2023 Fall Semester

=

Creative Algorithm

Project #1 : Making Korean Traditional Dancheong Patterns

Student Number : 20221124
Student Name : 최보현

1. Project Overview

1) Project Topic

In this project, I would like to create a traditional Korean pattern using Processing. Among them, Dancheong patterns(단청) will be produced.

2) Reason for the topic selection

I have a lot of interest in Korean traditional culture. On the strength of the trend of the emergence of the word "Hip Traditon', which recognizes that tradition is recognized as trendy, I wanted to break the stereotype that tradition is old through **digital expressions of traditional patterns**.

Also, in terms of technology, it was judged that it would be easy to make flowers by creating one flower petal using the rotational function after using the computing processing characteristic(repeat) that can efficiently perform repetitive tasks. In addition, it was expected that if the function of repetition was further utilized, a pattern could be created and copied to create a pattern. This could be transformed by other producers and developed into a grander art.

3) Visual Features

There is a **symmetry** in the Dancheong pattern. Divide into four, six, eight, or ten and draw a single petal on the divided part. This pattern is used in various fields such as korean tiled roof(기와) and clothes, and gives a sense of stability of symmetry. This part can utilize the element of accurately dividing and rotating, one of the technologies of the computer.

There are also various colors in those patterns. Similar colors are used, but also color expression is free, such as using complementary colors. It will utilize the random value generation function of the computer.

2. Process

The process of making patterns is as follows.

1. Make a petal(just one).
2. Make a flower by rotating petals.
3. Make several small flowers inside in the same way.
4. Select the polygon to be the background (hexagon/ pentagon or etc)

1) Making a petal

We have to use `bezier()`function to make a single petal. `bezier()` is a function that curves along the coordinates of four points given. I will use several curves to shape the water droplets instead of filling the inside of the bezier.

```
void setup() {
  size(600,400);
  background(255);
  noLoop();
}

void draw(){
  noFill();
  for (int x = 200;x <=400;x+=5)
  {
    strokeWeight(3);
    stroke(255,0,255,20);
    bezier(300,100,x,170,x,300,300,300);
  }
}
```

By adjusting the vertices that serve as Bezier's standard, I created curves of appropriate size and left several traces so that petals were formed in the shape of water droplets. This is a reference to the YouTube lecture at the link below.
https://www.youtube.com/watch?v=danJTwo14yA

In this next process, the values that control the sharpness and length of the petals were made into variables, and random values were assigned using the **random()** function to create various petal shapes.

```
void petals3(int petalnum) {
  pushMatrix();
  translate(width/2, height/2);

  petal_thin = random(100, 150);
  petal_sharp = random(200, 300);
  petal_folded = random(0, 300);
  r = random(0, 256);
  g = random(0, 256);
  b = random(0, 256);

  for (int i = 0; i < petalnum; i++) {
    for (int x = 0; x <= 120; x += 5) {
      strokeWeight(3);
      stroke(r, g, b, 40);
      bezier(60, 0, x, petal_thin, x, petal_sharp, 60, 300);
    }
    rotate(TWO_PI / petalnum);
  }
  popMatrix();
}
```

2) Making a flower

Now that I've made a petal, I have to rotate it to make one flower. Before that, I assigned **NUM_PETALS** array and **petalnum**, which are variables for the number of petals, because we had to decide how many petals to make first. NUM_PETALS is an array that include variables(the number of petals) and back_shape is the background type of polygon (polygons will be described later). Declare the array and variables and use the for statement to put the values into the array. NUM_PETALS contains 6, 8, and 10 petals, respectively, and back_shape contains from square to decagon.

```
int[] NUM_PETALS = new int[3];
int[] back_shape = new int[7];
int petalnum = 0;
int shape;

void setup() {
  size(1000, 800);
  background(0);
  noLoop();
  frameRate(5);
  int x = 6;
  int y = 4;

  for (int i = 0; i < 3; i++) {
    NUM_PETALS[i] = x;
    x += 2;
  }
  for (int i = 0; i < 7; i++) {
    back_shape[i] = y;
    y += 1;
  }

  petalnum = NUM_PETALS[int(random(0, 3))];
  shape = back_shape[int(random(0, 7))];
}
```

