MATLAB을 이용한

디지털 영상처리의 기초

영상의 모양을 변화시키는 여러 가지 방법들이 존재하는데, 특별한 변화에 대한 필요성이 이러한 문제에 동기를 부여하고 있다. 우리는 영상에서 무언가(잡음 등)를 제거할 필요가 있거나 영상의 모양을 개선하거나 영상에서 물체 등(사이즈, 위치, 성분의 수 등)을 계산하는 등이다.

그러나 영상처리와 또 다른 양상이 있는데, 영상에 몇 가지의 특수한 효과를 첨가 것이 이에 속한다. 우리는 영상을 효과적임 면이나 재미를 위해 변화시키기를 원하는 경우가 있다. 여러 가지 다른 효과가 가능하고, 이 장에서 그들의 몇 가지를 보기로 한다. 여기서 컴퓨터그래픽 영역으로 들어간다. 그러나 우리의 알고리즘은 화소들의 값 혹은 위치의 변화를 포함하고, 매우 간단한 의미를 가지고 많은 효과를 얻을 수 있다.

16.1 극좌표계

방사형태의 성질을 가지는 많은 특수효과들이 있다. 이들의 모양은 영상의 중심에서 바깥쪽으로 방출하는 형태이다. 이들의 효과를 달성하기 위해 영상을 직각좌표계에서 극좌표계로 변환하고 다시 역으로 변환할 필요가 있다.

원점

rows X cols 사이즈의 영상을 가정하자. 해당 차원이 홀수이면 중심 화소에 원점을 선택할 수 있다. 만일 차원이 짝수이면 오른쪽 아래의 4분면의 왼쪽 모서리에 원점을 선택한다. 그래서 7X9 영상에서 원점은 (4,5)의 화소에 있고, 6X8 영상은 원점이 (4,4)의 화소이다. 우리는 원점을 원칙적으로 아래와 같이 표현할 수 있다.

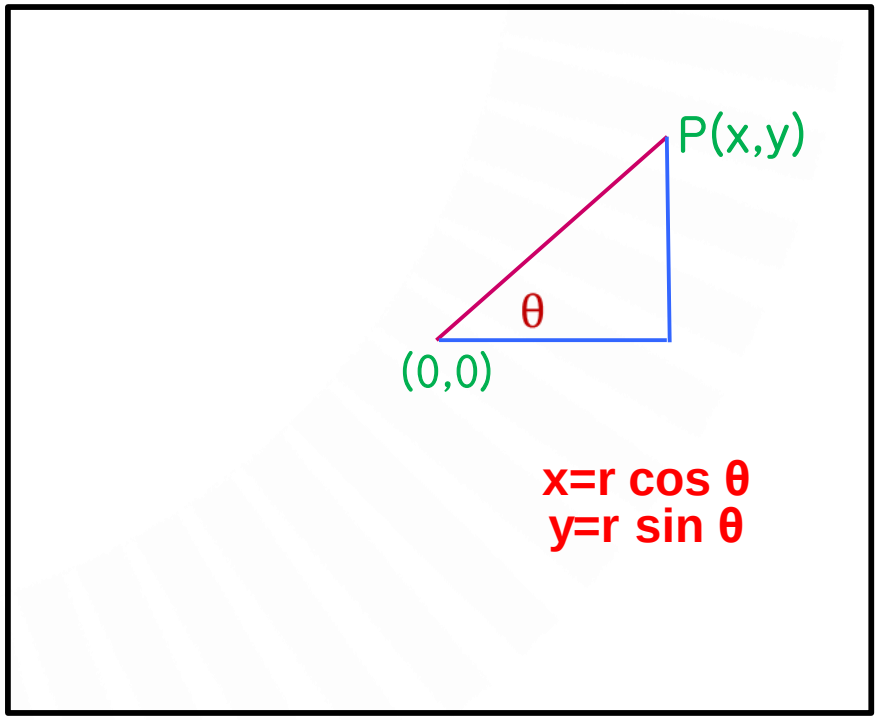
$$x_0 = \lceil (r+1)/2 \rceil$$

$$y_0 = \lceil (c+1)/2 \rceil,$$

여기서 r과 c는 각각 행들과 열들의 수이고, ⌈x⌉는 ceiling 함수이며, 이것은 x보다 큰 범위에서 가장 작은 수로 return된다. 물론 하나의 차원이 홀수이고, 나머지가 짝수이면 원점은 홀수차원의 중심과 짝수차원의 2번째 반의 시작점에 있다.

그러므로 원점의 좌표를 아래와 같이 구할 수 있다.

