

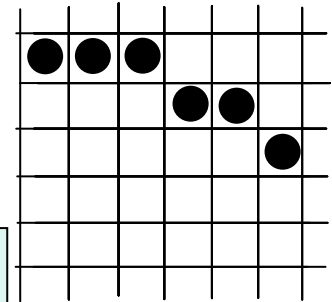
Computergrafik SS 2016

Oliver Vornberger

Vorlesung vom 12.04.2016

noch Kapitel 3
2D-Grundlagen

BresenhamCircle, die 2.



```
x = 0;
y = r;
delta = 5.0/4.0 - r;          d = 1 - r;
                                dx = 3;
                                dxy = -2*r + 5;

while (y >= x) {
    setPixel(x,y);
    if (delta < 0.0) {          (d < 0.0)
        delta = delta + 2*x + 3.0;    d = d + dx;
        x++;                          dx = dx + 2;
    else {                          dxy = dxy + 2;
        delta = delta + 2*x - 2*y + 5.0;    d = d + dxy;
        x++;                          dx = dx + 2;
        y--;                          dxy = dxy + 4;
    }
}                                     d:=delta-1/4    dx:=2x+3    dxy:= 2x-2y+5
}
```



Radius	delta		d	
r	$5/4 - r$	< 0	$1 - r$	< 0
5	-3.75	Ja	-4	Ja
4	-2.75	Ja	-3	Ja
3	-1.75	Ja	-2	Ja
	-0.75	Ja	-1	Ja
	0.25	Nein	0	Nein
	1.25	Nein	1	Nein

BresenhamCircle, die 3.

```
x=0; y=r; d=1-r; x=3; dx=3; dxy=-2*r+5;
while (y>=x){
    setPixel(+x,+y);
    setPixel(+y,+x);
    setPixel(+y,-x);
    setPixel(+x,-y);
    setPixel(-x,-y);
    setPixel(-y,-x);
    setPixel(-y,+x);
    setPixel(-x,+y);

    if (d<0) {d=d+dx; dx=dx+2; dxy=dxy+2; x++;}
    else    {d=d+dxy; dx=dx+2; dxy=dxy+4; x++;
             y--;}
}
```

