Snowflake 2D Animation

20150860 김태윤

1. Snow animation
   1. Koch snowflake mesh generator

I used a recursive method to create the Koch snowflake. A Koch snowflake starts with an equilateral triangle. To create this “base” triangle, I made a function (*Fractal::CreateBase*) that takes a float value for the length of a side of the equilateral triangle. The origin is set to be (0,0,0) to make rotation of the snowflake easier in the later steps. By placing the center of mass of the triangle at the origin, I constructed the three sides around it. The distance from the center of mass of an equilateral triangle to one side is given by

This value was calculated to offset the y coordinates and place the triangle at the origin. Trigonometry was used to calculate the coordinate of the top vertex. The three points found in this function was added to the list of vertices of the snowflake. Then the function *Fractal::SnowflakeRecursion* was used to recurse for each side of the base triangle. In the function, the base case is when the level decreases to 0, finishing the recursion. If the level is not 0 yet, the function is repeated for the four newly generated sides. The four new sides are found by the function *Fractal::CalculatePoints*. It returns the coordinates of the smaller triangle for each edge. The three vertices are found by first dividing the given edge into three sections. The two points dividing the edge become two of the three points. Then the top point is calculated by adding the perpendicular vector to the center point:

The three vertices are then added to the list of vertices. The order of insertion of these points are important because later in the *Fractal::FormMesh* function, the list is iterated to form a triangle every three vertices. If the order was not correct, a wrong mesh will be formed. In this step, a color is randomly selected from a list of defined colors then added to mesh as vertex data. When a snowflake was added to the world, the position was randomly chosen in the top area of the world. This is done in the constructor of the snowflake object.

* 1. Implementing smooth and natural animation

To implement the animation of the scene, a class called *Animation* exists. It contains a list of snowflakes, and the function *Animation::Animate*. The animation can be achieved by adjusting the position, orientation, scale etc for each frame of rendering. However, to implement a smooth and natural animation, the elapsed time between renders should be considered. For my scene, the snowflakes fall down in constant speed while rotating and swinging in random directions. In the algorithm, the elapsed time was used to calculate the next position and orientation of the snowflakes. The new position was calculated by adding the displacement to the current position:

In the velocity vector, the y component was a constant value, while the x component was a random number between -3 and 3. This way the snowflake seems as if it is swinging down from the sky. Another animation I have implemented is the rotation of the snowflakes. There is a function in the glm library that rotates an object. However, the axis of rotation is in the world coordinates. I wanted to make the snowflakes rotate around its center. To achieve this goal, I saved the current position in a variable, translated the snowflake to the world center, rotated around the z-axis, then translated back to the new position mentioned previously. Rendering after the transformation created a smooth translation and rotation animation. This animation was applied to all the snowflakes stored in the list of snowflakes.

1. Background
   1. Primitive objects with different colors

In the background, three kinds of objects are placed: abstract-style triangles, moon and stars. The triangles are created in the xy-plane and rotated before rendering. The axis of rotation is set so that the objects rotate in the 3-dimension. The two triangles have gradient colors that are different for each vertices. The moon object is made up of 10 triangles. The vertices are placed in the xy-plane so that the shape looks like a moon. The star object is made up of 6 triangles. The center square has long triangles on each side. Multiple stars are placed in the scene randomly.

* 1. Gradient background

The background object is created with two triangles connected together. The background is made large enough so that it fills the whole window. To create a natural gradient, the two vertices that overlap on the connecting edge have same colors.

1. Creativity

Keyboard input: The program takes four keyboard inputs. Pressing A on the keyboard adds two new snowflakes to the world. These snowflakes have random sizes and positions. Pressing W increases the fractal level of the snowflakes. Any snowflakes created afterwards will have increased level of recursion. Likewise, pressing Z decreases the level. The fractal level is declared as a global variable in the main class. Finally, pressing X removes all created snowflakes from the scene. This is done by clearing the list of snowflakes in the Animation class.

Moon color: The color of the moon is filled with a gradient. The center of the moon is lighter than the corners. The overlapping vertices of the triangles have same colors to achieve the gradient effect.

Snowflake size: The size of the newly generated snowflakes have random sizes. The function *Fractal::CreateBase* takes the size of the base triangle as a parameter and creates the base accordingly.