기하변환

스케일링, 로테이션, 쉬어링

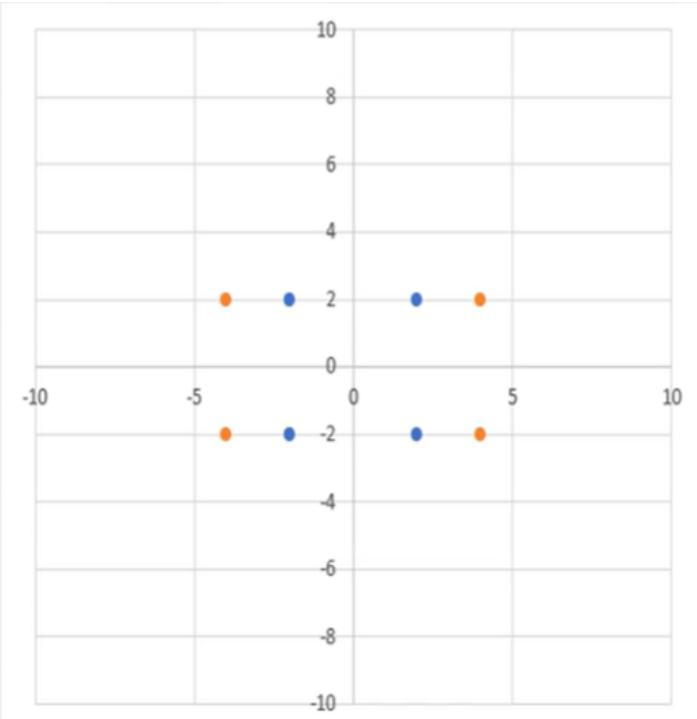
	A	B	C	D	E	F	G	H
1	Scaling							
2	1	0		-2	-2	2	2	
3	0	1		-2	2	-2	2	
4								
5	Rotation			=mmult(A2:B3,D2:G3)				
6								
7								

Ctrl+Shift+enter
여기선 단위 행렬을 곱했으니
동일한 출력이 나올 것

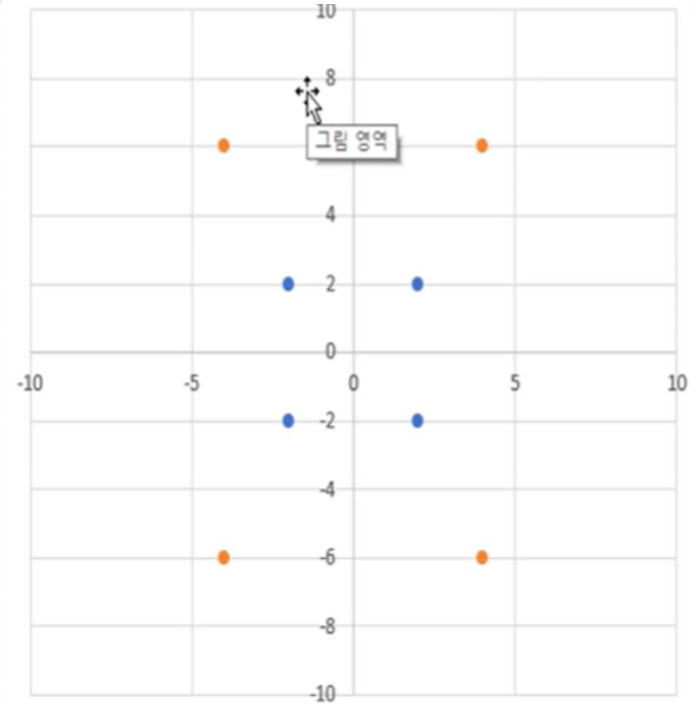
-2	-2	2	2
-2	2	-2	2

Scaling

	A	B	C	D	E	F	G	H	I
1	Scaling								
2	2	0		-2	-2	2	2		
3	0	1		-2	2	-2	2		
4									
5	Rotation			-4	-4	4	4		
6				-2	2	-2	2		
7									
8									