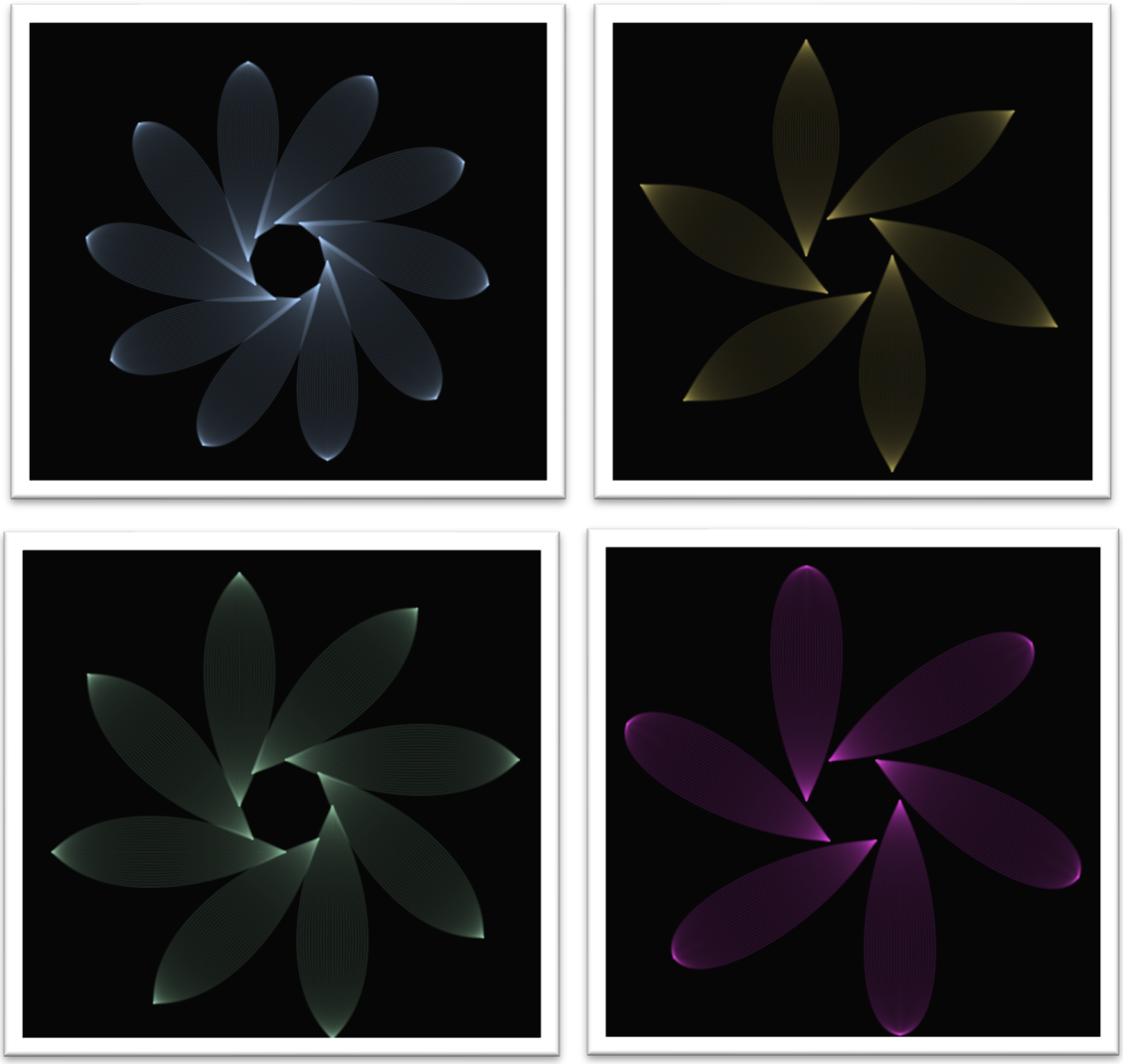
Based on the bezier described above, I tried to create each petals using **rotate()** and for loop. Since petal has to rotate one lap, it has to rotate the (2pi / num of petals) degree for total number of petals times to make a flower. For example, if a flower has eight petals, it must rotate eight times by an angle of (2Pi/8).

