§2 Rasterization

2.1 Overview

- Raster displays require the decomposition of all displayed geometric objects into pixels

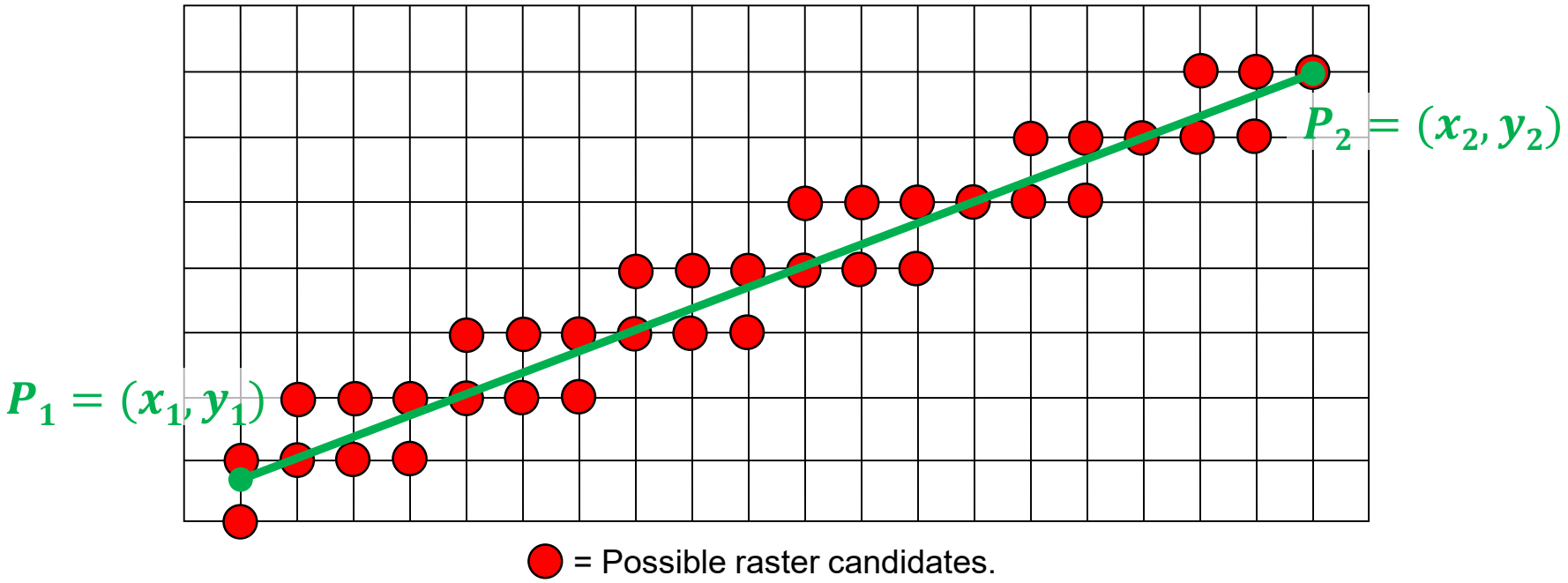
Rasterization

- It is one of the tasks of the image generation unit.

2.2 Rasterization of line segments

Problem

- Display of a line segment on a raster display requires the identification of optimal pixels in the raster/grid of the screen, i.e. appropriate integer rounding.



2.2 Rasterization of line segments

Requirements for the rasterization:

- lines should be straight,
- lines should have constant thickness,
- lines should have uniform brightness,
- brightness should not depend on direction of the line,
- end points should be exact,
- computation of lines should be fast,
- algorithm should be implementable in hardware.

2.2 Rasterization of line segments

Representation of lines

- Given: two points $P_1 = (x_1, y_1),\ \ P_2 = (x_2, y_2)$.
- Wanted: representation of line segment between P_1 and P_2

1. **parametric** $(\Delta x = x_2 - x_1, \Delta y = y_2 - y_1)$:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_1 + t \cdot \Delta x \\ y_1 + t \cdot \Delta y \end{pmatrix}$$

2. **functional** $(m = \Delta y / \Delta x)$:

$$y = m \cdot x + n$$

3. **implicit** $(a = \Delta y, b = -\Delta x)$:

$$F(x, y) = a \cdot x + b \cdot y + \ c = 0$$

2.2 Rasterization of line segments

Naïve algorithm

- Let P_1 and P_2 be given in integer-coordinates.
- Use functional representation: $y = m\,x + n.$
- Naïve algorithm:

  1) Iterate from x_1 to x_2 in pixel-steps.

  2) Compute y-values.

  3) Round.

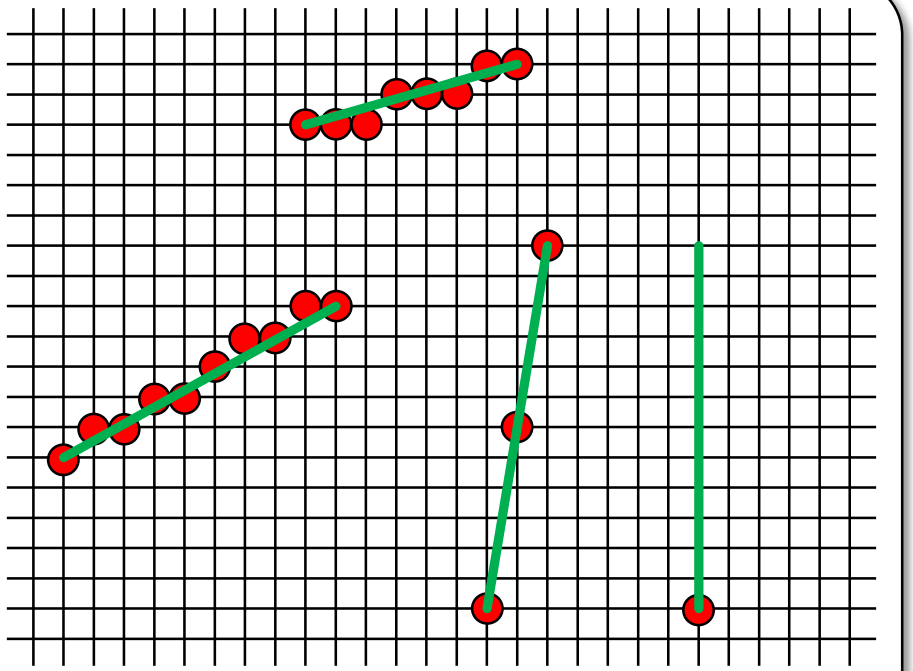
  4) Draw.

```
double m = (y2-y1)/(x2-x1);
for (int x=x1; x<=x2; x++)
{
   double y = m*x+n;
   setPixel(x,round(y));
}
```

2.2 Rasterization of line segments

Problems:

- Floating point values y and m,
- divisions and multiplications,
- rounding,
- vertical lines,
- appearance of lines depends on the slope m.

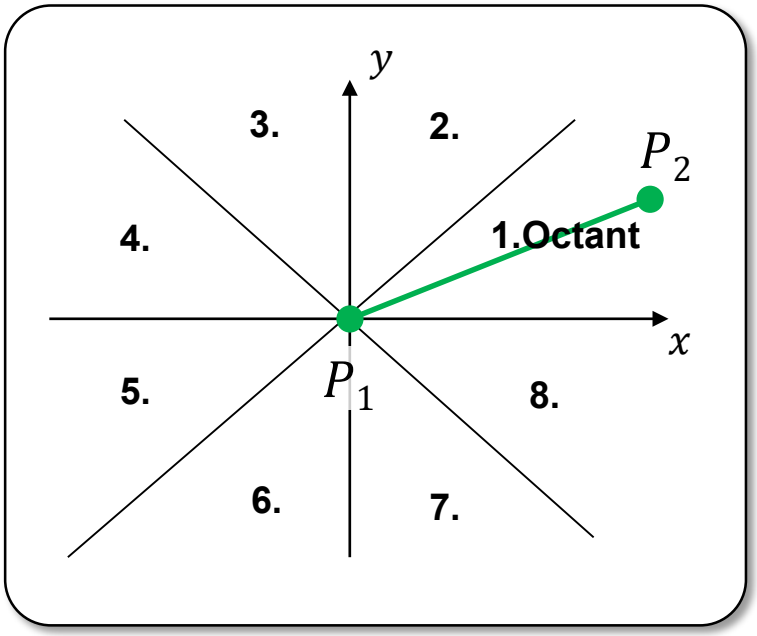


Rasterization using the naïve algorithm.

2.3 Rasterization of lines

Bresenham algorithm for lines

- Also known as Midpoint-Line-Algorithm.
- No divisions, only integer arithmetic.
- Uses the implicit representation: $F(x, y) = a\,x + b\,y + c = 0$ with $a = \Delta y$, $b = -\Delta x$.
- Assumptions:
  - Points P_1 and P_2 are raster points.
  - Line is in the 1st octant, i.e. $0 \leq \text{slope} \leq 1$.

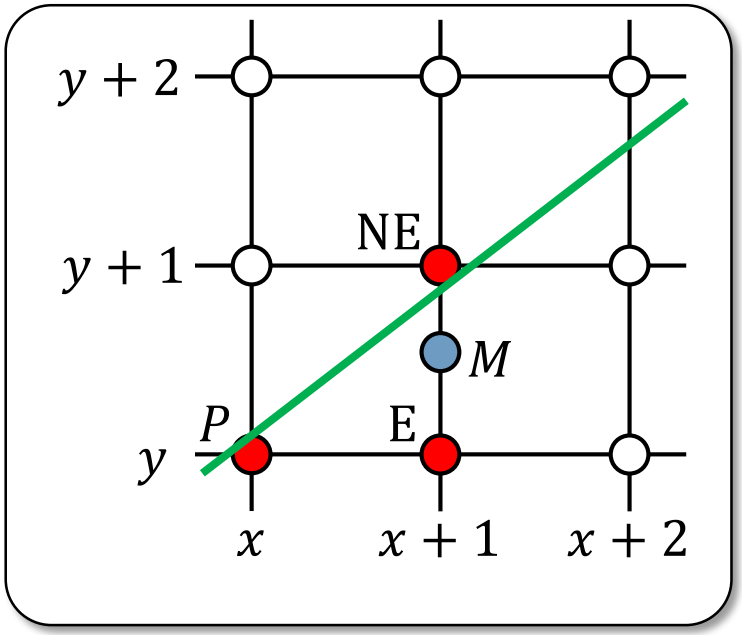


2.3 Rasterization of lines

Idea

- Pixel P has just been plotted.
- Is the next pixel to draw NE or E?
- **Easier:** Is $M = (x + 1, y + 1/2)$ above or below the line?

1. $F(M) > 0$, NE is the next pixel.
2. $F(M) \leq 0$, E is the next pixel.

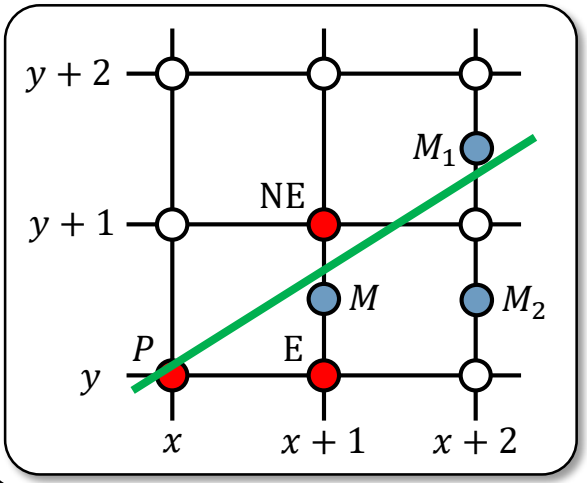


- Choose $d = F(M) = F(x + 1, y + 1/2)$ as decision variable.
- How to compute $d = F(M)$? $\longrightarrow$ **incrementally**
- $F(M) = a(x + 1) + b(y + 1/2) + c$

2.3 Rasterization of lines

Case 1

- NE is the next pixel.
- M_1 is the next midpoint.
- $F(M_1) = a(x+2) + b(y+3/2) + c$
- $F(M_1) = F(M) + a + b$
- $d = d + \Delta y - \Delta x$

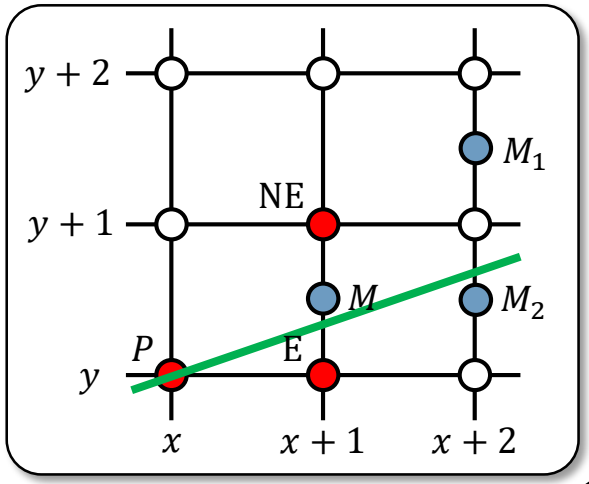


Summary

- $d = d + \begin{cases} \Delta_{NE} \\ \Delta_{E} \end{cases}$
- $\Delta_{NE} = \Delta y - \Delta x$
- $\Delta_{E} = \Delta y$

Case 2

- E is the next pixel.
- M_2 is the next midpoint.
- $F(M_2) = a(x+2) + b(y+1/2) + c$
- $F(M_2) = F(M) + a$
- $d = d + \Delta y$



2.3 Rasterization of lines

Initialization of d?

