멀티미디어 프로그래밍 과제 보고서

과제 4: Painerly Rendering

전공	소프트웨어학과
학번	21011778
이름	염지환
제출일	2024.05.13

[목차]

내용

1. 논문 분석

주어진 논문은 사진 이미지를 마치 화가가 그린 그림처럼 표현하도록 하는 Painterly Rendering 에 대해 다루고 있다. 이 논문에서는 그림의 요소를 각기 다른 Brush Size 를 가진 붓들을 이용한 Brush Stroke, 즉 붓질로 보고. 큰 Brush Size 를 가진 붓부터 작은 Brush Size 를 가진 붓 순으로 무작위 순서로 붓질을 그려내 그림을 그려내려고 한다.

2.1 에선 Brush Size 선언과 paint 함수, paintLayer 함수에 대해 다루고 있으며 paint 함수는 각 Brush Size 마다 Brush Size 에 비례하는 크기의 커널 사이즈로 가우시안 필터링을 진행한 reference 이미지를 만들어 paintLayer 함수에 매개변수로 주어 해당 Brush Size 를 가진 Stroke 들을 이용해 그림을 그린다. paintLayer 함수는 paint 함수에서 생성된 canvas 이미지와 reference 이미지, 그리고 Brush Size 를 매개변수로 받아 호출되며 Stroke 의 배열 S 를 만들고, canvas 와 reference 이미지의 오차를 저장할 배열 D 를 만든다. 그리고 가상의 그리드를 만드는데, 이때 한 그리드 칸의 크기는 Brush Size 에 비례하도록 만든다. 이후 각각의 그리드를 순회하며 그리드 안에서 가장 오차가 큰 지점을 Stroke 를 그릴 지점으로 정하고 그리드의 오차의 평균이 임계값 T 를 넘어갈 경우 makeStroke 함수를 통해 S 에 Stroke 를 추가한다. 모든 그리드의 순회가 끝나면 무작위 순서로 S 에 저장된 Stroke 들을 통해 canvas 에 원, 또는 Spline Stroke 를 그린다.

예시 프로그램에서 원을 이용한 그림 스타일은 지금까지의 내용으로 구현할 수 있으며 곡선을 만들기 위해서는 makeStroke 함수에 더 복잡한 기능이 필요하다. 이 논문의 2.2 부분에선 어떻게 해야 Curved Brush Stroke 를 구현할 수 있는지 설명한다. 먼저 스트로크의 색상은 reference 이미지 중 paintLayer 함수에서 매개변수로 받는 좌표의 픽셀에서 가져온다. 그리고 Spline Stroke 의 점의 집합에 매개변수로 받은 좌표를 추가하고, 현재 좌표 역시 매개변수로 받은 좌표로 초기화 한다. 그리고 직전 포인트의 (dx,dy)를 저장하기 위한 lastDx, lastDy 를 정의한다. 이후 반복문을 통해 Spline Stroke 의 직선을 연결할 점들을 생성한다. 먼저 현재 지점에서 reference 이미지와 canvas 사이의 색상 오차가 reference 이미지와 Stroke 색상의 색상 오차보다 작을 경우, 즉 Stroke 보다 canvas 가 색상이 reference 와 더

유사할 경우 함수를 종료한다. 또한 현재 좌표를 기준으로 reference 에서 현재 픽셀의 상 하, 좌 우 색상 변화가 0 일 경우, 즉 변화가 없을 경우 역시 함수를 종료한다. 이후 픽셀의 상 하, 좌 우 색상 변화량을 각각 gy, gx 라고 하여 가상의 벡터 (gx, gy)를 정의한다. 이 벡터의 방향은 색상이 더 밝게, 더 많이 변하는 방향이다. 우리는 Spline Stroke 를 만들 때 색상이 비슷한 부분을 선으로 이어야 하므로, 색상이 변하는 방향을 가진 벡터 (gx,gy)에 수직인 벡터를 구하여 색상이 변하지 않는 방향의 벡터 또한 얻을 수 있다. 색상이 변하지 않은 방향의 벡터를 (dx,dy) 라고 했을 때 dx = -gy, dy = gx 로 정의하여 (dx, dy)를 (gx, gy)에 수직인 벡터로 정의해 주었다. 이때 (dx, dy)가 Stroke 진행 방향의 거의 정 반대 방향일 경우 (dx, dy)를 (-dx, -dy)로 정의하여 반향을 반대로 바꿔준다. 그리고 (dx, dy)와 (lastDx, lastDy)를 특정 비율로 혼합하여 곡률을 바꿔주는 작업을 한다. Spline Stroke 의 점과 점 사이의 거리는 Brush Size 로 결정되므로 벡터의 크기가 필요 없기에 (dx, dy)를 단위벡터 로 바꾸어 준다. 그리고 x 와 y 를 (dx, dy)에 Brush Size 를 곱한 만큼 이동한 지점을 점들의 배열에 저장한다. 모든 점들의 저장이 끝나면 Stroke 를 반환한다.

