

1. 에지 검출을 하기 전에 평균값 필터링 또는 가우시안 필터링을 한 경우와 하지 않은 경우를 비교하고, 차이점을 기술하라.

1. 평균값 필터링 또는 가우시안 필터링을 적용한 경우:

- **노이즈 감소:** 평균값 필터링과 가우시안 필터링은 노이즈를 줄이는 데 효과적입니다. 따라서 필터링을 적용한 후 에지 검출을 수행하면, 노이즈 때문에 생기는 가짜 에지(false edges)가 적습니다.
- **에지 부드러움:** 필터링을 통해 이미지가 부드러워지므로, 에지도 약간 부드러워질 수 있습니다. 이는 디테일한 에지 정보의 일부 손실을 의미할 수 있습니다.
- **계산 시간 증가:** 필터링 단계가 추가되므로 전체 처리 시간이 약간 더 걸릴 수 있습니다.

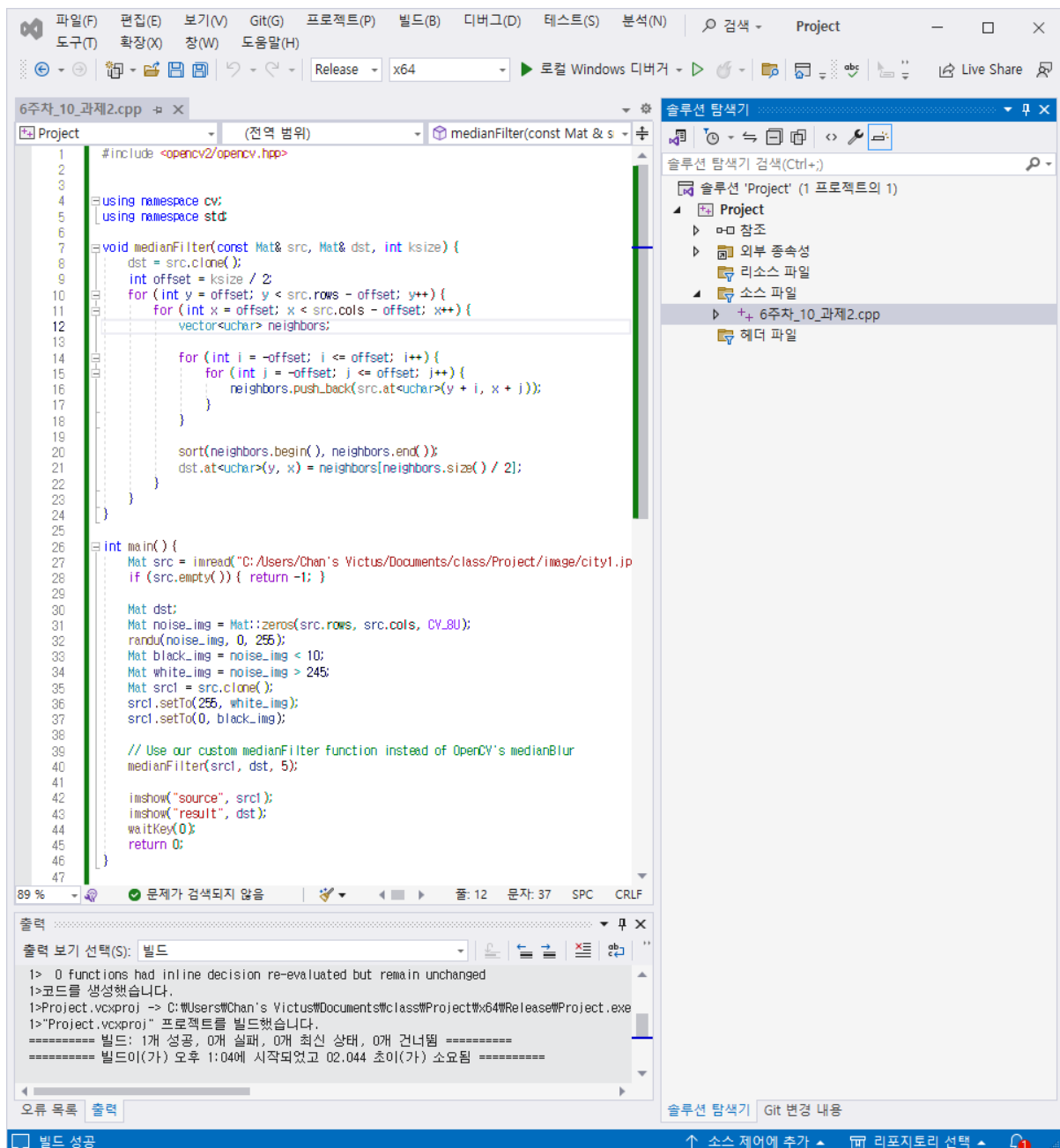
2. 평균값 필터링 또는 가우시안 필터링을 적용하지 않은 경우:

- **노이즈 민감성:** 이미지의 노이즈에 민감하게 반응하여, 노이즈로 인해 잘못된 에지가 많이 검출될 수 있습니다.
- **에지 디테일 유지:** 필터링을 하지 않아서, 원본 이미지의 세부 에지 정보가 유지됩니다. 그러나 이는 노이즈 때문에 유용한 정보와 혼동될 수 있습니다.
- **계산 시간 단축:** 필터링 단계가 없으므로 처리 시간이 빠를 수 있습니다.

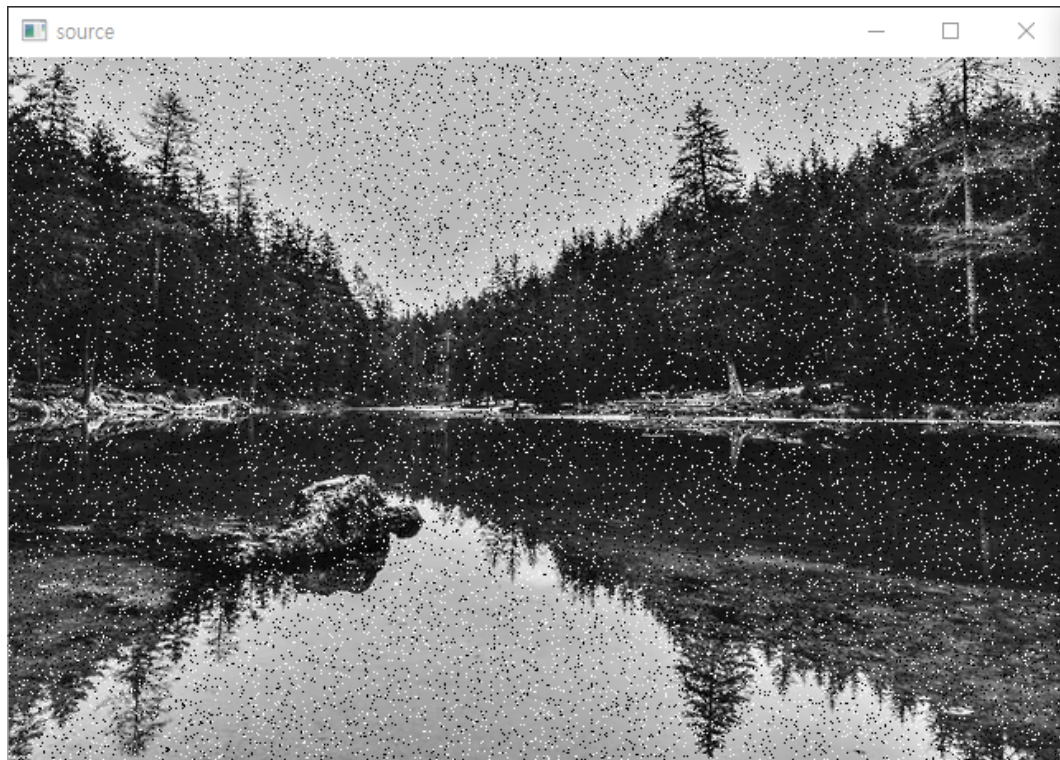
이미지의 특성과 목적에 따라 전처리 방식을 결정해야 합니다. 노이즈가 많은 이미지나 높은 품질의 에지 검출이 필요한 경우에는 가우시안 필터링 또는 평균값 필터링을 적용하는 것이 좋습니다. 하지만 처리 시간이 중요하거나 이미지의 노이즈가 적은 경우에는 필터링 없이 에지 검출을 수행하는 것도 고려할 수 있습니다.