What is important at this time is **translate(), pushMatrix(), and popMatrix().** Using the rotate() function in Processing changes the center point, so it is important to move the center point and return it to its original state. pushMatrix() is responsible for storing the current graphics conversion state in the stack and later calling popMatrix() to return to the previous state.
translate() is used to move the position of an object or figure in a graphic context.

The code expressing this is as follows.

```
void petals3(int petalnum) {
  pushMatrix();
  translate(width/2, height/2);

  petal_thin = random(100, 150);
  petal_sharp = random(200, 300);
  petal_folded = random(0, 300);
  r = random(0, 256);
  g = random(0, 256);
  b = random(0, 256);

  for (int i = 0; i < petalnum; i++) {
    for (int x = 0; x <= 120; x += 5) {
      strokeWeight(3);
      stroke(r, g, b, 40);
      bezier(60, 0, x, petal_thin, x, petal_sharp, 60, 300);
    }
    rotate(TWO_PI / petalnum);
  }
  popMatrix();
}
```

If flowers are generated using the above code, they are as follows. If you look at the picture, you can see various colors, the number of petals, and various shapes.

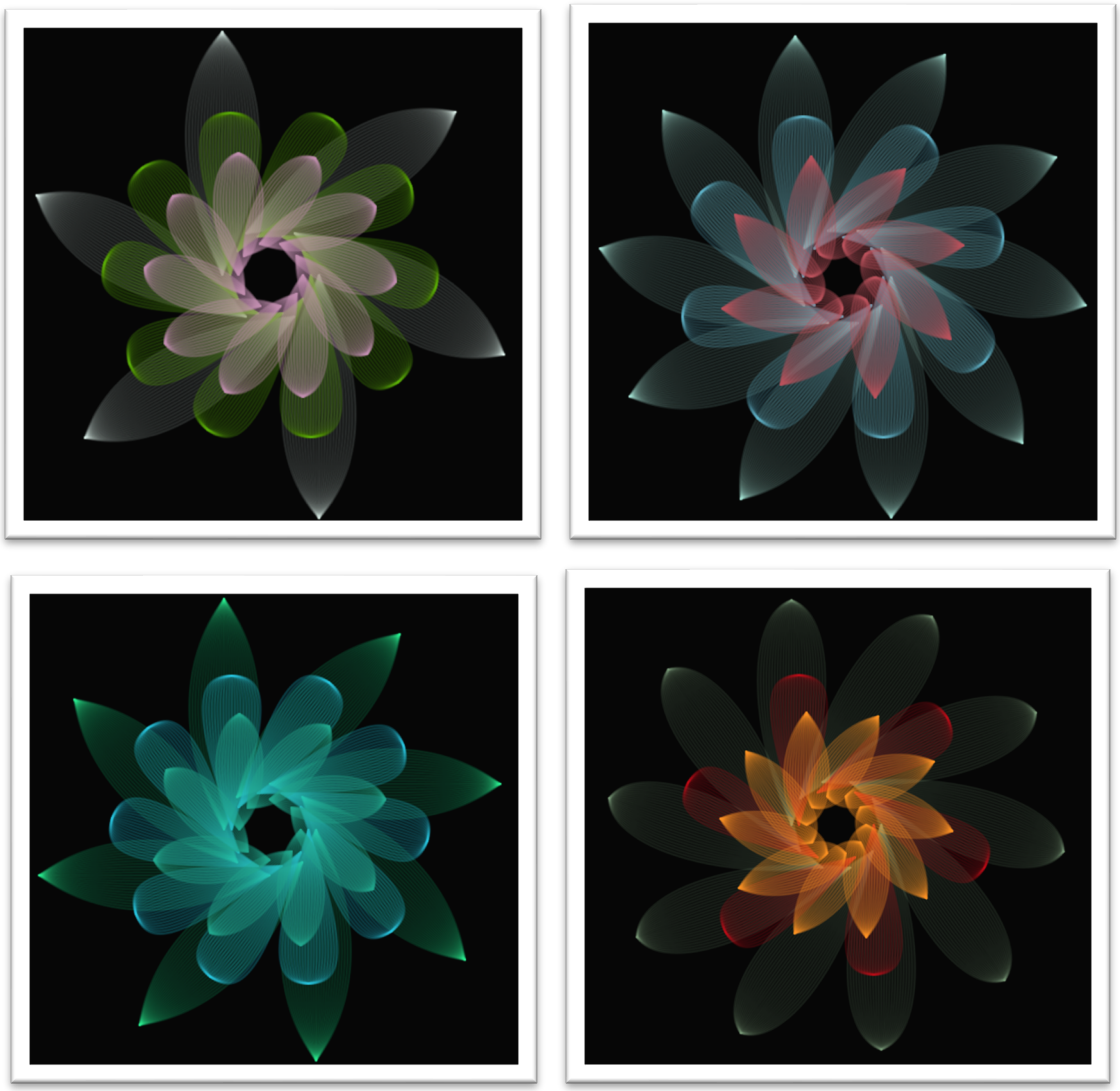