직각 좌표를 극좌표로 변경



```
>> ox=ceil((rows+1)/2)
>> oy=ceil((cols+1)/2)
```

극좌표

극좌표를 구하기 위해 아래의 일반적인 공식을 적용한다.

$$r = \sqrt{x^2 + y^2}$$

$$\theta = \tan^{-1}(y/x),$$

이것은 meshgrid 함수를 이용하여 하나의 배열로 구현될 수 있다. 여기서 indices(지수들)은 원점의 값들에 의해 아래와 같이 오프셋된다.

```
>> [y,x]=meshgrid([1:cols]-oy,[1:rows]-ox);
>> r=sqrt(x.^2+y.^2);
>> theta=atan2(y,x);
```

예) rows= 100, cols= 100 이라면
Ox=oy= 51
[y,x]=meshgrid ([-50:49, -50:49]);

우리는 극좌표계 (r, θ)에 대응하는 (x,y) 좌표계가 아래와 같다는 것을 이용하여 쉽게 되돌아 갈 수 있다.

$$x = r\cos\theta$$

$$y = r\sin\theta.$$

그러므로 아래와 같이 처리할 수 있다.

```
>> x2=round(r.*cos(theta))+ox
>> y2=round(r.*sin(theta))+oy
```

이 배열들은 원래의 배열들과 똑같은 지수들(indices)을 포함한다.

예제

우리는 3장에서 영상이 화소들로 표현된다는 것을 설명하였고, 이것은 imresize 함수를 이용하여 낮은 분해능의 큰 블록들로 표현된다는 것을 알고 있다. 그러나 mod 함수를 이용하여 같은 효과를 달성할 수 있다. 일반적으로 mod(x,n)은 x가 n으로 나눌 때 그 나머지이다. 예를 들면 다음을 보자.

```
>> x=1:12

x =

     1     2     3     4     5     6     7     8     9    10    11    12

>> mod(x,4)

ans =

     1     2     3     0     1     2     3     0     1     2     3     0
```

만일 x가 4로 나누어지면 나머지는 0이고, mod 함수는 0으로 return한다. 만일 원래의 값에서 mod 값들을 빼면 반복수를 얻는데, 이것은 영상에서 아래와 같은 특수효과를 만들 것이다.

```
>> x-mod(x,4)

ans =

     0     0     0     4     4     4     4     8     8     8     8    12
```

그러므로 방사형 변수에서 mod의 값들을 빼는 것은 방사형의 특수효과를 만들 것이다.

꽃의 영상으로 이를 시도할 수 있다. 간단히 하기 위해 아래와 같이 그레이스케일 영상으로 변환한다.

```
>> f=imread('flowers.tif');
>> fg=rgb2gray(f);
>> [rows cols]=size(fg)

rows =

   362


cols =

   500

>> ox=ceil((rows+1)/2)

ox =
   182

>> oy=ceil((cols+1)/2)

oy =

   251

>> [y,x]=meshgrid([1:cols]-oy,[1:rows]-ox);
>> r=sqrt(x.^2+y.^2);
>> theta=atan2(y,x);
```

이것은 표준 셋업방식이고, 특수효과를 아래와 같이 수행한다.

```
>> r2=r-mod(r,5);
>> theta2=theta-mod(theta,0.087);
```

각도는 라디안으로 주어지므로 modulus에서 작은 값을 사용하는데, 여기서 0.087 = $5\pi/180$을 선택한다. 여기서 다시 직각좌표계로 아래와 같이 되돌린다.

5°

```
>> x2=r2.*cos(theta2);
>> y2=r2.*sin(theta2);
```

디스플레이를 위한 영상을 얻기 위해 해당 지수들 x2와 y2를 ox와 oy로 더하고, 그들을 rounding off(⌈ ⌉)처리하며, 행의 지수가 1과 rows 사이에 있고 열의 지수가 1과 cols 사이에 있도록 아래와 같이 조정해야한다.

```
>> xx=round(x2)+ox;
>> yy=round(y2)+oy;
>> xx(find(xx>rows))=rows;
>> xx(find(xx<1))=1;
>> yy(find(yy>cols))=cols;
>> yy(find(yy<1))=1;
```

여기서 원래의 꽃 영상 매트릭tm fg로부터 영상을 얻기 위해 이들에 새로운 지수 xx와 yy를 사용한다. 이렇게 하기 위해 아래와 같이 2중 루프를 사용하거나,

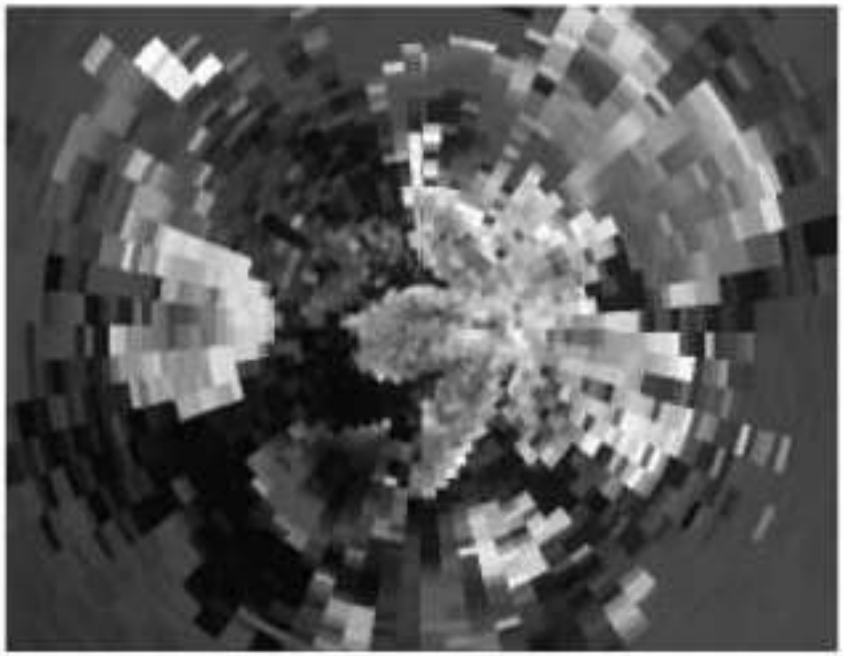
```
>> for i=1:rows,...
     for j=1:cols,...
        f2(i,j)=fg(xx(i,j),yy(i,j));...
     end;...
   end;
```