- Start point $P_1 = (x_1, y_1)$ is on the line.
- Then, the first midpoint is:

  $$F(x_1 + 1, y_1 + 1/2) = F(x_1, y_1) + a + b/2$$

  with $F(x_1, y_1) = 0$.
- But $d = a + b/2$ is not an integer.
- Only the sign of d is important for the decision, which does not change by a multiplication by 2:
  - $d = 2a + b$ for initialization
  - $d = d + 2\Delta_{\mathrm{NE}}$ for case 1
  - $d = d + 2\Delta_{\mathrm{E}}$ for case 2

2.3 Rasterization of lines

Bresenham algorithm for lines

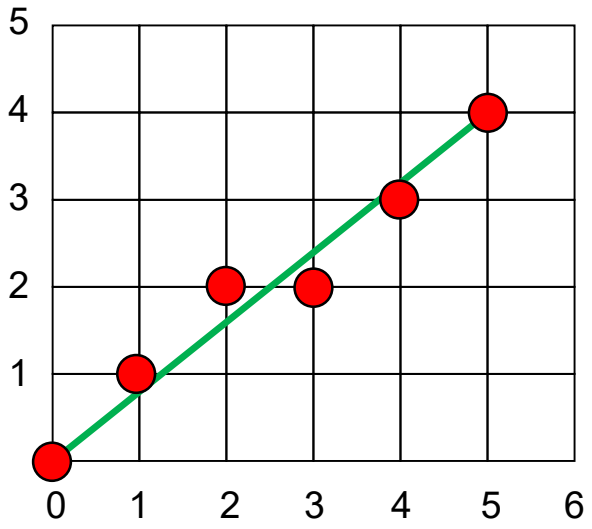
```
int x, y, Δx, Δy, d, ΔNE, ΔE;

x   = x1;          y  = y1;
Δx  = x2-x1;       Δy = y2-y1;
ΔNE = 2*(Δy-Δx); ΔE = 2*Δy;
d   = 2* Δy-Δx;
setPixel(x,y);
while ( x<x2 ) {
   if (d>=0) { d+=ΔNE; x++; y++; } // NE
   else      { d+=ΔE ; x++;      } // E
   setPixel(x,y);
}
```

- Pseudo-code is no optimized.
- Extension to lines with arbitrary slopes:
  - Exchange of x and y, signs, etc.

2.3 Rasterization of lines

Example:

$$P_1 = (0,0), P_2 = (5,4) \quad \blacktriangleright \quad \Delta_{NE} = -2, \Delta_E = 8$$

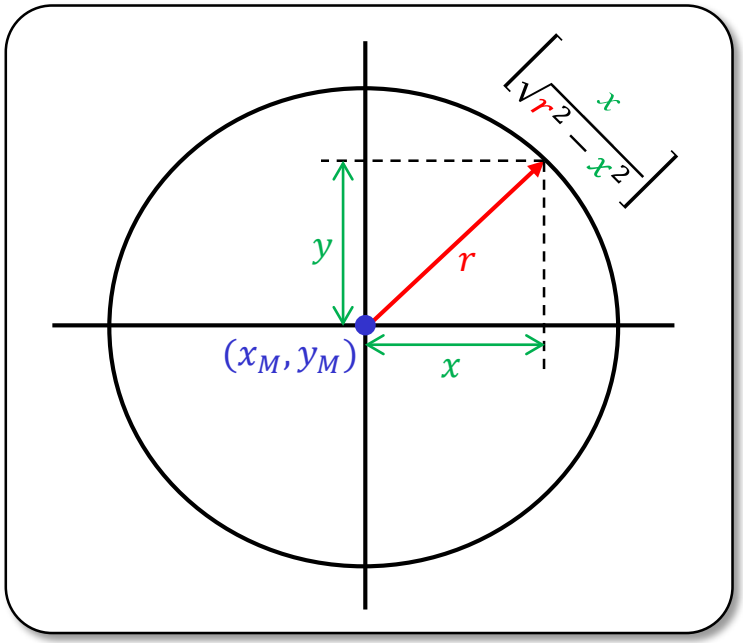


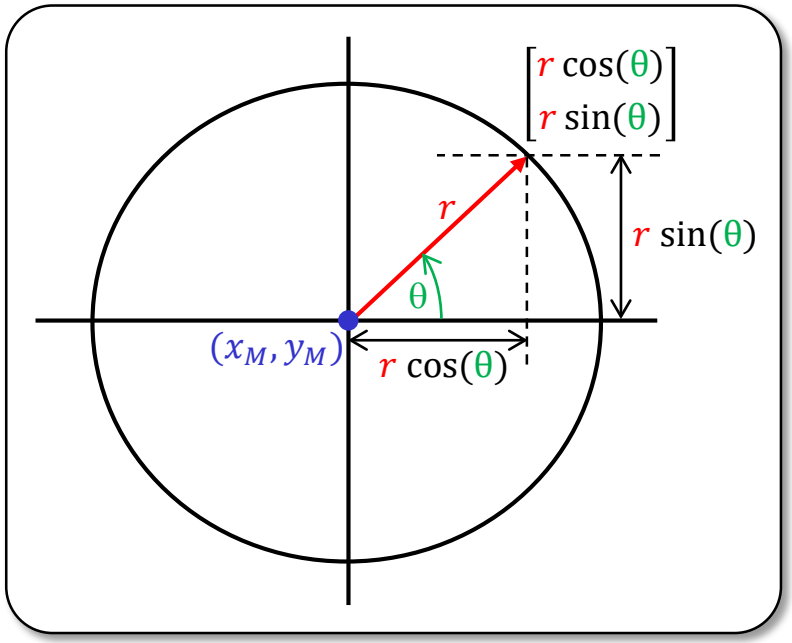
Δx	Δy	x	y	d	i	plot
5	4	0	0	3	1	(0,0)
		1	1	1	2	(1,1)
		2	2	-1	3	(2,2)
		3	2	7	4	(3,2)
		4	3	5	5	(4,3)
		5	4	3	6	(5,4)

2.4 Rasterization of circles

Representation of a circle at (x_M, y_M) with radius r:

1. **Implicit** $F(x,y) = (x - x_M)^2 + (y - y_M)^2 - r^2 = 0$
2. **Parametric** $x(\theta) = x_M + r\cos(\theta), y(\theta) = y_M + r\sin(\theta), \theta \in [0, 2\pi[$



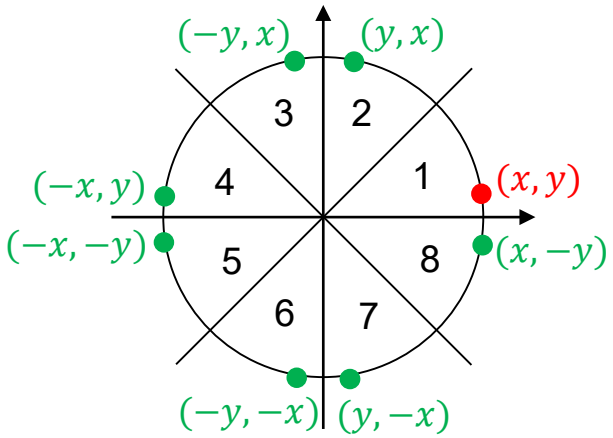


Problem: Both representation are computationally expensive and use expensive arithmetic operations.

2.4 Rasterization of circles

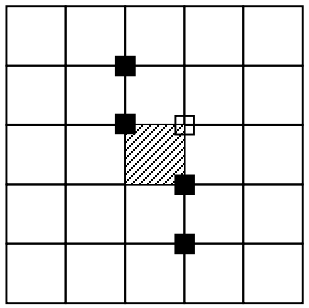
Observations

- Computing one point on the circle yields seven further circle points by symmetry.



- For a uniform brightness all pixels along the circle should by distributed as uniformly as possible.
- The assessment of the circle approxi-mation by the raster is subjective.
  - The usual criterion is the minimization of the residual

$$\left|x_i^2 + y_i^2 - r^2\right|.$$



For every raster quadrant only two of its corners should be plotted.

2.4 Rasterization of circles

Bresenham algorithm for circles

- The approach is analog to lines.
- Use the implicit representation:

$$F(x,y) = (x - x_M)^2 + (y - y_M)^2 - r^2 = 0.$$

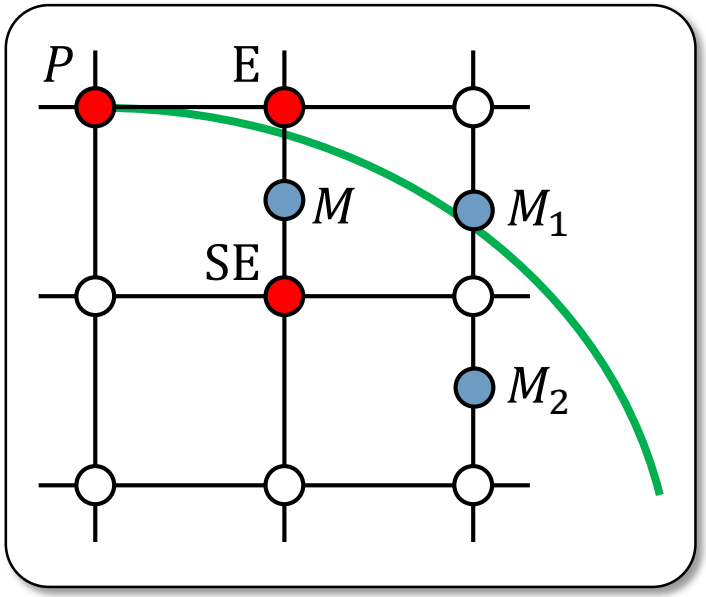
- Assumptions:
  - $(x_M, y_M) = (0,0)$,
  - the start point is on the raster,
  - radius $r \in \mathbb{N}$,
  - the circle segment is in the 2nd octant.

2.4 Rasterization of circles

Idea

- Pixel P has just been plotted.
- Is the next pixel to draw E or SE?
- Easier: Is $M = (x + 1, y - 1/2)$ inside or outside of the circle?

1. $F(M) < 0$, E is the next pixel.
2. $F(M) \geq 0$, SE is the next pixel.

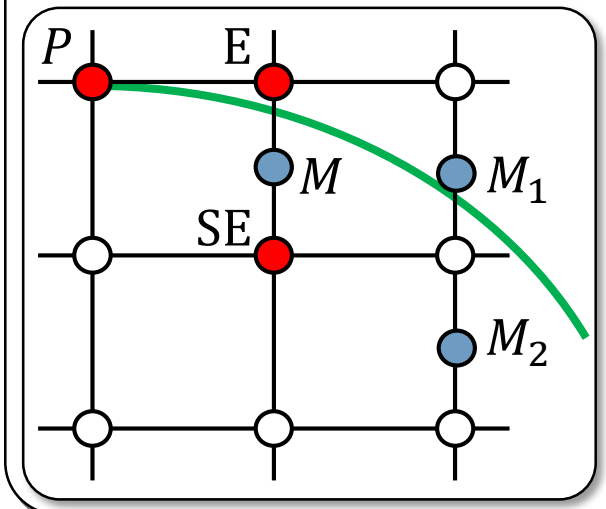


- Choose $d = F(M) = F(x + 1, y - 1/2)$ as decision variable.
- How to compute $F(M)$? $\longrightarrow$ **incrementally**
- $F(M) = (x + 1)^2 + (y - 1/2)^2 - r^2$

2.3 Rasterization of lines

Case 1

- E is the next pixel.
- M_1 is the next midpoint.
- $F(M_1) = (x + \quad 2)^2 + (y - 1/2)^2 - r^2$
- $F(M_1) = F(M) + 2x + 3$
- $d = d + 2x + 3$

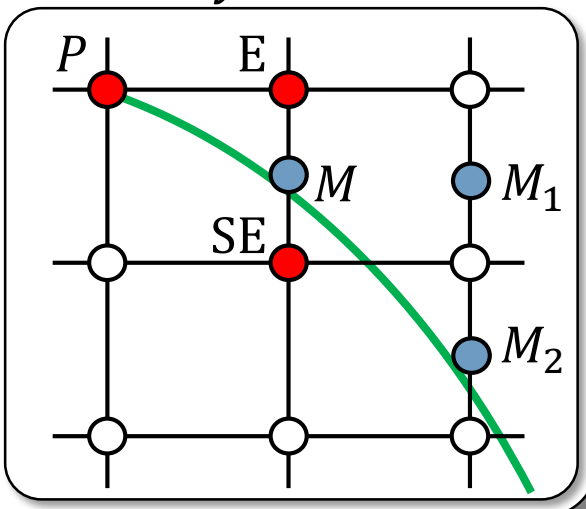


Summary

- $d = d + \begin{cases} \Delta_E \\ \Delta_{SE} \end{cases}$
- $\Delta_E = 2x + 3$
- $\Delta_{SE} = 2x - 2y + 5$

Case 2

- SE is the next pixel.
- M_2 is the next midpoint.
- $F(M_2) = (x + \quad 2)^2 + (y - 3/2)^2 - r^2$
- $F(M_2) = F(M) + 2x - 2y + 5$
- $d = d + 2x - 2y + 5$



2.4 Rasterization of circles

Initialization of d?