마지막으로 논문의 3.1 에선 그림 스타일을 바꾸는 몇가지 매개변수들을 설명해 두었다. T 는 임계값으로 오차가 얼마 이상일 때 Stroke 를 생성할 건지 정하는 값이다. Brush Size 는 말그대로 Stroke 가 그려질 때 반지름이다. Curvature Filter, fc 는 2.2 에서 곡률을 결정할 때 쓰인다. Blur Factor 는 paint 함수에서 Brush Size 의 몇배 사이즈의 커널로 가우시안 필터링을 진행시킬 지 결정하는 변수이다. Stroke Length 의 최소값과 최댓값도 스타일을 변경시키는 매개 변수이며, Grid Size, fg 는 그리드 하나의 크기가 Brush Size 의 몇배인지 결정하는 변수이다. 이런 매개변수들을 변경하여 그림의 스타일을 변경할 수 있다.

2. 프로그램 구현 과정

I. 구조체, 함수 선언 및 Main 함수 정의

```
#pragma warning(disable:4996)
#include <stdio.h>
#include <stdlib.h>
#include <cmath>
#include <opencv2/opencv.hpp>

//Stroke 구조체 정의
typedef struct {
    int r,x,y; // 브러쉬 사이즈, 원의 x,y좌표
    int point[20][2]; // Spline의 각 지점
    int length; // Stroke 길이
    CvScalar color; // Stroke의 색상
}Stroke;

char* inputFileName(); // 파일 이름을 입력받는 함수
IplImage* paint(IplImage* src, int R[]); // 브러쉬 사이즈 마다 printLayer 함수를 이용해 그림을 그려 canvas를 반환하는 함수
void paintLayer(IplImage* canvas, IplImage* ref, int R); // makeStroke 함수를 이용해 생성한 Stroke를 이용해 한 레이어를 그리는 함수
Stroke makeStroke(int y0, int x0, int R, IplImage* ref); // 원형의 Stroke를 만드는 함수
Stroke makeSplineStroke(int y0, int x0, int R, IplImage* ref, IplImage* canvas); // Spline Stroke를 만드는 함수

int MODE = 0; // 렌더링 스타일 결정 변수
```

먼저 프로그램을 구현하는데 필요한 구조체와, 함수들을 정의하였다. Stroke 를 구조체로 정의하여 Brush Size 를 저장할 변수 r, 그리고 Stroke 의 위치를 저장할 변수 x 와 y, 그리고 Stroke 의 색상을 저장할 CvScalar 변수를 선언하였다. 처음 원형 Stroke 만을 이용해 이미지를 그릴 때는 위의 4 가지 변수만 사용하여 Stroke 를 구성했지만 Spline Stroke 를 이용해 이미지를 그리기 위해선 spline 을 구성하는 점들을 저장할 배열과 spline 의 점들의 개수, 즉 Stroke 의 길이가 필요했다. 따라서 좌표들의 배열 point 와 Stroke 의 길이 length 를 추가로 정의해 주었다

```
int main()
{
    srand(time(NULL)); //seed 초기화

    int R[5] = { 32, 16, 8, 4 ,2 }; // Brush Size 정의

    // 파일 경로 입력받아 이미지 불러오기
    char *path = inputFileName();
    IplImage* img = cvLoadImage(path);

    // Drowing Mode 입력받기
    printf("Select Drawing Mode (0=circle, 1=stroke): ");
    scanf("%d", &MODE);

    //원본 이미지 출력 후 paint함수를 이용해 painterly rendering 진행
    cvShowImage("img", img);
    IplImage* canvas = paint(img, R);
    cvShowImage("canvas", canvas);
    cvWaitKey();
}
```

