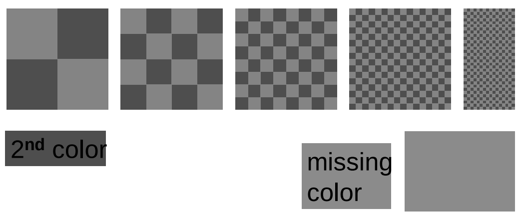
Dithering and Raster Graphics

Special Aspects of Raster Images

Dithering

missing colors are simulated by mixing existing colors



Dithering Methods (Digital Halftoning)

- **threshold dithering**
  - ordered dither
  - stochastic dither
  - dot diffusion
  - ....
- **error diffusion dithering (Floyd-Steinberg)**

Dithering in Printing Industry

- **newspapers**

black ink on light paper, rasterization of the images enables also grey levels, equal point density everywhere, variable size

- **color printing**

every primary color is rasterized separately, different printing angles ensure unbiased results

Threshold Dithering

every pixel is compared to a threshold **t**:

$$p < t \Rightarrow a$$
$$p > t \Rightarrow b$$

t can be:

- equal everywhere (e.g. (b–a)/2, arbitrary value, mean value, median, ...)
- location dependent (defined locally or globally)

Constant Threshold Dithering

sample image

1	7	6	5
1	6	5	4
1	5	4	3
1	4	2	1

threshold values

4.5	4.5	4.5	4.5
4.5	4.5	4.5	4.5
4.5	4.5	4.5	4.5
4.5	4.5	4.5	4.5

result

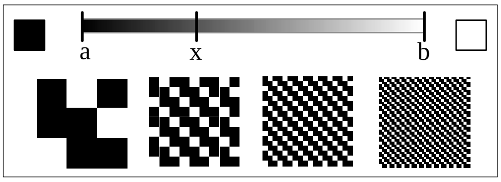
0	9	9	9
0	9	9	0
0	9	0	0
0	0	0	0

(values between 0 and 9)
corresponds to rounding

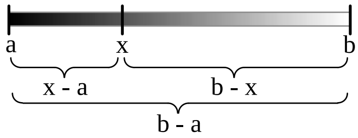
Principle of Dithering

available values a, b

missing value x between a and b shall be simulated by mixing a-pixels and b-pixels



Principle of Dithering (2)



to produce color value x there have to be:

$$100 * \frac{x-a}{b-a} \ \% \text{ b-pixels}$$

$$100 * \frac{b-x}{b-a} \ \% \text{ a-pixels}$$

Dithering a Uniform Area

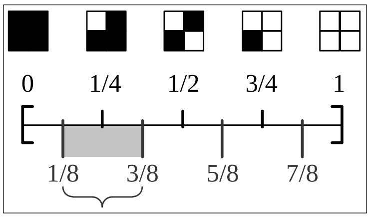
for a uniform area regular application of

this pattern

will produce

this grey tone

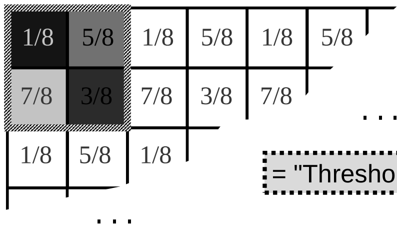
interval borders



all grey levels in this interval will be mapped to 1/4

Dithering a Uniform Area (2)

This can be done by using a different threshold for every pixel (using the interval borders)



= "Threshold Matrix"

