INHA UNIVERSITY

ICE4027 디지털영상처리설계

실습 12주차

보고서 작성 서약서

1. 나는 타학생의 보고서를 베끼거나 여러 보고서의 내용을 짜집기하지 않겠습니다.
2. 나는 보고서의 주요 내용을 인터넷사이트 등을 통해 얻지 않겠습니다.
3. 나는 보고서의 내용을 조작하지 않겠습니다.
4. 나는 보고서 작성에 참고한 문헌의 출처를 밝히겠습니다.
5. 나는 나의 보고서를 제출 전에 타학생에게 보여주지 않겠습니다.

나는 보고서 작성시 윤리에 어긋난 행동을 하지 않고 정보통신공학인으로서 나의 명예를 지킬 것을 맹세합니다.

2023년 05월 23일

학부 정보통신공학
학년 3
성명 김동한
학번 12191727

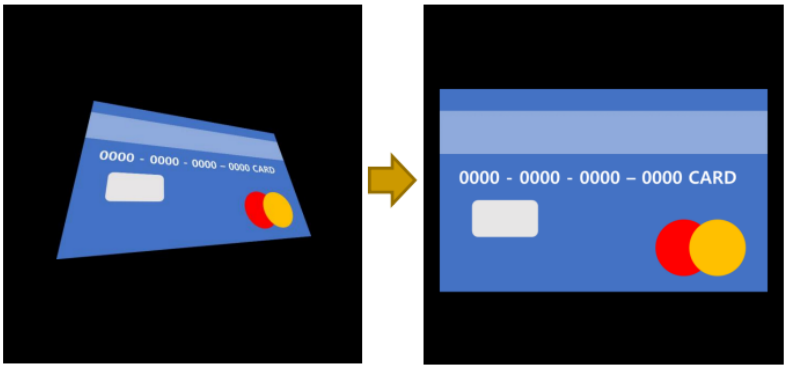
1. 개요

Homework

- 과제 1
  - getRotationMatrix()과 동일한 getMyRotationMatrix()함수를 직접 구현하고 두 결과가 동일한지 검증하라
  - Scale 변화는 구현하지 않아도 됨
  - 45도 변화 결과가 동일한지 비교하면 됨

Homework

- 과제 2
  - card_per.png 영상을 getPerspectiveTransform() 함수를 이용해 카드의 면이 시선 정면을 향하도록 정렬시켜라.
  - 입력 매개변수를 구하기 위해 네 꼭지점을 직접 찾으면 부분점수
  - 지난 실습 및 과제를 참고해 네 꼭지점을 자동으로 탐색하도록 만들면 만점
  - 둘 중 어떠한 방법으로 구현했는지 반드시 잘 보이게 명시할 것!



2. 상세 설계 내용

#1

```
Mat getMyRotationMatrix(Point center, double angle) {
    double a = cos(angle * CV_PI / 180);
    double b = sin(angle * CV_PI / 180);
    Mat matrix = (
        Mat_<double>(2, 3) <<
        a, b, (1 - a) * center.x - b * center.y,
        -b, a, b * center.x + (1 - a) * center.y
        );

    return matrix;
}
```

그림 1 Rotate 행렬 함수 구현

$$\begin{bmatrix} \alpha & \beta & (1-\alpha)\cdot \mathbf{center.x} - \beta \cdot \mathbf{center.y} \\ -\beta & \alpha & \beta \cdot \mathbf{center.x} + (1-\alpha)\cdot \mathbf{center.y} \end{bmatrix}$$

$$\alpha = \mathbf{scale} \cdot \cos \mathbf{angle},$$
$$\beta = \mathbf{scale} \cdot \sin \mathbf{angle}$$