main 함수에서는 먼저 무작위로 Stroke 를 그릴 때 rand()함수를 사용하기 위해 srand()를 사용하여 시드를 변경 해주었다. 그 후 Brush Size 를 지정해주었다. 주어진 예시 프로그램과 최대한 유사하게 동작하도록 각각 32, 16, 8, 4, 2 로 정해주었다. 다음으론 그림의 스타일을 변경하는 변수인 MODE 를 입력받아 사용자가 모드를 선택할 수 있도록 하였다. 그 후 inputFileName 을 이용해 파일 경로를 입력받아 원본 이미지 img 를 불러왔다. 그 후 img 를 출력하고 paint 함수를 이용해 렌더링이 끝난 함수를 canvas 에 저장하여 출력한다.

II. paint 함수 정의

```
IplImage* paint(IplImage* src, int R[])
{
    // 그림을 그릴 canvas 와 필터링한 이미지를 저장할 ref 선언
    CvSize size = cvGetSize(src);
    IplImage* canvas = cvCreateImage(size, 8, 3);
    IplImage* ref = cvCreateImage(size, 8, 3);
    cvSet(canvas, cvScalar(255, 255, 255)); // canvas를 흰색으로 초기화

    // fs = 3으로 선언(브러쉬 사이즈의 몇배 크기의 커널을 이용해 필터릴 할지 결정하는 상수)
    float fs = 3;

    // 각 브러쉬 사이즈마다 이미지를 필터링하여 printLayer함수를 이용해 그림 그리기
    for (int i = 0; i < 5; i++)
    {
        cvSmooth(src, ref, CV_GAUSSIAN, fs*R[i]+1); // src를 가우시안 필터로 블러처리하여 ref에 저장
        paintLayer(canvas, ref, R[i]); //printLayer함수에 브러쉬사이즈 전달
    }
    // 그림이 그려진 canvas 반환
    return canvas;
}
```

Paint 함수는 원본 이미지 src 와 Brush size 배열 R 을 전달받아 호출된다. 먼저 그림을 그릴 이미지인 canvas 와 원본을 필터링하여 저장할 ref 이미지를 원본 이미지와 같은 크기로 생성해주고 canvas 를 흰색으로 초기화해주었다. 그 후 가우시안 필터의 커널 크기를 Brush Size 의 몇 배로 할지 결정하는 상수인 fs 를 선언했고 위 소스코드에서는 3 으로 정의했다. 그 후 반복문을 통해 각 Brush Size 와 그에 맞게 필터링된 ref 이미지와 canvas 이미지를 매개변수로 하여 printLayer 함수를 호출한다. 마지막으로 반복문이 끝나면 완성된 이미지 canvas 를 반환한다.

III. paintLayer 함수 정의

```
void paintLayer(IplImage* canvas, IplImage* ref, int R)
{
    // ref의 너비와 높이를 w와 h에 저장
    CvSize size = cvGetSize(ref);
    int w = size.width;
    int h = size.height;

    // canvas 와 ref의 각 픽셀 사이의 오차를 저장할 2차원 배열 D를 ref의 픽셀 수만큼 할당
    int** D = (int**)malloc(sizeof(int*)*h);
    for (int i = 0; i < h; i++)
        D[i] = (int*)malloc(sizeof(int) * w);

    // ref와 canvas의 각 픽셀의 오차를 계산하여 D에 저장
    for (int y = 0; y < h; y++)
    {
        for (int x = 0; x < w; x++)
        {
            D[y][x] = 0;
            for (int k = 0; k < 3; k++) {
                D[y][x] += abs(cvGet2D(canvas, y, x).val[k] - cvGet2D(ref, y, x).val[k]);
            }
        }
    }
```

paintLayer 함수는 canvas 와 ref 이미지, 그리고 Brush size 를 매개변수로 받아 호출된다 먼저 원활한 이미지 처리를 위해 ref 의 가로와 세로 길이를 각각 w 와 h 에 저장한다. 그리고 canvas 와 ref 의 각 픽셀 사이의 오차를 저장하기 위한 2 차원 int 배열 D 를 이미지의 크기에 맞게 동적으로 할당해준다. 그 후 각 ref 와 canvas 를 순회하며 각 픽셀의 오차를 계산하여 D 에 저장한다.

```
// 그리드 사이즈 정하기
float fg = 1; // fg = 1로 선언(브러쉬 사이즈의 몇배를 그리드 하나의 사이즈로 정할지 결정하는 상수)
int grid = fg*R; // 그리드 사이즈 결정
int grid_x = w / grid; // x축 방향 그리드의 개수 저장
int grid_y = h / grid; // y축 방향 그리드의 개수 저장

// Stroke 배열을 그리드의 개수만큼 할당
Stroke* S = (Stroke*)malloc(sizeof(Stroke) * (grid_x*grid_y));
int stroke_cnt = 0; // Stroke의 갯수를 저장할 변수 stroke_cnt
```

