멀티미디어프로그래밍

과제 3 보고서

제출일	2024. 04. 12	전공	소프트웨어학과
과목	멀티미디어프로그래밍	학번	21011746
담당교수	박상일 교수님	이름	양현석

1. Fastest Mean Filter

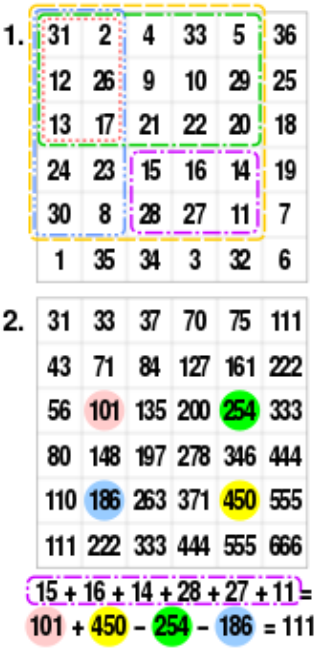
a) 문제 분석

Mean Filter를 수행하는 함수를 제작해야 하는데, cvSmooth() 함수 처럼 빠르게 작동하게 해야 하는 것이 주안점이다. 커널의 크기에 상관없이 작동하려면 Summed-area Table의 원리를 이용해야 한다.

b) 문제 풀이

i) Summed-area Table

먼저, Summed-area Table에 대해 설명하겠습니다. Summed-area Table은 그리드안의 원소들의 평균이나 합을 구할 때, 매우 빠르고 효율적으로 작동합니다.



왼쪽의 그림 1에서 알 수 있듯이, 보라색 사각형 내의 원소들의 합은 (노란색 사각형 원소들의 합) – (하늘색 사각형 원소들의 합) – (초록색 사각형 원소들의 합) + (빨간색 사각형 원소들의 합) 입니다.

(빨간색 사각형 원소들의 합)을 더해주는 이유는 하늘색과 초록색 원소들의 합을 뺄 때, 저 영역의 합이 2번 빠지기 때문입니다. 이 원리를 변형하여, 경우를 나누어서 CvScalar 형 2차원 배열에 값을 저장하겠습니다. (이미지에 넣게되면, 각 BGR값이 255보다 클 때 255로 저장되기 때문임) 그렇다면, CvScalar 형 2차원 배열에는 어떤 값이 들어가게 되는지 설명하겠습니다. 첫 번째, if(x== 0&& y==0) then Source Image(이하, src)의 (x, y)의 CvScalar를 그대로 가져옵니다. 두 번째, if(x==0 && y!=0) then src에서 (x, y)와 (x, y-1)의 CvScalar를 합산하여 저장합니다. 세 번째, if(y==0) then (x, y)와 (x-1, y)의 CvScalar를 합산하여 저장합니다. 마지막으로 위의 경우에 해당하지 않는 경우에는 (x, y) 값 + (x-1, y) 값 + (x, y-1) 값 - (x-1, y-1) 값을 저장합니다.

이렇게 하면, 그림 2와 같이 CvScalar 형 2차원 배열에 값들이 저장되게 됩니다. 이후에 입력된 커널 값을 기준으로 나누어 평균을 내고 출력할 이미지에 저장하면 코드가 완성됩니다.

ii) 속도가 빨라지는 원리

원래의 Mean Filter는 커널 값 K를 입력 받고, x와 y 값이 커질 때마다 그 좌표를 기준으로 닫힌 구간 [-K, K] (2차원) 의 평균을 내어 평균 값을 넣어야 합니다. 하지만 이 방법은 커널 값이 커질수록 시간이 매우 오래 걸린다는 단점이 있습니다. 하지만 위의 Summed-area Table의 원리를 이용하면 $((2 * K)^2$ 번의 연산을 해야 하는 것을 4번의 연산(평균내는 것까지 5번)으로 줄일 수 있게 됩니다. 이렇게 되면 K 값에 상관없이, K가 극단적으로 크다고 하더라도 일정한 실행 시간을 가질

수 있게 됩니다.

iii) 풀이

위의 연산들을 수행하기 위해 CvScalar 형 2차원 배열을 만들어야 합니다. 이는 2차원 배열 동적 할당을 이용합니다. 다음은 그 코드입니다.

```
//      Memory Allocation.
CvScalar ** arr;
arr = (CvScalar**)malloc(sizeof(CvScalar*) * h);

for (int i = 0; i < h; i++) {
            arr[i] = (CvScalar*)malloc(sizeof(CvScalar) * w);
}
```

(h : height of src, w : width of src)