	A	B	C	D	E	F	G	H	I
1	Scaling								
2	2	0		-2	-2	2	2		
3	0	1		-2	2	-2	2		
4									
5	Rotation			-4	-4	4	4		
6				-2	2	-2	2		
7									
8									



Rotation

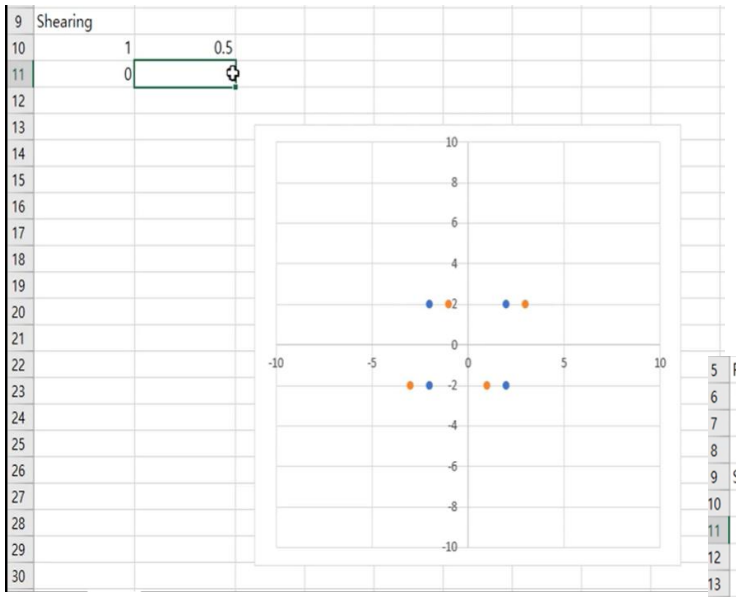
5	Rotation	
6	6.12574E-17	-1
7	1	6.12574E-17