그 후 가상의 그리드를 생성하기 위해 그리드 사이즈를 정한다. fg 는 그리드의 크기를 Brush size 의 몇배로 할 지 결정하는 상수로 위 소스코드에서는 1 로 설정하였다. 그 후 grid 에 R 에 fg 를 곱한 값을 저장하여 그리드의 크기를 저장해주고, w 와 h 를 grid 로 나누어 그리드의 가로 칸의 개수를 grid_x, 세로 칸의

개수를 grid_y 로 저장한다. 그리고 Stroke 배열 S 를 동적으로 할당하는데 이때 크기는 grid_x * grid_y 로 그리드의 전체 칸의 개수만큼 할당한다.

```
int T = 25; // Stroke를 생성되는 임계값 T는 25로 선언

// 각 그리드를 순회하며 그리드의 오차 계산
for (int y = grid/2 ; y < grid*grid_y; y += grid)
{
    for (int x = grid / 2; x < grid*grid_x; x += grid)
    {
        int max_error = 0; // 그리드 내에서의 최대 오차
        int max_x = 0; // 그리드 내 최대 오차를 가진 부분의 x좌표
        int max_y = 0; // 그리드 내 최대 오차를 가진 부분의 x좌표
        int grid_error = 0; // 그리드 내의 오차의 평균

        // 그리드를 순회하며 오차의 최댓값과 오차의 총합 연산
        for (int u = y - (grid / 2); u < y + (grid / 2)-1; u++)
        {
            for (int v = x - (grid / 2); v < x + (grid / 2)-1; v++)
            {
                // 오차값이 max_error보다 클 경우 max_error와 max_x,max_y 갱신
                if (D[u][v] > max_error) {
                    max_error = D[u][v];
                    max_x = v;
                    max_y = u;
                }
                // 오차값을 grid_error에 더함
                grid_error += D[u][v];
            }
        }
        grid_error /= (grid * grid); // grid_error를 grid의 제곱으로 나누어 평균을 구함

        // grid_error가 임계값보다 클 경우
        if (grid_error > T) {
            if (MODE == 0) // mode가 1일 땐 makeStroke로 max_x, max_y 위치의 원형 Stroke를 생성하여 S에 추가
                S[stroke_cnt] = makeStroke(max_y, max_x, R, ref);
            else if (MODE == 1) // mode가 1일 땐 makeSplineStroke로 max_x, max_y 위치의 Spline Stroke를 생성하여 S에 추가
                S[stroke_cnt] = makeSplineStroke(max_y, max_x, R, ref, canvas);
            stroke_cnt++; // stroke_cnt 1 증가
        }
    }
}
```

다음으로는 각 그리드를 순회하며 앞서 만든 2 차원 배열 D 를 이용해 그리드 내에서 ref 와 canvas 사이의 오차값이 가장 큰 지점의 좌표를 찾아내고, 그리드 내 픽셀의 오차들을 전부 더해 그리드의 총 픽셀 수로 나누어 그리드의 평균 오차를 구했다. 이렇게 구한 평균 오차가 임계값 T 보다 클 경우에 Stroke 를 생성하도록 하였다. 위 소스코드에서는 임계값을 25 로 정의하였다. Stroke 를 생성할 때 MODE 가 0,일 경우 원형 스트로크를 만드는 makeStroke 를 이용해 Stroke 를만들었고, MODE 가 1 일 경우 Spline Stroke 를 만드는 makeSplineStroke 함수를 이용해 Stroke 를 만들어 S 에 저장하고 stroke_cnt 를 증가시켜 Stroke 의 총 개수를 카운트하였다.

```
// 랜덤 순서 만들기
int* order = (int*)malloc(sizeof(int) * grid_x * grid_y);
//order를 순회하며 stroke_cnt 보다 작은 수를 무작위로 저장
for (int i = 0; i < stroke_cnt; i++)
{
    order[i] = rand() % (stroke_cnt);
    //이미 저장되어있는 수일 경우 무시하고 다시 저장하게 하여
    for (int j = 0; j < i; j++)
    {
        if (order[i] == order[j])
        {
            i--;
            break;
        }
    }
}
```