Threshold Matrix

distances between interval borders are equal, therefore it suffices to define the sequence of the pixel values in the matrix:

instead of

1/8	5/8
7/8	3/8

only

0	2
3	1

i.e. for an nxn matrix: values $\in [0, n^2-1]$

value k corresponds to threshold value: $\frac{2k+1}{2n^2}$

Dither Matrix Example

dither matrix → threshold matrix

7	2	3
0	4	8
5	6	1

$\frac{15}{18}$	$\frac{5}{18}$	$\frac{7}{18}$
$\frac{1}{18}$	$\frac{9}{18}$	$\frac{17}{18}$
$\frac{11}{18}$	$\frac{13}{18}$	$\frac{3}{18}$

value k corresponds to threshold value: $\frac{2k+1}{2n^2}$

Threshold Matrix Dithering Example

sample image

1	7	6	5
1	6	5	4
1	5	4	3
1	4	2	1

threshold values

1.1	5.6	1.1	5.6
7.9	3.4	7.9	3.4
1.1	5.6	1.1	5.6
7.9	3.4	7.9	3.4

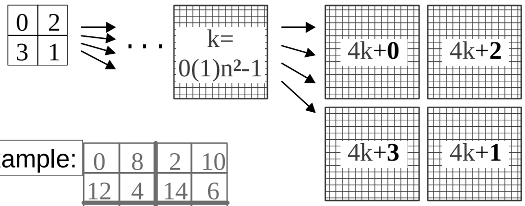
result

0	9	9	0
0	9	0	9
0	0	9	0
0	9	0	0

(values between 0 and 9)

Generation of Threshold Matrices (1)

recursive method: 4 copies of smaller matrices



example:

0	8	2	10
12	4	14	6
3	11	1	9
15	7	13	5

Generation of Threshold Matrices (2)

direct method: use of magic squares

example

0	14	3	13
11	5	8	6
12	2	15	1
7	9	4	10

magic squares produce
fewer diagonal stripes

Dithering between Grey Levels

threshold values have to lie between a and b:

$$k \Rightarrow a + \frac{2k+1}{2n^2} \cdot (b-a)$$

calculation is done separately for every pixel
(not once for a dithering matrix)

Grey Level Dithering Example

4 grey values are available: 0, 3, 6, 9

dither matrix:

0	2
3	1

sample image

1	7	6	5
1	6	5	4
1	5	4	3
1	4	2	1

threshold values

0.4	7.9	3.4	4.9
2.6	3.4	5.6	4.1
0.4	4.9	3.4	1.9
2.6	4.1	2.6	1.1

result

3	6	6	6
0	6	3	3
3	6	6	3
0	3	0	0

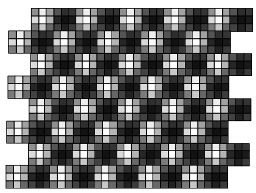
(values between 0 and 9)

Dot Diffusion Dithering

ordering of the threshold values generates
larger dot areas

example:

11	16	12	8	4	5
15	17	13	3	0	1
10	14	9	7	2	6



simulates traditional printing techniques
for high resolution devices

Stochastic Dithering?

use of random numbers as threshold values

- expectation value of total error = 0
- no regular artificial patterns possible

unfortunately: **very bad results!**
(due to bad distribution of random numbers)

Forced Random Matrix Dithering

improved "random" matrices ⇒ very good results

method: insert threshold values one by one into matrix, always use the position farthest away from all previous points

repulsive force field: $f(r) = e^{-\left(\frac{r}{s}\right)^p}$

precalculate large threshold matrices: 300x300
very good results!

Error Diffusion Dithering
(Floyd-Steinberg)

the rounding error of every pixel is propagated to neighbor pixels and compensated there

variations: which neighbor pixels are affected?

Simplest Error Diffusion Dithering

pixel line						...
correct value	k_1	k_2	k_3	k_4	k_5	...
rounded value	r_1	r_2	r_3	r_4	r_5	...

$r_1 := \text{round}(k_1)$ $\quad \text{error}_1 := r_1 - k_1$
$r_2 := \text{round}(k_2 - \text{error}_1)$
$\text{error}_2 := r_2 - (k_2 - \text{error}_1)$
...
$r_i := \text{round}(k_i - \text{error}_{i-1})$
$\text{error}_i := r_i - (k_i - \text{error}_{i-1})$

Error Diffusion Dithering Example

example: the values 0, 3, 6, 9 are available

k:	1	1	1	2	3	4	7	1	5	...
r:	0	3	0	3	3	3	6	3	3	...
f:	-1	1	0	1	1	0	-1	1	-1	...