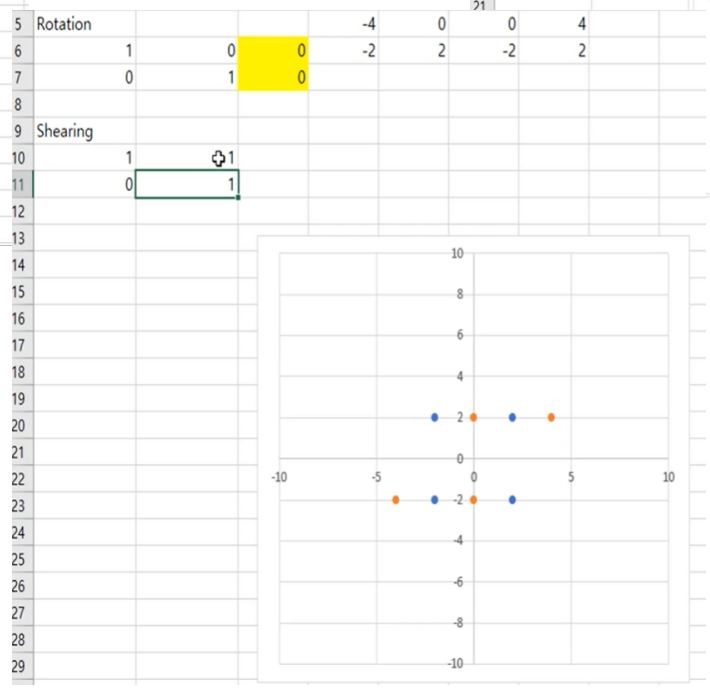
0이라고 보면 되고,
90도 Rotation 한 결과이다

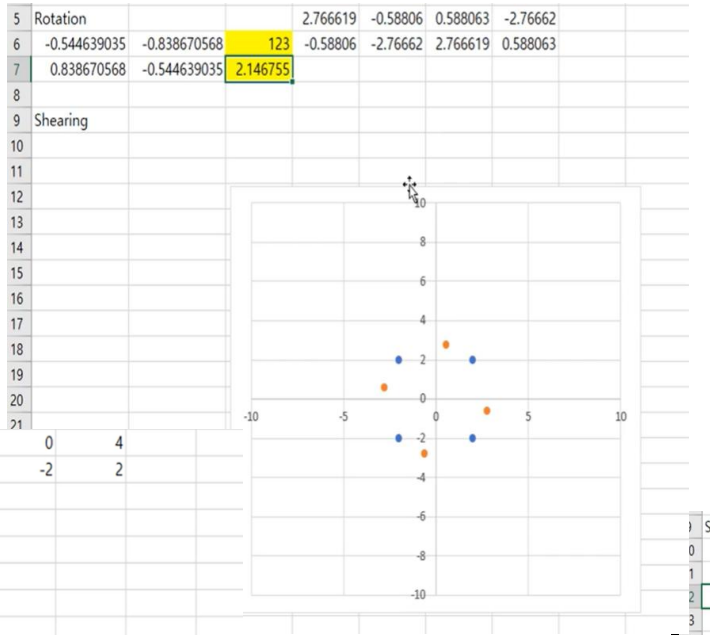
5	Rotation		
6	1	2.4503E-16	360
7	-2.4503E-16	1	6.283185

