20190002_강다혜_HW1

1. 구현 1
    a. Translate() 함수 구현
    b. Rotate() 함수 구현
    c. Translate 후 Rotate 함수 실행
    d. Rotate 후 Translate 함수 실행
    e. c, d 결과 비교
2. 선택 2
    a. Translate() 함수 구현
    b. Rotate() 함수 구현
3. 추가 검토사항
    a. 자료형 및 픽셀 값 문제
    b. Affine matrix

1. 구현 1

`Translate()`, `Rotate()` 함수를 nearest 보간법을 이용해서 구현한다.

다음은 함수 호출에 필요한 인자들이다.

```
% "nearest" or "bicubic"
translateMethod = "nearest";
rotationMethod = "nearest";
tx = 20;
ty = 30;
rotateAngle = 30;
```

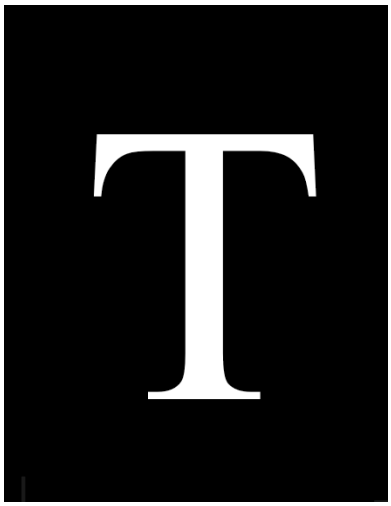
- tx : x 방향(column 방향)으로 어느 만큼 움직일지 결정한다.
- ty : x 방향(row방향)으로 어느 만큼 움직일지 결정한다.
- rotateAngle : degree를 기준으로 어느 만큼 회전할 지 결정한다. rotateAngle=30은 반시계방향으로 30도 회전한다.

a. Translate() 함수 구현

```
ret = Translate(img, tx, ty, translateMethod);
```

`Translate` 함수는 인자로 (원본 이미지, x축으로 이동할 양, y축으로 이동할 양, 보간법)을 받고 이동된 이미지를 반환한다.

letterT.tif 이미지를 (20, 30)만큼 이동시키면 아래와 같은 결과를 얻을 수 있다.



b. Rotate() 함수 구현

```
ret = Rotate(img, rotateAngle, rotationMethod);
```

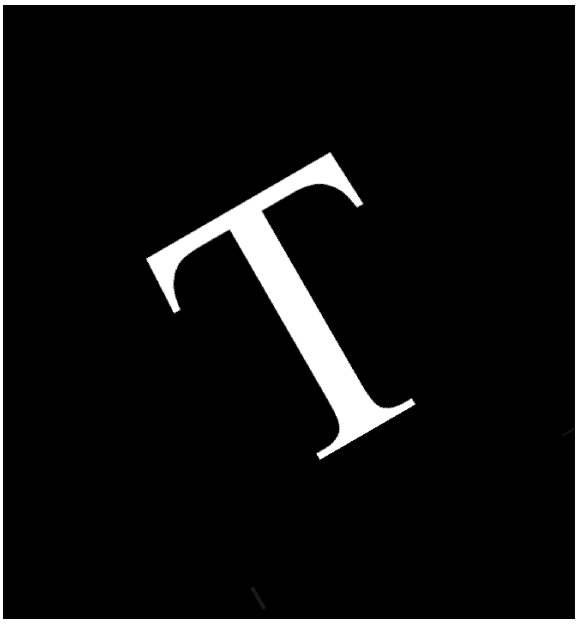
`Rotate` 함수는 인자로 (원본 이미지, 회전 각도, 보간법)을 받고 회전된 이미지를 반환한다.

letterT.tif 이미지를 30도만큼 회전시키면 아래와 같은 결과를 얻을 수 있다.

c. Translate 후 Rotate 함수 실행

```
translated_nearest = Translate(img, tx, ty, translateMethod);
rotated_nearest = Rotate(translated_nearest, rotateAngle, rotationMethod);
```