$r_i := \text{round}(k_i - f_{i-1})$ $\qquad f_i := r_i - k_i + f_{i-1}$

Diffusion Direction Variations

to gain better results, the error is distributed to *several* neighbors (with weights)

sum of all weights = 1

often used weighting distributions:

$\frac{1}{2}$ x

x	1
1	

$\frac{1}{16}$ x

	x	7
3	5	1

$\frac{1}{48}$ x

		x	7	5
3	5	7	5	3
1	3	5	3	1

$\frac{1}{42}$ x

		x	8	4
2	4	8	4	2
1	2	4	2	1

Error Diffusion Dithering Example

example: the values 0, 3, 6, 9 are available

error distribution: $\frac{1}{2}$ x

x	1
1	

sample image

1	7	6	5
1	6	5	4
1	5	4	3
1	4	2	1

error

-1	1.5	.75	1.37
1.5	1.5	-.87	-.75
-.25	-1.37	.87	.06
-1.12	.75	-1.12	1.47

result

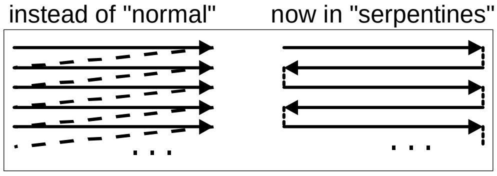
0	9	6	6
3	6	3	3
0	3	6	3
0	6	0	3

(values between 0 and 9)

Serpentine Method

artificial stripes can be reduced drastically by processing the scanlines in serpentine order

instead of "normal" now in "serpentines"



no additional memory necessary

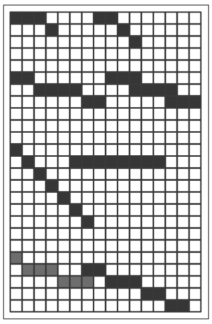
Raster Conversion

converting primitives (lines etc.) into pixels

very frequent operations, therefore necessary:

- efficiency
- possibility to implement in hardware

Raster Conversion of Lines



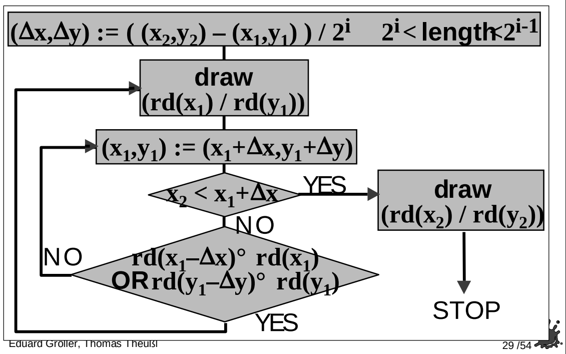
lines should appear straight

lines should appear uniformly bright

lightness should be independent of direction

endpoints should be "exact"

Symm. DDA for Line $(x_1,y_1) \rightarrow (x_2,y_2)$



Simple DDA

symmetric DDA produces lines of variable breadth, but if Δx and Δy are chosen such that

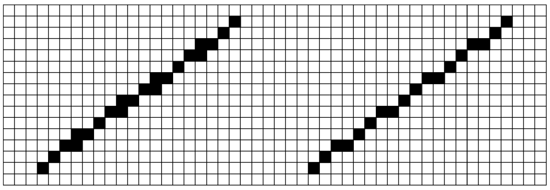
$$\max(|\Delta x|, |\Delta y|) = 1$$

$\Rightarrow$ only 1 pixel per unit in the longer direction

problem: requires a real division

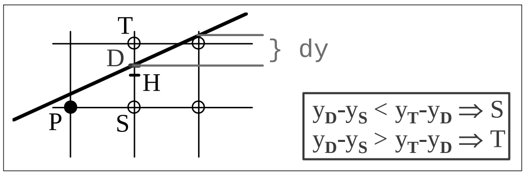
solution: **Bresenham Algorithm**
image equal to simple DDA, but no division

Comparision Symmetric - Simple DDA



symmetric DDA   simple DDA

Raster Conversion of Lines
(Bresenham's Line Algorithm)



only for lines with angle 45º
other lines by mirroring/rotating with 90º/180º...

