소프트웨어 개발백서

프로젝트명 : 디더링
제작자 : 노지홍
작성일 : 2020.04.09. ~ 2020.05.20 (42일)

1. 작업 개요

Floyd Steinberg 디더링 알고리즘의 한계점을 해결하기 위한 다른 디더링 알고리즘 연구.

2. 연구한 알고리즘

1) Floyd Steinberg 알고리즘에서 가중치를 2방향으로 조정.
2) 기존에 Floyd Steinberg 알고리즘이 에러 값을 확산 시키는 알고리즘을 거꾸로 적용하여 에러 값을 모은 후에 현재 픽셀 값을 빼는 방법.
3) Direct Binary Search

3. DBS 알고리즘 설명

1) DBS 알고리즘의 과정
- 정규분포 난수 발생 알고리즘으로 하프톤 이미지 생성.
- 가우시안 필터 생성. (시그마 값 = 1.2, 필터 사이즈 = 7 X 7)
- 생성한 가우시안 필터로 자기상관 행렬(CPP) 생성.
- 자기상관 행렬(CPP)와 에러 값을 이용해 상호상관 행렬(CEP) 생성.
- 하프톤 이미지와 원본 이미지의 각 픽셀에 대해 에러 값 계산.
- DBS 알고리즘 반복 수행.

2) DBS 알고리즘의 핵심 아이디어
- 현재 픽셀의 주변 픽셀을 대상으로 Swapping과 Toggling(자기 자신을 토글)을 하면서 에러 값을 계산하여 최소화 하는 방법.
- 에러 값이 0보다 작으면 오차가 생긴다는 결과를 이용해 모든 픽셀에서 0보다 작은 경우가 안 나올 때까지 알고리즘을 반복 적용 시킨다.

4. 결과

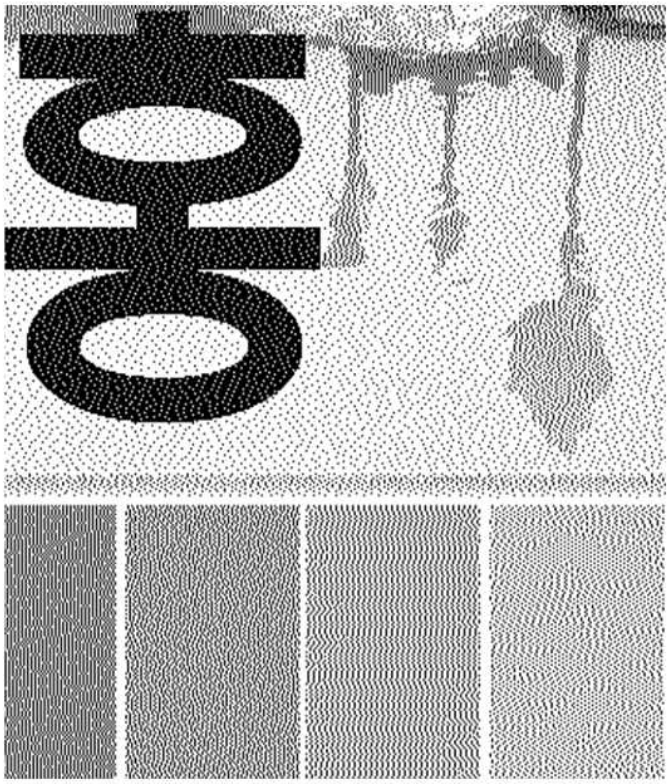
1) 속도

속도 비교(초)			
이미지 사이즈	Floyd Steinberg	DBS	
512 X 512	CPU : 0.0035	CPU : 8.351	GPU : 0.2138
2048 X 2560	CPU : 0.056	CPU : 113.3	GPU : 3.158
4096 X 3072	CPU : 0.497	CPU : 346.6	GPU : 7.248

