# 멀티미디어프로그래밍

## 과제 4 보고서


| | | | |
|---|---|---|---|
| 제출일 | 2024. 05. 13 | 전공 | 소프트웨어학과 |
| 과목 | 멀티미디어프로그래밍 | 학번 | 21011746 |
| 담당교수 | 박상일 교수님 | 이름 | 양현석 |

1. Painterly Rendering

a) 논문 분석

입력된 사진을 화가풍의 이미지로 전환하는 프로그램을 만들어야 합니다. 방법은 다양한 사이즈의 브러쉬로 캔버스를 덧대어 칠해서 디테일을 더하는 것이라 논문에 작성되어 있습니다. 이는 Circle이나 Spline을 칠하는 방식 2가지로 나누어서 할 수 있는데, 먼저 Circle부터 보겠습니다.

b) 문제 풀이

먼저, 이미지를 받은 후에 이미지의 사이즈와 동일한 캔버스(cvScalar(255,255,255)로 초기화) 를 제작합니다. 이제 붓사이즈를 정할 것인데, 제공받은 실행파일과 최대한 비슷하게 만들기 위해 5개로 정해주겠습니다. 다음과 같습니다.

```
int R[5] = { 31,19,11,5,3 };
```

다음은 논문 상 Pseudo-Code로 제공된 paint함수를 구현하겠습니다. 원본 이미지를 cvSmooth함수로 처리를 저장할 ref를 생성합니다. 이 이미지를 paintLayer함수로 넘겨 줄 것입니다. 알다시피 인자로 들어가는 kernel값은 항상 홀수이어야 하고, 또한 이 값은 브러쉬의 사이즈와 비례해야 한다고 명시되어 있습니다. 따라서 코드는 다음과 같습니다.

```
void paint(IplImage* src, IplImage* dst, int R[], int flag) {

	IplImage* ref = cvCreateImage(cvGetSize(src), 8, 3);
	int kernel;

	for (int i = 0; i < 5; i++) {
		kernel = R[i] * 3;

		cvSmooth(src, ref, CV_GAUSSIAN, kernel);
		cvShowImage("dst", dst);
		cvWaitKey(1000);

		if (R[i] == 3 && flag == 1) {
			R[i] = 2;
		}
		paintLayer(dst, ref, R[i], flag);
	}
}
```

매개변수 flag는 circle로 칠할 것인지, spline으로 칠할 것인지 정하는 값입니다. 보라색으로 처리

된 부분은 앞에서 설명했던 것처럼 브러쉬의 사이즈와 kernel 값이 비례해야 한다고 했었지만, spline으로 그림을 그릴 때 더 디테일한 브러쉬 사이즈가 필요했기에 R이 3일 때만 2로 조정해주었습니다.

다음은 ref를 넘겨받은 paintLayer함수를 구현해보겠습니다. 이 단계에서는 Stroke 구조체를 만들어서 사용합니다. 캔버스와 ref의 에러 값의 평균을 계산하고, 임계값(T)를 넘을 경우에 가장 에러 값이 큰 지점에 색을 칠합니다. Circle일 경우 바로 색을 칠하고, Spline이면 makeSplineStroke함수에 넘겨주어 추가적인 단계를 진행합니다. Stroke 구조체와 코드는 다음과 같습니다.

```
typedef struct Stroke {
	CvPoint P;
	CvScalar C;
}Strk;
```

```
void paintLayer(IplImage* dst, IplImage* ref, int R, int flag) {

	Strk* S;

	int idx = 0;

	int h = cvGetSize(ref).height;
	int w = cvGetSize(ref).width;

	// 그리드 한 변의 길이 gridX, girdY
	int divX = w / R;
	int divY = h / R;
	int gridX = w / divX;
	int gridY = h / divY;

	//		int S_size = divX * divY;
	//		메모리 효율을 위해 위처럼 사이즈를 정하려 했으나, 상수 값으로 고정시킴.
	int S_size = 10000000;

	S = (Strk*)malloc(sizeof(Strk) * S_size);


	for (int y = 0; y < h; y += gridY) {
		for (int x = 0; x < w; x += gridX) {

	//		err : T와 비교하기 위한 전체 평균 변수, sum_err : Max_ERR를 구하기 위한 변수
			float err = 0.f;
			int ct = 0;
			int m_x = 0;
```

```
int m_y = 0;

float sum_err;
float Max_ERR = 0.f;

// 평균 에러 구하기
for (int v = 0; v < gridY; v++) {
	for (int u = 0; u < gridX; u++) {

		//	initialization
		sum_err = 0;

		if (x + u < 0 || x + u > w - 1) continue;
		if (y + v < 0 || y + v > h - 1) continue;

		//	canvas와 ref의 에러 값 구하기
		CvScalar f = cvGet2D(dst, y + v, x + u);
		CvScalar g = cvGet2D(ref, y + v, x + u);

		for (int k = 0; k < 3; k++) {
			err += abs(f.val[k] - g.val[k]);
			sum_err += abs(f.val[k] - g.val[k]);
		}

		if (Max_ERR < sum_err) {
			Max_ERR = sum_err;
			m_x = x + u;
			m_y = y + v;
		}
		ct++;
	}
}

//	ct로 카운팅 후 평균 내기
err /= (float)ct;

//	임계값 T는 임의로 지정 -> 변경 가능 -> Thresholding의 원리
if (err > 30) {
	S[idx].C = cvGet2D(ref, m_y, m_x);
	S[idx].P = cvPoint(m_x, m_y);
	idx++;

//	만일 배열의 크기를 벗어나게 될 경우 반복문을 탈출한다.
```

```
                                        if (idx >= S_size - 1) { break; }
                                }
                        }
                }

                if (flag == 0) {
                        //      paint all strokes in S on the canvas in random order
                        shuffleArray(S, idx);

                        for (int i = 0; i < idx; i++) {
                                cvCircle(dst, S[i].P, R, S[i].C, -1);
                        }
                }

                else {
                        //      paint all strokes in S on the canvas in random order
                        //      항상 Circle이 그려지는 지점이 spline의 시작이다.
                        shuffleArray(S, idx);

                        for (int i = 0; i < idx; i++) {
                                makeSplineStroke(S[i].P.x, S[i].P.y, R, ref, dst);
                        }
                }

                free(S);
}
```