Some Bresenham Mathematics

$$y_D - y_S < y_T - y_D \Leftrightarrow y_D - y_H < 0$$
$$y_D - y_S > y_T - y_D \Leftrightarrow y_D - y_H > 0$$

if $y_D - y_H < 0 \Rightarrow y_S, y_T$ do not change,
if $y_D - y_H > 0 \Rightarrow y_S := y_S+1,\ y_T := y_T+1$

in every case $y_D := y_D + dy$
and (x_S/y_S) are drawn

let $d = y_D - y_H$ decision variable

Bresenham's Line Algorithm (1)

```
ys:=y1;
d:=-0.5;    {d=yD–yH}
dy:=(y2-y1)/(x2-x1);
FOR xs:=x1 TO x2 DO
BEGIN SetPixel(xs,ys);
  d:=d+dy;
  IF d>0
  THEN BEGIN ys:=ys+1;
    d:=d-1  {because yH:=yH+1}
  END
END
```

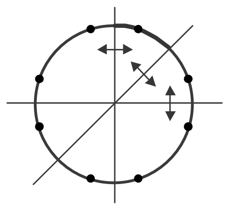

Bresenham's Line Algorithm (2)

```
ys:=y1;
e:=-(x2-x1)div 2;   {d*(x2-x1)}
de:=(y2-y1);        {dy*(x2-x1)}
FOR xs:=x1 TO x2 DO
BEGIN SetPixel(xs,ys);
  e:=e+de;
  IF e>0
  THEN BEGIN ys:=ys+1;
    e:=e-(x2-x1)
  END
END
```

only integers!
only addition, subtraction, shift!

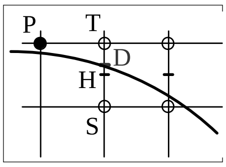
Raster Conversion of Circles
(Bresenham's Circle Algorithm)

utilize threefold symmetry!



only one eighth has to be calculated

Bresenham's Circle Algorithm



criterion:

$y_D - y_S > y_T - y_D \Rightarrow T$
$y_D - y_S < y_T - y_D \Rightarrow S$

(only for 2nd octant)

Bresenham's Circle Algorithm

$y_D - y_S > y_T - y_D \Leftrightarrow$ H inside the circle

that is if $x_H^2 + y_H^2 < r^2$
or if $f(x_H, y_H) = x_H^2 + y_H^2 - r^2 < 0$

$f(x_H, y_H) < 0 \Rightarrow T$
$f(x_H, y_H) > 0 \Rightarrow S$

d = "decision variable"
i.e. $d_{old} = d = f(x_p+1, y_p - {}^1/_2)$,
then d_{new} can be calculated as follows:

Bresenham's Circle Algorithm

if $d_{old} < 0$ then $H_{new} = H_{old} + (1,0)$,
ie.

$d_{new} = f(x_p+2, y_p - {}^1/_2) = (x_p+2)^2 + (y_p - {}^1/_2)^2 - r^2$

$d_{new} = d_{old} + (2x_p+3)$

if $d_{old} > 0$ then $H_{new} = H_{old} + (1,-1)$,
ie.