Stroke 배열 S 를 완성한 뒤에는 Stroke 배열을 무작위로 순회하기 위해 무작위로 정렬된 인덱스들의 집합 order 을 만들었다. Order 의 크기는 그리드의 개수가 되도록 동적할당 하였다. 그 후 반복문을 통해 stroke_cnt 보다 작은 수들을 무작위로 order 에 추가하였으며, 만약 중복되는 숫자가 추가될 경우 그 추가를 무시하고 다시 추가하는 방식으로 인덱스의 중복을 피하여 무작위로 정렬된 인덱스들의 배열을 만들 수 있었다.

```
//모드에 따라 canvas에 Stroke를 이용해 무작위 순서로 원 또는 선을 그림
if (MODE == 0)
{
    for (int i = 0; i < stroke_cnt; i++)
    {
        Stroke K = S[order[i]];
        cvCircle(canvas, cvPoint(K.x, K.y), R, K.color, -1); // canvas에 Stroke의 x, y좌표에 꽉찬 원을 그림
    }
}
else if (MODE == 1)
{
    for (int i = 0; i < stroke_cnt; i++)
    {
        Stroke K = S[order[i]];
        //Stroke의 길이가 최소길이보다 길 경우 Stroke의 point배열에 저장된 지점들을 따라 선을 그어 spline을 형성
        if (K.length >= 4)
        {
            for (int j = 0; j < K.length - 1; j++)
                cvLine(canvas, cvPoint(K.point[j][0], K.point[j][1]), cvPoint(K.point[j + 1][0], K.point[j + 1][1]), K.color, R);
        }
    }
}
```

이후 MODE 값에 따라 Stroke 배열 S 를 canvas 에 적용하였다. MODE 가 0 일 경우 order 에 저장된 무작위 순서에 따라, makeStroke 으로 만들어진 원형 Stroke 들을 cvCircle 함수를 이용해 꽉찬 원으로 canvas 위에 그리게 하였다. MODE 가 1 일 경우 역시 무작위 순서에 따라 그렸으며 makeSplineStroke 로 만들어진 Spline

Stroke 를 각 Stroke 의 길이가 최소 길이보다 클 경우, point 에 저장된 점들을 따라 그려지는 직선을 canvas 위에 그리게 하였다.

```
// 동적할당 해제
for (int i = 0; i < h; i++)
    free(D[i]);
free(D);
free(S);
free(order);

// canvas를 출력하고 1초 후 다시 함수 종료
cvShowImage("canvas", canvas);
cvWaitKey(1000);
```

마지막으로 동적할당 되었던 배열 D, S, order 의 할당을 해제하고 cvShowImage 를 통해 각 레이어가 칠해질 때 마다 canvas 를 출력하게 하였으며 cvWaitKey(1000)을 이용해 레이어가 칠해지는 반복문이 1 초 간격을 두고 실행되도록 하였다.

IV. makeStroke 함수 정의

```
Stroke makeStroke(int y, int x, int R, IplImage* ref)
{
    Stroke K; // 반환할 Stroke K 선언
    K.r = R; // r에 브러쉬 사이즈 저장
    K.color = cvGet2D(ref,y,x); // ref에서 전달받은 (max_x, max_y)픽셀의 컬러를 가져와 저장
    K.y = y; // Stroke를 그릴 위치인 max_y 저장
    K.x = x; // Stroke를 그릴 위치인 max_x 저장

    return K; // K 반환
}
```

원형의 Stroke 를 생성하는 makeStroke 함수는 위와 같다. 먼저 새로운 Stroke K 를 선언하고 paintLayer 에서 구한 그리드 내에서 오차가 가장 큰 픽셀의 x 좌표와 y 좌표를 Stroke 의 위치 K.x, K.y 로 저장하고 Brush Size 는 K.r 에 저장, ref 이미지에서 K.x, K.y 에 위치한 픽셀의 색상을 불러와 K.color 에 저장하였다. 그리고 K 를 반환하여 반환된 Stroke 값을 배열에 추가할 수 있도록 하였다.

