine the fog color. For a realistic



(a) *Simplex noise* **(b)** *Classic Perlin noise*

Figure 5: *This figure illustrates how the fog shape looks after noise has been added to the outline.*

fog, the background image should become blurry when fog is applied to the windowpane. This can be done is by extracting the texture coordinates for the surrounding fragments, and then applying a filter kernel, in order to get a weighted average for each fragment. This, however, requires a large filter kernel, in order to obtain good results, and thereby many texture lookups which significantly decreases the performance. Since the texture that is filtered is static, this operation can instead be pre-computed, which is the approach that was chosen for this project, see Figure 6.



(a) *No fog added* **(b)** *Fog added*

Figure 6: *These figures illustrate what the fog would look like if a blurry background color is displayed as the fog color.*

At this stage, the fog needs to become a mixture between this blurry background color and another color in order to make it more visible. In Figure 7, three colors were tested.

Figure 7: *This figure illustrates how the fog looks when the blurry background color is mixed with another color. From left to right: scene with no fog, scene with fog colored white, light gray and gray.*

The result from this test was that the a white fog color gave the best result, since the results with the gray colors gave too dark results.

In order to get a color for the fog that is not constant, a noise was also added to the background color to add a subtle variation in fog thickness. This feature can be activated by the user in the GUI.

5.1 Hidden Features

For the hidden features on the windowpane, a separate image file is used. This image file is colored white and black, with black representing the hidden features that shall be displayed on the windowpane, e.g. fingerprints. When the color for the fog is to be determined for a fragment, the color for the hidden feature is checked. If the color is darker than 0.7, then a mixture between the background color, the fog color and the blurry color is made, see Figure 8.

Figure 8: *Hidden features displayed with white fog color combined with the blurry color.*

6 Results and Conclusion

The result for this project is an interactive web application that enables the user to create fog on a windowpane in order to find hidden features. The thickness and the shape of the fog are created procedurally, whilst the color of the fog is achieved using a combination of textures and static colors, in order for it to be transparent. The application is interactive and enables the user to navigate the scene, using different key strokes, and change the fog properties utilizing the GUI.

6.1 Limitations

One limitation with this project is that the shader only handles one clicked mouse position, meaning that if the user presses at a new position, a new fog will be created and the old one will instantly disappear. One way that this limitation may be solved is to store the clicked mouse positions in an array, which is updated to the shader for each new position that is added. This would also require an array containing the varying radii for all clicked positions.

References

GUSTAVSSON STEFAN (2005). Simplex noise demystified. *http://webstaff.itn.liu.se/ stegu/simplexnoise/simplexnoise.pdf* [2016-02-29]