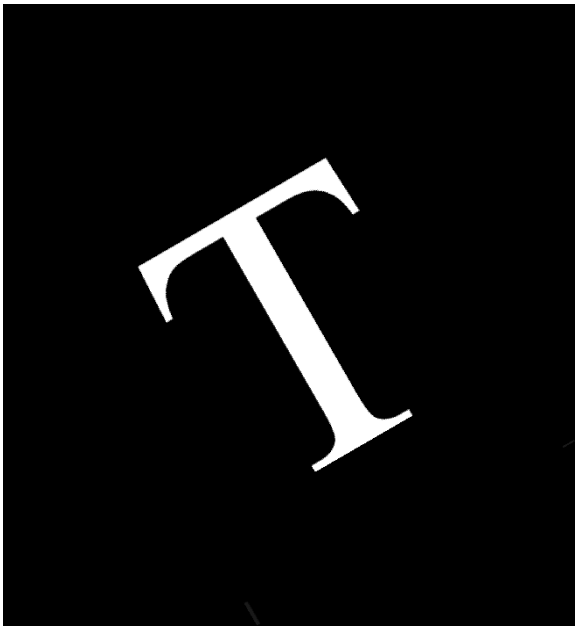
실행 결과는 아래와 같다.



d. Rotate 후 Translate 함수 실행

```
rotated_nearest = Rotate(img, rotateAngle, rotationMethod);
translated_nearest = Translate(rotated_nearest, tx, ty, translateMethod);
```

실행 결과는 아래와 같다.



e. c, d 결과 비교

c, d의 결과 이미지를 각각 투명도 50으로 설정해서 겹친 결과는 다음과 같다.

영상의 사이즈와 문자의 위치가 다른 것을 확인할 수 있다. 이는 원본 영상에 곱해지는 affine matarix의 형태가 다르기 때문이다.

반시계방향으로 θ만큼 회전시키는 affine matrix를 $\begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$ 으로, x축 방향으로 t_x만큼, y축 방향으로 t_y만큼 이동시키는 affine matrix를 $\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$ 로 나타낸다.

c에서 영상에 곱해지는 affine matrix는 이렇게 계산할 수 있다.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

한편, d에서 영상에 곱해지는 affine matrix는 이렇게 계산할 수 있다.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translate, Rotate 함수를 구현하기 위해서는 backward 방법을 사용한다.

따라서 Translate 함수에서는 $\begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ 처럼 원래 좌표값을 찾아야 한다.

Rotate 함수에서는 원점을 기준으로 회전한다. 원본 영상의 이미지 크기를 row * col, 회전된 영상에 외접하는 사각형의의 크기를 newRow * newCol이라고 했을 때,

$\begin{bmatrix} 1 & 0 & row/2 \\ 0 & 1 & col/2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos(-\theta) & -sin(-\theta) & 0 \\ sin(-\theta) & cos(-\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & newRow/2 \\ 0 & 1 & newCol/2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ 행렬 계산을 통해서 회전된 이미지의 x', y'에 대응하는 원본 이미지의 x, y 좌표값을 구해야 한다.

2. 선택 2

a. Translate() 함수 구현

`Translate()` 함수 내에서 affineMatrix와 r, c에 대응하는 원본 영상의 좌표값을 찾을 때, 보간법에 따라서 pixel의 값을 불러오도록 했다.

```
translatedCoords = affineMatrix * [r; c; 1];
if (method == "nearest")
    ret(r, c, ch) = nearest(img, row, col, ch, translatedCoords(1), translatedCoords(2));
end
if (method == "bicubic")
    ret(r, c, ch) = bicubic(img, row, col, ch, translatedCoords(1), translatedCoords(2));
end
```

`bicubic` 함수는 `Rotate()` 함수에서도 bicubic 보간법 선택 시 호출되는 함수이다.

bicubic 함수의 구현은 다음과 같다.