3) Making several flowers

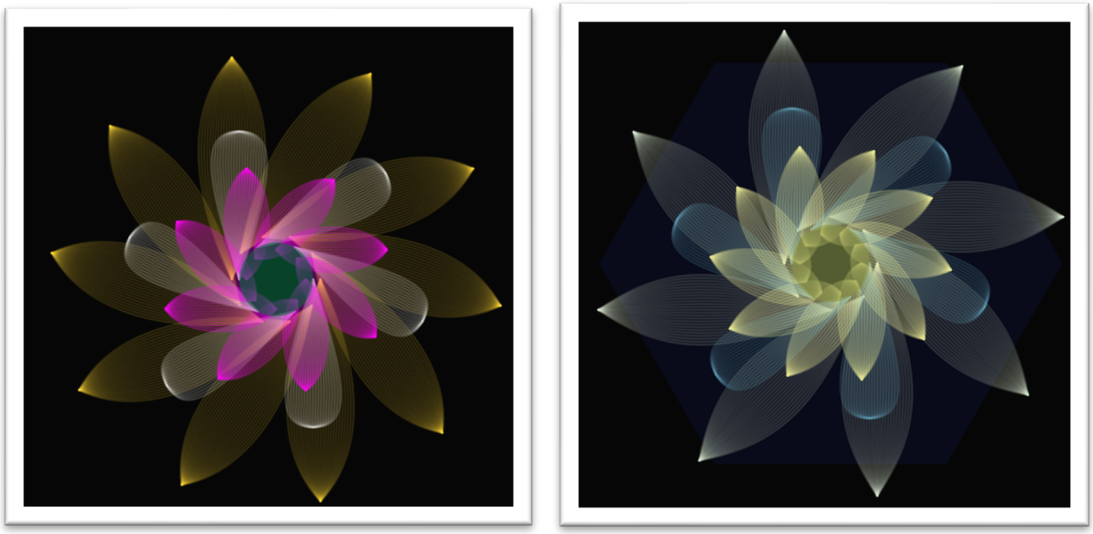
Now that we've made one flower, we have to adjust the size and stack several flowers in layers. Bezier's value can be adjusted appropriately. Here are two codes created by transforming the petals3() code and their results.

```
void petals2(int petalnum) {
  pushMatrix();
  translate(width/2, height/2);

  r = random(0, 256);
  g = random(0, 256);
  b = random(0, 256);

  for (int i = 0; i < petalnum; i++) {
    for (int x = 0; x <= 100; x += 5) {
      strokeWeight(3);
      stroke(r, g, b, 70);
      bezier(50, 0, x, 150, x, 200, 50, 200);
    }
    rotate(TWO_PI / petalnum);
  }
  popMatrix();
}
```

```
void petals1(int petalnum) {
  pushMatrix();
  translate(width/2, height/2);

  petal_thin = random(0, 50);
  petal_sharp = random(100, 150);
  petal_folded = random(0, 300);
  r = random(0, 256);
  g = random(0, 256);
  b = random(0, 256);

  noFill();
  for (int i = 0; i < petalnum; i++) {
    for (int x = 0; x <= 80; x += 5) {
      strokeWeight(3);
      stroke(r, g, b, 100);
      bezier(40, 0, x, petal_thin, x, petal_sharp, 40, 150);
    }
    rotate(TWO_PI / petalnum);
  }
  popMatrix();
}
```