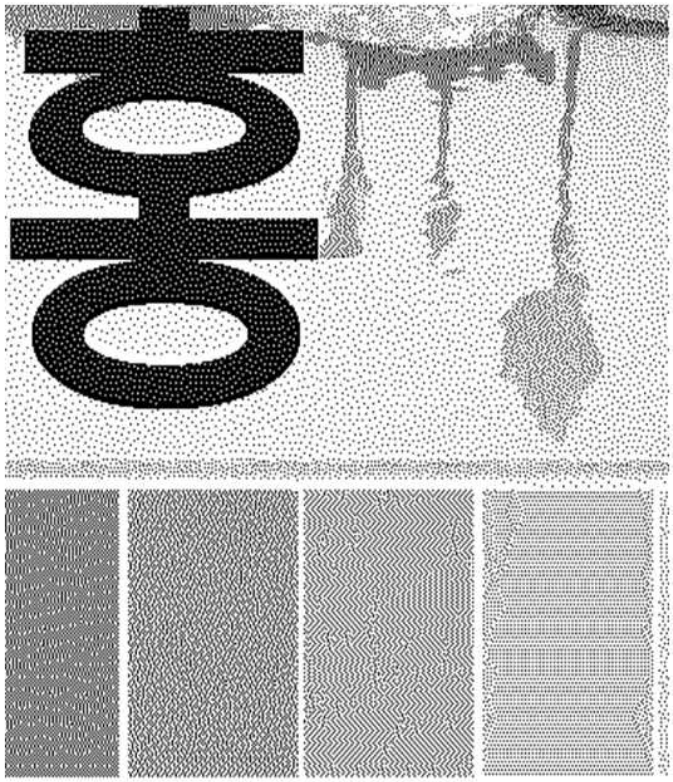
2) 사진

- 가중치를 2방향으로 조정

가중치 2개 7:3 비율 (양방향)

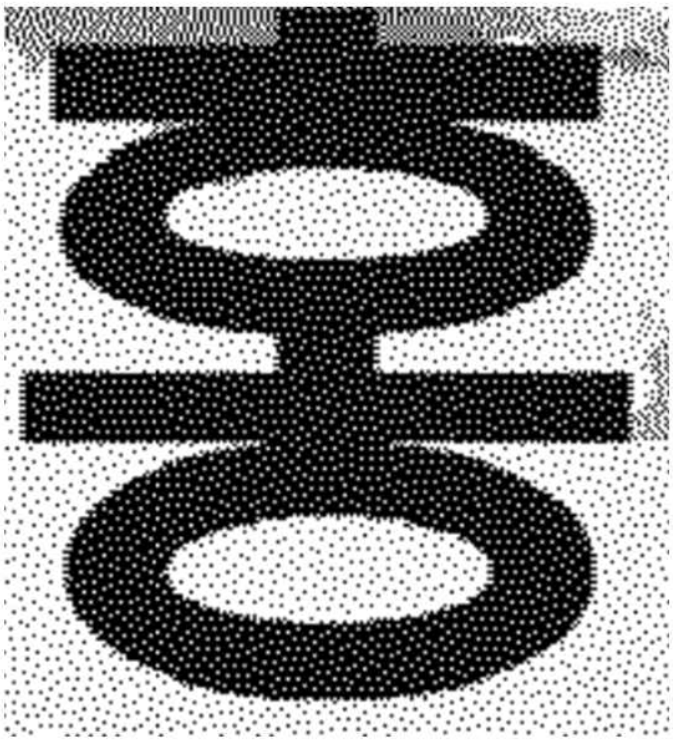


Floyd Steinberg (양방향)