중심점을 기준으로 처리

그림 2 강의노트6pg rotation행렬

강의노트를 참고하여 opencv의 getRotation 함수와 같은 기능을 하는 getMyRoatationMatrix를 구현했다. α=a, β=b 이고, 강의노트의 공식을 적용해 cos함수와 sin함수로 값을 계산하였다. 이를 cvRotation함수에 넣어 rotation을 구현했다. scale변화는 구현할 필요없어 구현하지않았다.

```
void cvRotation() {
    Mat src = imread("C:\\images\\Lenna.png", 1);
    Mat dst,dst2, matrix;

    Point center = Point(src.cols / 2, src.rows / 2);

    //CV
    matrix = getRotationMatrix2D(center, 45.0, 1.0);
    warpAffine(src, dst, matrix, src.size());
    //Rotation
    matrix = getMyRotationMatrix(center, 45.0);
    warpAffine(src, dst2, matrix, src.size());



    imshow("nonrot.jpg", src);
    imshow("rot", dst);
    imshow("my_rot", dst2);
    waitKey(0);

    destroyAllWindows();

}
```

그림 3 cvRotation함수

getMyRotationMatrix함수의 인자로 center(Point 클래스), 45.0(angle)을 입력하여 45°회전하였다.

warpAffine으로 dst2에 회전한 이미지를 저장해 출력했다.

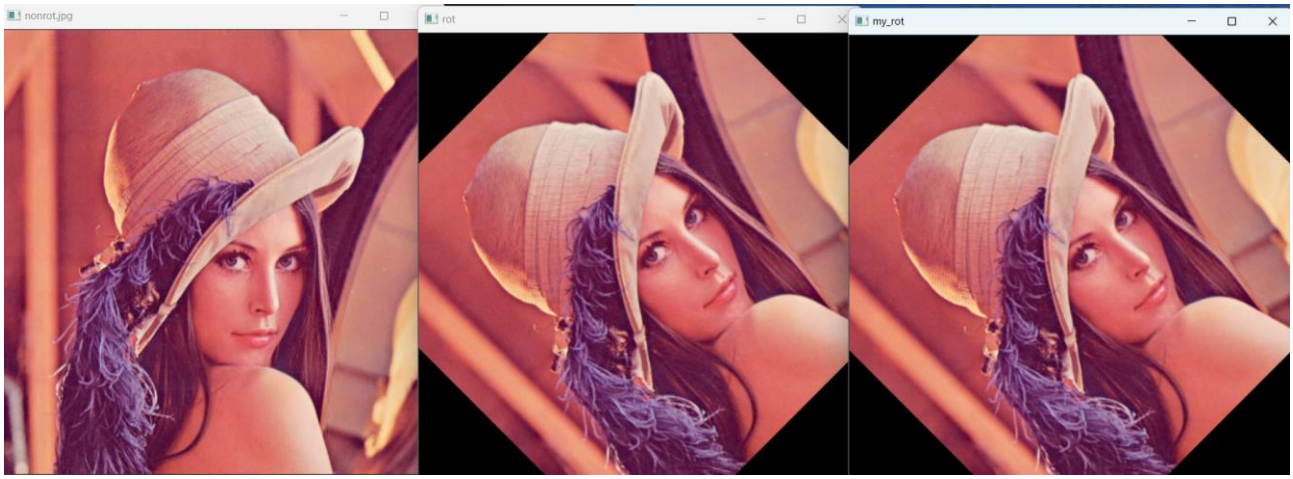


그림 4 실행결과

실행결과 45°회전이 된 것을 확인할 수 있다.