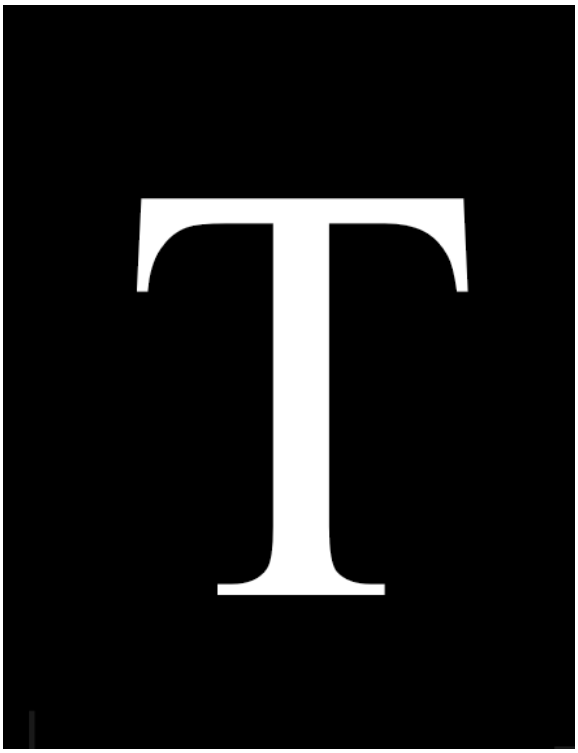
```
function retValue = bicubic(img, maxRow, maxCol, ch, targetR, targetC)
retValue = 0;

baseRow = floor(targetR) - 1;
```

```
baseCol = floor(targetC) - 1;
pixelValue=[];

pixelValue(1, 1) = getPixel(img, maxRow, baseRow + 0, maxCol, baseCol + 0, ch);
pixelValue(2, 1) = getPixel(img, maxRow, baseRow + 1, maxCol, baseCol + 0, ch);
pixelValue(3, 1) = getPixel(img, maxRow, baseRow + 2, maxCol, baseCol + 0, ch);
pixelValue(4, 1) = getPixel(img, maxRow, baseRow + 3, maxCol, baseCol + 0, ch);

% (중략)

pixelValue(4, 4) = getPixel(img, maxRow, baseRow + 3, maxCol, baseCol + 3, ch);
matt = [(baseRow + 0) ^ 3, (baseRow + 0) ^ 2, (baseRow + 0), 1;
    (baseRow + 1) ^ 3, (baseRow + 1) ^ 2, (baseRow + 1), 1;
    (baseRow + 2) ^ 3, (baseRow + 2) ^ 2, (baseRow + 2), 1;
    (baseRow + 3) ^ 3, (baseRow + 3) ^ 2, (baseRow + 3), 1];
matt = getInverseMatrix(matt);

target = [pixelValue(1, 1); pixelValue(2, 1); pixelValue(3, 1); pixelValue(4, 1)];
ans1 = matt * target;
y1 = targetR ^ 3 * ans1(1) + targetR ^ 2 * ans1(2) + targetR * ans1(3) + ans1(4);

% (중략)

target = [pixelValue(1, 4); pixelValue(2, 4); pixelValue(3, 4); pixelValue(4, 4)];
ans4 = matt * target;
y4 = targetR ^ 3 * ans4(1) + targetR ^ 2 * ans4(2) + targetR * ans4(3) + ans4(4);

matt = [baseCol + 0 ^ 3, baseCol + 0 ^ 2, baseCol + 0, 1;
    (baseCol + 1) ^ 3, (baseCol + 1) ^ 2, (baseCol + 1), 1;
    (baseCol + 2) ^ 3, (baseCol + 2) ^ 2, (baseCol + 2), 1;
    baseCol + 3 ^ 3, baseCol + 3 ^ 2, baseCol + 3, 1];
matt = getInverseMatrix(matt);

target = [y1; y2; y3; y4];
ans5 = matt * target;
final = targetC ^ 3 * ans5(1) + targetC ^ 2 * ans5(2) + targetC * ans5(3) + ans5(4);
end
```