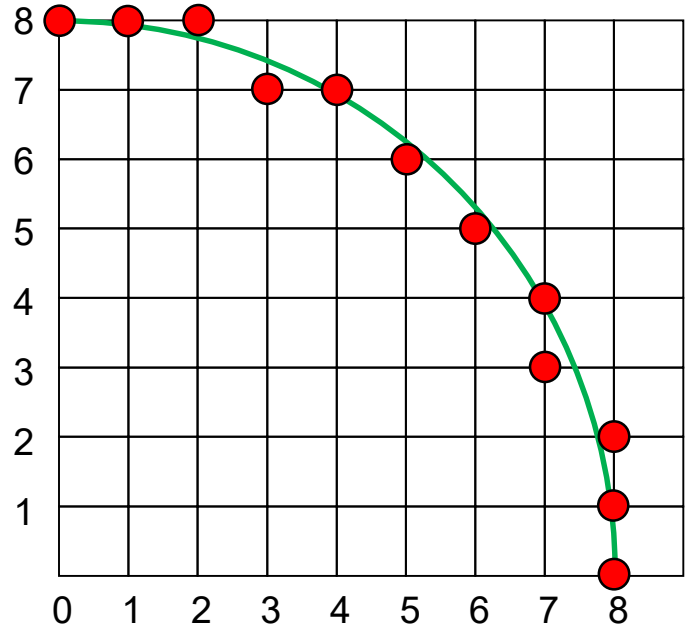
- Start point is $(0, r)$, i.e. $d = F(1, r - 1/2) = 5/4 - r$.
- Only the sign of d is important for the decision, multiplication by 4.

```
int x, y, d, Δ_SE, Δ_E;
x  = 0; y = r;
d  = 5 - 4 r;
setPixel(x,y); setPixel(-x,y);.....                    // Symmetry
while (y>x) {
   if(d>=0) { Δ_SE = 4*(2*(x-y)+5); d +=Δ_SE; x++; y--; } // SE
   else     { Δ_E  = 4*(2*x+3);     d +=Δ_E ; x++;      } // E
   setPixel(x,y); setPixel(-x,y); .....                 // Symmetry
}
```

- Pseudo-code is no optimized.
- Extensions:
  - Incremental computation of Δ_E and Δ_{SE} → no multiplications

2.4 Rasterization of circles

Example: $r = 8$



Δ_{SE}	Δ_{E}	d	x	y	Pixel
-	-	-27	0	8	1
-	12	-15	1	8	2
-	20	5	2	8	3
-28	-	-23	3	7	4
-	36	13	4	7	5
-4	-	9	5	6	6

2.5 Rasterization of ellipses*

- Implicit representation of an ellipse around the origin an two semi-axes a and b:

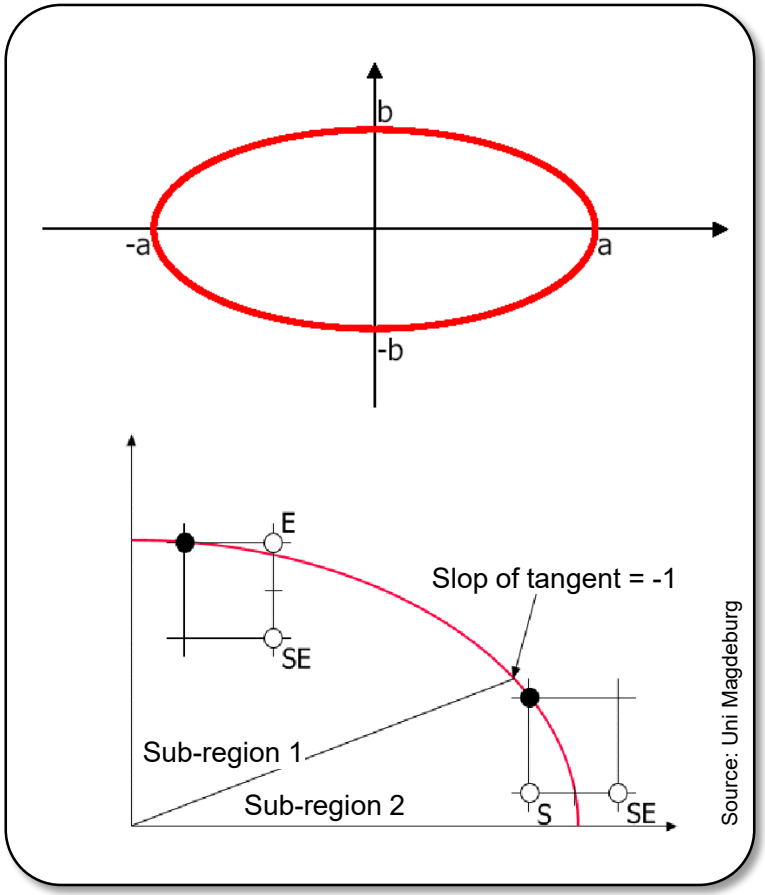
$$F(x,y) = b^2x^2 + a^2y^2 - a^2b^2 = 0$$

- Algorithm for the 1st quadrant and reflections for the rest.
- Two sub-regions with transition at

$$y^2(y - 1/2) \leq b^2(x - 1)$$

1. Decision between E and SE
2. Decision between S and SE

- In each sub-region analog to the circle-algorithm using adapted d, Δ_{SE}, Δ_E, and Δ_S.



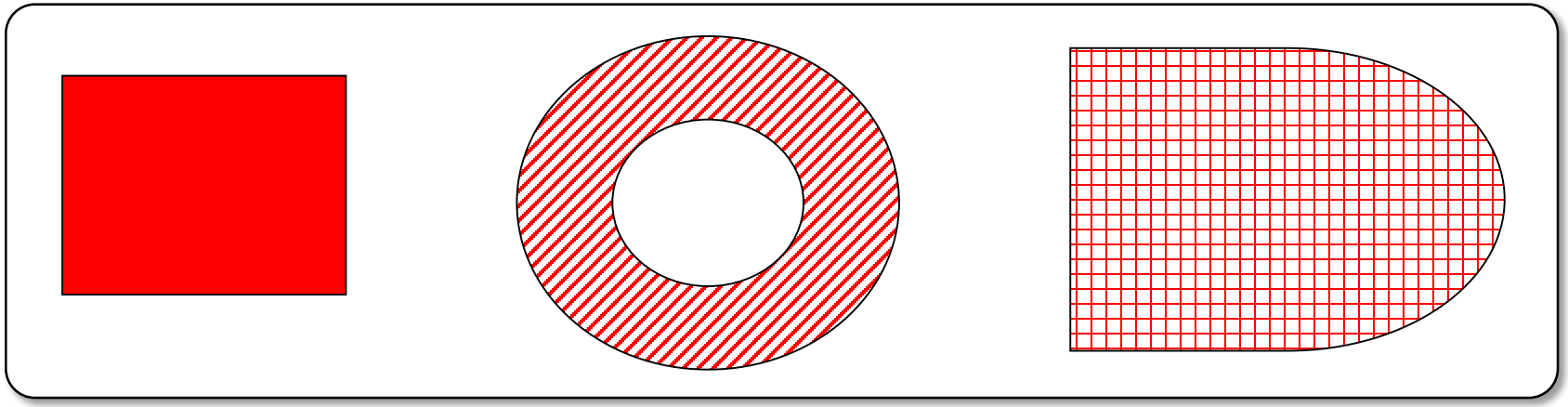
2.6 Filling algorithms

Input: Polygon or bounded area defined as

1. set of pixels,
2. geometric description (polygon, circle, etc.).

Output: Coloring of the area with a single color, a pattern or a hatching.

Examples: Bar diagrams, surfaces, solids, etc.



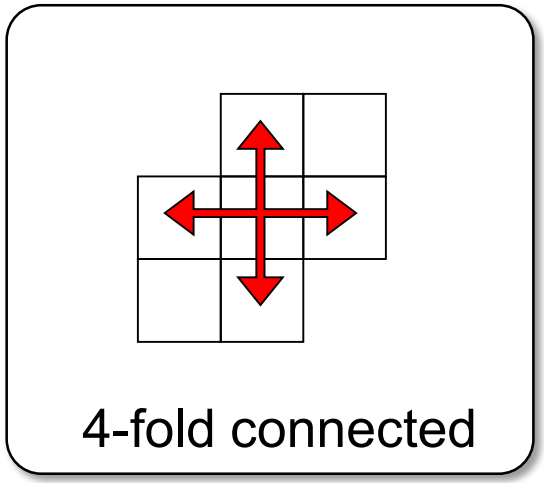
2.6 Filling algorithms

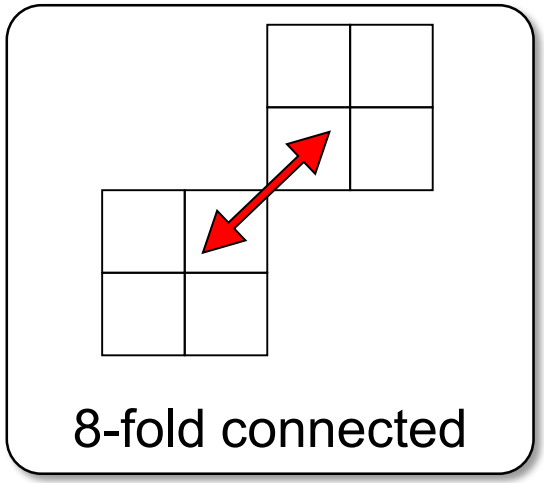
2.6.1 Filling of pixel-sets, seed fill

Input:
- Rasterized polygon defined as a boundary-pixel-set.
- Start pixel (seed), that determines which side of the boundary is the inside.

Output: All inner pixels that need to be colored.

Connectedness of the area: What are the neighbors of a pixel?





2.6 Filling algorithms

Remarks

- Filling algorithms with 8 degrees of freedom (directions) can be used to fill 4-fold connected areas.
- **Problem:** 4-fold connected areas sharing a common corner (leakage).
- Filling algorithms with 4 degrees of freedom cannot be used to fill 8-fold connected areas.

2.6 Filling algorithms

The approach depends on the definition of the area:

(i) Boundary-Fill-Algorithm for boundary-defined areas

Input: **1.** start pixel (aka seed),
2. **boundary-color**
3. fill-color or pattern.

Algorithm:

- start at the seed and
- re-color all neighboring pixels recursively,
- until a pixel has the **boundary-color** (or is already re-colored),
- then stop.

(ii) Flood/Interior-Fill-Algorithm for interior-defined areas

Input: **1.** start pixel (aka seed),
2. **interior-color**
3. fill-color or pattern.

Algorithm:

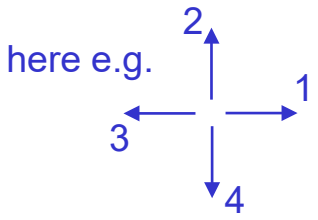
- start at the seed and
- re-color all neighboring pixels recursively,
- until a pixel does not have the **interior-color** (or is re-colored),
- then stop.

2.6 Filling algorithms

Simple seed-algorithm

(4-fold connected, boundary-defined area, FILO/LIFO-principle)

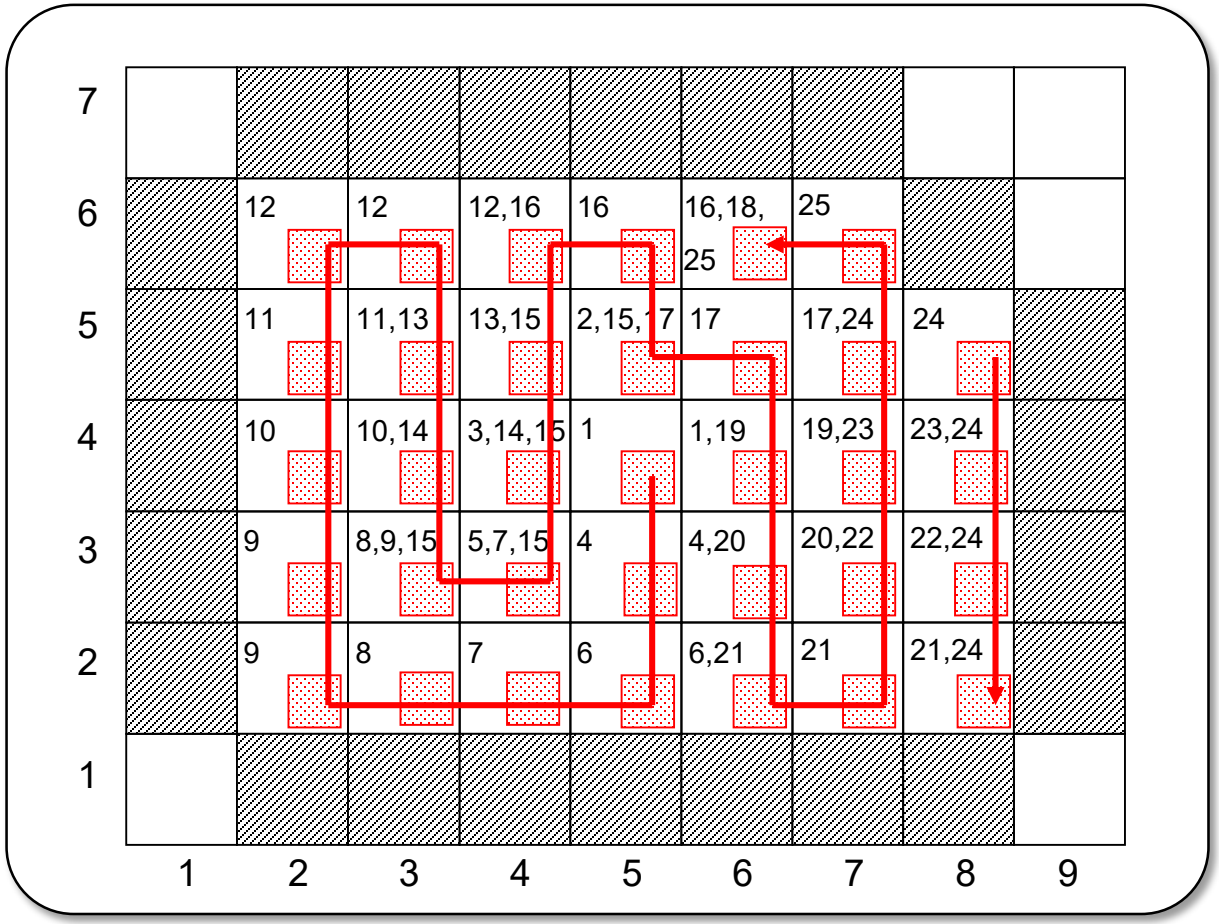
```
Empty(stack);
Push(stack, seed-pixel);

while(stack not empty) {
  Pixel p = Pop(stack);
  setColor(p, FillColor);
  for ( each of the 4-connected neighbor pixels p_i of p )
  {
    if(! (p_i==boundary_pixel || colorOf(p_i)==FillColor))
      Push(stack, p_i);
  }
}
```



- Possibly pixels are pushed on the stack multiple times to avoid search requests on the stack!