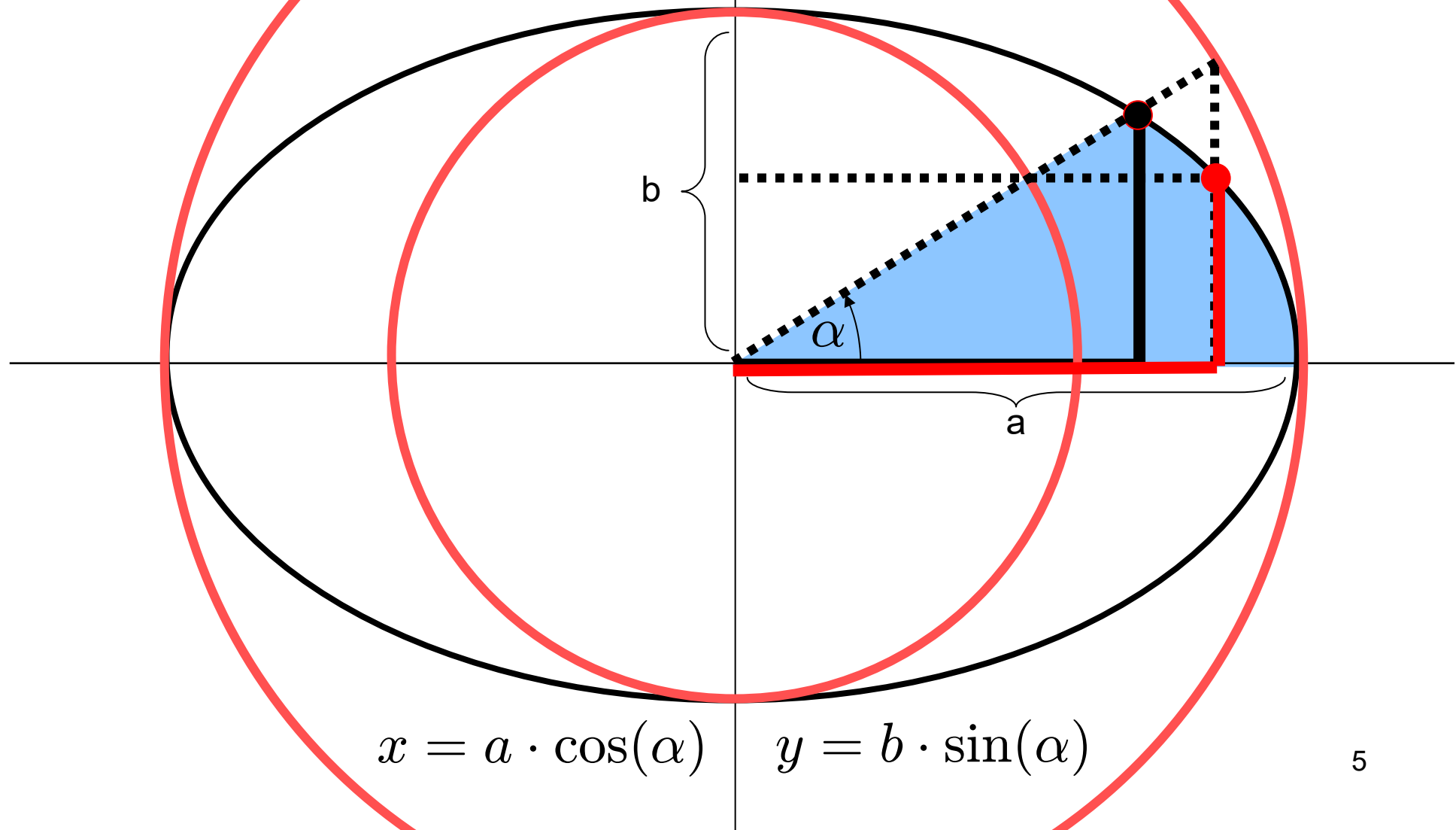
Source: ~cg/2016/skript/Sources/drawBresenhamCircle.jav

