**\*\*\*\*rotation.cpp\*\*\*\***

**Code Explanation**

Purpose of the code: perform image rotation using inverse warping and interpolation(both nearest neighbor and bilinear interpolation)

1. template <typename T> Mat myrotate(const Mat input, float angle, const char\* opt)

-input: input matrix to be rotated

-angle: rotation angle

-opt: interpolation method (if nearest-neighbor method then "nearest", else if bilinear method then "bilinear")

1) estimate size of a rectangle surrounding a rotated image(sq\_row, sq\_col)

float sq\_row = ceil(row \* sin(radian) + col \* cos(radian));

float sq\_col = ceil(col \* sin(radian) + row \* cos(radian));

2) output image를 sq\_row행 sq\_col열 input.type() 타입의 영행렬로 초기화한다

Mat output = Mat::zeros(sq\_row, sq\_col, input.type());

3) inverse warping with nearest neighbor or bilinear interpolation

//nearset integer grid의 좌표를 구한다

float y1 = floor(y);

float y2 = ceil(y);

float x1 = floor(x);

float x2 = ceil(x);

float mu = y - y1;

float lambda = x - x1;

//nearest neighbor interpolation -> use function R(u)

if (!strcmp(opt, "nearest")) {

output.at<T>(i, j) = R(lambda \* (-1)) \* (R(mu \* (-1)) \* input.at<T>(y1, x1) + R(1 - mu) \* input.at<T>(y2, x1))+ R(1 - lambda) \* (R(mu \* (-1)) \* input.at<T>(y1, x2) + R(1 - mu) \* input.at<T>(y2, x2));

}

//bilinear interpolation

else if (!strcmp(opt, "bilinear")) {

output.at<T>(i, j) = lambda \* (mu \* input.at<T>(y2, x2) + (1 - mu) \* input.at<T>(y1, x2)) +

(1 - lambda) \* (mu \* input.at<T>(y2, x1) + (1 - mu) \* input.at<T>(y1, x1));

}

2. float R(float u)

:function for nearest-neighbor interpolation

-> 1 if -0.5<u<=0.5 otherwise 0

3. int main()

-input: read "lena.jpg" using imread() function

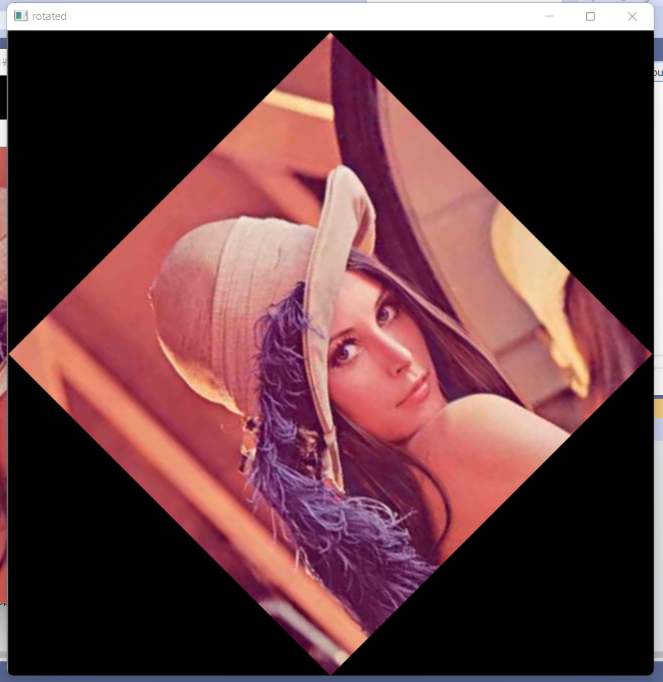
-rotated: rotated image matrix

**Results**

**텍스트, 의류, 머리카락이(가) 표시된 사진

자동 생성된 설명** original image

**** 45 degree rotation(nearest neighbor)

**** 45 degree rotation(bilinear)

* Image with bilinear method has better quality than nearest neighbor method

**\*\*stitching.cpp\*\***

**Code Explanation**

Purpose of the code: perform image stitching with two images "Img01.jpg" and "Img02.jpg" using affine transformation

1. template <typename T> Mat cal\_affine(int ptl\_x[], int ptl\_y[], int ptr\_x[], int ptr\_y[], int number\_of\_points)

:function to obtain affine transform

-ptl\_x[], ptl\_y[], ptr\_x[], ptr\_y[]: corresponding pixels which will be used to calculate affine matrix

-number\_of\_points: number of points

->return x= (M^T \* M)^(−1) \* M^T \* b , so we can find a,b,c,d,e,f of affine matrix A

2. void blend\_stitching(const Mat I1, const Mat I2, Mat& I\_f, int bound\_l, int bound\_u, float alpha)