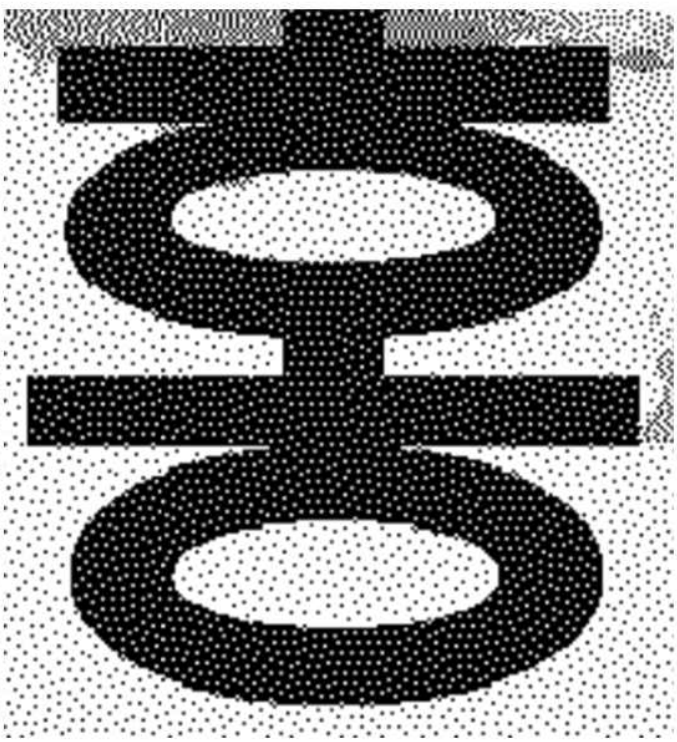


- 에러 값을 모은 후에 픽셀 값을 뺌

픽셀 값을 그냥 뺀 결과



픽셀 값에 2/5를 곱하고 뺀 결과

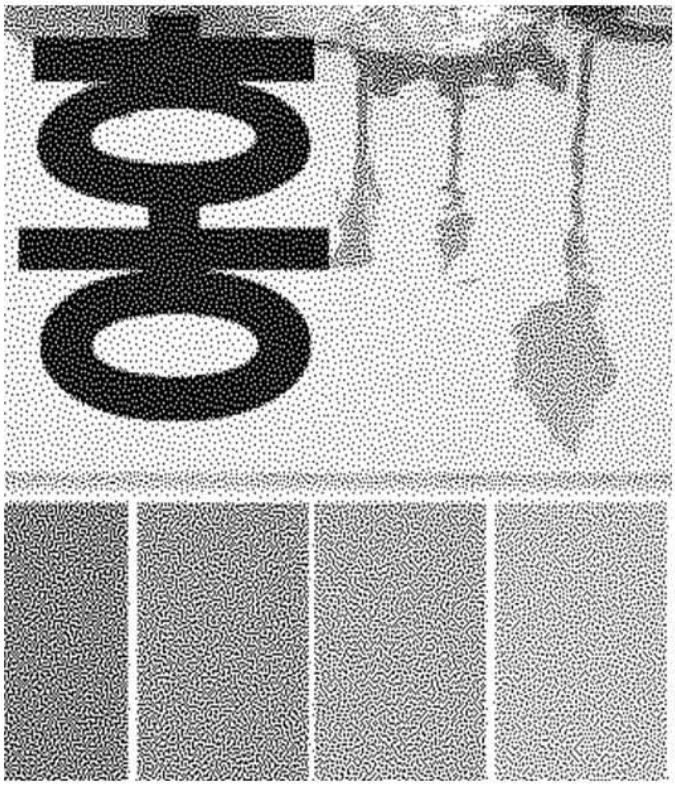


- 에러 값을 모으는 방법 코드

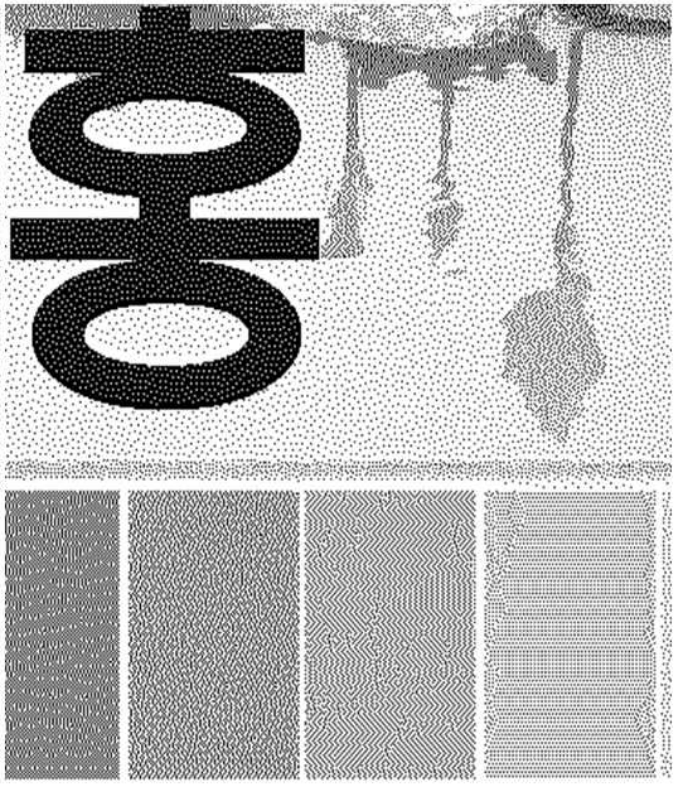
```
// 각 가중치 7, 3, 5, 1에 대한 에러 값
int err7, err3, err5, err1;

for (int y = 1; y < bph - 1; y++)
{
    if (y % 2 == 1)
    {
        for (int x = 1; x < bpl - 1; x++)
        {
            // 각 픽셀의 7, 3, 5, 1 가중치에 대한 에러 값 계산
            err7 = pixE[y * bpl + x - 1] - pix[y * bpl + x - 1];
            err1 = pixE[(y - 1) * bpl + x + 1] - pix[(y - 1) * bpl + x + 1];
            err5 = pixE[(y - 1) * bpl + x] - pix[(y - 1) * bpl + x];
            err3 = pixE[(y - 1) * bpl + x - 1] - pix[(y - 1) * bpl + x - 1];

            // pixE : 픽셀의 에러 값, 에러 값을 모아준다.
            pixE[y * bpl + x] += err7 * 7 / 16;
            pixE[y * bpl + x] += err1 * 1 / 16;
            pixE[y * bpl + x] += err5 * 5 / 16;
            pixE[y * bpl + x] += err3 * 3 / 16;
            pixE[y * bpl + x] += pix[y * bpl + x];

            // 점이 찍히는 순간