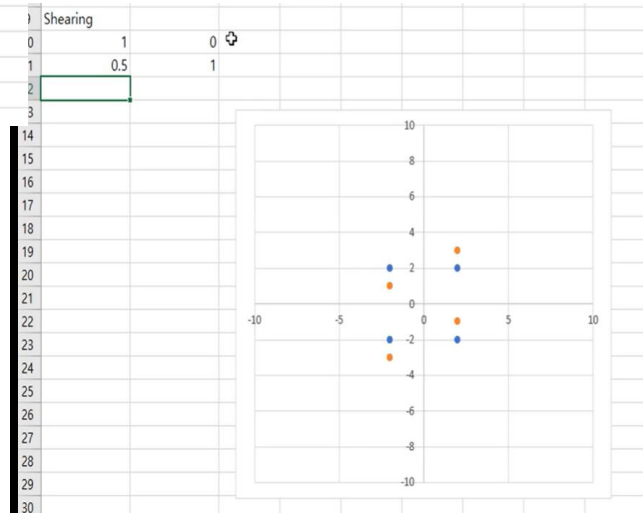
이건 360도 회전한 결과

shearing


Shearing
1 0.5
0


Rotation
1 0 0
0 1 0
Shearing
1 1
0 1

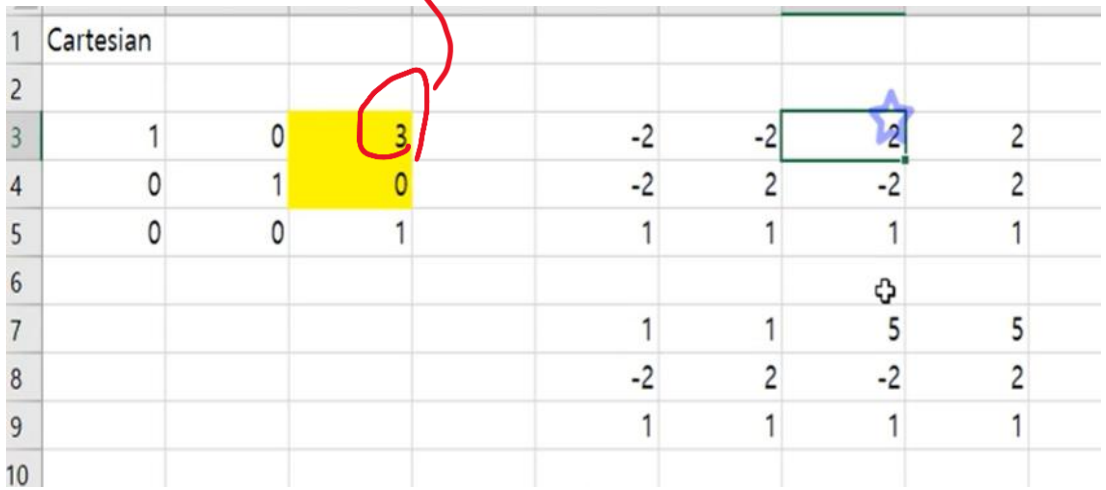

Rotation
-0.544639035 -0.838670568 123
0.838670568 -0.544639035 2.146755
Shearing


Shearing
1 0
0.5 1

Cartesian

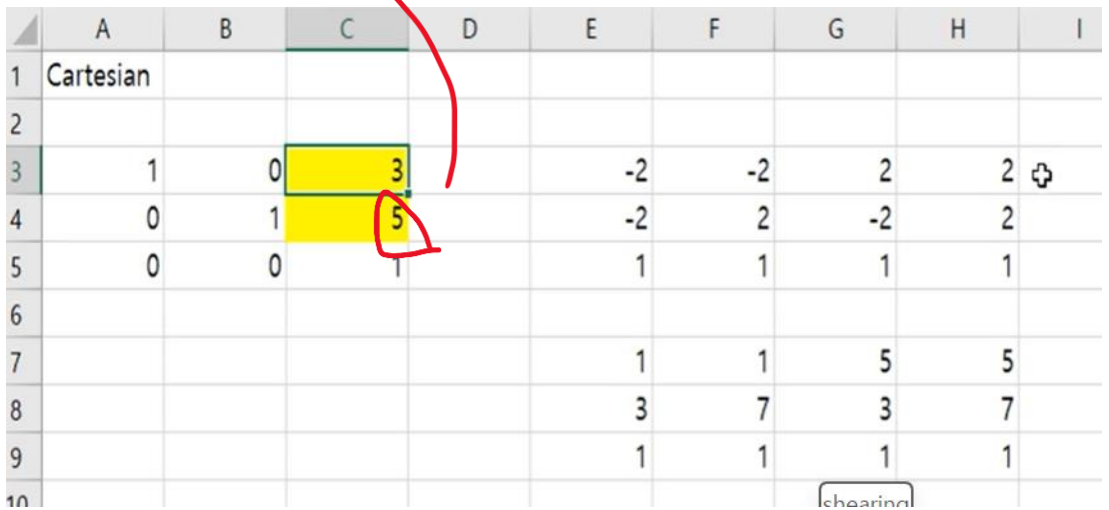
- 모두 행렬의 곱셈으로 이루어져있어
- Translation 이건 유일하게 덧셈으로 이루어져 있어
- So 디카르트가 카르테시안(Cartesian) 표기법을 만들어냄
- 이동은 행렬의 덧셈으로만 가능했어
- But 행 렬 하나씩 추가하는 방법을 한다면, x와 y의 이동량을 알려주는 값
- 하나씩 행 렬을 추가한다면 끝에 두개만 성분을 바꿔서 x와 y의 이동량을 표현하여 행렬의 덧셈이 아니라 곱셈을 이용하여 이동을 볼 수 있어
- 복합적인 기하변환을 수행할 때 하나의 행렬로 만들어서 적용할 수 있어 원랜 덧셈인데 카르테시안으로 적용하면 다른 것과 섞일 수 있다는 장점이 있어