Elipse has also been added to the center of the flower so that it can feel more like a flower.

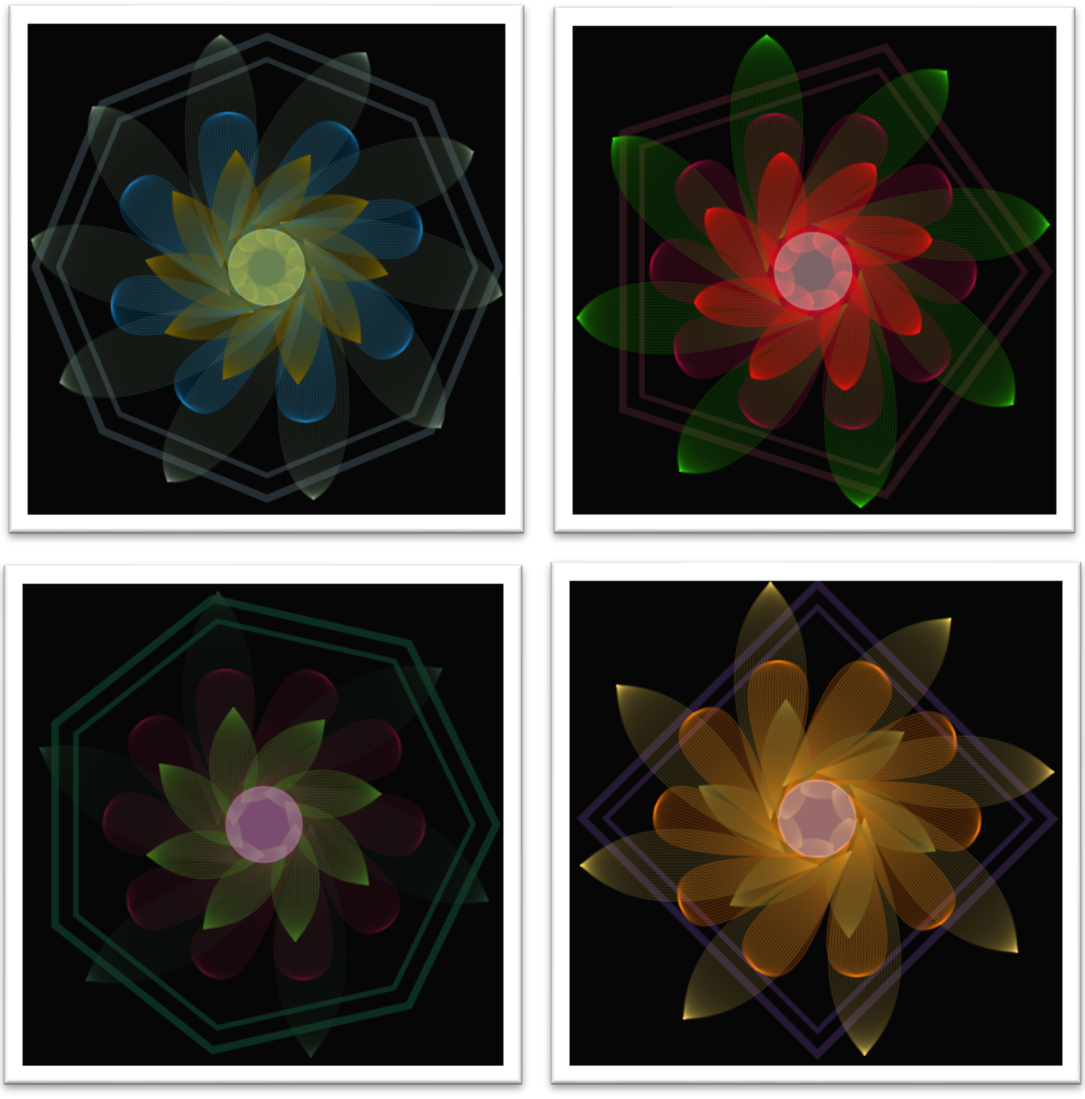


4) Select the polygon to be the background

The Dancheong pattern has a polygon around the flower. After creating a polygon() function, a custom function that creates a polygon, the values of back_shape, the array declared above, were taken one by one and the background was created behind it.

```
void polygon(float x, float y, float radius, int npoints) {
  float angle = TWO_PI / npoints;
  beginShape();
  for (float a = 0; a < TWO_PI; a += angle) {
    float sx = x + cos(a) * radius;
    float sy = y + sin(a) * radius;
    vertex(sx, sy);
  }
  endShape(CLOSE);
}
```

```
void draw() {
  background(0);
  r = random(0, 256);
  g = random(0, 256);
  b = random(0, 256);
  noFill();
  strokeWeight(10);
  stroke(r, g, b, 50);
  polygon(width/2, height/2, 300, shape);
  strokeWeight(7);
  polygon(width/2, height/2, 270, shape);
```



Making dancheong pattern by using processing is done.

3. **Appendix** : How to reduce memory

I was working on Processing and suddenly I got this error.

It was an error of using too much memory, which seemed to have occurred because it created too many variables and called functions that function the same many times. So, unlike the final code created above, I tried to change the code by minimizing allocation to variables and making the most of random() value or for loop.

The code is follows.

```
int shape;

void setup() {
  size(1000, 800);
  background(0);
  noLoop();
  frameRate(5);
}

void draw() {
  background(0);
  noFill();
  shape = (int)random(4,11);
  strokeWeight(10);
  stroke(random(0,255),random(0,255),random(0,255),50);
  polygon(width/2,height/2, 300, shape);