Java-Applet: ~cg/2016/skript/Applets/2D-basic/App.html

$$U \approx 2\pi \sqrt{\frac{1}{2}(a^2 + b^2)}$$

$$A = \pi ab$$

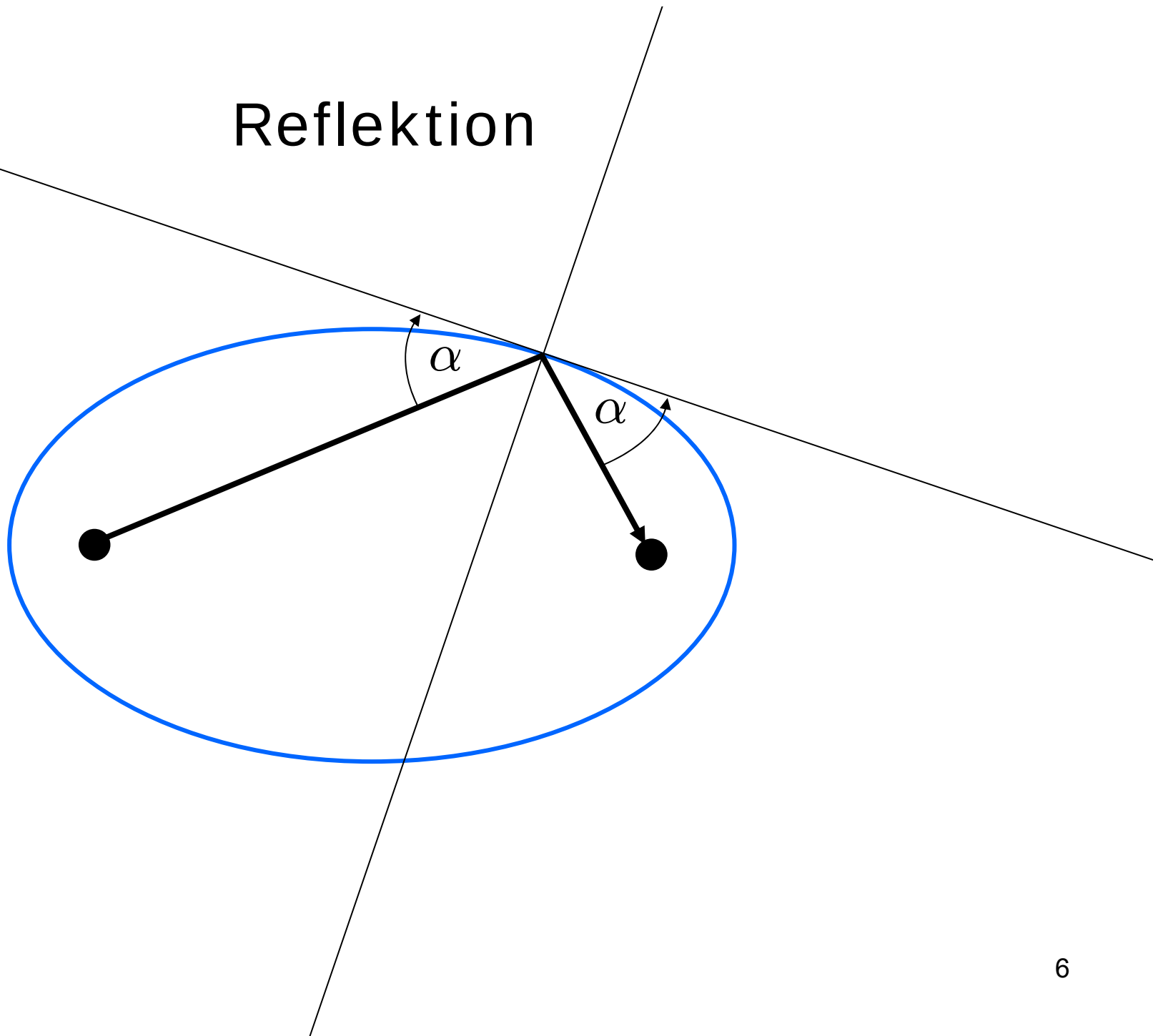