더욱 간단하고 정확히 같은 효과를 내는 sub2ind 함수를 아래와 같이 사용한다.

```
>> f2=fg(sub2ind([rows,cols],xx,yy));
```

여기서 원 영상과 그 결과를 디스플레이할 수 있다. 이 결과를 그림 16.1에 보였다.

The original fg



The pixelated f2

그림 16.1 방사형 특수효과

```
f=imread('flowers.tif');
fr=f(:,:,1);  % red channel 분리
fg=f(:,:,2); % green channel 분리
fb=f(:,:,3); % blue channel 분리
[rows cols]=size(fg);
ox=ceil((rows+1)/2);
oy=ceil((cols+1)/2);
[y,x]=meshgrid([1:cols]-oy, [1:rows]-ox);
r=sqrt(x.^2+y.^2);
theta=atan2(y,x);
r2=r-mod(r,5);
theta2=theta-mod(theta,0.087);
x2=r2.*cos(theta2);
y2=r2.*sin(theta2);
xx=round(x2)+ox;
yy=round(y2)+oy;
xx(find(xx>rows))=rows;
xx(find(xx <1))=1;
yy(find(yy>cols))=cols;
yy(find(yy <1))=1;

ffr=fr(sub2ind([rows,cols],xx,yy)); %channel 별 변환
ffg=fg(sub2ind([rows,cols],xx,yy)); %channel 별 변환
ffb=fb(sub2ind([rows,cols],xx,yy)); %channel 별 변환
imshow(cat(3,ffr,ffg,ffb));  % channel concatenation
```

16.2 리플효과

우리는 2가지 다른 리플효과를 알아보기로 한다. 그 하나는 욕실유리의 리플, 이것은 욕실에서 젖은 유리를 통해 볼 수 있는 영상의 효과와이고, 또 하나는 pond(연못) 리플로서, 이것은 연못의 표면에서 반사되는 효과이다.

moduli의 빼기는 특수효과를 만든다는 것을 보았다. 욕실의 유리효과는 moduli를 더하여 아래와 같이 얻을 수 있다.

```
>> x=1:12;
>> x+mod(x,4)

ans =

     2     4     6     4     6     8    10     8    10    12    14    12
```

이것은 왼쪽에서 오른쪽으로 그 값들을 증가시킨다는 것을 알 수 있고, 그들은 작은 길이로 중복되는 모양을 보인다. 그 모양은 영상의 전체에 걸쳐 리플로 나타난다. 먼저 직각좌표계를 이용하여 아래와 같이 실험해보자.

```
>> [y,x]=meshgrid(1:cols,1:rows);
>> y2=y+mod(y,32);
>> y2(find(y2<1))=1;
>> y2(find(y2>cols))=cols;
```

여기서 바로 열에 걸쳐서 moduli를 더한 후에 열방향의 범위에 이 값들이 차지하는 것을 확인하기 위해 그 결과를 조정한다. 그러면 새로운 영상을 만들기 위해 sub2ind 함수를 아래와 같이 사용할 수 있다.

```
>> ripple1=fg(sub2ind([rows cols],x,y2));
>> imshow(ripple1)
```

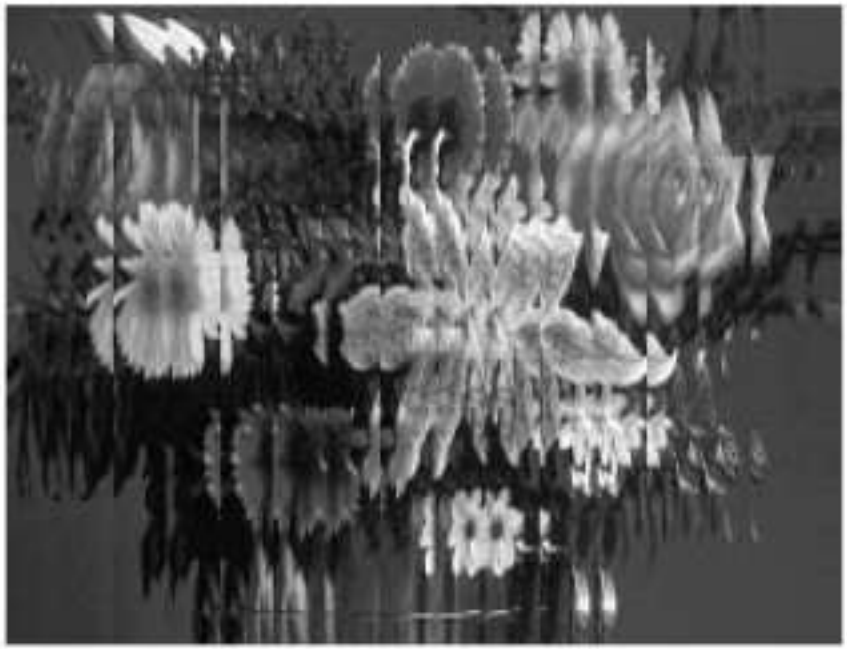
물론 행의 방향으로 같은 방법으로 아래와 같이 처리할 수 있다.

```
>> x2=x+mod(x,32);
>> x2(find(x2<1))=1;
>> x2(find(x2>rows))=rows;
>> ripple2=fg(sub2ind([rows cols],x2,y));
>> imshow(ripple2)
```