            // v1 / vsum 은 가중치 값으로 2 / 5 값이 결과가 가장 괜찮았음.
            pix[y * bpl + x] = (pixE[y * bpl + x] - (pix[y * bpl + x] * v1 / vsum)) / 128 * 255;
        }
    }
    else
    {
        for (int x = bpl - 2; x >= 1; x--)
        {
            // 각 픽셀의 7, 3, 5, 1 가중치에 대한 에러 값 계산
            err7 = pixE[y * bpl + x + 1] - pix[y * bpl + x + 1];
            err1 = pixE[(y - 1) * bpl + x - 1] - pix[(y - 1) * bpl + x - 1];
            err5 = pixE[(y - 1) * bpl + x] - pix[(y - 1) * bpl + x];
            err3 = pixE[(y - 1) * bpl + x + 1] - pix[(y - 1) * bpl + x + 1];

            // pixE : 픽셀의 에러 값, 에러 값을 모아준다.
            pixE[y * bpl + x] += err7 * 7 / 16;
            pixE[y * bpl + x] += err1 * 1 / 16;
            pixE[y * bpl + x] += err5 * 5 / 16;
            pixE[y * bpl + x] += err3 * 3 / 16;
            pixE[y * bpl + x] += pix[y * bpl + x];


            // 점이 찍히는 순간
            // v1 / vsum 은 가중치 값으로 2 / 5 값이 결과가 가장 괜찮았음.
            pix[y * bpl + x] = (pixE[y * bpl + x] - (pix[y * bpl + x] * v1 / vsum)) / 128 * 255;
        }
    }
}
```