Ellipse um (0,0)
parametrisiert



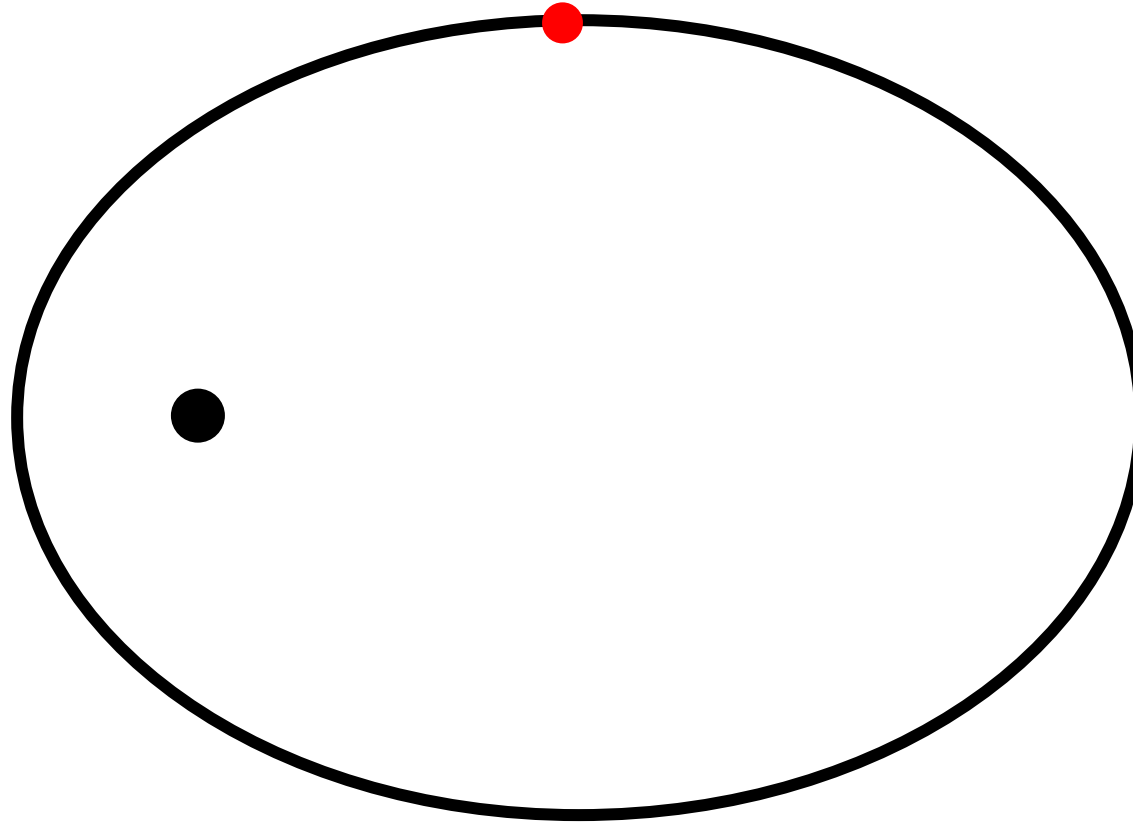
$$x = a \cdot \cos(\alpha)$$

$$y = b \cdot \sin(\alpha)$$

Reflektion