2. 중간값 필터링용 OpenCV 함수 medianBlur() 와 동일한 기능의 medianFilter() 함수를 직접 구현하고, PPT 21쪽의 실습 프로그램에 삽입하여 medianBlur() 함수와 비교하라.

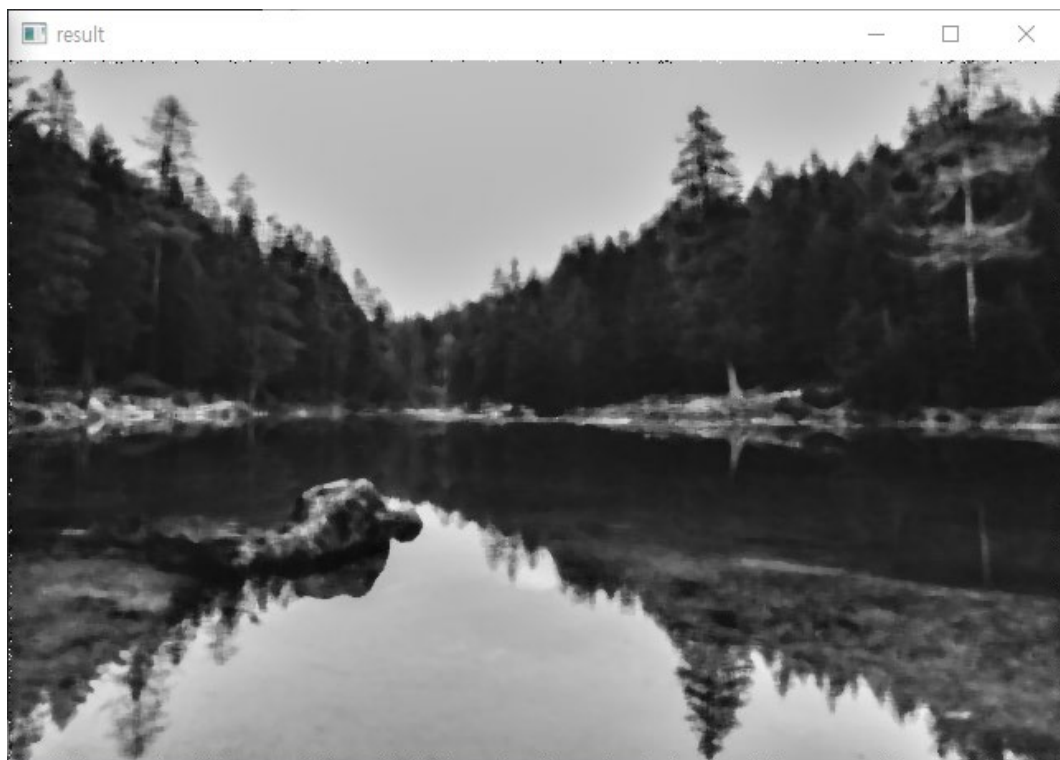
void medianFilter(Mat input, Mat& output, int ksize)