또는 한 번에 행과 열의 방향으로 동시에 아래와 같이 처리할 수도 있다

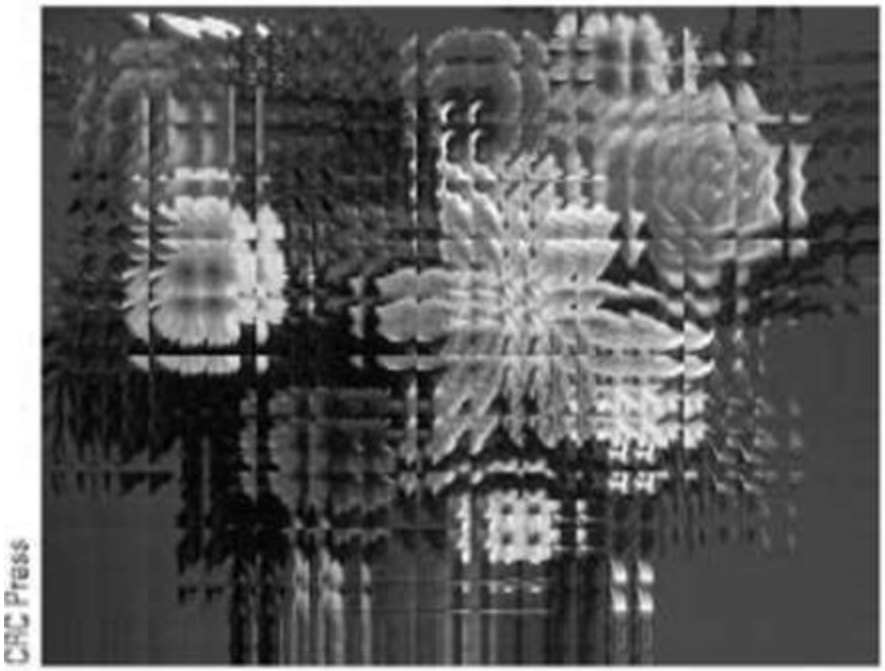
```
>> ripple3=fg(sub2ind([rows cols],x2,y2));
>> imshow(ripple3)
```

모든 결과는 그림 16.2에 보였다.



ripple1

ripple2



ripple3

연못 리플효과를 얻기 위해 사인파를 이용하여 주위의 화소들을 움직인다. 직각의 효과를 얻기 위해 아래와 같이 사용한다.

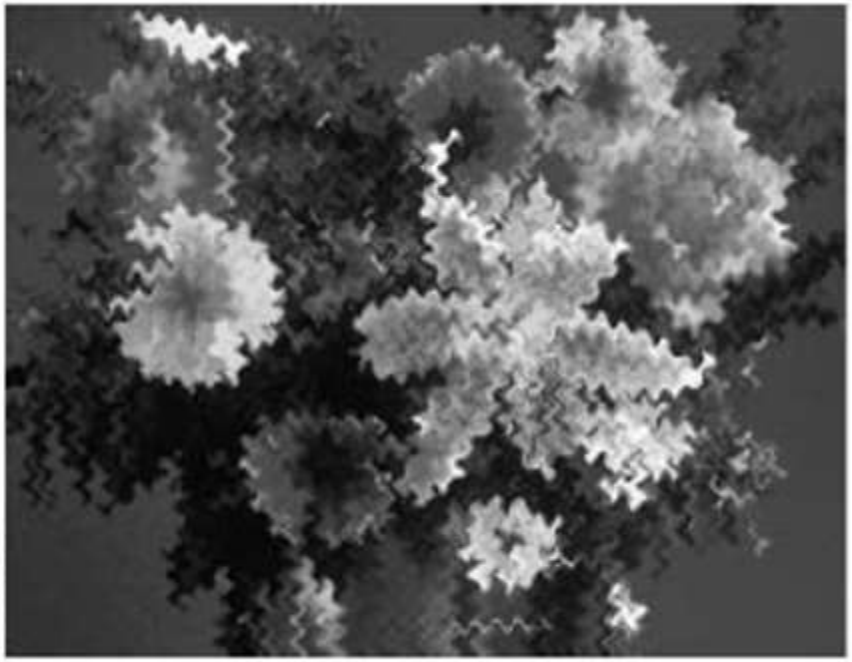
```
>> [y,x]=meshgrid(1:cols,1:rows);
>> y2=round(y+3*sin(x/2));
>> x2=round(x+3*sin(y/2));
```