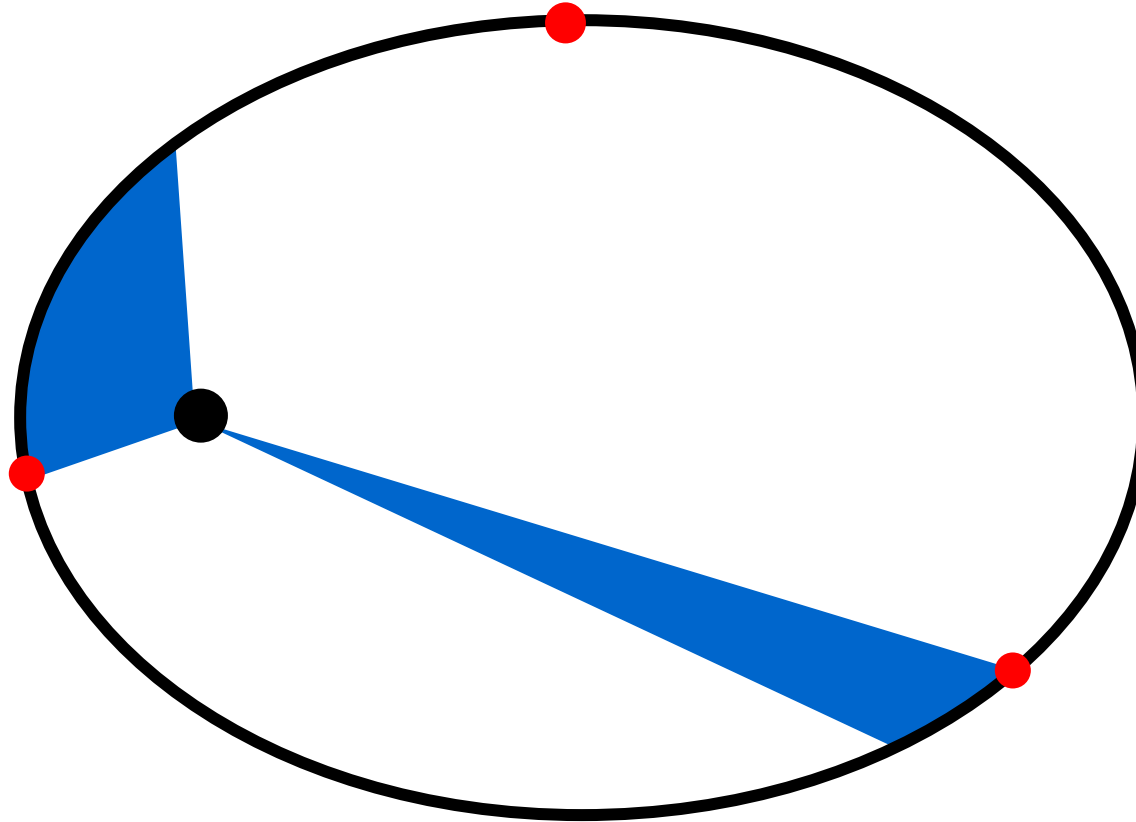


1. Keplersches Gesetz



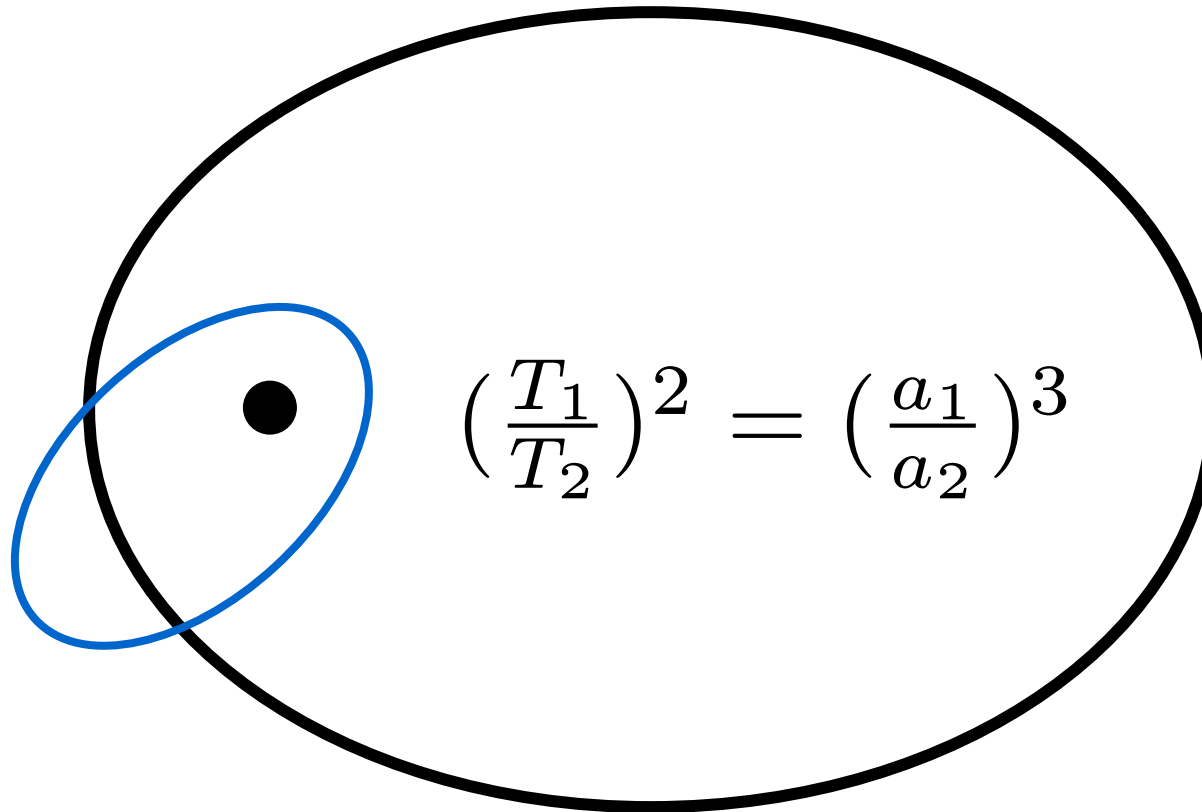
Die Planeten umkreisen die Sonne auf einer Ellipse