$d_{new} = f(x_p+2, y_p - {}^3/_2) = (x_p+2)^2 + (y_p - {}^3/_2)^2 - r^2$

$d_{new} = d_{old} + (2x_p - 2y_p + 5)$

Bresenham's Circle Algorithm

```
PROCEDURE Circle (r:INTEGER);
VAR x,y: INTEGER; d: REAL;
BEGIN ... {initialize x,y,d}
  REPEAT
    Draw_8(x,y)
    IF d < 0
    THEN d:=d+2*x+3
    ELSE
    BEGIN d:=d+2*(x-y)+5;
      y:=y-1
    END;
    x:=x+1;
  UNTIL y < x
END;
```

Bresenham's Circle Algorithm

initialization: restriction to integer radii

x = 0 and **y = r**

$H = (1, r - {}^1/_2)$
$\Rightarrow d = f(H) = 1 + (r^2 - r + {}^1/_4) - r^2$
$= {}^5/_4 - r$

Bresenham's Circle Algorithm

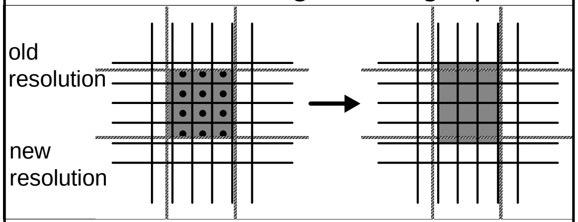
```
PROCEDURE Circle (r:INTEGER);
VAR x,y: INTEGER; d: REAL;
BEGIN x:=0; y:=r; d:=5/4 - r;
  REPEAT
    Draw_8(x,y)
    IF d < 0
    THEN d:=d+2*x+3
    ELSE
    BEGIN d:=d+2*(x-y)+5;
      y:=y-1
    END;
    x:=x+1;
  UNTIL y < x
END;
```

Raster Transformations

how to apply geometrical transformations to raster images?

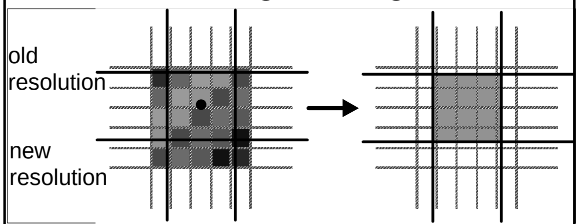
- **translation:** trivial
- **scaling:** resampling necessary
- **shearing:** by line conversion
- **rotation:** partition in three shearings

Raster Scaling: Scaling Up



center point of pixel in new resolution defines its color = **"resampling"**

Raster Scaling: Scaling Down

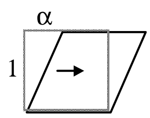


center point of pixel in new resolution defines its color = **"resampling"**

Shearing

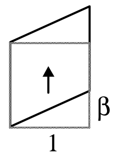
x-shearing

$$(x\ \ y) \cdot \begin{pmatrix} 1 & 0 \\ \alpha & 1 \end{pmatrix} = (x+\alpha y\ \ y)$$

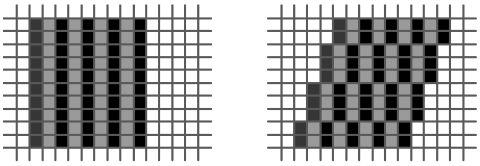


y-shearing

$$(x\ \ y) \cdot \begin{pmatrix} 1 & \beta \\ 0 & 1 \end{pmatrix} = (x\ \ y+\beta x)$$



Raster Shearing



can be seen as the multiple application of a line raster conversion algorithm (e.g. Bresenham)