위와 같이 값들 x2와 y2를 행과 열의 경계영역 내에 아래와 같이 고정되었는지 확인한다.

```
>> ripple5=fg(sub2ind([rows cols],x,y2));
>> ripple6=fg(sub2ind([rows cols],x2,y));
>> ripple7=fg(sub2ind([rows cols],x2,y2));
```

ripple5

ripple6



ripple7

16.3 일반적인 왜곡효과

앞 절에서 리플효과들은 왜곡(distortion)효과라고 하는 더욱 일반적인 효과의 예이다. 여기서 이 효과들은 화소들이 그들의 위치에 따라 변화량이 그렇게 많지 않다. 같은 방법으로 많은 왜곡효과들은 단계적인 순서로 묘사할 수 있다. 직각좌표계에서만 포함되는 하나의 효과를 다음과 같이 생각하자.

1. 사이즈가 m×n인 영상 매트릭스 p(i,j)로 시작하라.
2. 2개의 새로운 지수형 배열(동일 사이즈의 매트릭스) x(i,j) 및 y(i,j)를 만들어라.
3. 새로운 영상 q(i,j)를 아래와 같이 만들어라.

$$q(i,j) = p(x(i,j), y(i,j)).$$

우리는 배열 x(i,j)와 y(i,j)가 분수가 되거나 혹은 영상 매트릭스의 경계들의 바깥에 있는 값들을 만드는 과정에 의해 생성되는 것을 위에서 보았다. 그러므로 우리는 지수형의 배열들이 정수들만을 포함하고, 또 모든 i 및 j가 각각 $1 \le x(i,j) \le m$ 및 $1 \le y(i,j) \le n$이 됨을 확인할 필요가 있다.

만일 극좌표를 요구하는 효과를 가지면 위의 과정에 대하여 몇 가지의 단계를 아래와 같이 추가한다.

1. 사이즈가 m×n인 영상 매트릭스 p(i,j)로 시작하라.
2. 각 위치 (i,j)의 극좌표계를 포함하는 배열 r(i,j) 및 θ(i,j)를 만들어라.
3. r과 θ의 값들을 조정하여 새로운 $s(i,j)$와 $\phi(i,j)$를 마들어라.
4. 이들 새 배열들을 직가좌표의 지수형 배열들(매트릭스로서 같은 사이즈)을 아래와 같 이 출력하라.

$$x(i,j) = s(i,j)\cos(\phi(i,j)) + o_x$$
$$y(i,j) = s(i,j)\sin(\phi(i,j)) + o_y$$

여기서 (o_x, o_y)는 극좌표에서의 원점이다.

5. 새로운 영상 q(i,j)를 아래와 같이 만들어라.

$$q(i,j) = p(x(i,j), y(i,j)).$$

위와 같이 지수형 배열들이 정수만을 포함하고 적절한 경계를 가지는지 확인해야 한다.

위리는 위에서 MATLAB에서 여러 가지 왜곡효과들을 구현하는 방법을 보았다. 그래서 지수형 배열들 혹은 극좌표계를 왜곡하는데 사용된 함수들을 변화시켜서 여러 가지 다른 효과들을 달성할 수 있다.

어안렌즈효과(hisheye)

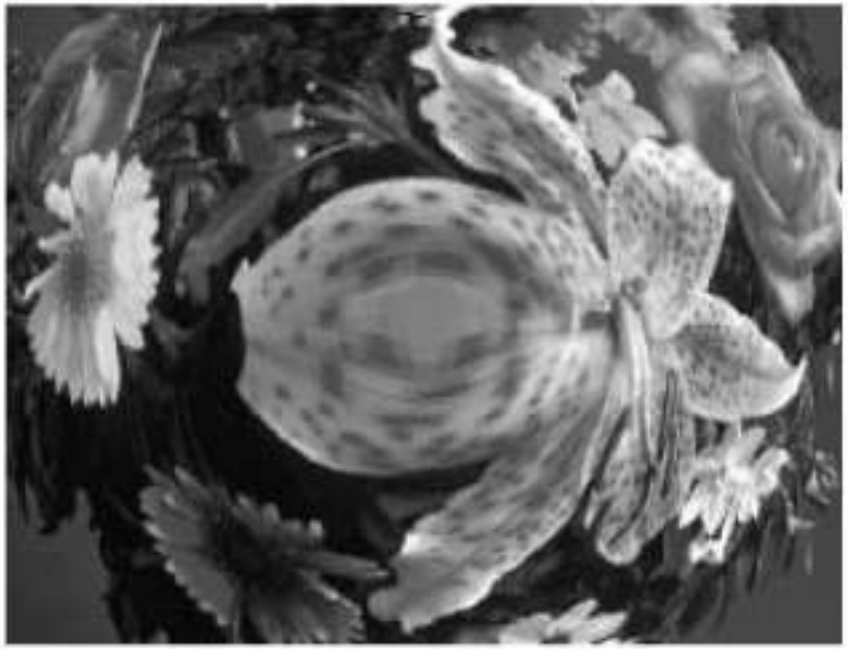
어안렌즈효과는 매우 쉽게 극좌표로서 얻을 수 있다. 여기서 적절한 명령은 아래와 같다.

```
>> R=max(r(:));
>> r=r.^2/R;
```

이들을 전후관계로 두기 위해서 우리는 꽃의 매트릭스 fg를 이용하기 위해 전체 명령의 순서를 아래와 같이 리스트한다.