#2

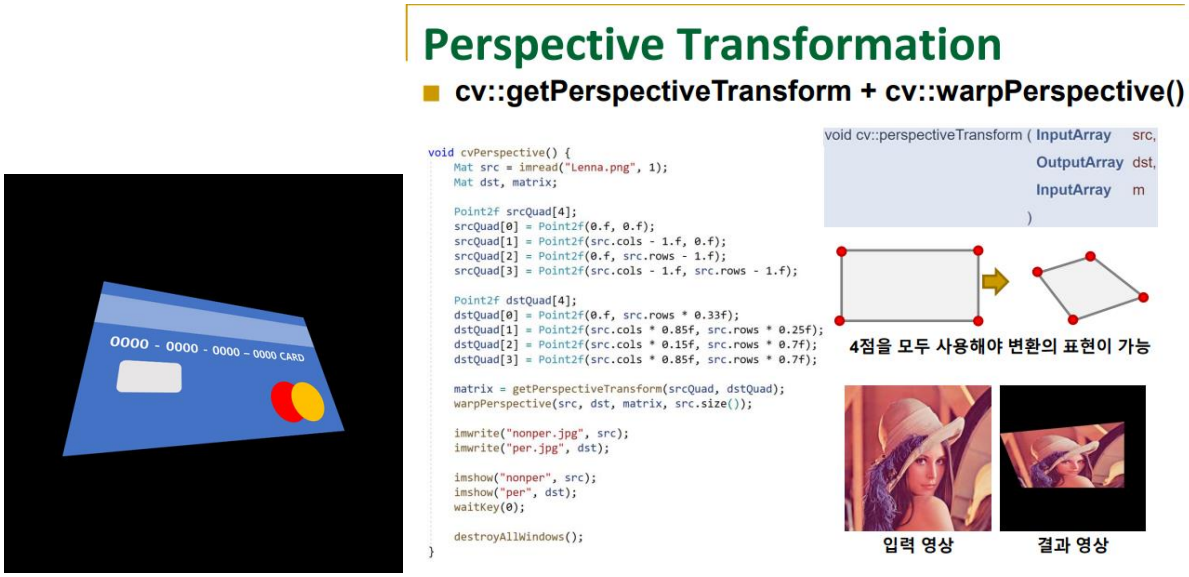


그림 5 card_per.png , getPerspectiveTransform함수

getPerspectiveTransform 함수를 구현하기 위해선, 4개의 점의 위치를 정확하게 알아야한다. 여기서, 11주차에 활용한 harris corner detection함수를 적용하기로했다. 하지만, 11주차에서의 harris corner detection 함수는 이미지를 반환하는 함수였기 때문에, 각 꼭짓점의 좌표 즉 point점들을 반환하는 함수로 변형하였다. 여기서 문제는 카드 내에 적혀있는 글자들도 corner로 detect한다는 점이였다.