:function to merge two images I1, I2

-I1, I2: matrix of image to be blended

-I\_f: merged image

-bound\_l, bound\_u: boundary for merged image(I\_f)

-alpha: when blending I1 and I2 use alpha\*I1 + (1-alpha)\*I2

I-f={ alpha\*I1 + (1-alpha)\*I2 (if both I1 and I2 are valid)

I1 (if only I1 is valid)

I2(if only I2 is valid)

0 (otherwise) }

if (cond\_I2) //if both l1, l2 are valid

I\_f.at<Vec3f>(i - bound\_u, j - bound\_l) = alpha \* I1.at<Vec3f>(i, j) + (1 - alpha) \* I\_f.at<Vec3f>(i - bound\_u, j - bound\_l);

else //if only l1 is valid

I\_f.at<Vec3f>(i - bound\_u, j - bound\_l) = I1.at<Vec3f>(i, j);

3. int main()

1) read each image

I1 = imread("C:\\Users\\MIN\\Desktop\\Lab02\\Img01.jpg", IMREAD\_COLOR);

I2 = imread("C:\\Users\\MIN\\Desktop\\Lab02\\Img02.jpg", IMREAD\_COLOR);

2) convert l1(l2) into : l1\*alpha + beta (l2\*alpha + beta )

(alpha=1.0/255 , beta= 0.0(default parameter) )

I1.convertTo(I1, CV\_32FC3, 1.0 / 255);

I2.convertTo(I2, CV\_32FC3, 1.0 / 255);

3) I1\_row, I1\_col, I2\_row, I2\_col: height(row), width(col) of each image

4) compute corners (p1, p2, p3, p4) using A21

-> A21.at<float>(0), ..., A21.at<float>(5)는 affine matrix A21의 a,b,c,d,e,f

Point2f p1(A21.at<float>(0) \* 0 + A21.at<float>(1) \* 0 + A21.at<float>(2), A21.at<float>(3) \* 0 + A21.at<float>(4) \* 0 + A21.at<float>(5));

Point2f p2(A21.at<float>(0) \* 0 + A21.at<float>(1) \* I2\_row + A21.at<float>(2), A21.at<float>(3) \* 0 + A21.at<float>(4) \* I2\_row + A21.at<float>(5));

Point2f p3(A21.at<float>(0) \* I2\_col + A21.at<float>(1) \* I2\_row + A21.at<float>(2), A21.at<float>(3) \* I2\_col + A21.at<float>(4) \* I2\_row + A21.at<float>(5));

Point2f p4(A21.at<float>(0) \* I2\_col + A21.at<float>(1) \* 0 + A21.at<float>(2), A21.at<float>(3) \* I2\_col + A21.at<float>(4) \* 0 + A21.at<float>(5));

5) bound\_u, bound\_b, bound\_l, bound\_r: compute boundary for merged image

6) initialize merged image

Mat I\_f(bound\_b - bound\_u + 1, bound\_r - bound\_l + 1, CV\_32FC3, Scalar(0));

7) inverse warping with bilinear interplolation using A12

for (int i = bound\_u; i <= bound\_b; i++) {

for (int j = bound\_l; j <= bound\_r; j++) {

float x = A12.at<float>(0) \* j + A12.at<float>(1) \* i + A12.at<float>(2) -bound\_l;

float y = A12.at<float>(3) \* j + A12.at<float>(4) \* i + A12.at<float>(5) - bound\_u;

//nearest integer grid

float y1 = floor(y);

float y2 = ceil(y);

float x1 = floor(x);

float x2 = ceil(x);

float mu = y - y1;

float lambda = x - x1;

if (x1 >= 0 && x2 < I2\_col && y1 >= 0 && y2 < I2\_row)

I\_f.at<Vec3f>(i - bound\_u, j - bound\_l) = lambda \* (mu \* I2.at<Vec3f>(y2, x2) + (1 - mu) \* I2.at<Vec3f>(y1, x2)) +(1 - lambda) \* (mu \* I2.at<Vec3f>(y2, x1) + (1 - mu) \* I2.at<Vec3f>(y1, x1));

}

}

8) image stitching with blend

blend\_stitching(I1, I2, I\_f, bound\_l, bound\_u, 0.5);

I\_f = 0.5\*I1 + 0.5\*I2’

9) show l1,l2, and stitched image

10) convert l\_f into : l\_f\*alpha + beta ( alpha=255.0 , beta= 0.0(default parameter) )

I\_f.convertTo(I\_f, CV\_8UC3, 255.0);

11) write I\_f into “result.png”

imwrite("result.png", I\_f);

**Results**

**텍스트, 하늘, 실외, 나무이(가) 표시된 사진

자동 생성된 설명** left image

텍스트, 실외, 하늘, 잔디이(가) 표시된 사진

자동 생성된 설명 right image

텍스트, 실외이(가) 표시된 사진

자동 생성된 설명

Stitched image(alpha=0.5)