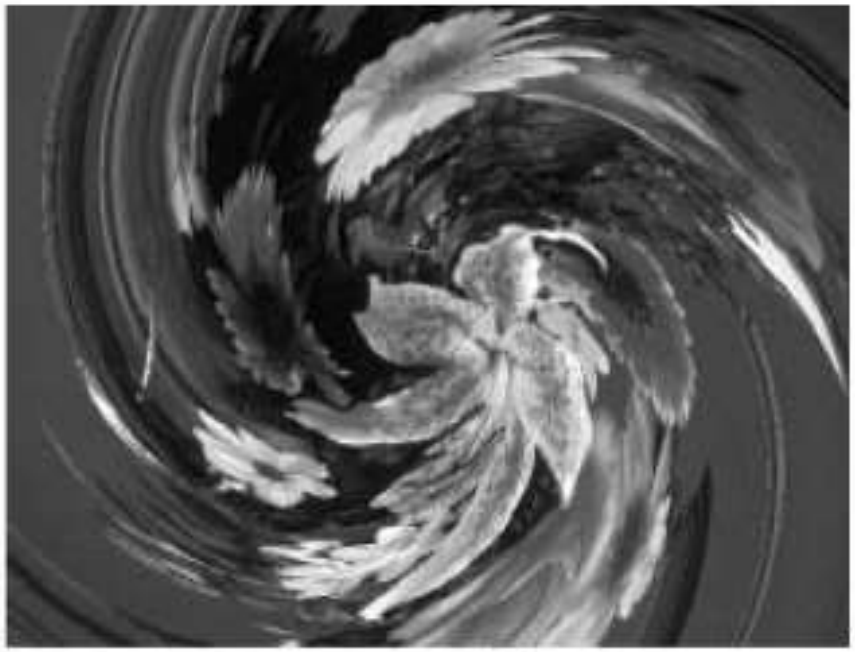
```
>> [rows,cols]=size(fg);
>> ox=ceil((rows+1)/2);
>> oy=ceil((cols+1)/2);
>> [y,x]=meshgrid([1:cols]-oy,[1:rows]-ox);
>> r=sqrt(x.^2+y.^2);
>> theta=atan2(y,x);
>> R=max(r(:));                    % Here is where we implement the fisheye effect
>> s=r.^2/R;
>> x2=round(s.*cos(theta))+ox; % Now we write out to the indexing arrays...
>> y2=round(s.*sin(theta))+oy;
>> x2(find(x2<1))=1;               % ...and ensure that their bounds are correct
>> x2(find(x2>rows))=rows;
>> y2(find(y2<1))=1;
>> y2(find(y2>cols))=cols;
>> fisheye=fg(sub2ind([rows cols],x2,y2));  % Create the new image...
>> imshow(fisheye)                          % ...and view it
```

그림 16.4에 이 결과를 보였다.

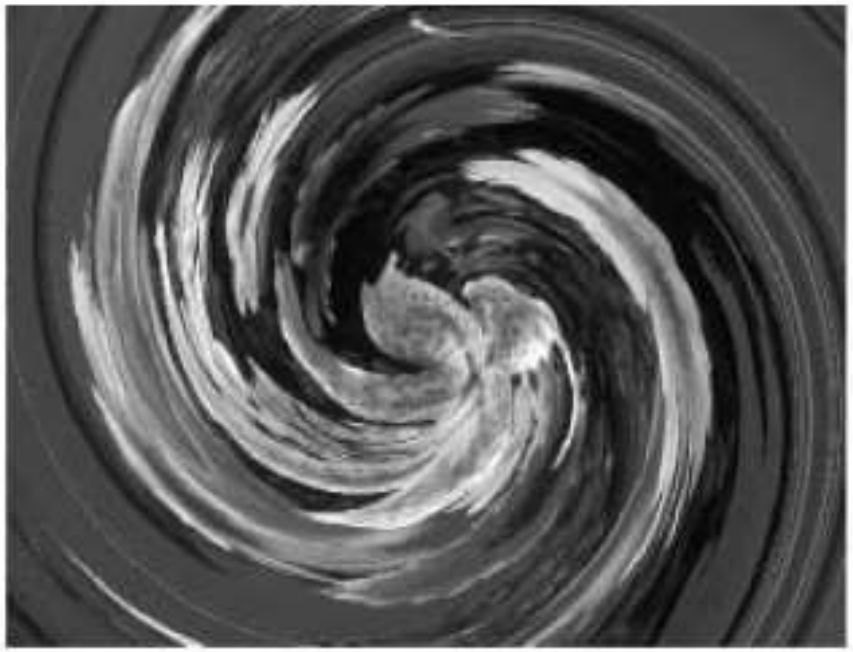


fisheye

A twirl with K=200



A twirl with K=100



A twirl with K=50

그림 16.4 어안(fisheye)와 비틀림(twirls)