X방향 증가



1	Cartesian						
2							
3	1	0	3	-2	-2	2	2
4	0	1	0	-2	2	-2	2
5	0	0	1	1	1	1	1
6							
7				1	1	5	5
8				-2	2	-2	2
9				1	1	1	1
10							

y방향 증가



	A	B	C	D	E	F	G	H	I
1	Cartesian								
2									
3	1	0	3		-2	-2	2	2	
4	0	1	5		-2	2	-2	2	
5	0	0	1		1	1	1	1	
6									
7					1	1	5	5	
8					3	7	3	7	
9					1	1	1	1	
10							shearing		

```
691     //기하변환_이동 [Translation] -> 화소의 위치가 바뀌겠지? so, x랑 y 이동량 변수를 설정해야지
692
693     int Tx = 50, Ty = 30;
694     for (int i = 0; i < H; i++)
695     {
696         for (int j = 0; j < W; j++)
697         {
698             //입력 영상의 특정 픽셀을 출력 영상으로 보내야해
699             //1st 순방향 사상
700             if ((i + Ty < H && i + Ty >= 0) && (j + Tx < W && j + Tx >= 0))
701             {
702                 Output[(i + Ty) * W + (j + Tx)] = Image[i * W + j]; //영상의 범위를 벗어나게 돼 --> 예외처리 필수!
703             }
704
705         }
706     }
```

```
520     // Translation
521     int Tx = 50, Ty = -30;
522     for (int i = 0; i < H; i++) {
523         for (int j = 0; j < W; j++) {
524             if((i+Ty < H && i+Ty >= 0) && (j+Tx < W && j+Tx >=0))
525                 Output[(i+Ty)*W + (j+Tx)] = Image[i*W + j];
526         }
527     }
```

```
610  void Translation(BYTE* Image, BYTE* Output, int W, int H, int Tx, int Ty)
611  {
612
613      //Ty 부호 바꿔주어야 해 --> 사람 기준으로 움직이는 걸 기대하려면, 이 상태에서 *-1을 곱해야해
614
615      Ty = Ty * -1;
616
617      for (int i = 0; i < H; i++)
618      {
619          for (int j = 0; j < W; j++)
620          {
621              //입력 영상의 특정 픽셀을 출력 영상으로 보내야해
622              //1st 순방향 사상
623              if ((i + Ty < H && i + Ty >= 0) && (j + Tx < W && j + Tx >= 0))
624              {
625                  Output[(i + Ty) * W + (j + Tx)] = Image[i * W + j]; //영상의 범위를 벗어나게 돼 --> 예외처리 필수!
626              }
627
628          }
629      }
630
631  }
```

//Ty 부호 바꿔주어야 해 --> 사람 기준으로 움직이는 걸 기대하려면, 이 상태에서 *-1을 곱해야해

VerticalFlip -> 위 아래 뒤집기

```
648
649     //기하변환_이동 [Translation] -> 화소의 위치가 바뀌겠지? so, x랑 y 이동량 변수를 설정해야지
650
651
652
653     VerticalFlip(Image, W, H);
654
655
656     int Tx = 50, Ty = 30;
657
658
659
660     for (int i = 0; i < H; i++)
661     {
662         for (int j = 0; j < W; j++)
663         {
664             //입력 영상의 특정 픽셀을 출력 영상으로 보내야해
665             //1st 순방향 사상
666             if ((i + Ty < H && i + Ty >= 0) && (j + Tx < W && j + Tx >= 0))
667             {
668                 Output[(i + Ty) * W + (j + Tx)] = Image[i * W + j]; //영상의 범위를 벗어나게 돼 --> 예외처리 필수!
669             }
670
671         }
672     }
673
674
```

뒤집힌 상태로 좌, 하단에 마진을 남김
Flip -> 이동

HorizontalFlip

```
623  void HorizontalFlip(BYTE* Img, int W, int H)
624  {
625      for (int i = 0; i < W/2; i++)
626      {
627          for (int j = 0; j < H; j++) //y축 한 줄
628          {
629              swap(&Img[j * W + i], &Img[j* W + (W-1-i)]);
630          }
631      }
632  }
```