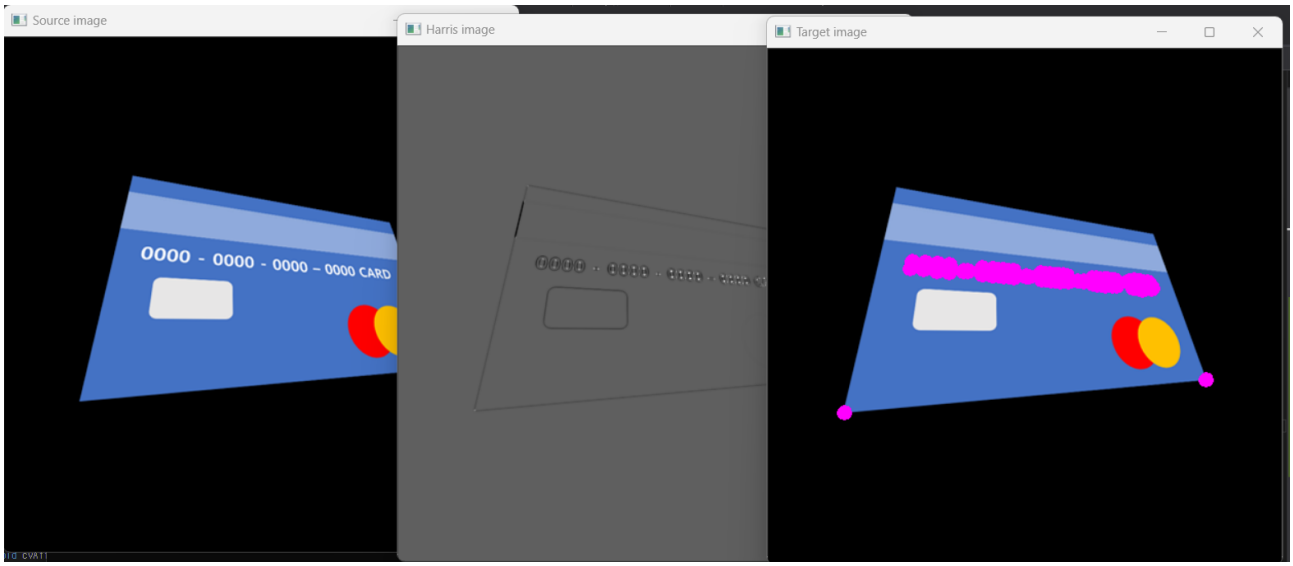
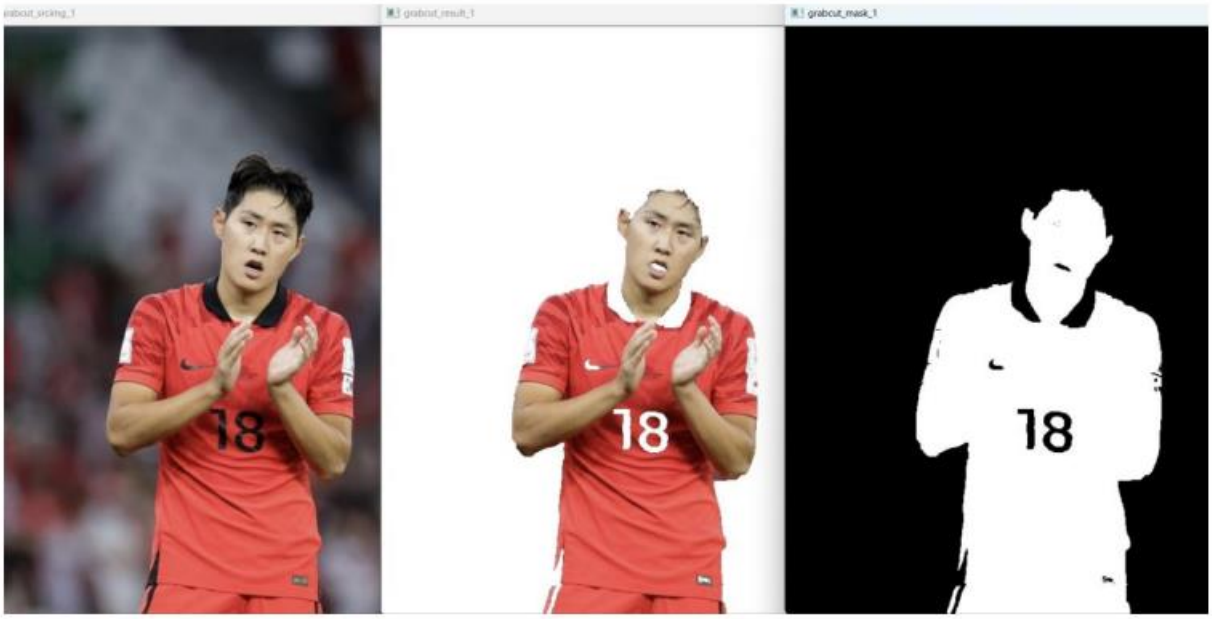


그림 6 corner detect failure

이를 해결할 방법으로 grabcut을 사용하였다.

}



위의 예시는 grabcut이 잘 이루어지지 않은 예시이다. 이강인 선수의 머리색이 배경과 유사한 픽셀값을 가지기 때문에 같이 처리된 것이다. grabcut 알고리즘은 이미지의 전경과 배경을 구분하는 윤곽선이 명확하지않으면 불분명하게 나오게되는것이다. 아래의 두 영상은 전경과 배경의 구분이 확실한 case이기