- DBS

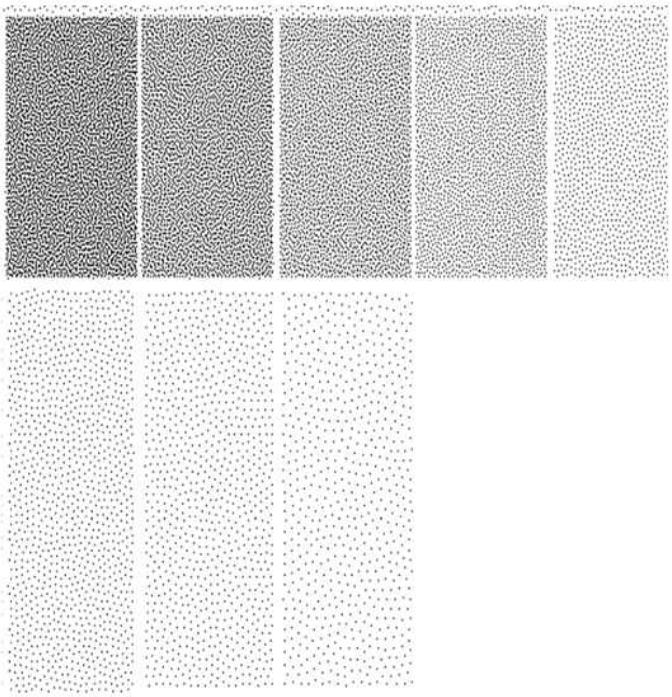
DBS 결과



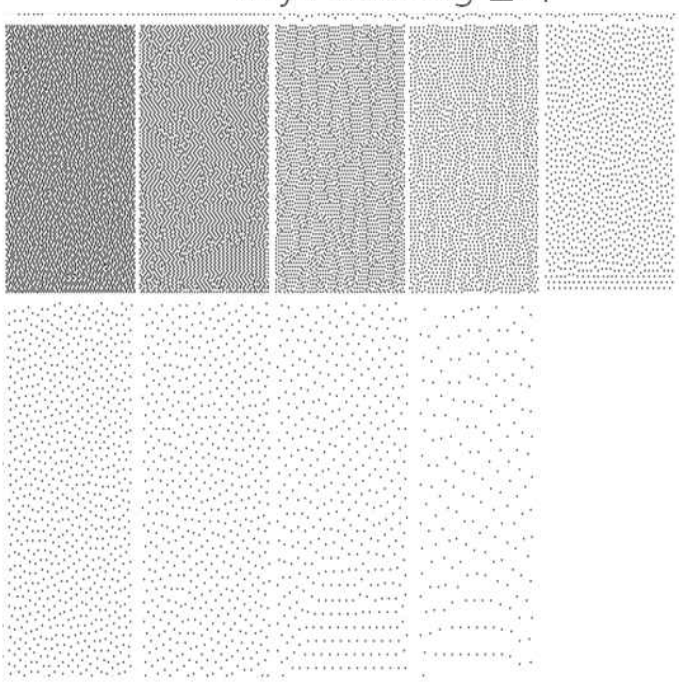
Floyd Steinberg 결과



DBS 결과



Floyd Steinberg 결과



5. 개발에 어려웠던 점

1) Direct Binary Search의 한계점인 Clipping현상을 해결하는 Clipping Free DBS라는 알고리즘을 분석하고 개발하는 과정 중에 임계 값 행렬을 구현하는 부분에서 막힘.

6. 미처리 사항

1) Direct Binary Search의 속도가 매우느림.
2) Direct Binary Search의 Clipping 현상 해결.
3) 이미지의 경계선 부근에서의 살짝 번짐 현상. (가우시안 필터 영향)
4) 매우 큰 이미지를 대상으로 디더링을 할 때 메모리 최적화를 해야 함. (라인단위로 읽어들이기)
5) Direct Binary Search를 GPGPU로 병렬처리 할 수 있는 가능성.

7. DBS와 Clipping현상의 관계

- 가우시안 필터에서 시그마 값이 1.2일때 결과가 가장 좋다는 연구결과가 나옴.
- 가우시안 필터 size 는 (7 X 7) 일 때 가장 좋았으며 그 이상의 size로 해도 차이가 거의 안남.
- Clipping 현상이 나타나는 경계 값 D = 5/255 와 6/255 사이임. D가 감소할수록 시그마 값이 증가함.

8. 첨부 자료 내용

1) DBS 전체코드 텍스트파일
2) DBS 논문 PDF 파일들