Scaling 순방향

```
687     //factor < 1 ==> 축소
688
689     double SF_X = 1.3, SF_Y = 1.5; // -> 확대 --> hole 이 나타날 것
690     int tmpX, tmpY;
691
692     for (int i = 0; i < H; i++)
693     {
694         for (int j = 0; j < W; j++)
695         {
696             tmpX = int(j * SF_X);
697             tmpY = int(i * SF_Y);
698             //스케일링 값은 음수가 될 수 없어
699             if (tmpY<H && tmpX<W)
700             {
701                 Output[tmpY*W+tmpX] = Image[i * W + j]; //영상의 범위를 벗어나게 돼 --> 예외처리 필수!
702             }
703
704
705         }
706     }
707
```

Scaling 역방향

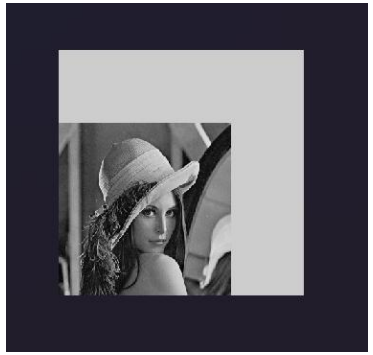
```
689     double SF_X = 1.3, SF_Y = 1.5; // -> 확대 --> hole 이 나타날 것
690     int tmpX, tmpY;
691
692     for (int i = 0; i < H; i++)
693     {
694         for (int j = 0; j < W; j++)
695         {
696
697             //역방향 연산 하기
698             tmpX = int(j / SF_X);
699             tmpY = int(i / SF_Y);
700             if (tmpY < H && tmpX < W)
701             {
702                 Output[i*W+j] = Image[tmpY*W + tmpX]; //영상의 범위를 벗어나게 돼 --> 예외처리 필수!
703             }
704
705         }
706     }
707
```

Scaling 확대 & 축소

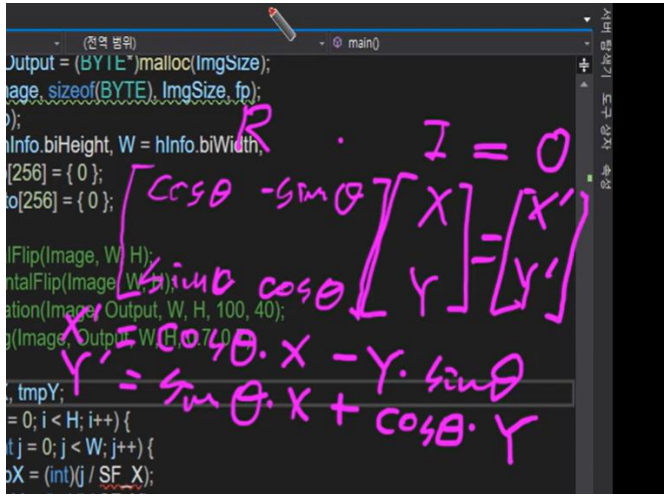
```
656
657 void Scaling(BYTE* Image, BYTE* Output, int W, int H, double SF_X, double SF_Y)
658 {
659     int tmpX, tmpY;
660     for (int i = 0; i < H; i++)
661     {
662         for (int j = 0; j < W; j++)
663         {
664
665             //역방향 연산 하기
666             tmpX = int(j / SF_X);
667             tmpY = int(i / SF_Y);
668             if (tmpY < H && tmpX < W)
669             {
670                 Output[i * W + j] = Image[tmpY * W + tmpX]; //영상의 범위를 벗어나게 돼 --> 예외처리 필수!
671             }
672
673         }
674     }
675 }
676
```