위의 코드에서 주석으로 표시되었듯이, spline의 시작점은 Circle의 시작점과 동일합니다. 그렇기에 Stroke 배열의 Circle과 마찬가지로 랜덤으로 섞어주어서 makeSplineStroke함수로 넘겨줍니다. 랜덤으로 섞는 것은 rand()함수를 이용하여 만들어 주었습니다. i를 배열의 끝부터 인덱스 1까지 읽어오며 i번째와 j(rand()값을 i+1로 나눈 나머지)번째 값을 swap해줍니다. 코드는 따로 달지 않겠습니다.

마지막으로 makeSplineStroke입니다. 첫 시작점과 색상을 인자로 받습니다. 또한 선을 잇는 최대 횟수와 최소 횟수를 정해주어야 하는데, 최대 횟수는 무한루프를 돌지 않기 위해 설정합니다. 여기에서 새로운 Stroke 배열을 만들어주는데, 이의 각 원소는 Control Point에 해당합니다.

첫 번째 지점은 인자로 넘겨받은 그 지점을 기준으로 세우고 두 번째 지점부터 반복문을 통해 구합니다. 여기에서 Gradient라는 개념을 사용하는데, 색의 변화율을 의미하는 것으로 사용하겠습니다. 하지만 색을 칠할 때는 색이 밝아지는 방향과 수직으로 칠해야 합니다.(색 변화율이 사실상 없는 지점을 따라간다고 생각하고 진행함.) 이때 Gradient와 수직 방향인 벡터를 구하게 되면 2가지의 벡터가 나오게 됩니다. 그 중 spline의 방향을 현재의 벡터와 이전의 벡터의 내적이 양수가

되는 벡터를 사용해야 합니다. 만일 내적이 음수라면 벡터의 방향을 180도 돌려줍니다. (이때 index가 1인 경우에는 이전의 벡터가 (0,0)이므로 내적이 음수뿐만 아니라 0일 때에도 반대로 돌려주는 방식을 채택함. – 논문의 의사코드와 다른 점) 다음은 벡터를 단위벡터로 바꾸고, 움직임을 좀 더 부드럽게 하기 위해 fc라는 값을 도입합니다. 이를 이용해 이전 벡터와 현 벡터를 선형 보간의 원리로 더하여 현 벡터에 대입합니다. 이 값은 0.5로 고정해주었으며, 실제로 브러쉬의 이동이 조금 더 부드러워지는 것을 확인할 수 있습니다.

이렇게 계산한 벡터(dx, dy)를 이용해 (x, y)값을 조정합니다. 움직이는 거리는 R(브러쉬 사이즈)입니다. (x, y)를 배열에 저장하여 출력합니다.