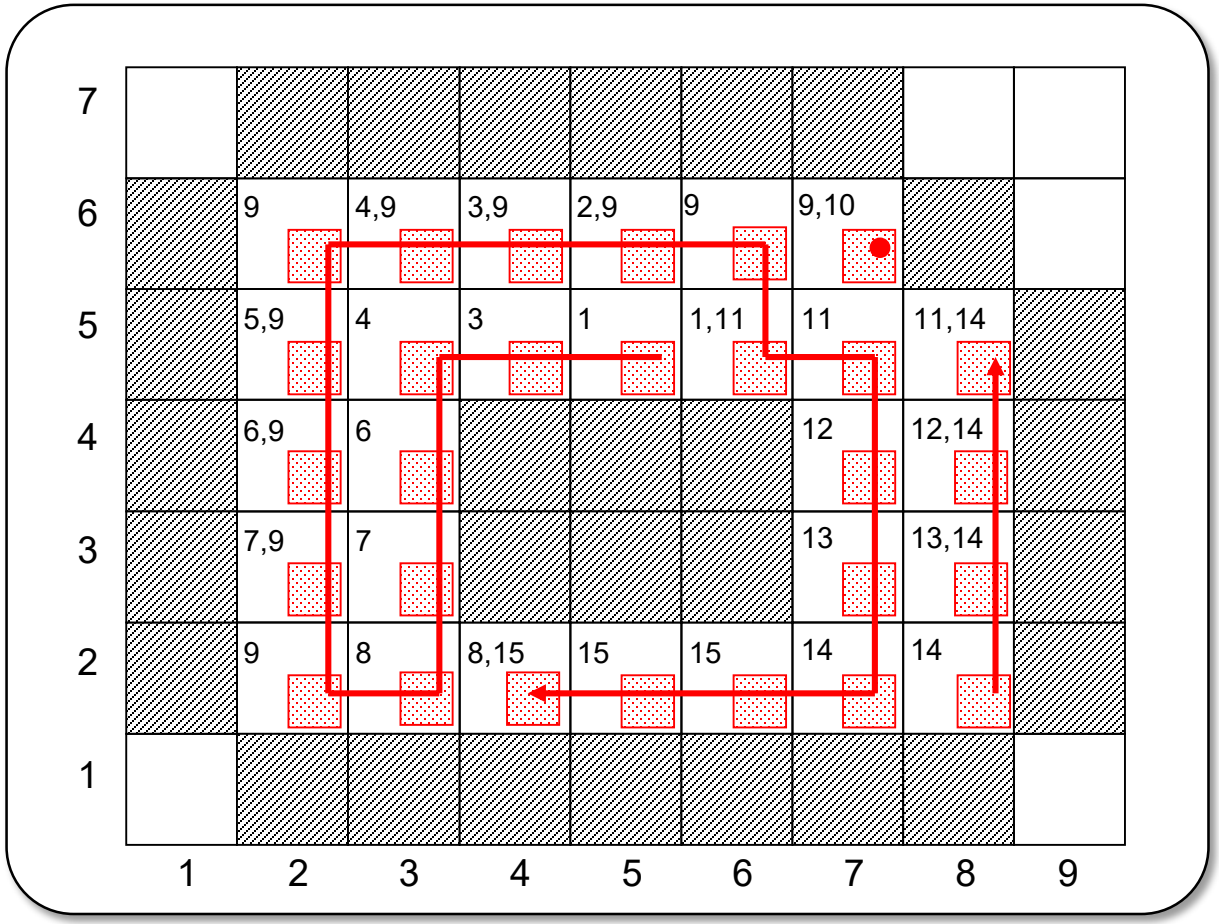
2.6 Filling algorithms

Example: The numbers denote the position of the pixel in the stack.



2.6 Filling algorithms

Example: Area with a hole.



2.6 Filling algorithms

2.6.2 Scan-Line-Methods*

- Also known as ***Scan Conversion***.
- Proceeds from the top raster rows to the bottom raster row.
- A pixel of the actual row (***scan line***) is re-colored, if it is inside the polygon.

```
// simplest approach for rectangles:
for(y=y_min ; y<=y_max ; y++)                    // row
  for(x=x_min ; x<=x_max ; x++)                  // column
    if(Inside(polygon, x, y))
      SetPixel(x,y);
```

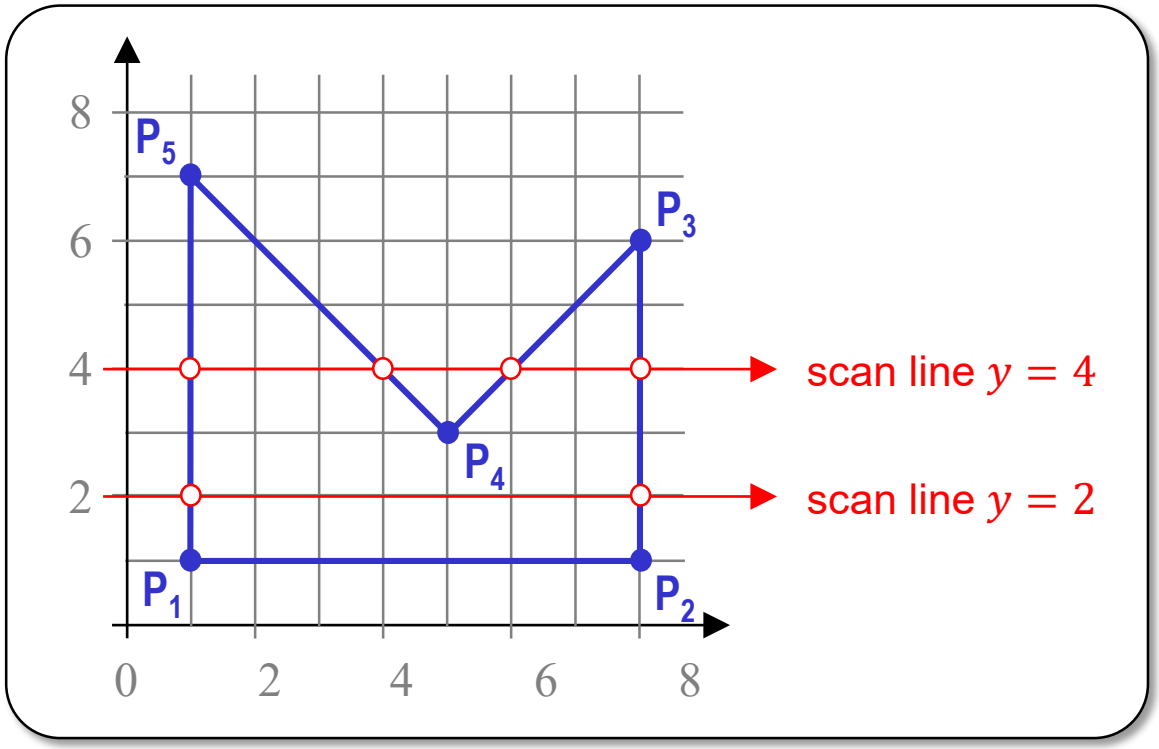
- Can be used for geometric and pixel-wise defined areas,…
- …but is very slow.

2.6 Filling algorithms

- Improvement using coherence.
- Use row-coherence: adjacent pixels on the same row have most likely the same intensity.
- The pixel characteristic (intensity) changes only, if the scan line intersects a polygon edge.
- A scan line segment between two intersections is either completely inside or outside the polygon.

2.6 Filling algorithms

Example:

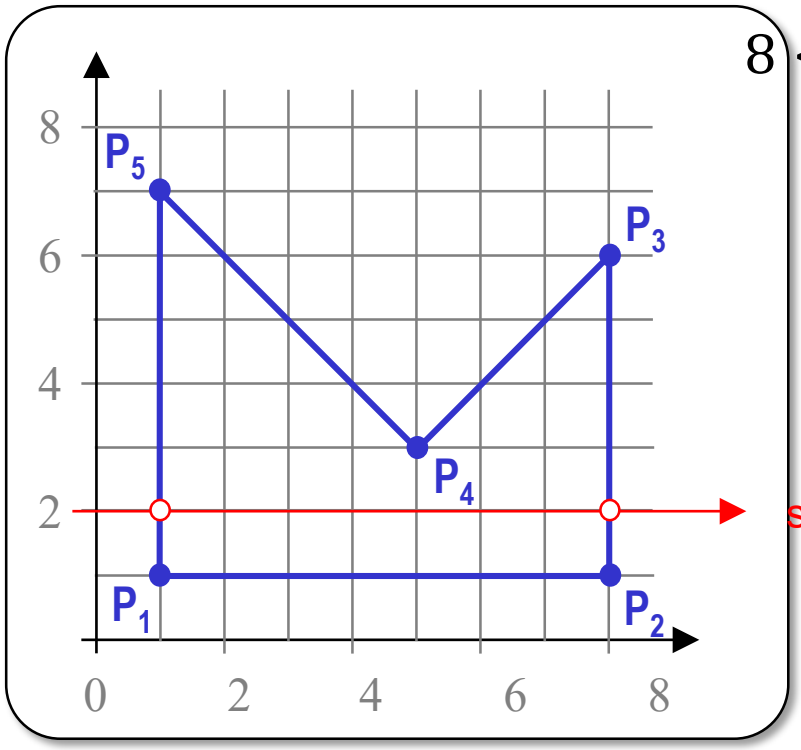


- Scan line $y = 2$: Intersection with polygon at $x \in \{ 1, 8 \}$
- Scan line $y = 4$: Intersection with polygon at $x \in \{ 1, 4, 6, 8 \}$

2.6 Filling algorithms

Example: Scan line $y = 2$: Subdivide the scan line into 3 segments:

$$x < 1 \quad : \quad \text{outside the polygon}$$
$$1 \leq x \leq 8 \quad : \quad \text{inside the polygon}$$
$$8 < x \quad : \quad \text{outside the polygon}$$



2.6 Filling algorithms

Example: Scan line $y = 4$: Subdivide the scan line into 5 segments:

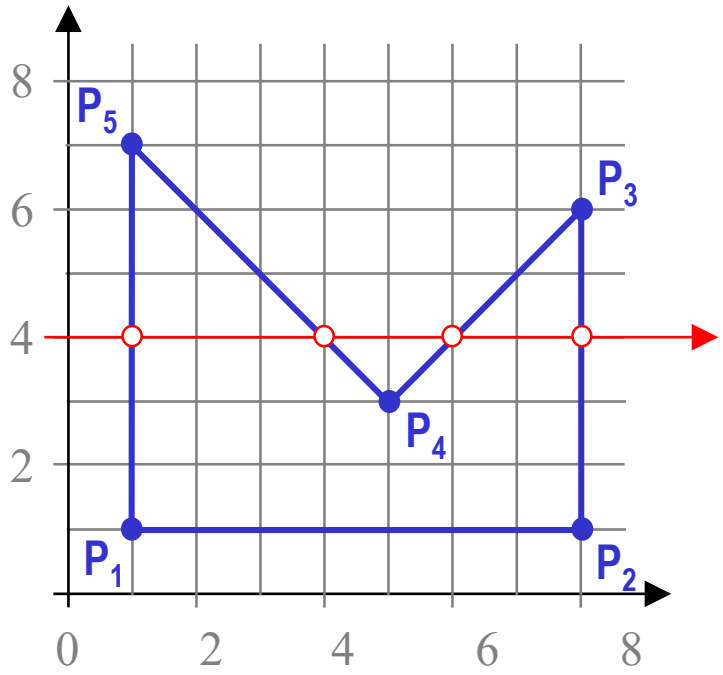
$x < 1$: outside the polygon

$1 \leq x \leq 4$: inside the polygon

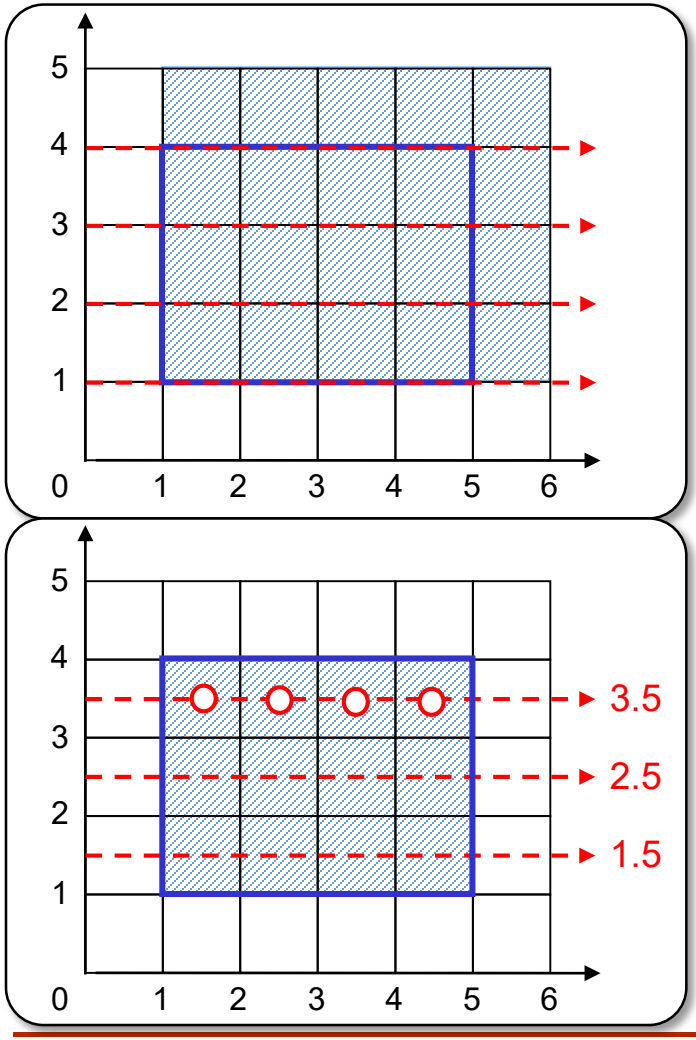
$4 < x < 6$: outside the polygon

$6 \leq x \leq 8$: inside the polygon

$8 < x$: outside the polygon



2.6 Filling algorithms



Choice of scan lines

- Scan lines at $y = n$ with integer n.

  Pixel-activation: left pixel side is in the interval.

  ▶ Problem: Possibly too many pixels are activated.

- Scan lines through pixel midpoint

  Pixel-activation: Pixel midpoint is right of the intersection of the scan line and the polygon.

2.6 Filling algorithms

Simple ordered edge list algorithm

Principle: a) Pre-Processing

b) Scan Conversion

a) Pre-Processing:

- Determine (e.g. using Bresenham-algorithm) for each polygon edge the intersections with the scan lines at pixel midpoints.
- Ignore horizontal edges.
- Store all intersections $(x, y + ½)$ in a simple list.
- Sort list top-down and left-to-right.

Since scan lines traverse pixel midpoints!