  strokeWeight(7);
  polygon(width/2,height/2, 270, shape);

  for (int k=0;k<3;k++) {
    petals((int)random(3,6) * 2,k*10);
  }

  noStroke();
  fill(random(0,255),random(0,255),random(0,255),150);
  ellipse(width/2,height/2,100,100);

}

void petals(int petalnum,int t) {
  pushMatrix(); // pushMatrix()와 popMatrix()를 사용하여 변환 행렬을 보존
  translate(width/2, height/2);

  noFill();
  int r = (int)random(0,255);
  int g = (int)random(0,255);
  int b = (int)random(0,255);

  for (int i=0; i<petalnum; i++) {
    for (int x=0;x<=80+(t*2);x+=5) {
      strokeWeight(3);
      stroke(r,g,b, 100-(t*3));
      bezier(40+t,0,x,random(0+5*t,50+5*t),x,random(100+5*t,150+5*t),40+t,150+5*t);
    }
    rotate(TWO_PI / petalnum);
  }
  popMatrix(); // 변환 행렬을 복원
}


void polygon(float x, float y, float radius, int npoints) {
  float angle = TWO_PI / npoints;
  beginShape();
  for (float a = 0; a < TWO_PI; a += angle) {
    float sx = x + cos(a) * radius;
    float sy = y + sin(a) * radius;
    vertex(sx, sy);
  }
  endShape(CLOSE);
}
```

The for-loop was used to declare only one function(petals()) and allow it to be called multiple times. In addition, the same behavior was made by using random() to generate random values within an appropriate range without creating an array. However, the use of only one patals function changed the shape of the flower a little. In the previous code, the patals1, patals2, and patals3 functions looked similar, but the values assigned to bezier() were slightly different.

However, in this code, by reducing or increasing to a certain value, a certain percentage of the value was substituted, resulting in a change in shape. Elements such as neatness decreased, but more **symmetry** could be seen. It's more closer to the Dancheong pattern!