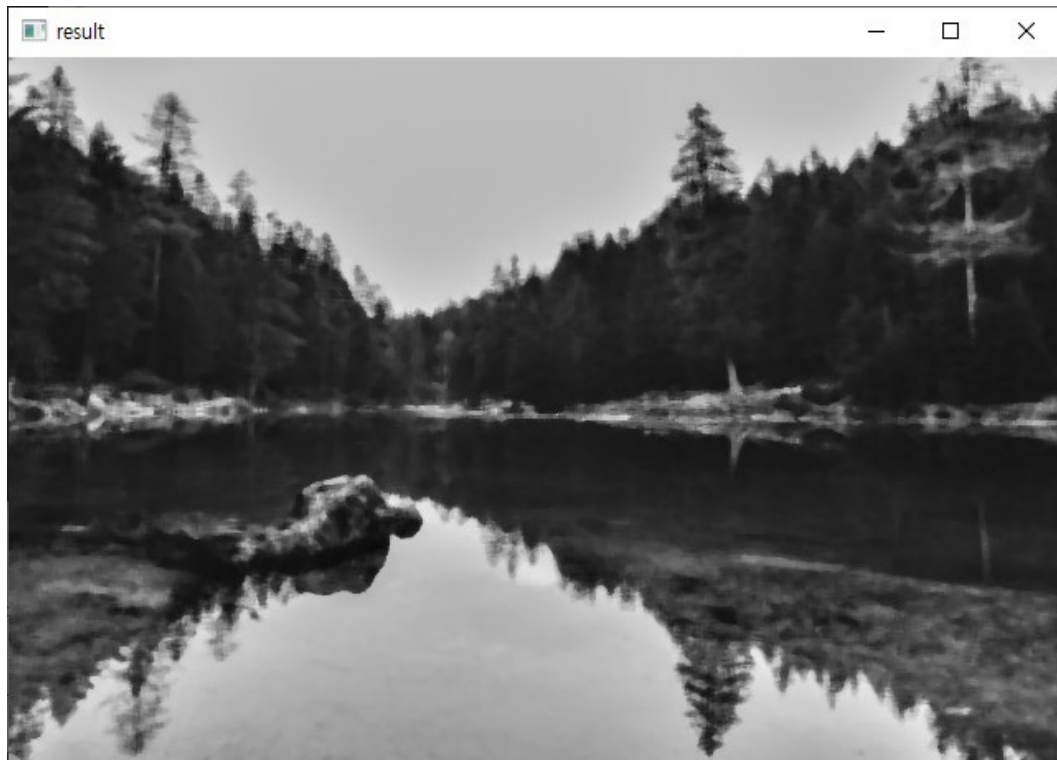
1.원본



2.medianFilter()



3. medianBlur()



직접 구현한 **medianFilter()** 함수는 중간값 필터링을 기본적으로 수행합니다. 이 함수는 이중 for 루프와 리스트 정렬이 포함되어 있기 때문에 실행 속도 측면에서는 상대적으로 느릴 수 있습니다. 특히 큰 이미지나 큰 커널 크기를 다룰 때 더욱 느려질 것입니다.