2.6 Filling algorithms

- Problems at singularities:

  Scan line intersects the polygon in a corner.

- Different cases:
  - If the corner is a local extremum, count the intersections twice.
  - If the corner is no local extremum, count the intersections once.

- Local extremum:

  The y-values of the end points of the polygon edges beginning in a corner are both larger/smaller than the y-value of the intersected corner.

2.6 Filling algorithms

b) Scan Conversion:

- Consider two subsequent intersections (x_1, y_1) und (x_2, y_2) in the list, i.e.
  - List elements 1 and 2,
  - List elements 3 and 4,
  - etc.
- Because of the pre-processing, the scan line at y satisfies

$$y = y_1 = y_2 \text{ and } x_1 \leq x_2.$$

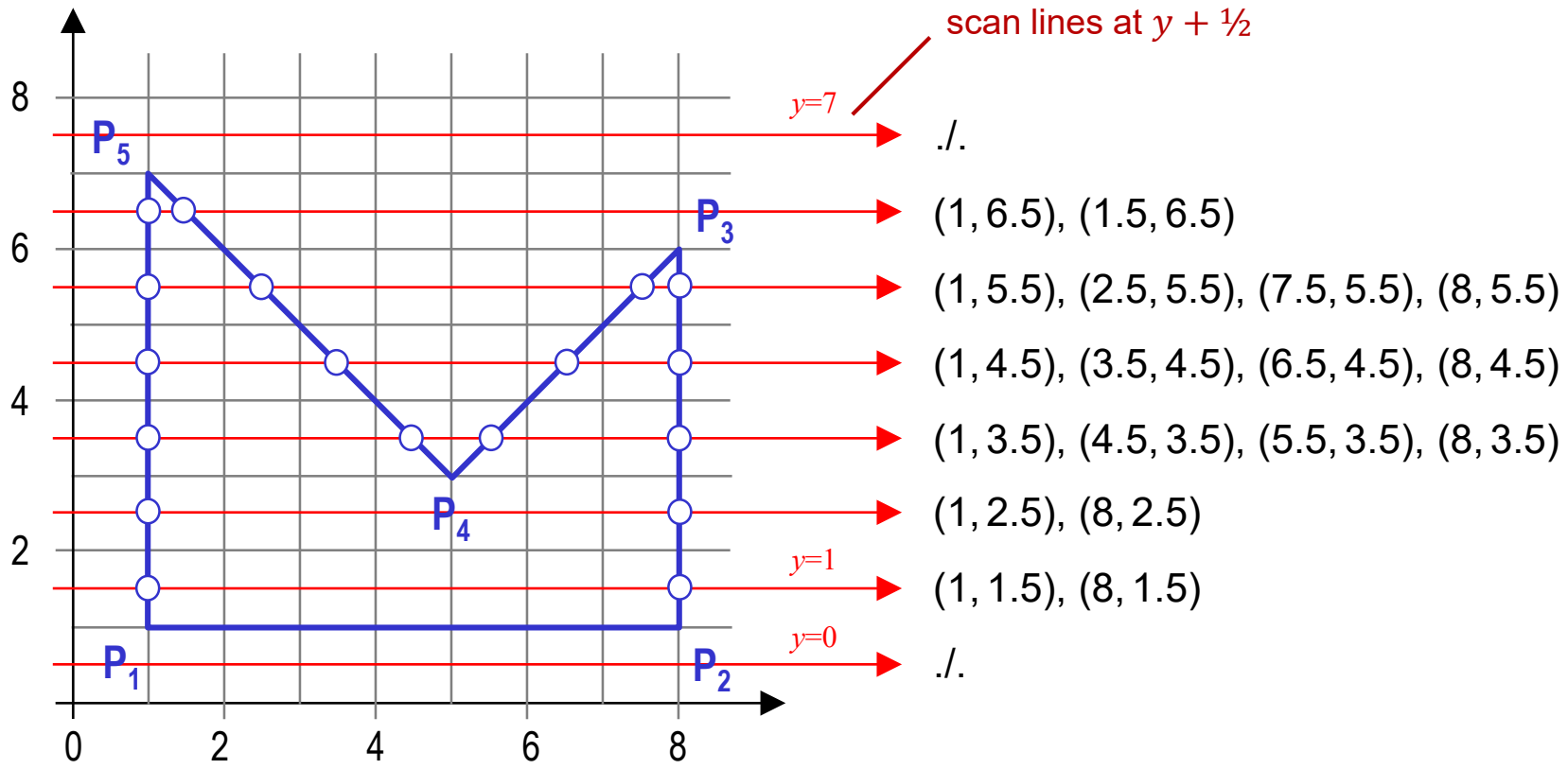
- Plot all pixels on the scan line at y, if

$$x_1 \leq x \leq x_2 \text{ with integer } x.$$

2.6 Filling algorithms

Example:

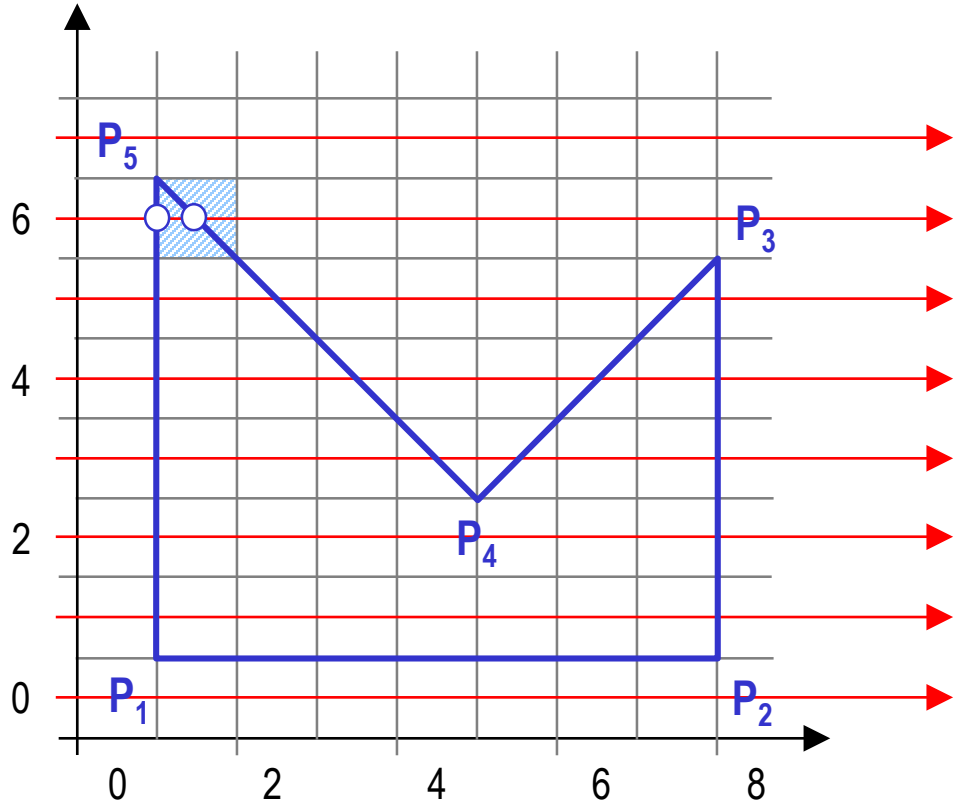
a) Pre-processing:



2.6 Filling algorithms

Example:

a) Scan Conversion:



Attention:
These are only the pixel coordinates!

./.

(1, 6)

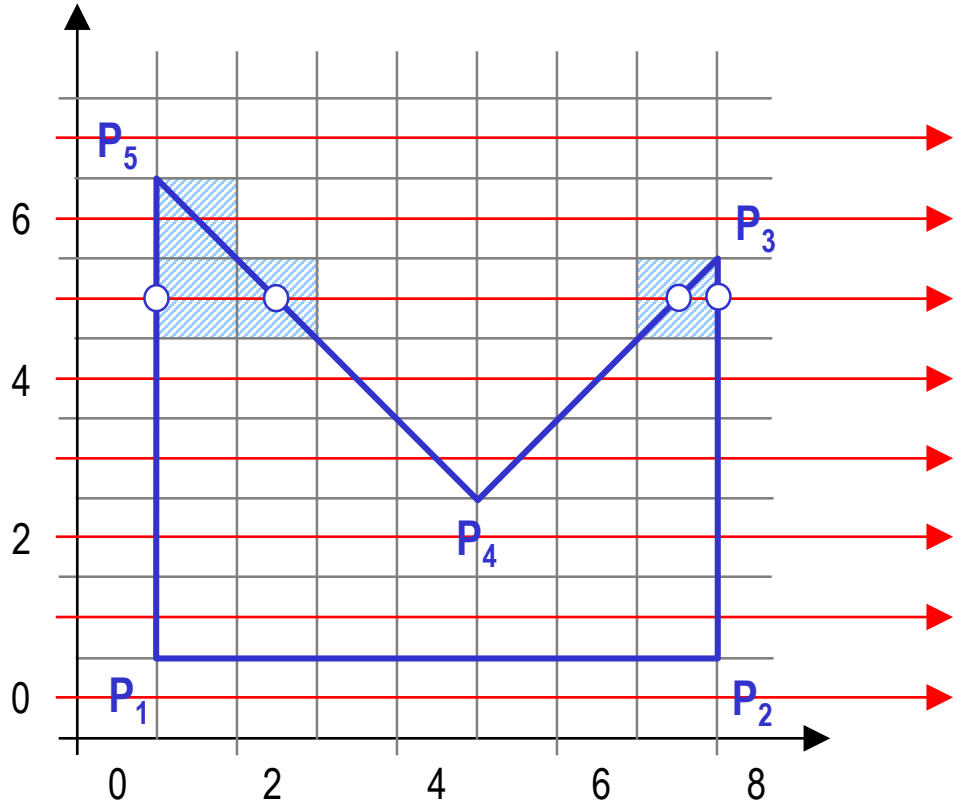
Because $y = 6$ yields

$$1 \le x + \tfrac{1}{2} \le 1.5 \;\Rightarrow\; x = 1$$

2.6 Filling algorithms

Example:

a) Scan Conversion:



Attention:
These are only the pixel coordinates!

./.

(1, 6)

(1, 5), (2, 5), (7, 5)

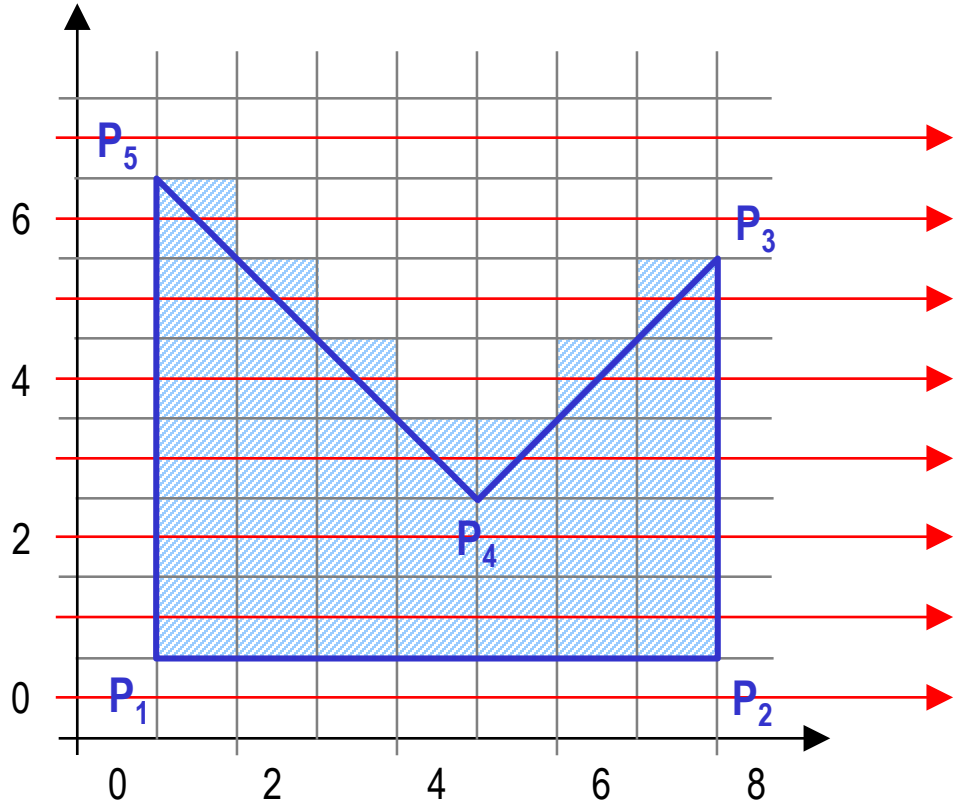
Because $y = 5$ yields:

$$1 \le x+½ \le 2.5 \Rightarrow x \in \{1,2\}$$
$$7.5 \le x+½ \le 8 \Rightarrow x = 7$$

2.6 Filling algorithms

Example:

a) Scan Conversion:



./.

(1, 6)

(1, 5), (2, 5), (7, 5)

(1, 4), (2, 4), (3, 4), (6, 4), (7, 4)

(1, 3), (2, 3), …, (7, 3)

(1, 2), (2, 2), …, (7, 2)

(1, 1), (2, 1), …, (7, 1)

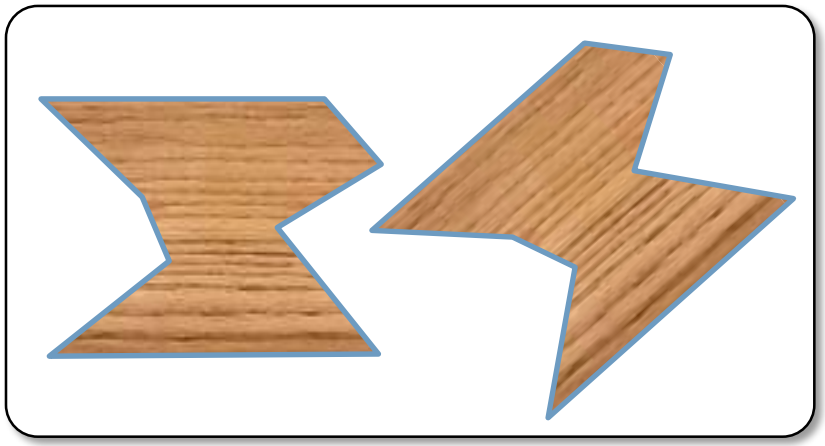
./.