그 후 Summed-area Table을 깔끔하게 수행하기 위해 arr에 i)에서 기술한대로 CvScalar를 저장해줍니다. 다음은, 입력 받은 커널 값을 기준으로 가상의 그리드를 만들어 평균을 내줍니다. 여기에서 Summed-area Table을 활용해줍니다. 하지만 기준 (x ,y)에서 K를 더하거나 뺄 때 이미지의 범위를 벗어나게 될 수 있습니다. 이를 방지하기 위해서 다음과 같이 조건문을 작성해야 합니다. 일단 먼저 저장하는 식부터 작성해보겠습니다.

```
CvScalar f;
int tempY1 = y + K;
int tempY2 = y – K;
int tempX1 = x + K;
int tempX2 = x – K;

for (int k = 0; k < 3; k++) {
            f.val[k] = arr[tempY1][tempX1].val[k] - arr[tempY2][tempX1].val[k] - arr[tempY1][tempX2].val[k] +
            arr[tempY2][tempX2].val[k];
             f.val[k] /= ((tempY1 - tempY2 + 1) * (tempX1 - tempX2 + 1));
}

cvSet2D(dst, y, x, f);
```

이제 조건문을 작성해야 합니다. 크게 4가지 경우로 나눌 수 있는데, (tempY1 >= h), (tempY2 < 0), (tempX1 >= w), (tempX2 < 0)의 경우입니다.

```
if (tempY1 >= h) {
            tempY1 = h - 1;
}
if (tempY2 < 0) {
            tempY2 = 0;
```

```
}
if (tempX1 >= w) {
		tempX1 = w - 1;
}
if (tempX2 < 0) {
		tempX2 = 0;
}
```

범위를 벗어나게 되면, 범위의 마지막 값을 각각 대입해주었고 또한 그 범위에 맞추어 그리드의 범위를 맞추기 위해 같은 변수를 갖는 다른 temp~ 값도 조정해줍니다. 이렇게 한다면, Summed-area Table을 사용한 Mean Filter가 완성됩니다.

c) 시행 착오

시행 착오 중 가장 난해했던 부분은 코드를 짤 때 가장 중요한 부분은 조건문을 거는 것이었습니다. 조건문을 걸지 않으면 애초에 에러가 발생하는 것은 더불어, for 구문의 범위를 변경하더라도 사진이 잘리게 되었기 때문입니다. 완성된 코드에서는 굉장히 간단한 알고리즘을 잘 짰었지만, 처음 arr에 저장하는 것부터 엉망이었습니다.

```
if (x == 0) {		//		y == 0 일때에도 포함
		if (y == 0) {		// x==0 && y==0
			for (int k = 0; k < 3; k++) {
				arr[y][x].val[k] = f.val[k];
			}
		}

		else {
			for (int k = 0; k < 3; k++) {
				arr[y][x].val[k] = f.val[k] + arr[y - 1][x].val[k];
			}
		}

}

else if (y == 0) {
		//		이미 x, y == 0 인 경우를 해결함.
		for (int k = 0; k < 3; k++) {
			arr[y][x].val[k] = f.val[k] + arr[y][x - 1].val[k];
		}
}

else {				//		y>0 && x>0
```

```
		for (int k = 0; k < 3; k++) {
			arr[y][x].val[k] = f.val[k] + arr[y - 1][x].val[k] + arr[y][x - 1].val[k] - arr[y - 1][x - 1].val[k];
		}
}
```

한눈에 보기에도 알겠지만, 굉장히 난잡하고 더군다나 작동도 되질 않습니다. 완성된 코드와는 다르게 각각의 경우에서 다르게 대입을 합니다. 다음은 조건문 2에 해당하는 부분입니다.

```
CvScalar f;
if (x + K < w && y + K < h) {
		if (x > n && y > n) {
			for (int k = 0; k < 3; k++) {
				f.val[k] = arr[y + K][x + K].val[k] - arr[y - n + K][x + K].val[k] - arr[y + K][x - n + K].val[k]
+ arr[y - n + K][x - n + K].val[k];
		}
}
else if (x > n) {
		for (int k = 0; k < 3; k++) {
			f.val[k] = arr[y + K][x + K].val[k] - arr[y + K][x + K - n].val[k];
		}
}
else if (y > n) {
		for (int k = 0; k < 3; k++) {
			f.val[k] = arr[y + K][x + K].val[k] - arr[y + K - n][x + K].val[k];
		}
}
else {
		for (int k = 0; k < 3; k++) {
			f.val[k] = arr[y + K][x + K].val[k];
		}
}

for (int k = 0; k < 3; k++) {
		f.val[k] = f.val[k] / (float)(n * n);
}
for (int i = -K; i <= K; i++) {
		for (int j = -K; j <= K; j++) {
			cvSet2D(dst, y, x, f);
		}

}
```

이 코드 역시 가독성이 굉장히 떨어질 뿐더러, 제대로 작동하지 않습니다. 아마 제대로 작동하지 않는 이유는 대입하는 경우에서 Summed-area Table의 원리를 제대로 적용하지 못하지 않았을까

추측하고 있습니다. 이 조건문을 작성하면서 느낀점은 무조건 모든 조건을 다 걸어서 하려고 하면 안되고, 체계적으로 나누어서 혹은 변수를 만들어서 최대한 간결하게 작성해야 함을 절실히 느꼈습니다. 평소 대충 쓰던 조건문을 깊게 고민해볼 수 있는 좋은 기회였습니다.

이상입니다.