Laboratory work 12

1 Objectives

This laboratory presents you with basic notions about the concepts of fog effect, blending of colors and discarding of fragments.

2 Fog

We can increase the realism of a 3D scene by adding a fog effect which will create a particular atmosphere. By manipulating the attributes associated with the fog effect we can personalize the atmosphere and also we enhance the perception of depth.

The clear color should be chosen accordingly with the color of the fog.

```
glClearColor(0.5, 0.5, 0.5, 1.0);
```

In the fixed pipeline of the OpenGL there were three methods to compute fog (linear, exponential and square exponential). The same functionality we can accomplished in the programmable pipeline by adding the fog computation in our fragment shader.

2.1 Linear fog

The first method of computing fog is by using a linear interpolation function which will blend the fog color with the fragment color based on the distance of the fragment (computed in eye space).

$$fogFactor = \frac{fogEnd - fragmentDistance}{fogEnd - fogStart}$$

The fogFactor values should be between 0.0 and 1.0.

2.2 Exponential fog

A better result can be obtained by taking into consideration the reduction of light intensity over the distance. The attenuation factor which will be using represents the fog density and it is constant over the scene. The result is a fast decrease of the fog factor compared to the linear approach.

$$fogFactor = e^{-fragmentDistance * fogDensity}$$

2.3 Square exponential fog

The formula is basically the same with the previous one, the only difference come from the fact that we squared the fragment distance and the fog density.

$$fogFactor = e^{-(fragmentDistance * fogDensity)^2}$$

2.4 Tutorial

Add the following function inside the fragment shader:

```
float computeFog()
{
    float fogDensity = 0.05f;
    float fragmentDistance = length(fragmentPosEyeSpace);
    float fogFactor = exp(-pow(fragmentDistance * fogDensity, 2));

    return clamp(fogFactor, 0.0f, 1.0f);
}
```

The clamp function constrains a value to lie between two values (minValue and maxValue).

```
float fogFactor = computeFog();
vec4 fogColor = vec4(0.5f, 0.5f, 0.5f, 1.0f);
fColor = mix(fogColor, color, fogFactor);
```

The mix function linearly interpolates between two values. The resulting color can be computed also by:

```
fColor = fogColor * (1 – fogFactor) + color * fogFactor;
```

The color variable represents the color computed using the light sources specified in your scene.

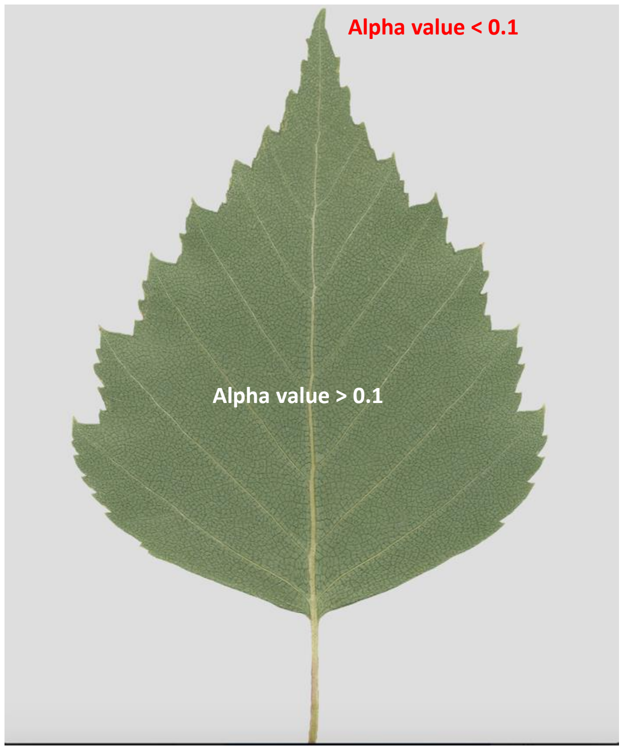
3 Discarding fragments

The alpha value from textures can be used to discard a fragment based on this value. For instance, you want to display a leaf and the 3D model is represented by a quad on which there is mapped a leaf texture. We can check the value of the alpha channel and discard the fragment if the value is less than 0.1.

```
vec4 colorFromTexture = texture(diffuseTexture, interpolatedTexCoords);
if(colorFromTexture.a < 0.1)
        discard;
fColor = colorFromTexture;
```

Update the call of the glTexImage2D function inside the ReadTextureFromFile method (from the Model3D class):

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, x, y, 0, GL_RGBA, GL_UNSIGNED_BYTE, image_data);
```



4 Blending colors

We can make objects to be transparent (or semi-transparent) by blending the object's color and the current color from the framebuffer (representing the objects already rendered) into a single color. The amount of transparency is defined by the alpha channel from the color representation. If the 4th component of the color is 1.0f then the object is opaque, if the value is 0.0f then the object is transparent. Values inside the (0.0f, 1.0f) interval represents a different amount of transparency.

Blending of colors is enabled in OpenGL by using the following function:

```
glEnable(GL_BLEND);
```

The resulting color after applying this operation is computed using the following equation:

$color = sourceColor * sourceFactor + destinationColor * destinationFactor$ where,

- *sourceColor* – represents the color of the object that is currently rendered
- *sourceFactor* – defines the influence of the source color
- *destinationColor* – represents the color from the framebuffer
- *destinationFactor* – defined the influence of the destination color

The sourceFactor and destinationFactor are specified by using the glBlendFunction. In general, we use the alpha value of the source as the sourceFactor and (1 – alpha) as the destinationFactor. For this we have to call the following function:

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

The order in which we render objects in this situation is very important. When drawing such scene, we need to:

- draw all opaque objects
- sort the transparent objects
- draw the transparent objects in order

5 Assignment

- Implement all the topics presented in this laboratory work.