2. Keplersches Gesetz



In gleichen Zeiten überstreicht der Fahrstrahl gleiche Flächen

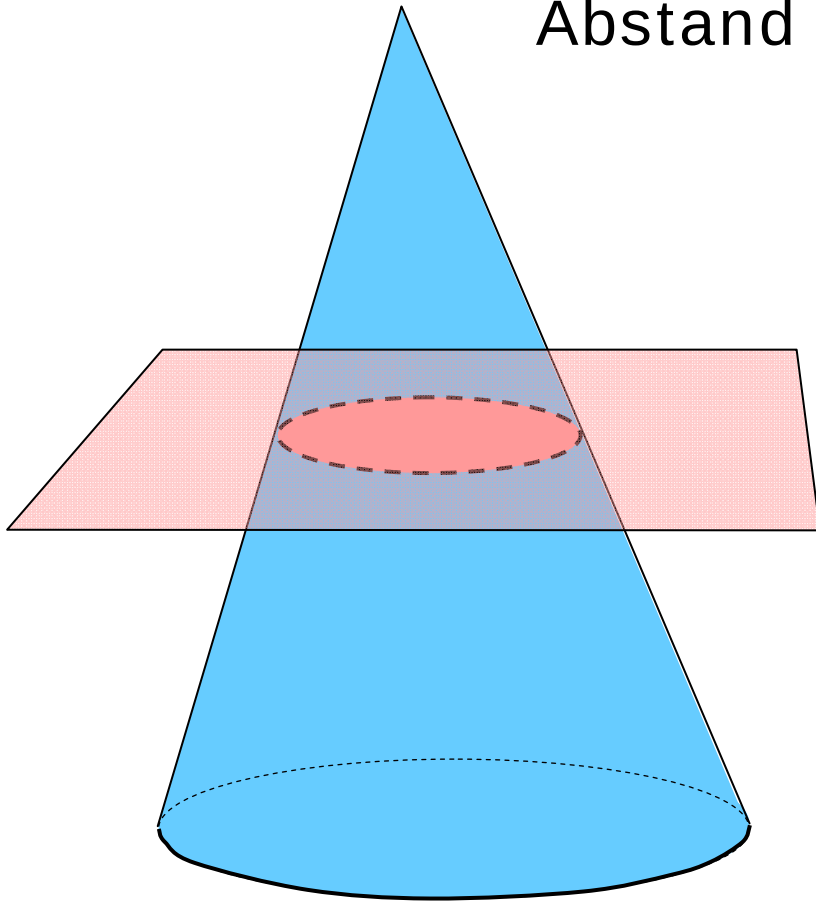
3. Keplersches Gesetz



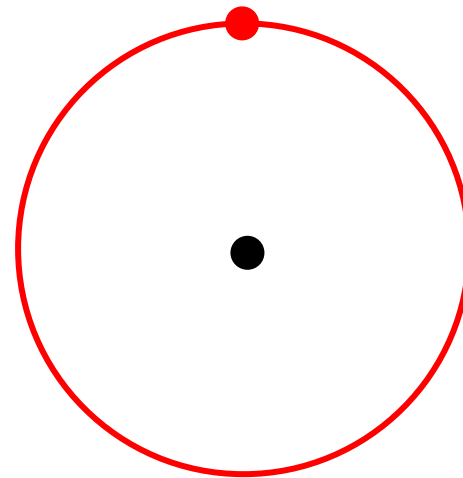
Die Quadrate der Umlaufzeiten verhalten sich
wie die Kuben der großen Halbachsen