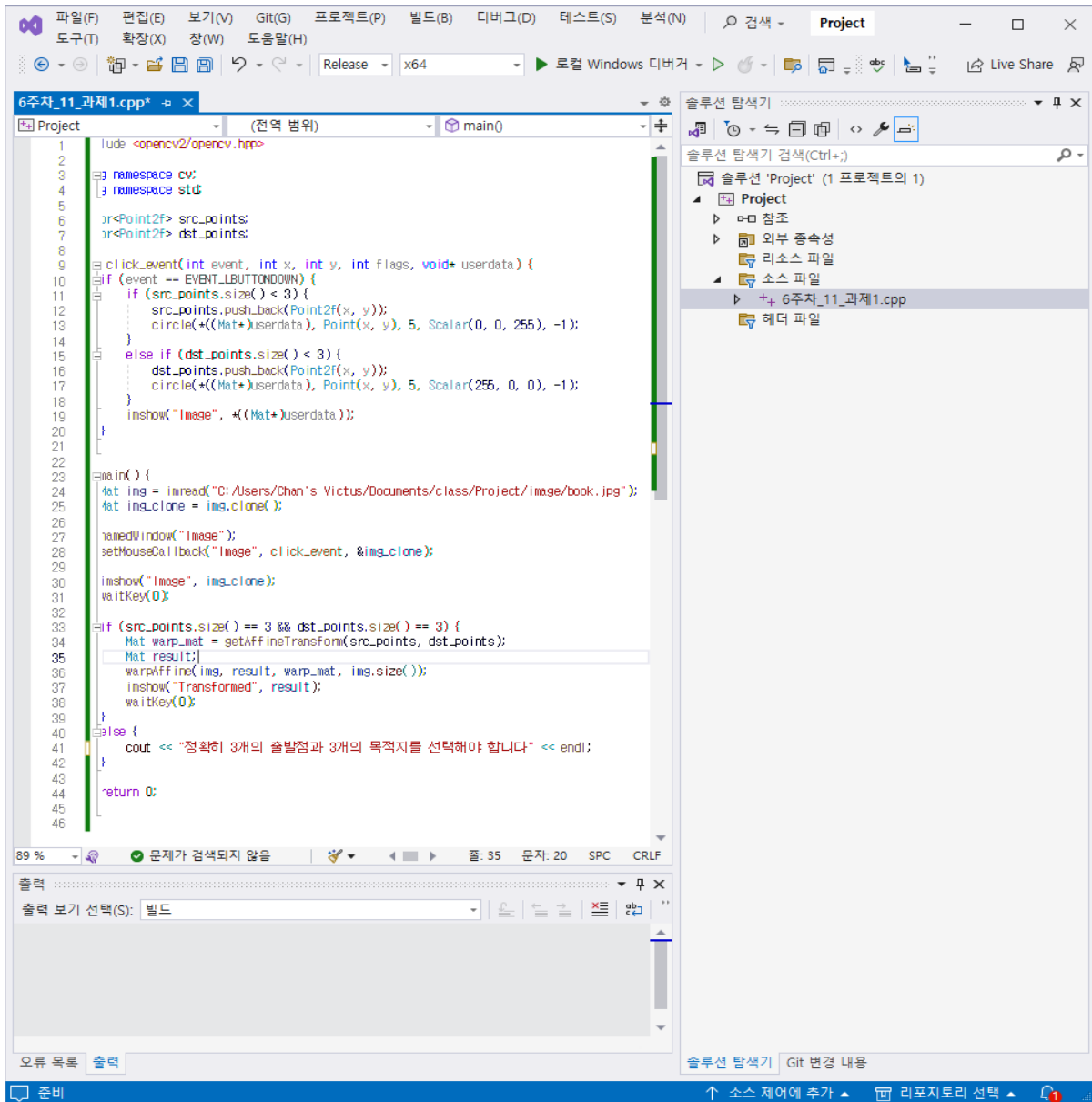
반면, OpenCV의 **medianBlur()** 함수는 이미 최적화된 방식으로 구현되어 있습니다. 따라서 대부분의 상황에서는 **medianFilter()**보다 더 빠른 성능을 보여줄 것입니다.

또한, 두 함수는 중간값 필터링의 원리에 기반하므로 주어진 입력에 대해 유사한 결과를 생성해야 합니다. 그러나 구현 방식의 차이로 인해 아주 미세한 결과의 차이가 발생할 수 있습니다.