비틀림(twirl)

커피의 컵 위에서 스푼으로 휘젓는 것과 같이 비틀림 혹은 소용돌이(swirl) 효과도 매우 쉽게 할 수 있다. 이것도 극좌표 효과이다. 비틀림 효과는 아래 식으로 얻어진다.

$$s(i,j) = \theta(i,j) + r(i,j)/K,$$

여기서 K는 비틀림의 양을 조정하기 위해 변화될 수 있다. 하나의 작은 값은 더욱 비틀리게 하고, 큰 값은 적게 비틀린다. 그래서 배열들 r, θ 및 K가 주어지면 아래의 명령으로 처리할 수 있다.

```
>> phi=theta+(r/K);
>> x2=round(r.*cos(phi))+ox;
>> y2=round(r.*sin(phi))+oy;
```

그림 16.4는 다른 K의 값을 가진 비틀림 처리의 결과를 나타내었다.

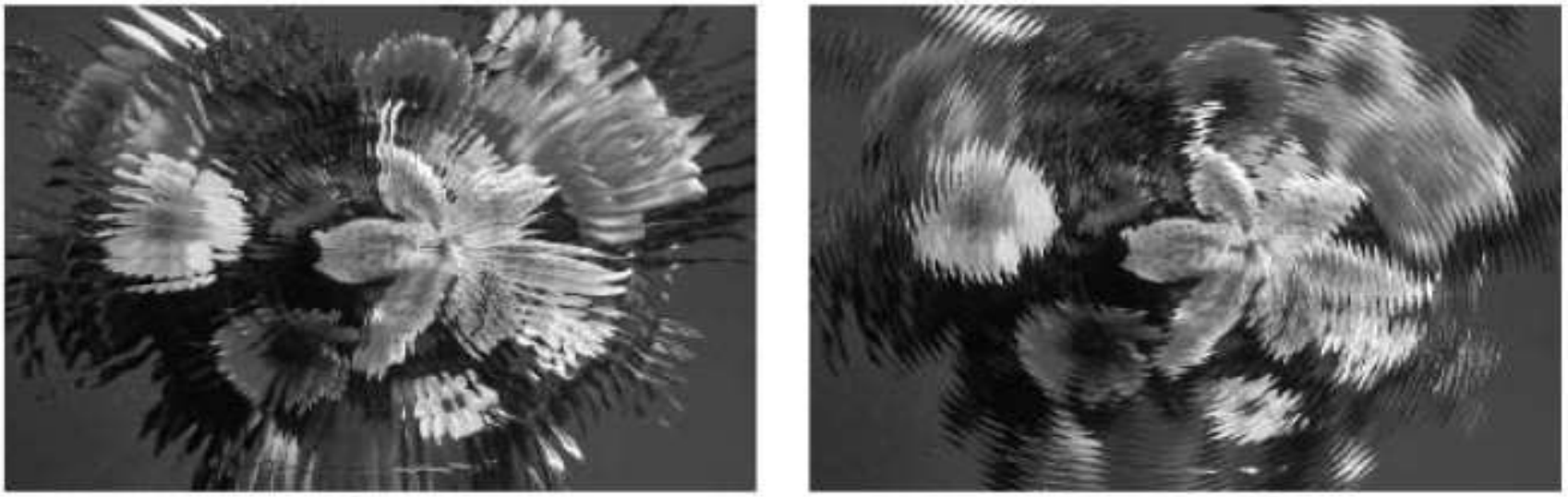
지터(jitter)

지터효과는 이미 설명한 것보다 더 쉽게 구현된다. 이것은 방사형 욕실 유리효과와 비슷하지만, 리플효과를 얻기 위해 r 매트릭스를 조작하는 대신에 우리는 θ 매트릭스를 아래와 같이 조작한다.

```
>> phi=theta+mod(theta,0.1396)-0.0698;
```

0.1396 =8 도, 0.0698= 4도

여기서 0.316 = $8\pi/180$, 및 0.0698 = $4\pi/180$ 이다. 그 결과는 그림 16.5에 보였다.



Jitter Circular slice

그림 16.5 지터와 원형 슬라이스

원형 슬라이스

이것은 각도를 조작하여 얻어지는 또 하나의 효과이다. 이것의 구현은 아래와 같이 지터효과와 비슷하다.

```
>> phi=theta+mod(r,6)*pi/180
```

여기서 라디안으로 변환하기 위해 여분의 $\pi/180$ 계수를 포함한다. 이 결과는 리플 모양을 가지며, 그림 16.5에 보였다.