⇒ no information lost

Raster Rotation

90°-rotation, 180°-rotation, etc. trivial

principle of other rotations:
subdivide rotation in a series of three shears

$$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \alpha & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & \beta \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ \alpha & 1 \end{pmatrix}$$

where $\alpha = -\tan(\theta/2)$
and $\beta = \sin\theta$

Rotation = 3 Shears

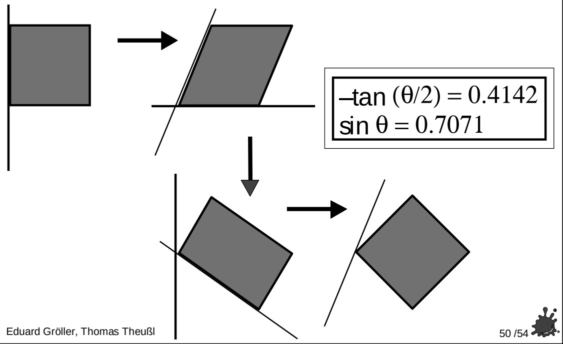
$$\tan(\theta/2) = (1-\cos\theta)/\sin\theta$$

$$\begin{pmatrix} 1 & 0 \\ \frac{\cos\theta-1}{\sin\theta} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & \sin\theta \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ \frac{\cos\theta-1}{\sin\theta} & 1 \end{pmatrix} =$$

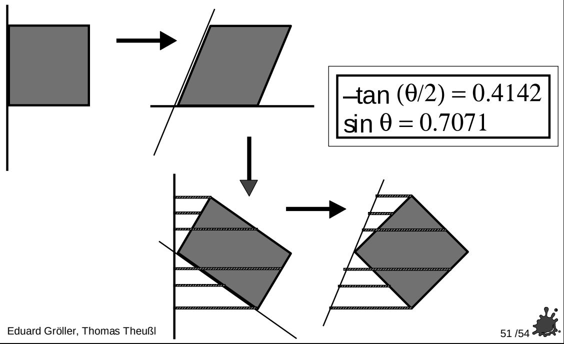
$$\begin{pmatrix} 1 & \sin\theta \\ \frac{\cos\theta-1}{\sin\theta} & \cos\theta \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ \frac{\cos\theta-1}{\sin\theta} & 1 \end{pmatrix} =$$

$$\begin{pmatrix} 1+\cos\theta-1 & \sin\theta \\ \frac{\cos\theta-1+\cos^2\theta-\cos\theta}{\sin\theta} & \cos\theta \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

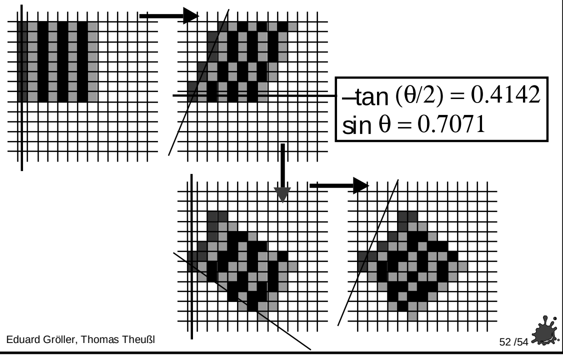
Example: –45° Rotation



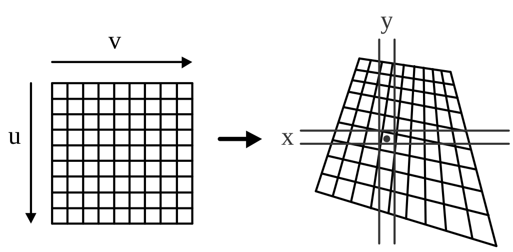
Example: –45° Rotation



Example: –45° Rotation

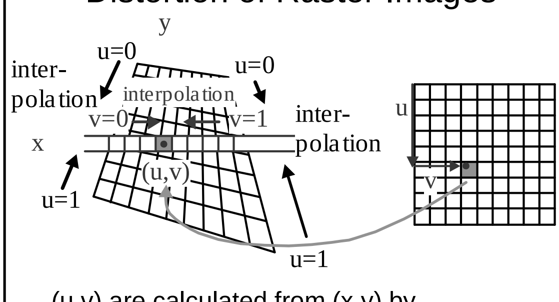


Distortion of Raster Images



we are looking for the (u,v)-coordinates of the center of pixel (x,y)

Distortion of Raster Images



(u,v) are calculated from (x,y) by interpolation from the corner coordinates