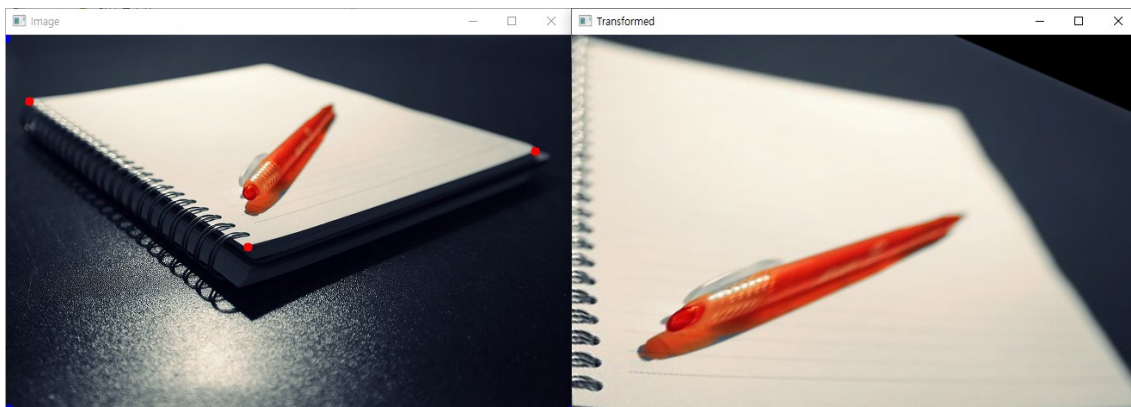
하지만, **medianFilter()**의 가장 큰 장점은 확장성과 수정 용이성입니다. 우리가 이 함수를 직접 작성하였기 때문에 필요에 따라 알고리즘을 수정하거나 추가적인 기능을 구현하는 것이 훨씬 쉽습니다. 반대로, **medianBlur()** 함수는 OpenCV 라이브러리의 일부로, 직접 수정하기는 어렵습니다.

결과적으로, 일반적인 상황에서 빠른 성능을 원한다면 **medianBlur()**를 사용하는 것이 좋습니다. 그러나 특정한 요구 사항이 있거나 알고리즘의 동작 방식을 조절하고 싶다면 **medianFilter()**와 같이 사용자 정의 함수를 사용하는 것이 유용하다고 볼 수 있습니다.

1. 사용자로부터 3개의 점을 받아서 어파인 변환하는 프로그램을 작성하라. 마우스를 이용하여 점들의 초기 위치와 변환 후의 위치를 입력할 수 있도록 한다.



```
1 | #include <opencv2/opencv.hpp>
2 |
3 | namespace cv;
4 | namespace std;
5 |
6 | struct Point2f {
7 |     float x;
8 |     float y;
9 | };
10 |
11 | click_event(int event, int x, int y, int flags, void* userdata) {
12 |     if (event == EVENT_LBUTTONDOWN) {
13 |         if (src_points.size() < 3) {
14 |             src_points.push_back(Point2f(x, y));
15 |             circle(*(Mat*)userdata, Point(x, y), 5, Scalar(0, 0, 255), -1);
16 |         }
17 |         else if (dst_points.size() < 3) {
18 |             dst_points.push_back(Point2f(x, y));
19 |             circle(*(Mat*)userdata, Point(x, y), 5, Scalar(255, 0, 0), -1);
20 |         }
21 |         imshow("Image", *(Mat*)userdata);
22 |     }
23 | }
24 |
25 | main() {
26 |     Mat img = imread("C:/Users/Chan's Victus/Documents/class/Project/image/book.jpg");
27 |     Mat img_clone = img.clone();
28 |
29 |     namedWindow("Image");
30 |     setMouseCallback("Image", click_event, &img_clone);
31 |
32 |     imshow("Image", img_clone);
33 |     waitKey(0);
34 |
35 |     if (src_points.size() == 3 && dst_points.size() == 3) {
36 |         Mat warp_mat = getAffineTransform(src_points, dst_points);
37 |         Mat result;
38 |         warpAffine(img, result, warp_mat, img.size());
39 |         imshow("Transformed", result);
40 |         waitKey(0);
41 |     }
42 |     else {
43 |         cout << "정확히 3개의 출발점과 3개의 목적지를 선택해야 합니다" << endl;
44 |     }
45 |
46 |     return 0;
47 | }
```



2. 사용자로부터 4개의 점을 받아서 원근 변환하는 프로그램을 작성하라. 마우스를 이용하여 점들의 초기위치와 변환 후의 위치를 입력할 수 있도록 한다.