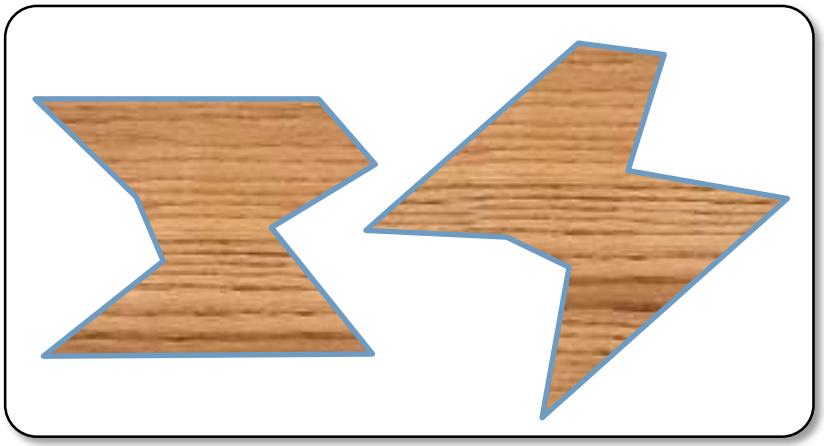
Attention:
These are only the pixel coordinates!

2.6 Filling algorithms

2.6.3 Filling with patterns*

- Pattern given as bitmap (texture).
- Non-uniform color, but mapping of pixels of the image to pixels of the texture.
- Depends on the anchor of the texture, i.e. the position of the texture relative to the polygon:
  - anchor in the left lower polygon corner (left example) or
  - anchor on the background (right example).





2.7 Aliasing

Aliasing (notion from signal theory)

In general aliasing-effects occur for incorrect reconstructions of a (continuous) input signal from the sampled signal, which is discretely sampled at a too low sampling frequency.

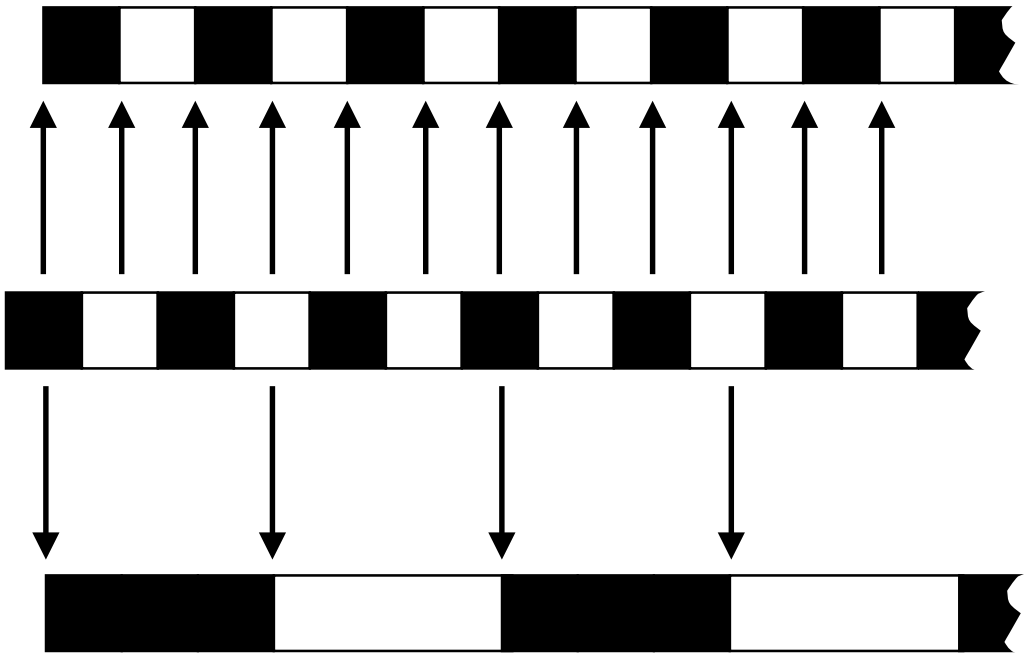
Background: **Shannon-sampling-theorem**

A band-limited signal needs to be sampled at least with the double cut-off frequency (**Nyquist-frequency**) to allow for an exact reconstruction.

2.7 Aliasing

Pixels sampled with the double detail frequency.

Pixels sampled with too small sampling frequency.



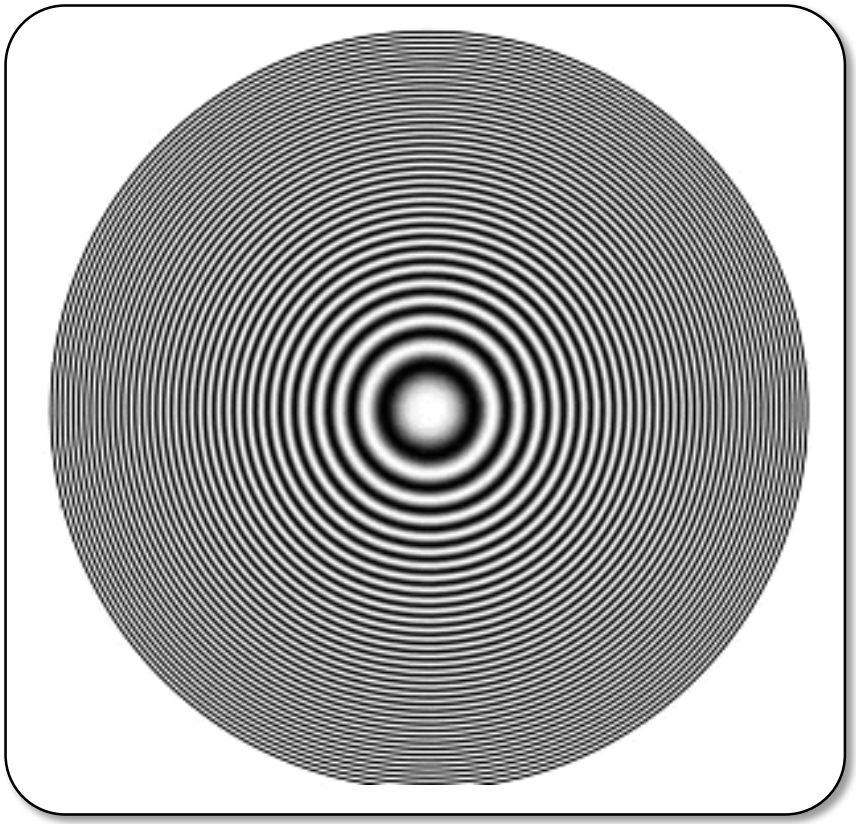
Correctly sampled line: details are (almost) exactly reconstructed.

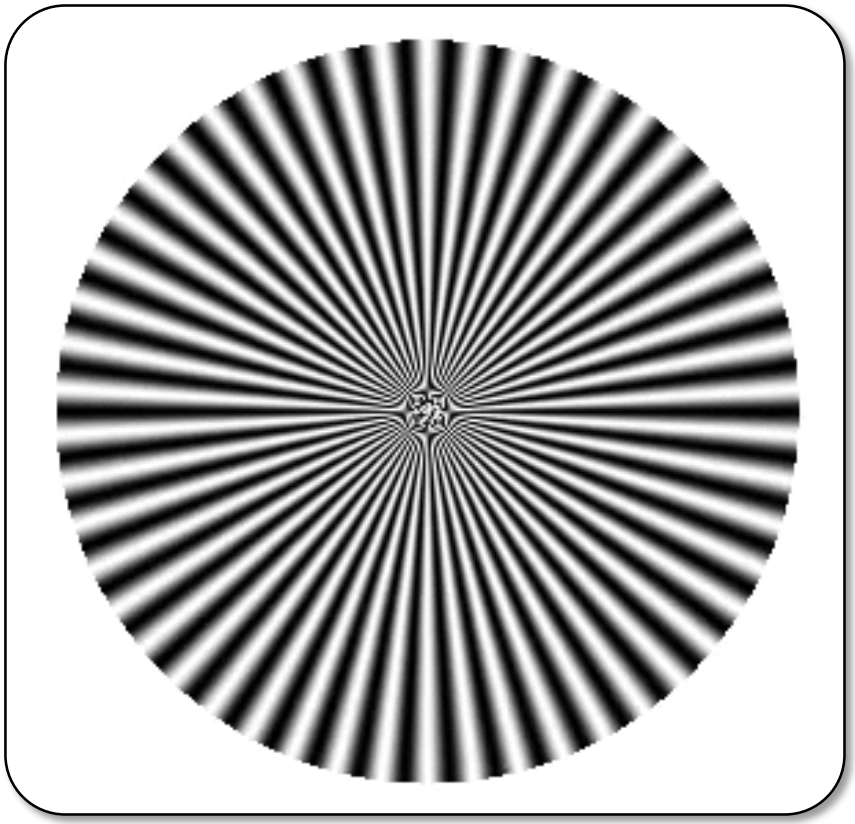
Line in an image with intensity details.

Subsampled line: Details occur at a lower frequency aliased.

2.7 Aliasing

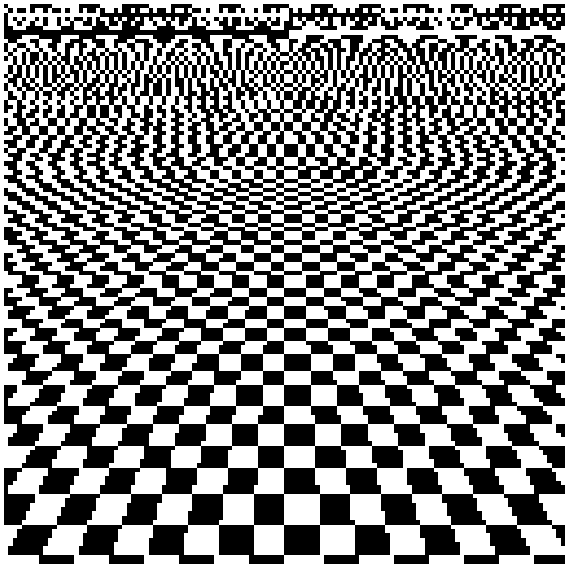
Visual Effects



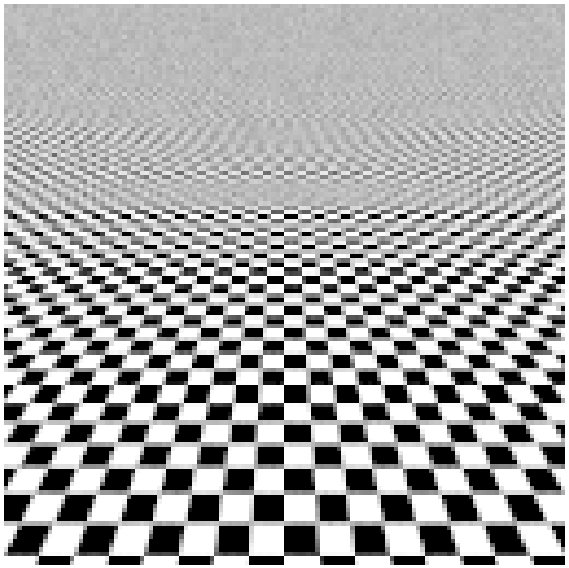


2.7 Aliasing

Visual Effects + Visual Artifacts: infinite checker-board pattern



Visual effects: periodicity of the texture is in the magnitude of the pixels.
Visual artifacts: jagged edges



Anti-aliasing reduces both effects.

2.7 Aliasing

Aliasing in the context of computer graphics

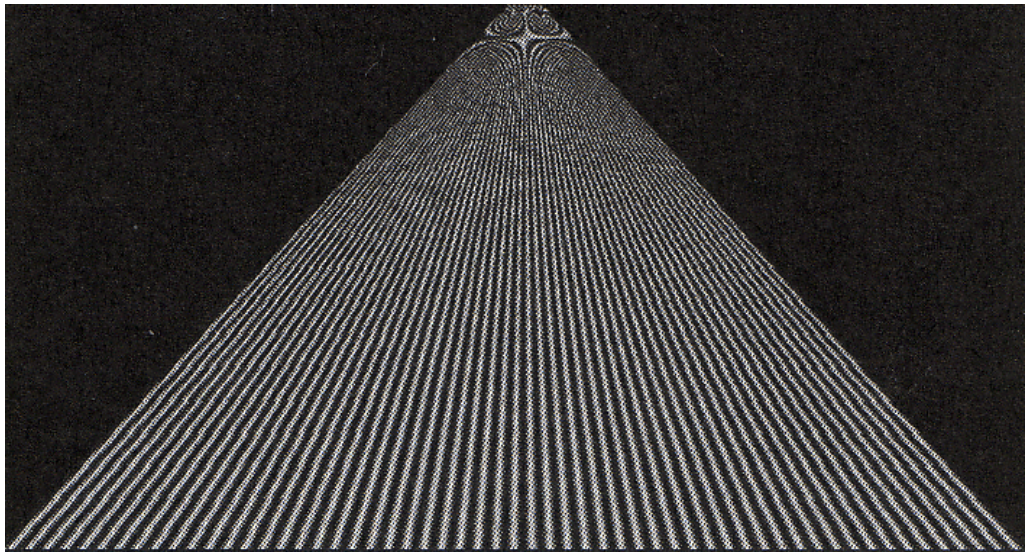
- **Visual effects**, true aliasing effects caused by spatial under-sampling, e.g.
  - checkboard pattern, Moiré-effects, etc.
- **Visual artifacts**, cause by raster conversion, e.g.
  - jagged edges for slanted lines,
  - vanishing of objects, that are smaller than one pixel,
  - vanishing of long, thin objects,
  - loss of detail in complex images, etc.
- Distinction between **spatial** and **temporal aliasing**, e.g.
  - *Effect:* seemingly backward turning wheels,
  - *Artifact:* blinking of small moving objects or in animations, etc.