그림 7 이전 grabcut과제 참조사진

위의 사진은 배경과 유사한 픽셀값에 의해 잘 구분되지않았지만, 주어진 card_per.png파일은 배경과 카드의 픽셀값이 확실하게 구분되기 때문에, 이미지의 전경과 배경을 구분하는 윤곽선만 본뜰 수 있었다.

```
Rect rect = Rect(Point(55, 105), Point(450, 370));
grabCut(src, grabcut_result,
    rect,
    bg_model,
    fg_model,
    5,
    GC_INIT_WITH_RECT);

compare(grabcut_result, GC_PR_FGD, grabcut_result, CMP_EQ);

Mat mask(src.size(), CV_8UC3, cv::Scalar(255, 255, 255));
src.copyTo(mask, grabcut_result);
```

그림 8 grabcut code

grabcut으로 카드의 모양만 본뜬뒤, 그이미지를 hariscorner detect하였다. 어차피 꼭짓점의 좌표만 알면 되는 상황이었기 때문이다.

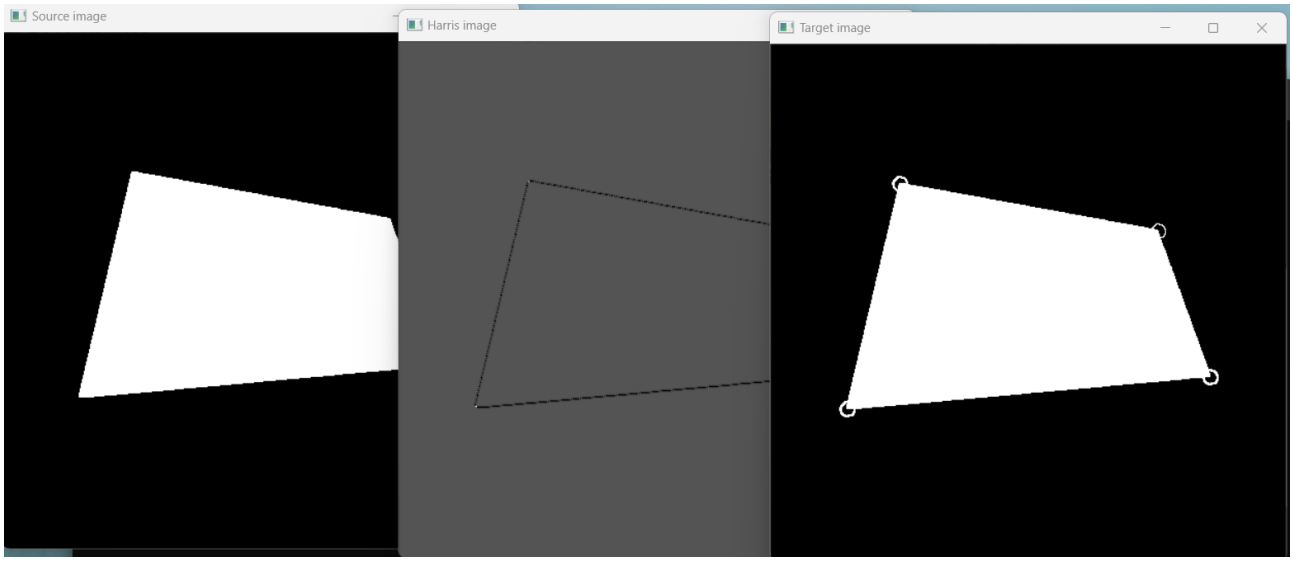


그림 9 corner detect success

이제 corner로 detect한 꼭짓점들의 좌표값을 getperspectiveTransform함수에 전달해야하기 때문에, 함수의 반환값을 바꿔주었다.

```
vector <Point2f> cvHarrisCorner(Mat img) {
    Mat gray = img.clone();

    // < Do Harris corner detection >
    Mat harr;
    cornerHarris(gray, harr, 2, 3, 0.05, BORDER_DEFAULT);
    normalize(harr, harr, 0, 255, NORM_MINMAX, CV_32FC1, Mat());

    // < Get abs for Harris visualization >
    Mat harr_abs;
    convertScaleAbs(harr, harr_abs);

    // < Print corners >
    int thresh = 100;
    int min_pixel = 0;

    vector <Point2f>xy;
    Mat result = img.clone();

    for (int y = 0; y < harr.rows; y++) {