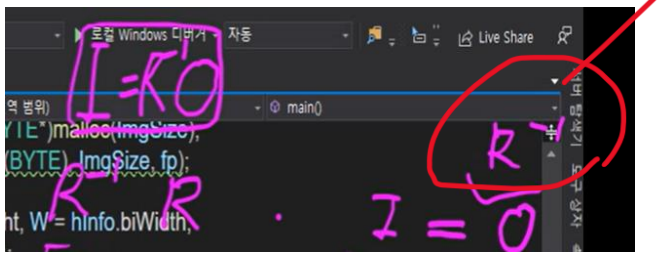
```
703
704
705     //스케일링
706     //factor > 1 ==> 확대
707     //factor < 1 ==> 축소
708
709     Scaling(Image, Output, W, H, 2.0, 0.7);
710
711
```

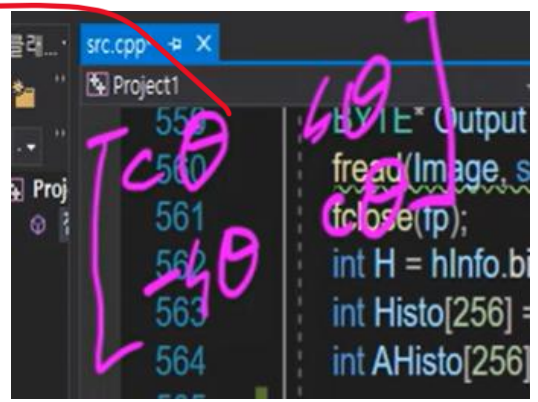
```
704
705     //스케일링
706     //factor > 1 ==> 확대
707     //factor < 1 ==> 축소
708
709     Scaling(Image, Output, W, H, 0.7, 0.7); // 이렇게 동일하게 스케일링 되면 Uniform Scaling이라고 함
710
711
712
```



회전영상 수식


main()
Output = (BYTE*)malloc(ImgSize);
age, sizeof(BYTE), ImgSize, fp);
hInfo.biHeight, W = hInfo.biWidth,
[256] = { 0 };
o[256] = { 0 };
IFlip(Image, W, H);
ntalFlip(Image, W, H);
ation(Image, Output, W, H, 100, 40);
(Image, Output, W, H, 70, 40);
, tmpY;
= 0; i < H; i++) {
t j = 0; j < W; j++) {
oX = (int)(j / SF_X);

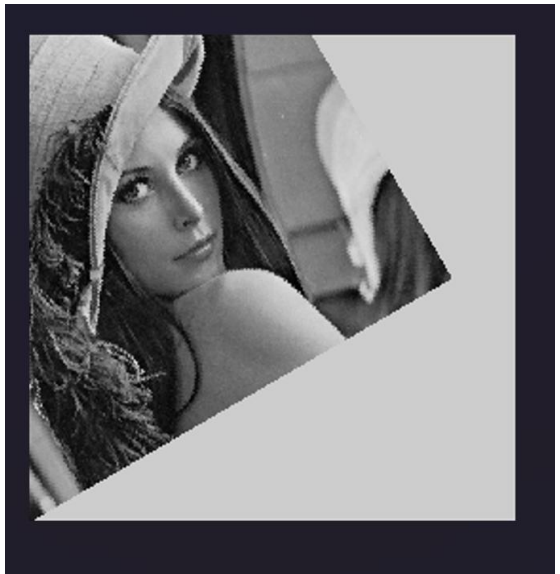

로컬 Windows 디버거
자동
Live Share
main()
(BYTE) ImgSize, fp);
nt, W = hInfo.biWidth,


src.cpp
Project1
561
563
564
BYTE* Output
fread(Image, s
fclose(fp);
int H = hInfo.bi
int Histo[256]
int AHisto[256]