2.7 Aliasing

Visual Effects:

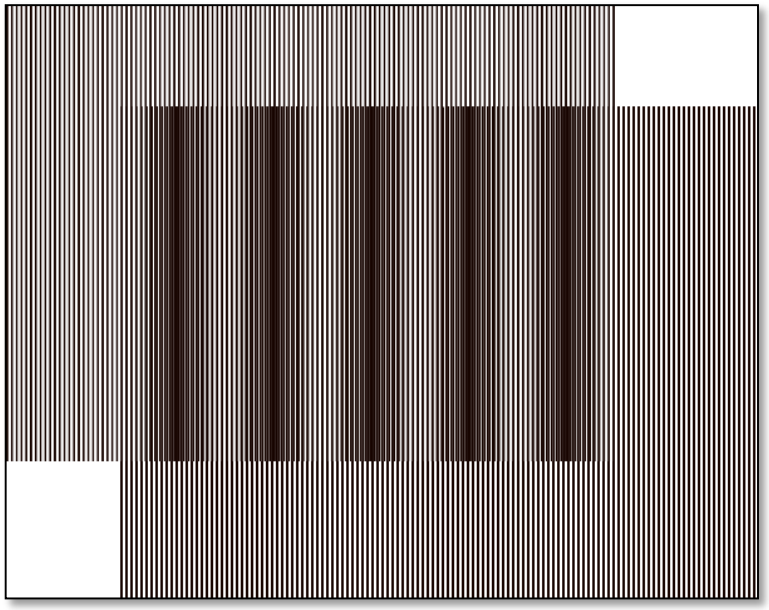
Spatially high-frequent (under-sampled) details/objects do not simply vanish, but occur at lower, wrong frequencies

Example:
spatial aliasing



2.7 Aliasing

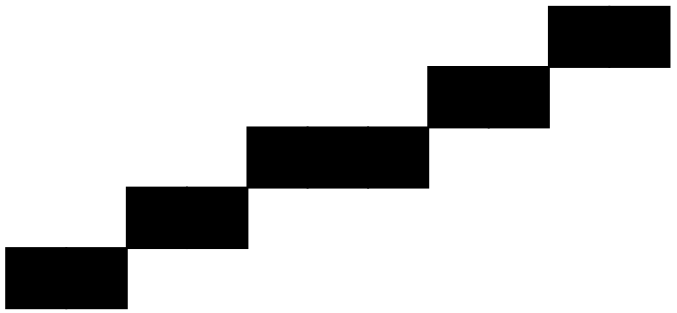
Visual Effects: Moiré-effect, overlay of the sampling grid with a pattern in the image.



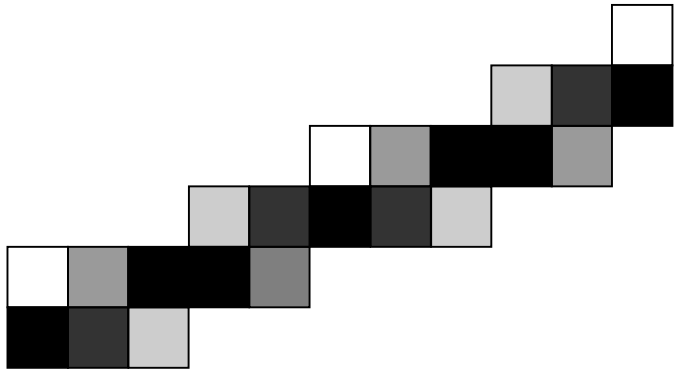
Source: Wikipedia

2.7 Aliasing

Visual artifact: jagged edges, jaggies



Jagged edges, because points can only be activated at raster positions.

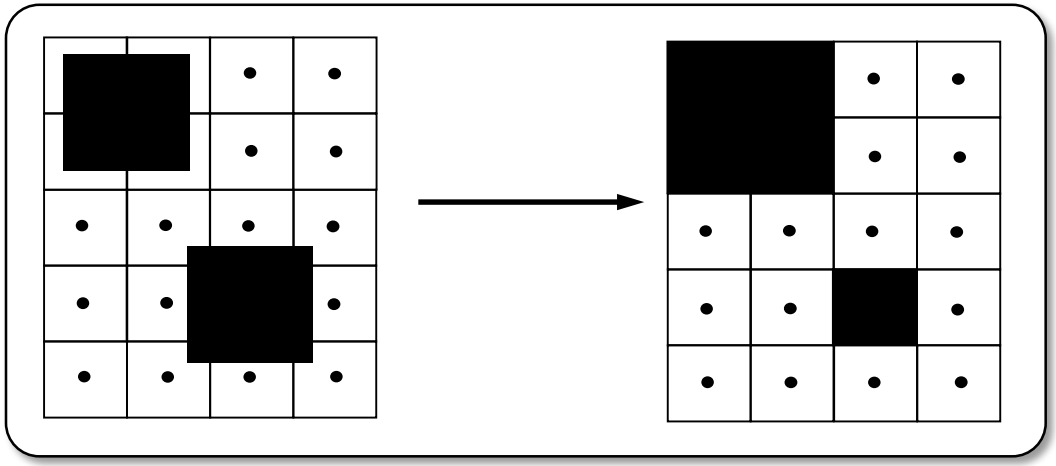


Smoothed lines by activation of neighbor pixels with varying intensities.

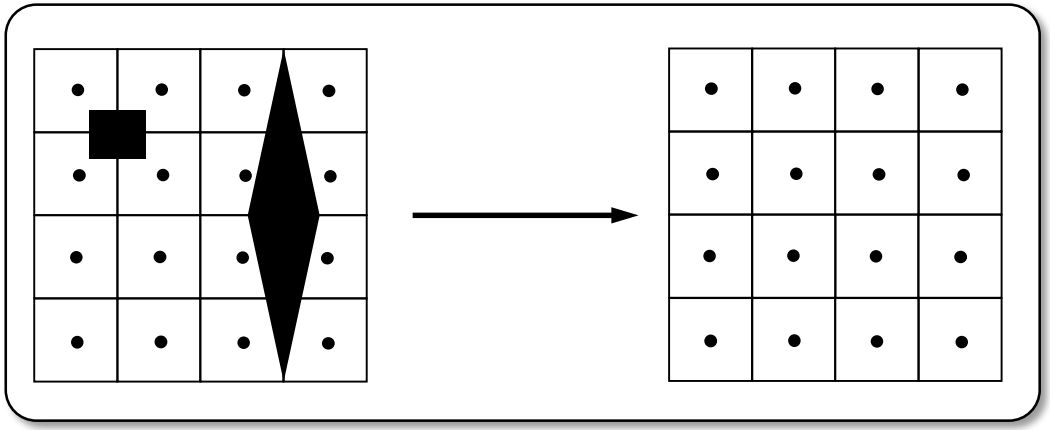
2.7 Aliasing

Visual artifact: Aliasing for polygons

- Objects with the size are plotted with different sizes.



- Vanishing of objects, that are smaller than one pixel.
- Vanishing of long, thin objects.



2.7 Aliasing

Temporal Aliasing

Occurs, if some part of an animation move/change relative to the frame rate too fast.

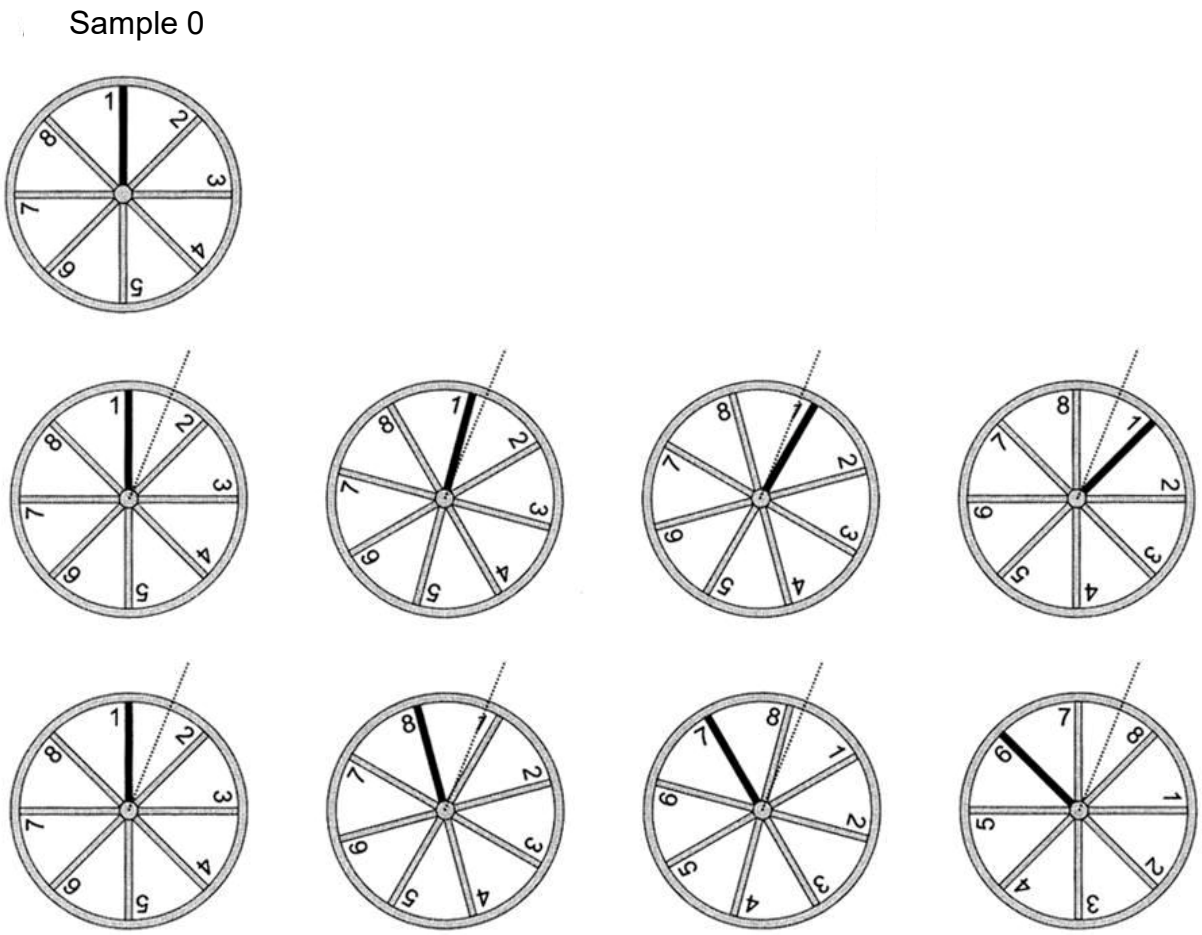
Example: Cartwheels in western movies.

- Perception of the movement via moving sprockets.
- Wheel with 8 sprockets
  - Repetition rate f_W for $\mathbf{1/8}$ full rotation of the wheel.
  - Sampling frequency $f_A = 50\text{Hz}$, $1/f_A = 0.02\,\text{sec.}$
- Dependent on the sampling of the movement, the wheels stand still, turn (fast/slow) backward, or forward.
- Particularly distracting: varying wheel speeds around

$1/16$ full rotations.

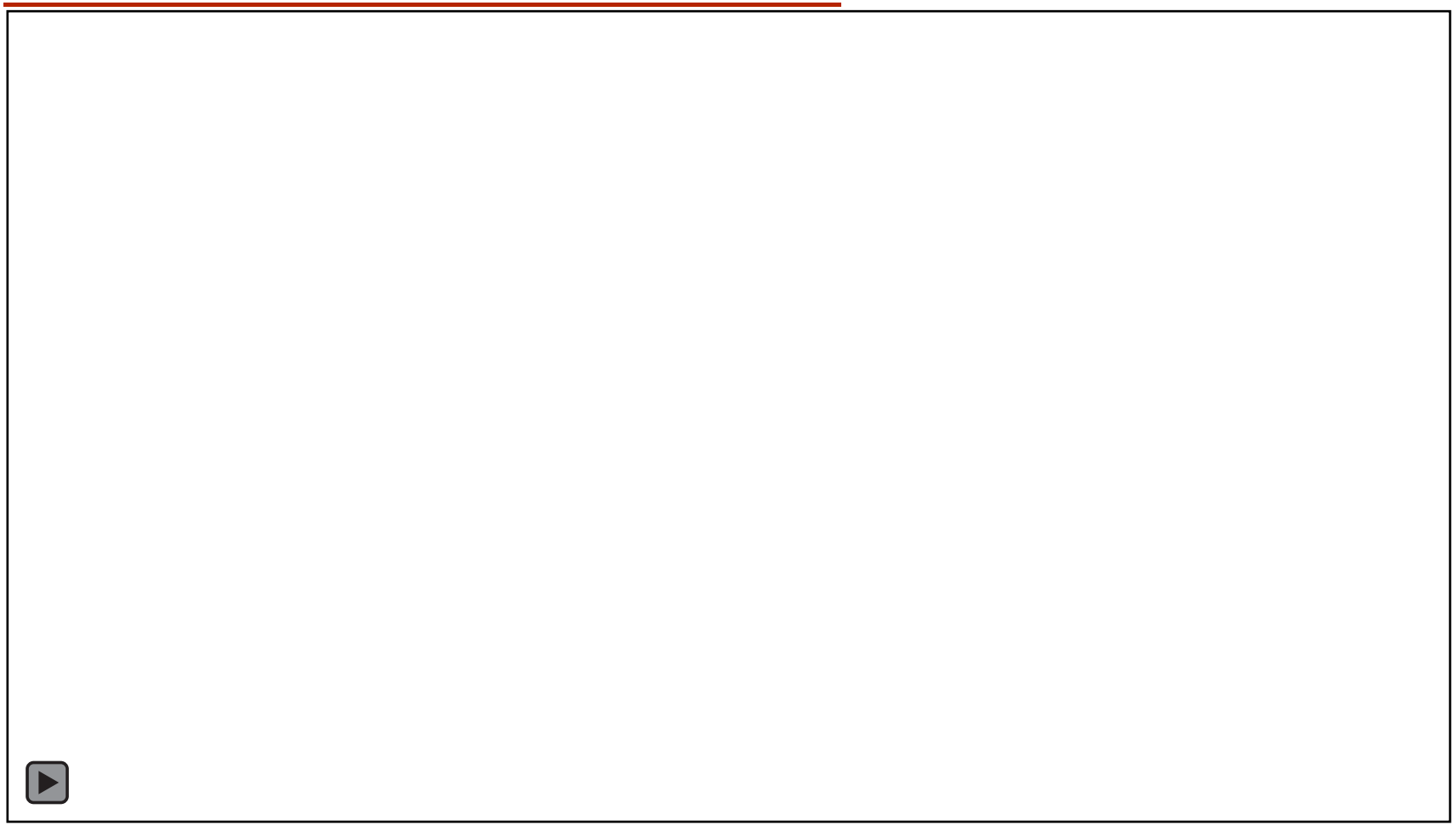
2.7 Aliasing

- $f_W = fA$, i.e. $1/8$ rotations per sampling cycle.
- $2\,f_W < f_A$, i.e. $< 1/16$ rotations per sampling cycle.
- $2\,f_W > f_A$, i.e. $> 1/16$ rotations per sampling cycle.