        for (int x = 0; x < harr.rows; x++)
        {
            if ((int)harr.at<float>(y, x) > thresh) {

                circle(result, Point(x, y), 7, Scalar(255, 0, 255), 0, 4, 0);
                if (xy.size() == 0) {
                    xy.push_back(Point(x, y));
                }
                else
                {
                    if (x != xy[min_pixel].x && y != xy[min_pixel].y) {
                        if ((x<xy[min_pixel].x - 5 || x > xy[min_pixel].x + 5) &&
                            (x<xy[min_pixel].y - 5 || x > xy[min_pixel].y + 5))
                        {
                            xy.push_back(Point(x, y));
                            min_pixel++;
                        }
                    }
                }

            }
        }
    }
```

```
    imshow("Target image", result);
    waitKey(0);
    cout << xy[0]<<endl;
    cout << xy[1]<<endl;
    cout << xy[2]<<endl;
    cout << xy[3]<<endl;
    return xy;
}
```

그림 10 cvHarriscorner함수

vector형으로 반환값을 설정한 뒤, threshold값보다 큰값을 출력하고 circle만 그렸었는데 해당 Point를 xy vector에 삽입해주었다. 오차는 그후 vector에 넣은 좌표의값과 Point좌표의 x,y값이 다르다면, 오차범

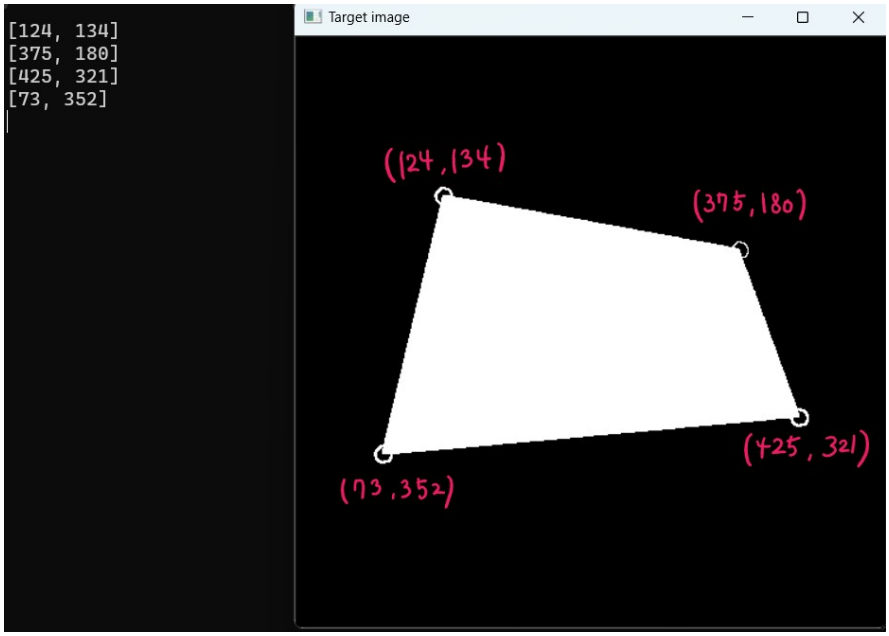
위 +-5 안에 없으면 pushback한뒤, vector에 넣어둔 좌표값을 1더 키워 비교했다. 이로써 xy[0]부터 xy[3]까지 꼭짓점을 저장할 수 있었다.