V. makeSplineStroke 함수 정의

```
Stroke makeSplineStroke(int y0, int x0, int R, IplImage* ref, IplImage* canvas)
{
    int maxStrokeLength = 16; // Spline Stroke의 최대 길이
    int minStrokeLength = 4; // Spline Stroke의 최소 길이
    float fc = 0.5; // fc = 0.5로 선언 (곡률 조절 상수)
    Stroke K; // 반환할 Stroke K 선언
    K.r = R; // r에 브러쉬 사이즈 저장
    K.color = cvGet2D(ref, y0, x0); // ref에서 전달받은 (max_x, max_y)픽셀의 컬러를 가져와 저장
    K.point[0][0] = x0; // Stroke를 그리기 시작할 위치인 max_x를 첫번째 포인트로 저장
    K.point[0][1] = y0; // Stroke를 그리기 시작할 위치인 max_y를 첫번째 포인트로 저장
    int y = y0; // gx, gy를 계산할 현재 점 y를 max_y로 초기화
    int x = x0; // gx, gy를 계산할 현재 점 x를 max_x로 초기화
    float lastDy = 0; // 이전 포인트에서 구한 dy를 저장할 변수 lastDy를 0으로 초기화
    float lastDx = 0; // 이전 포인트에서 구한 dx를 저장할 변수 lastDx를 0으로 초기화
```

makeStroke 함수를 이용해 원형 Stroke 를 이용해 그림을 그리는데 성공한 뒤에는 makeSplineStroke 함수를 정의하였다. makeSplineStroke 함수는 makeStroke 가 받는 매개변수들을 받고 이에 더해 canvas 이미지도 매개변수로 받는다. 함수의 첫 부분에서는 SplineStroke 를 만들기 위한 변수들을 선언한다. Stroke 길이의 최댓값과 최솟값, Stroke 의 곡률을 조정할 상수 fc 를 선언하고, 새 Stroke 구조체 변수 K 를 선언하여 Brush Size 와 color 를 저장한 뒤에 Spline 을 이루는 점들의 배열인 K.point 배열의 첫번째 인덱스에 x0, y0 을 저장한다. 그리고 gx, gy 를 계산할 현재 위치의 좌표를 x0, y0 으로 초기화하고 직전의 dx 와 dy 값을 저장할 변수 lastDx 와 lastDy 변수를 선언하였다.

```
    int i;
    for (i = 1; i < maxStrokeLength; i++)
    {
        // x, y가 1보다 작거나 각각 width-2, heigth-2보다 클 경우 함수 종료
        if (x < 1 || y < 1 || x > ref->width-2 || y > ref->height-2) {
            K.length = i;
            return K;
        }

        int r_cError = 0; // ref 와 canvas 사이의 오차
        int r_sError = 0; // ref 와 Stroke 사이의 오차

        // ref 와 canvas 사이의 오차 계산
        for (int k = 0; k < 3; k++)
            r_cError += abs(cvGet2D(ref, y, x).val[k] - cvGet2D(canvas, y, x).val[k]);
        // ref 와 canvas 사이의 오차 계산
        for (int k = 0; k < 3; k++)
            r_sError += abs(cvGet2D(ref, y, x).val[k] - K.color.val[k]);

        // i가 최소 스트로크 길이보다 길고 ref와 canvas의 오차보다 ref와 stroke의 오차가 클 경우 함수 종료
        if (i > minStrokeLength && r_cError < r_sError) {
            K.length = i;
            return K;
        }
```