Source: Bender, Brill.

2.7 Aliasing



Source: Jaguar.

2.8 Anti-Aliasing

Anti-aliasing in the context of computer graphics

- Methods (e.g. over-sampling, filtering), to counteract aliasing-effects and –artifacts.
- A complete elimination is often impossible:
  - e.g. signal is not band-limited;
  - here increasing the sampling frequency (over-sampling) helps, but does not eliminate the problem.
- For artifacts caused by raster conversion, anti-aliasing-methods are also called *edge smoothing*.

2.8 Anti-Aliasing

2.8.1 Super-sampling

- Simple, global (i.e. for the complete image) anti-aliasing-method:
  - Over-sampling or super-sampling.
  - Every pixel is computed with a higher resolution than the true display resolution, i.e. every pixel is refined into sub-pixels.
  - The pixel gets the grey-value or color-value of its average sub-pixels.
  - This approach corresponds to a filter process.

2.8 Anti-Aliasing

- Usual filter kernels (Crow, 1981):

3 x 3

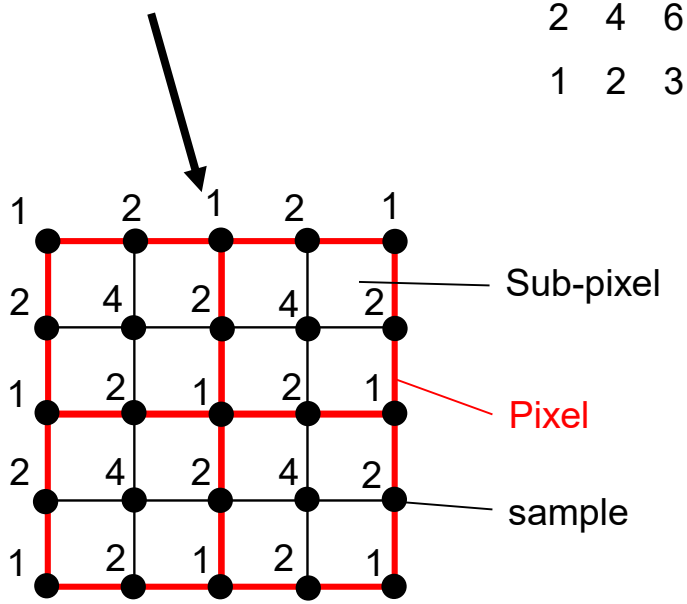
1	2	1
2	4	2
1	2	1

5 x 5

1	2	3	2	1
2	4	6	4	2
3	6	9	6	3
2	4	6	4	2
1	2	3	2	1

7 x 7

1	2	3	4	3	2	1
2	4	6	8	6	4	2
3	6	9	12	9	6	3
4	8	12	16	12	8	4
3	6	9	12	9	6	3
2	4	6	8	6	4	2
1	2	3	4	3	2	1



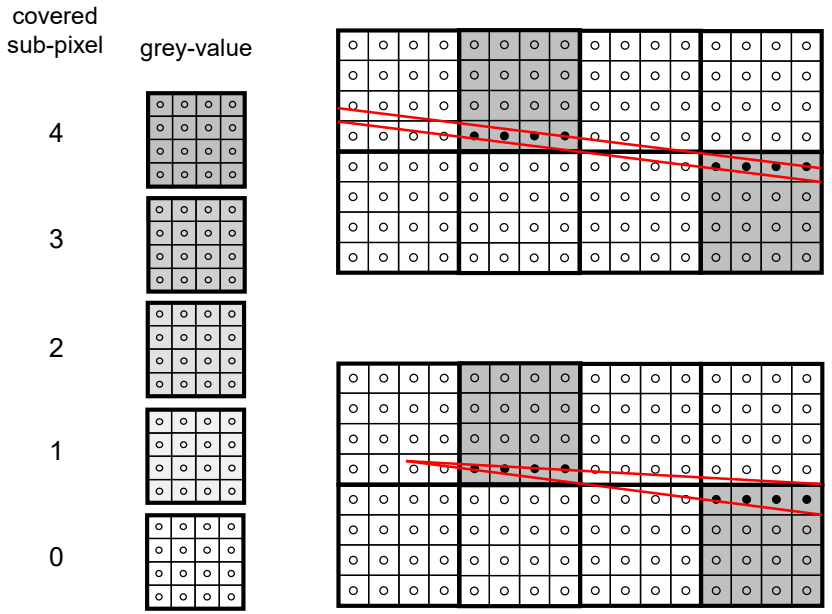
Example:

W	W	B
W	B	B
B	B	B

Average of colors → 68,75 % B

2.8 Anti-Aliasing

- Despite super-sampling, there are still surprising effects for lines and thin triangles (thin polygons):



Lines vanish partially, if no sub-pixels are covered!

ditto for triangles!

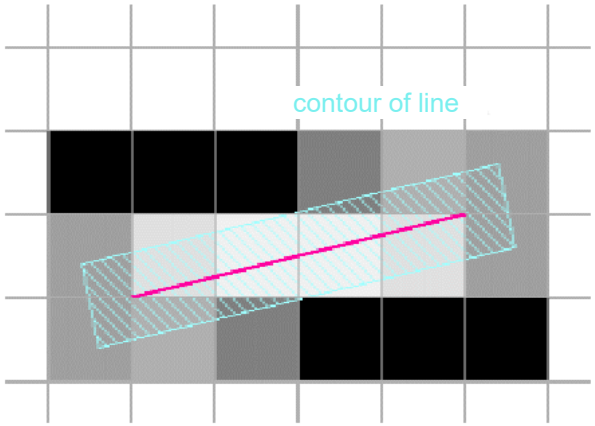
Remarks:

- Covered sub-pixels are black.
- The shading shows the grey-value dependent on the number of covered sub-pixels.

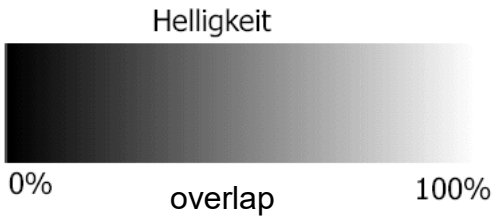
2.8 Anti-Aliasing

2.8.2 Un-weighted overlap area

- Solution only possible computing the correct overlap of the line with each pixel.
- **Idea:** Pixel intensity is proportional to ratio of overlap.
- **Assumption:** Every pixel has a square footprint of size 1x1, i.e. raster cell.



ratio of (pixel area) / (line overlap) yields the pixel intensity

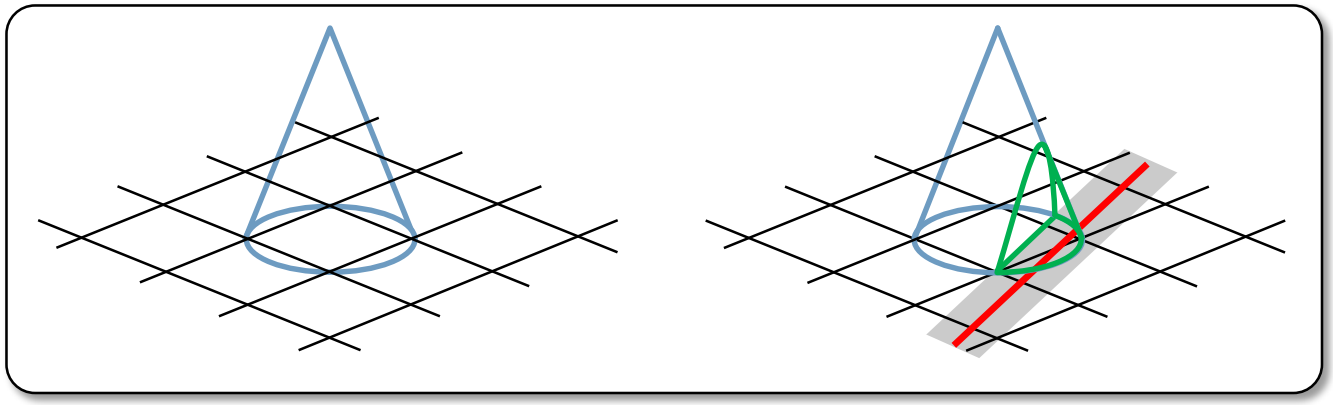


Source: FH München

2.8 Anti-Aliasing

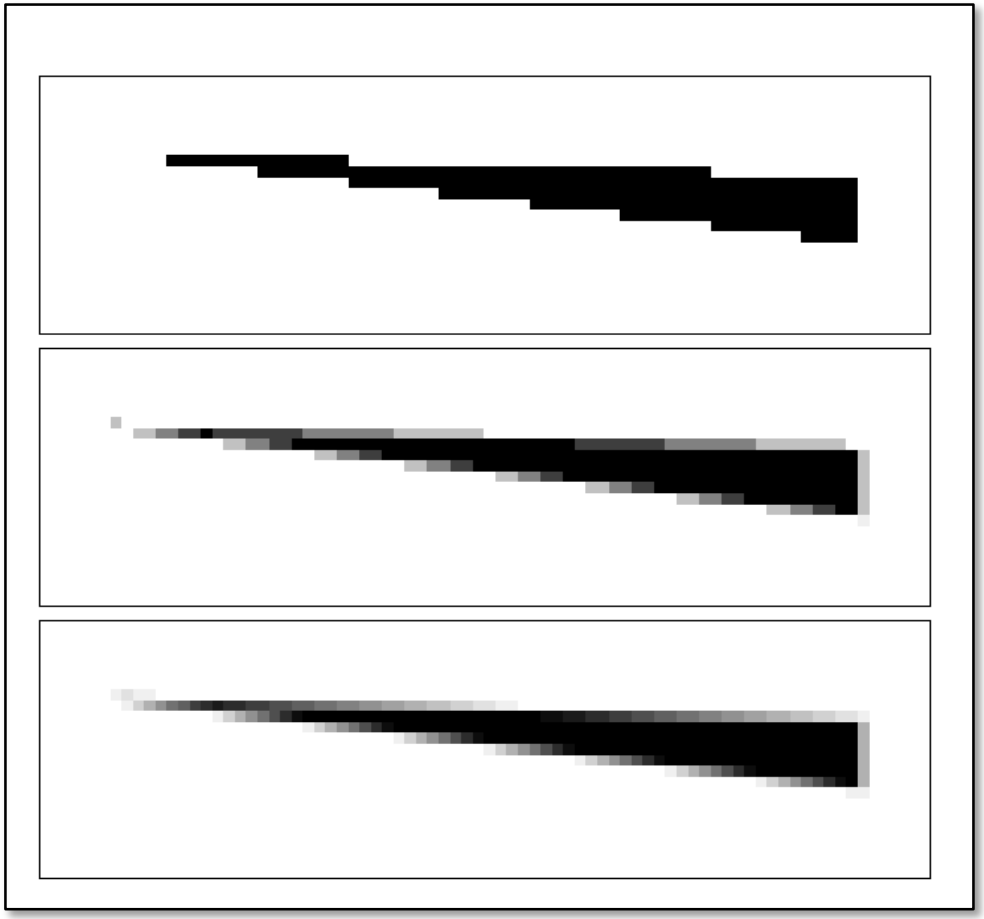
2.8.3 Weighted overlap area

- Un-weighted overlap area, i.e. lighted volume is a cube.
- Pixel footprint is a circle, i.e. lighted volume is a cylinder.
- Pixel area is not uniformly lighted, the center is brighter, and neighbor pixels are also lighted, i.e. lighted volume is a **cone** with base circle overlapping to neighboring pixels.
- Weighted overlap area, i.e. pixel intensity corresponds to volume of **intersected cone-section**.



2.8 Anti-Aliasing

Example:



Acute triangle without smoothing.

Acute triangle with over-sampling.

Acute triangle with correct computation of area overlap.

2.8 Anti-Aliasing

2.8.4 Stochastic methods

- Stochastic sampling:
  - Over-sampling using Monte-Carlo-Methods,
  - intensity is computed at random positions inside a pixel and the resulting intensities are averaged.
- Monte-Carlo-Methods can also be used to compute the area overlap.
- Stochastic methods increase efficiency, but tend to flicker for animated scenes.

2.8 Anti-Aliasing

Nvidia-Technologies

Global anti-aliasing in the pixel-shader for the total rendered image.

Disadvantage: The image is smoothed, i.e. it becomes blurry.

- FSAA (Full scene anti-aliasing): Super-sampling for every pixels of the scene, very expensive.
- MSAA (Multi-sample anti-aliasing): Super-sampling of the depth/stencil-values of every pixels of the scene
- FXAA (Fast-approximate anti-aliasing): Edge detection and smoothing of edge-pixels only.
- TXAA (Temporal anti-aliasing): Filtering of multiple samples inside and outside the pixel with pixels of previous frames.
- DLSS (Deep learning super sampling): A NN is trained (an stored on the driver) to render the anti-aliased high resolution image.

Goals

- How do you rasterize lines?
- How do you rasterize circles?
- How do you fill polygons?
- What is the principle of seed-fill-algorithms?
- What is the principle of scan-line-algorithms?
- What is aliasing?
- What is spatial / temporal aliasing?
- How can aliasing-effects or aliasing-artifacts be reduced?