사각형 슬라이스

사각형 슬라이스는 직각 효과이다. 이것은 영상을 작은 사각형으로 잘라서 처리하고, 이들을 약간 혼란시키게 한다. 이것은 부호함수를 사용하고 아래와 같이 정의한다.

$$\mathbf{sign}(x) = \begin{cases} 1 & \text{if } x > 0, \\ 0 & \text{if } x = 0, \\ -1 & \text{if } x < 0. \end{cases}$$

이것을 아래와 같이 시작한다.

```
>> [y,x]=meshgrid(1:cols,1:rows);
```

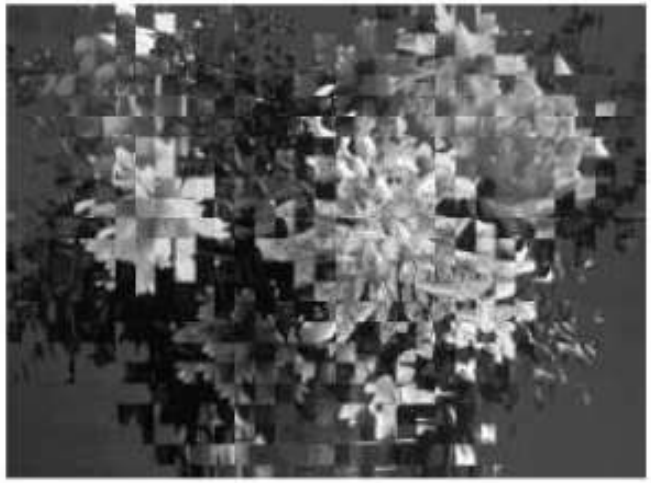
그리고 아래와 같이 슬라이스 왜곡을 구현한다.

```
>> K=8;
>> Q=10;
>> x2=round(x+K*sign(cos(y/Q)));
>> y2=round(y+K*sign(cos(x/Q)));
```

여기서 K와 Q는 각각 사각형의 사이즈와 혼란의 양에 영향을 준다. K의 큰 값은 매우 혼란스러운 영상을 만들고, Q의 큰 값은 큰 사각형을 만든다. rounding처리 후에, 경계를 지정하여, 새로운 영상으로 출력한다. 이 결과는 그림 16.6에 보였다.

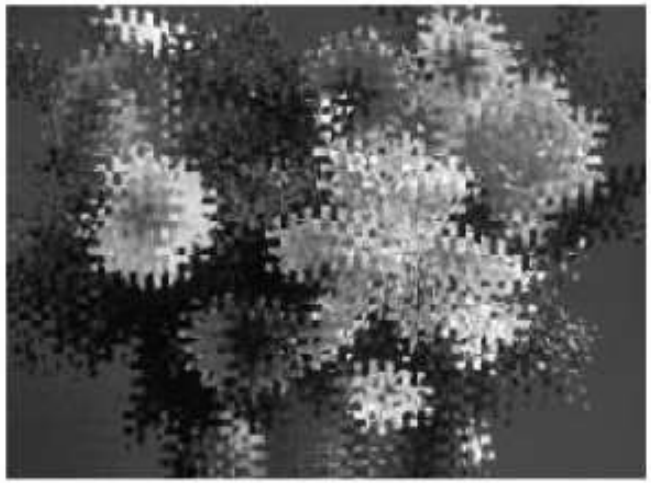
```
f=imread('flowers.tif');
fr=f(:,:,1);
fg=f(:,:,2);
fb=f(:,:,3);
[rows cols]=size(fg);
ox=ceil((rows+1)/2);
oy=ceil((cols+1)/2);
[y,x]=meshgrid([1:cols]-oy, [1:rows]-ox);
K=input('K값을 입력하시오:');
Q=input('Q값을 입력하시오:');
x2=round(x+K*sign(cos(y/Q)));
y2=round(y+K*sign(cos(x/Q)));
xx=round(x2)+ox;
yy=round(y2)+oy;
xx(find(xx>rows))=rows;
xx(find(xx <1))=1;
yy(find(yy>cols))=cols;
yy(find(yy <1))=1;
 ffr=fr(sub2ind([rows,cols],xx,yy));
ffg=fg(sub2ind([rows,cols],xx,yy));
ffb=fb(sub2ind([rows,cols],xx,yy));
imshow(cat(3,ffr,ffg,ffb));
```

K=8;Q=10



K=10;Q=5

K=20;Q=10



K=10;Q=5

그림 16.6 사각형 슬라이스 효과

퍼지효과

이 효과는 그 이웃화소들로부터 불규칙하게 선택된 화소로서 각 화소를 치환하여 얻어진다. 이렇게 하기 위해 지수(index)매트릭스 x와 y에 불규칙한 값들을 단순히 더한다. 만일 7X7 이웃으로부터 값들을 선택하면, -3, -2, -1, 0, 1, 2, 3으로부터 선택된 정수 값들로 지수들을 불규칙하게 처리한다. 아래와 같이 이것을 달성할 수 있다.

```
>> x2=x+floor(7*rand(rows,cols)-3);
>> y2=y+floor(7*rand(rows,cols)-3);
```

5×5 7×7

그림 16.7 다른 이웃화소들을 이용한 퍼지효과