1. img(r, c)의 값을 구할 때, bicubic 보간법을 사용하기 위한 좌표 r_1, r_2, r_3, r_4와 c_1, c_2, c_3, c_4를 구한다. 여기에서는 r_1을 $\mathrm{flooor}(r) - 1$로 두고, $r_2 = r_1 + 1, r_3 = r_2 + 1, r_4 = r_3 + 1$로 했다. c좌표 역시 같은 방법으로 정의한다.
2. $(r_1, c_1), (r_1, c_2), ..., (r_4, c_4)$에 대응하는 원본 이미지의 픽셀값 16개를 구한다.
3. r_1에서 $(c_1, f(c_1)), (c_2, f(c_2)), (c_3, f(c_3)), (c_4, f(c_4))$를 지나는 하는 3차식 $f(x) = a * x^3 + b * x^2 + c * x + d$의 해를 구한다.
4. 이렇게 완성한 3차식에서 $f(r)$값을 구한다.
5. 3~4번을 반복해 $(r, c_1), (r, c_2), (r, c_3), (r, c_4)$에서의 픽셀값을 구한다.

6. 같은 방법으로 3차식을 세워서 (r, c)에서의 픽셀값을 구한다.

해당 구현의 실행 결과는 아래와 같다.

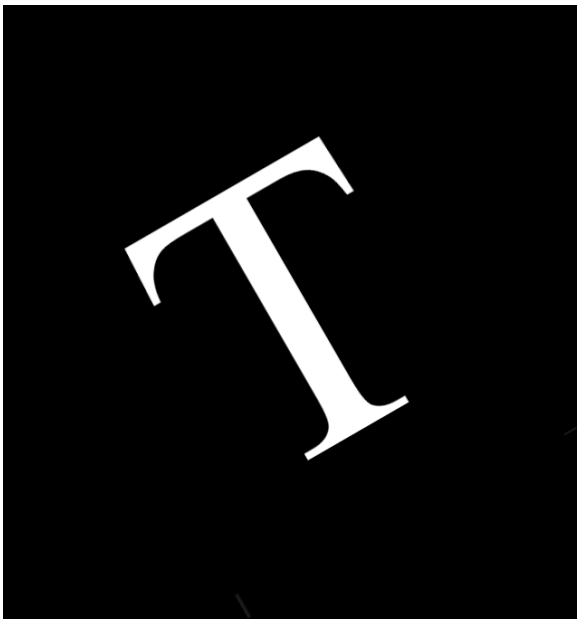


b. Rotate() 함수 구현

`Translate()` 함수와 마찬가지로 `bicubic` 함수를 호출하도록 작성했다.

```
rotatedCoords = affineMatrix * [r; c; 1];
if (method == "nearest")
    ret(r, c, ch) = nearest(img, row, col, ch, rotatedCoords(1), rotatedCoords(2));
end
if (method == "bicubic")
    ret(r, c, ch) = bicubic(img, row, col, ch, rotatedCoords(1), rotatedCoords(2));
end
```

실행 결과는 아래와 같다.



3. 추가 검토사항

a. 자료형 및 픽셀 값 문제

우리가 사용하는 영상의 픽셀값은 8bit로, 0~255 사이이다. 하지만 bicubic 연산 중에 결과값이 음수나 255를 넘는 값이 나올 수 있다. 이 문제를 해결하기 위해서, 음수인 픽셀 값은 0으로, 255를 넘는 값은 255로 변환했다.

b. Affine matrix

affine matrix에서는 음수값이 등장할 수 있다. translation matrix $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ 에서 t_x, t_y의 값이 음수라면 -x, -y 방향으로 이동하게 된다. scale matrix $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ 등에서도 s_x, s_y가 음수라면 원본 이미지보다 변형된 이미지가 더 작아진다.