xy[0]=124,134

xy[1]=375,180

xy[2]=425,321

xy[3]=73,352




```
void cvPerspective(Mat src,vector <Point2f> xy) {
    Mat dst, matrix;

    Point2f srcQuad[4];
    srcQuad[0] = xy[0];
    srcQuad[1] = xy[1];
    srcQuad[2] = xy[3];
    srcQuad[3] = xy[2];

    Point2f dstQuad[4];
    dstQuad[0] = Point2f(xy[3].x, xy[0].y);
    dstQuad[1] = Point2f(xy[2].x, xy[0].y);
    dstQuad[2] = Point2f(xy[3].x, xy[3].y);
    dstQuad[3] = Point2f(xy[2].x, xy[3].y);

    matrix = getPerspectiveTransform(srcQuad, dstQuad);
    warpPerspective(src, dst, matrix, src.size());

    imshow("nonper", src);
    imshow("per", dst);
    waitKey(0);
    destroyAllWindows();

}
```

끝점으로 dstQuad를 설정했다. 예를들어 첫번째 dstQuad[0]은 x좌표가 73, y좌표가 134로 들어간것이다. 이는 가로방향으로 좌측 끝쪽 값과 세로방향으로 제일 위쪽에있는값으로 변형하는 것이다. 같은느낌으로, dstQuad[1]도 우측의 끝쪽 x값과 상하로 따졌을 때 제일 끝에 있는 값을 넣어줌으로써 perspective transform을 구현했다.

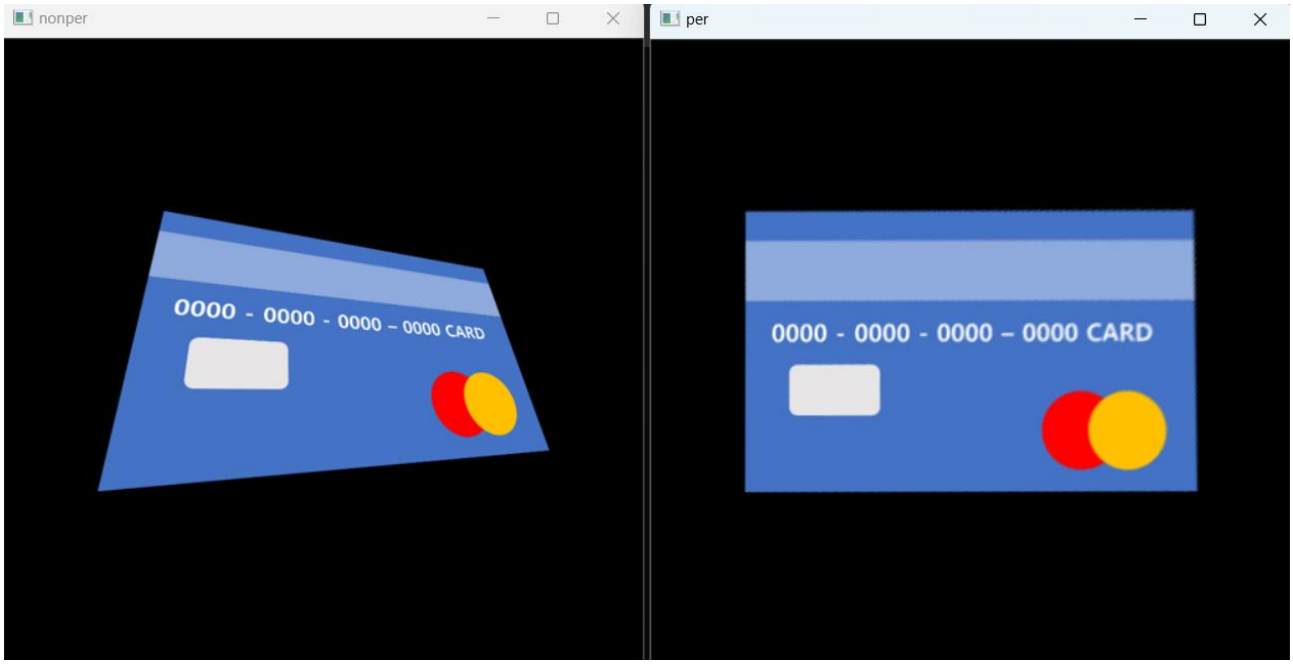


그림 11 실행결과