변수 선언을 마친 뒤에는 반복문을 통해 spline 을 이루는 점들을 배열에 추가해가며 Spline Stroke 를 만들어가야 했다. 우선, (x,y)에서 상하 좌우의 색상 변화량 (gx,gy)를 계산할 때 변수가 이미지 범위를 벗어나거나 Spline Stroke 의 point 가 이미지 밖에 위치하는 경우를 막기 위해 특정 범위가 넘어가는 경우 바로 함수를 종료하게 하였다 그 후 (x,y)에서 ref 와 canvas, ref 와 Stroke 색상 사이의 오차를 비교하여 Stroke 의 길이가 최솟값보다 길고 ref 의 색상이 Stroke 보다 canvas 에 가까울 경우, 함수를 종료하고 K 를 반환하도록 하였다.

```
float gy = 0; // y방향 색상 변화량 gy
float gx = 0; // x방향 색상 변화량 gx

// x, y를 기준으로 상,하,좌,우의 색 변화량을 계산하여 y방향 색상 변화량, x방향 색상 변화량을 구해 벡터(gx,gy) 구하기
for (int k = 0; k < 3; k++)
    gx += cvGet2D(ref, y, x+1).val[k] - cvGet2D(ref, y, x-1).val[k];
for (int k = 0; k < 3; k++)
    gy += cvGet2D(ref, y+1, x).val[k] - cvGet2D(ref, y-1, x).val[k];

// gx와 gy가 모두 0일 경우 함수 종료
if (sqrt(gx * gx + gy * gy) == 0) {
    K.length = i;
    return K;
}

// 색상이 변하는 방향의 벡터 (gx,gy)와 수직인 벡터(dx,dy)를 계산
float dx = -gy;
float dy = gx;

// dx,dy의 방향이 lastDx, lastDy의 방향과 반대일 경우 방향을 반대로 설정
if (lastDx * dx + lastDy * dy < 0) {
    dx = -dx;
    dy = -dy;
}
```

이후 (x,y)를 기준으로 y 방향 색상 변화량을 gy, x 방향 색상 변화량을 gx 로 선언하였다. 그리고 계산을 통해 (gx,gy)의 값을 얻었다. 그 후 만약 gx 와 gy 가 둘 다 0, 즉 상하좌우 픽셀의 색상에 변화가 없을 경우에도 함수를 종료시키고 K 를 반환하였다. 이후 벡터 (gx,gy)와 방향이 수직인 벡터를 저장하기 위해 (dx,dy)를 선언하고 dx = -gy, dy = gx 로 초기화 하여 색상이 변하지 않는 방향을 가리키는 벡터인 (dx,dy)를 얻어내었다. 이때 (dx,dy)의 방향이 Stroke 의 진행방향과 정 반대일 경우 dx = -dx, dy = -dy 로 (dx,dy)의 방향을 반대로 바꿔주었다.

```
        // 상수 fc를 통해 dx,dy 와 lastDx,lastDy의 비율을 조절해 더하여 dx dy에 저장하여 곡률 조정
        dx = fc * dx + (1 - fc) * lastDx;
        dy = fc * dy + (1 - fc) * lastDy;

        // (dx,dy)를 단위벡터로 만드는 연산
        dx = dx / sqrt(dx * dx + dy * dy);
        dy = dy / sqrt(dx * dx + dy * dy);

        //브러쉬 사이즈 만큼 (dx,dy) 방향으로 포인트 이동
        x = x + R * dx;
        y = y + R * dy;

        //lastDx, lastDy에 dx dy 저장
        lastDx = dx;
        lastDy = dy;

        // (x,y) 를 K.point에 추가하여 spline의 한 점 추가
        K.point[i][0] = x;
        K.point[i][1] = y;
    }
    K.length = i; // K.length에 i 저장
    return K; // 완성된 Spline Stroke K 반환
}
```

그 뒤에는 상수 fc 를 이용해 (dx,dy)와 (lastDx,lastDy)를 특정 비율로 더하여 SplineStroke 의 곡률을 정하도록 하였다. 그 뒤 (dx,dy)를 벡터의 크기만큼 나누어 단위벡터로 만들었다. 이후 x 와 y 에 각각 R*dx, R*dy 를 더하여 (x,y)에서 (dx,dy)방향으로 Brush Size 만큼 이동한 곳을 새로운 (x,y)로 지정하고 이 좌표를 point 배열에 저장하여 Spline 을 이루는 점을 추가하였다. 이후 i 를 K.length 에 저장하여 Stroke 의 길이를 저장하고 완성된 Stroke K 를 반환한다

3. 결과

논문에서 주어진 상수들을 조절하여 최대한 주어진 예시 코드와 유사하게 동작하도록 프로그램을 구성하도록 하였고 그 결과 최종적으로 결정된 상수는 위의 소스코드와 같이 fg = 1, fs = 1, fc = 0.5, maxStrokeLength = 16, minStrokeLength = 4, T = 25 였다.

다음은 프로그램을 통해 변형한 이미지들의 예시이다, 모든 예시는 SplineStroke 를 이용해 그린 이미지들이다.

Source 1

Result 1

Source 2

Result 2

Source 3

Result 3