회전 순방향사상 & 역방향사상

```
709
710     int tmpX, tmpY;
711     int Angle = 30;
712     //각도를 라디안으로 받아 참고해
713     double Radian = Angle * 3.141592 / 180.0; //라디안은 pi/180 을 곱하면 된다
714     for (int i = 0; i < H; i++)
715     {
716         for (int j = 0; j < W; j++)
717         {
718             int tmpX = cos(Radian) * j - sin(Radian) * i;
719             int tmpY = sin(Radian) * j + cos(Radian) * i;
720
721             if ((tmpY < H && tmpY >= 0) && (tmpX < W && tmpX >= 0))
722             {
723                 // 순방향 사상
724                 // hole이 어쩔 수 없이 발생하게 돼
725                 // 입력에서 30도만큼 회전한 게 Output으로 가야해
726                 // 출력영상 기준으로 -30도만큼 이동한 게 입력에서 30도 이동 한 거랑 같겠지
727                 Output[tmpY * W + tmpX] = Image[i * W + j];
728             }
729         }
730     }
731
```

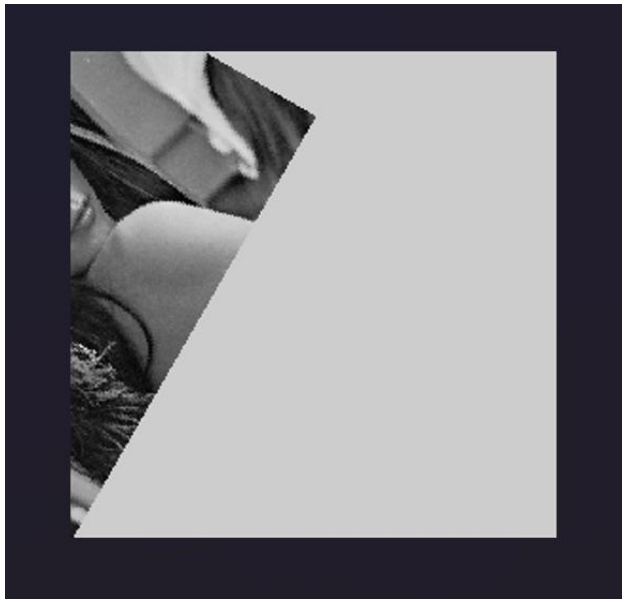
```
710     int tmpX, tmpY;
711     int Angle = 30;
712     //각도를 라디안으로 받아 참고해
713     double Radian = Angle * 3.141592 / 180.0; //라디안은 pi/180 을 곱하면 된다
714     for (int i = 0; i < H; i++)
715     {
716         for (int j = 0; j < W; j++)
717         {
718             int tmpX = (int)(cos(Radian) * j + sin(Radian) * i);
719             int tmpY = (int)(- sin(Radian) * j + cos(Radian) * i);
720
721             if ((tmpY < H && tmpY >= 0) && (tmpX < W && tmpX >= 0))
722             {
723                 // 역방향사상
724                 Output[i*W+j] = Image[tmpY * W + tmpX];
725             }
726         }
727     }
728
```



Rotation 함수

지금까지 영상의 0,0을 중심으로 구했는데 센터를 중심으로 해서 구하는 것도 해보기 !

```
677  void Rotation(BYTE* Image, BYTE* Output, int W, int H, int Angle)
678  {
679
680      int tmpX, tmpY;
681      //각도를 라디안으로 받아 참고해
682      double Radian = Angle * 3.141592 / 180.0; //라디안은 pi/180 을 곱하면 된다
683      for (int i = 0; i < H; i++)
684      {
685          for (int j = 0; j < W; j++)
686          {
687              int tmpX = (int)(cos(Radian) * j + sin(Radian) * i);
688              int tmpY = (int)(-sin(Radian) * j + cos(Radian) * i);
689
690              if ((tmpY < H && tmpY >= 0) && (tmpX < W && tmpX >= 0))
691              {
692                  // 역방향사상
693                  Output[i * W + j] = Image[tmpY * W + tmpX];
694              }
695          }
696      }
697
698  }
699
```



```
731
732      Rotation(Image, Output, W, H, 60);
733
734
```