CS380 Introduction to Computer Graphics
Homework Assignment #1

Snowflake 2D Animation

20160042 Inyong Koo

In this document, I will explain how my implementation (shown in Figure 1) satisfies the given specifications. I will not copy the code itself, but rather explain in plain text. So if you want to see how the actual implementation looks like, please refer to the code lines and comments from attached files. I'll note where to look for each task.

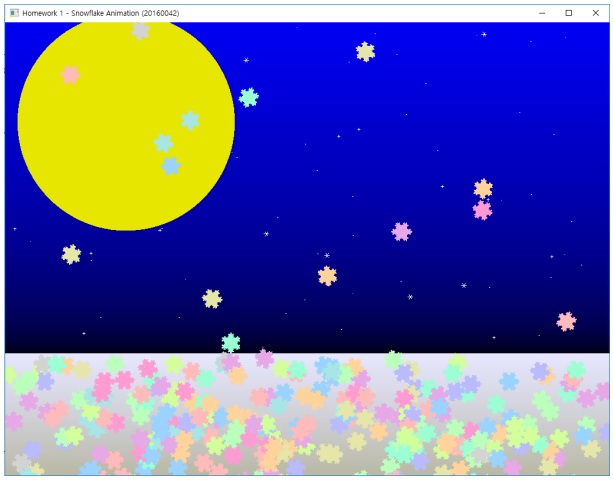

Figure 1: Execution screen

1 Snow animation

1.1 Koch snowflake

First objective is to implement Koch snowflake mesh generator. In `GenerateSnowflake` function, I initialize an equilateral triangle, and call `DrawKochLine` function for each edge. `DrawKochLine` function determines vertices of next iteration using vector operations, and recursively calls itself for newly defined edges. The implemented Koch snowflake is made out of 5 iterations.

Note that we should keep in mind the order of adding vertex data into mesh. In draw mode `GL_TRIANGLES`, we should add vertices in counter-clockwise order to make the triangle face front.

(*Please refer to* `Fractal.cpp`)

1.2 Animating snowflakes

Our next objective is to implement Animation and Snowflake class for smooth and natural motion. `Animate` function is repeatedly called in `main` function's while loop. By using timing function `glfwGetTime()`, I was able to learn time duration between function calls. Since the duration (*deltaTime*) is not constant, I predefined a constant value *FRAME_RATE* and set independent clock inside

the `Animate` function. 'motions' are executed only if the clock value gets larger than *FRAME_RATE*, where the clock value is returns to 0. Therefore, the 'motions' are executed over same time interval. Using the same principle, a snowflake is generated outside the viewing rectangle every 0.25 second. (This happens in `main` function.)

The 'motions' are defined in Snowflake class; `SpinSnowflake` function changes orientation of each snowflake in constant rate. `FallSnowflake` function changes position of each snowflake. A snowflake moves in constant speed towards y-axis, and randomly choose direction and amount of x-axis translation for each frame. But such implementation showed random, somewhat chaotic behavior, so I kept x-axis translation of its previous frame and applied weighted average with it to acquire smooth motion.

`CheckFallen` function determines whether the snowflake has 'fallen'. Each snowflake has random, different *_ground* value. When a snowflake reaches the *_ground*, it stops its motion and stay. However, since a snowflake is generated on every 0.25 second, keeping all those snowflake objects will cause lags, or even overflow error. So I removed the 'fallen snowflakes' when there are more than 300 snowflake objects in *snowflakes* vector.

(*Please refer to* `Animation.cpp` *and* `Snowflake.cpp`)

2 Background

Another specification for assignment 1 is to construct a gradient background scene with two different kinds of objects with different colors.

2.1 Gradient rectangle

I draw two adjacent rectangles to depict a snowing, night scenary. blue-black gradient color was used in upper rectangle to express the night sky, and white-yellow gradient color was used in lower rectangle to express the snow covered ground illuminating yellow moonlight. The rectangles are drawn in draw mode `GL_TRIANGLE_STRIP`.

(*Please refer to* `GenerateGradientRectangle` *function in* `main.cpp`)

2.2 Mesh 1 - Moon object

I draw the moon using draw mode `GL_TRIANGLE_FAN`. Moon object is simply a yellow circle, where first vertex is the center, and other vertices are laid on the circle, counter-clockwise order. Since `DefaultMaterial` class sets its color white, I updated its color later using `UpdateColor` function.

(*Please refer to* `GenerateBackgroundMeshType1` *function in* `main.cpp`)

2.3 Mesh 2 - Stars object

I draw the stars using draw mode `GL_LINES`. My original intention was to use `GL_POINTS`, but our vertex shader does not handle `GL_POINTS` and caused error while rendering. Instead, I used short lines to implement similar effect. Furthermore, I was able to differentiate stars into 3 groups; small ($\cdot$) , middle ($+$), and large ($*$). I randomly selected position of the stars in 'sky' region.

(*Please refer to* `GenerateBackgroundMeshType2` *function in* `main.cpp`)

3 Creativity

20 points of this assignment's grading is 'creativity' score. It means that my animation should not only meet the specifications, but also take a step further to make it look interesting and nice. I put a lot of effort to make my 2D animation to compose a 'scenary'. (For instance, I could have just put different two colored triangles for background meshs, but instead tried to depict the night sky by implementing the moon and the stars.) In this section, I'd like to appeal some 'creative' features of my implementation.

- A snowflake randomly chooses its color from 13 options, and stays on the ground after it has 'fallen'. Since snowflake is '눈꽃' in korean, I thought it would be interesting to make some kind of flower bed from fallen snowflakes, rather than let them just disappear below the window. Since snowflakes are constructed on same depth (z value), I had to alter rendering order to get later snowflakes gets above the older snowflakes.
- Using random variable with weighted average on snowflake animation allowed each snowflake to have independent, smooth, and natural descending path.
- I used background rectangle meaningfully by seperating regions - sky and ground. Snowflakes halt when they're in ground region, and stars are created only in sky region.
- I tried to differentiate not only colors of two background objects, but also their draw mode, so that I could express my understanding on different modes.