이 코드 내에서 예외 처리를 해야 할 지점이 몇 가지 있습니다. 첫 번째, 캔버스에 칠을 하는데 이미 칠해져 있는 것보다 지금 칠할 것이 더 큰 에러 값을 가져올 때입니다. 이때는 index가 minStroekLength값보다 클 경우에만 해당합니다. 두 번째, Gradient를 계산할 때에 이미지의 범위를 벗어나는 경우입니다. 세 번째, Gradient 벡터가 0일 경우입니다. 한 마디로 색의 차이가 없는 것입니다. 마지막으로, x와 y가 이미지의 범위를 벗어날 때입니다. 이렇게까지 해준다면 코드를 완성할 수 있습니다. 다음은 그 코드입니다.

```
void makeSplineStroke(int x0, int y0, int R, IplImage* ref, IplImage* dst) {
	CvScalar strokeColor = cvGet2D(ref, y0, x0);

	const int maxStrokeLength = 30;
	int minStrokeLength = 7;
	int idx = 1;

	Stroke K[maxStrokeLength];

	//	first element(에러 값이 가장 컸던 첫 지점 - 처음 점을 찍은 곳)
	K[0].C = strokeColor;
	K[0].P = cvPoint(x0, y0);


	int lastDx = 0, lastDy = 0;

	int x = x0, y = y0;

	//	논문에는 나와있지만, 교수님이 안 해도 된다고 한 것.
	float fc = 0.5f;

	for (int i = 1; i < maxStrokeLength; i++) {

		if (i > minStrokeLength && (abs(getBri(cvGet2D(ref, y, x)) - getBri(cvGet2D(dst, y, x))) < abs(getBri(cvGet2D(ref, y, x)) - getBri(strokeColor)))) {
			break;
		}

		//	gradient를 구할 범위
```

```
		int n = 1;

		//		범위를 벗어날 때
		if (x + n > ref->width - 1 || x - n < 0 || y + n > ref->height - 1 || y - n < 0) continue;

		//		a vector of gradient
		float gx = getBri(cvGet2D(ref, y, x + n)) - getBri(cvGet2D(ref, y, x - n));
		float gy = getBri(cvGet2D(ref, y + n, x)) - getBri(cvGet2D(ref, y - n, x));

		//		색의 차이가 없다!
		if (gx == 0 && gy == 0) {
			break;
		}

		//		수직 방향의 벡터
		float dx = -gy;
		float dy = gx;

		//		reverse direction
		if (lastDx * dx + lastDy * dy <= 0) {
			dx = -dx;
			dy = -dy;
		}

		//		빼도 되는 부분이지만 테스트로 해보았습니다.
		dx = fc * dx + (1 - fc) * lastDx;
		dy = fc * dy + (1 - fc) * lastDy;

		// 벡터의 크기
		float L = sqrt(dx * dx + dy * dy);

		// 크기를 1로 변경
		dx /= L;
		dy /= L;

		// R만큼 이동 (반지름)
		x = x + R * dx;
		y = y + R * dy;

		lastDx = dx;
		lastDy = dy;

		if (x > ref->width - 1 || x < 0 || y > ref->height - 1 || y < 0) break;

		K[idx].P = cvPoint(x, y);
		K[idx].C = strokeColor;
		idx++;
	}
```

```
		for (int i = 0; i < idx - 1; i++) {
			cvLine(dst, K[i].P, K[i + 1].P, strokeColor, R);
		}
}
```

안에서 사용된 getBri함수는 B, G, R 값의 평균을 구하는 함수입니다.

c) 시행 착오

가장 시간을 많이 끌었던 부분은 makeSplineStroke를 구현할 때였습니다. lastDx, lastDy에 x, y값을 대입한다던지, (lastDx, lastDy)와 (dx, dy)의 내적이 음수뿐만 아니라 0일 때에도 처리를 해야한다는 것, Gradient를 구하는 범위를 바꿔보기도 하는 등이 있을 것 같습니다. 하지만 이런 예외처리의 예외라던지, 휴먼에러 같은 부분이 결과물에 지대한 영향을 끼친다는 것을 알게 되었고, 예외처리에 특히 신경을 많이 써야겠다고 생각했습니다.

두 번째 시행착오는 생성한 SplineStroke를 랜덤으로 섞는 과정이었습니다. 사실 포인터를 잘만 활용하면 충분히 할 수 있는 부분이지만, 구조체를 이중 포인터를 활용해서 만드는 것이 생각보다 쉬운 일이 아니었습니다. 더군다나 각각의 SplineStroke 배열들은 길이가 다 제각기였기에 이를 처리하는 것이 어렵다 판단하고, 배열의 시작점이 Circle의 시작점과 동일하다는 것을 이용하여 먼저 Stroke 배열을 섞어준 후 SplineStroke를 생성하고 바로 그려주도록 했습니다.

세 번째는 논문에서 가우시안 필터의 kernel값이 브러쉬 사이즈와 비례하다고 나와있었는데, 이를 제대로 이해하지 못하고 브러쉬 사이즈가 변하는 것에 비례하지 않게 kernel값을 변경시켰다는 것입니다. 이렇게 하면 절대 정교한 이미지를 그릴 수 없고, Painterly rendering이 제대로 이루어지지 않음을 파악하게 되었습니다. 다음은 수정 전의 코드입니다.

```
void paint(IplImage* src, IplImage* dst, int R[], int flag) {
		IplImage* ref = cvCreateImage(cvGetSize(src), 8, 3);
		int kernel = 21;

		for (int i = 0; i < 5; i++) {
			cvSmooth(src, ref, CV_GAUSSIAN, kernel);
			cvShowImage("dst", dst);
			cvWaitKey();
			paintLayer(dst, ref, R[i], flag);
			kernel -= 2;
		}
}
```

그 외엔 논문을 제대로 이해하고 의사코드를 구체화시키는 과제였기에, 코드를 짜는데 어려운

점이 많지는 않았습니다. 하지만 브러쉬의 사이즈, maxStrokeLength, minStrokeLength, 임계값(T) 등 다양한 변수들이 조금씩 바뀔 때마다 나오는 그림이 천차만별로 달라지는 것에서 난항을 겪었습니다.

이상입니다.