Kegelschnitt: Kreis

Abstand zu einem Punkt ist konstant

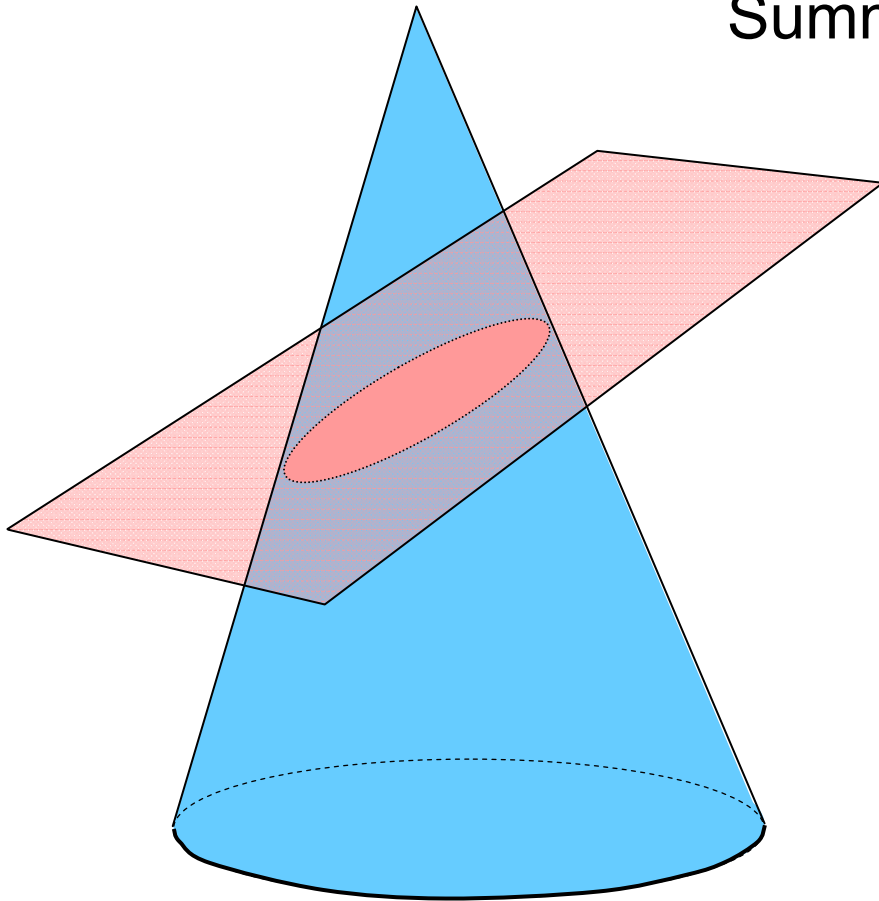


$$x^2 + y^2 = 1$$

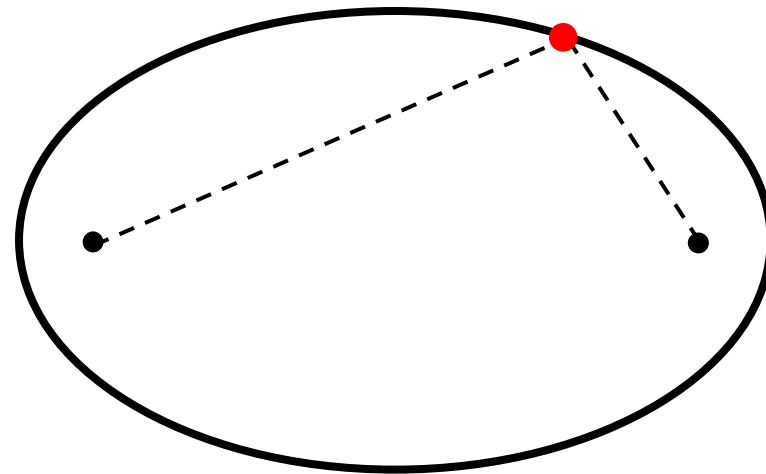


Kegelschnitt: Ellipse

Summe der Abstände zu 2 Punkten
ist konstant



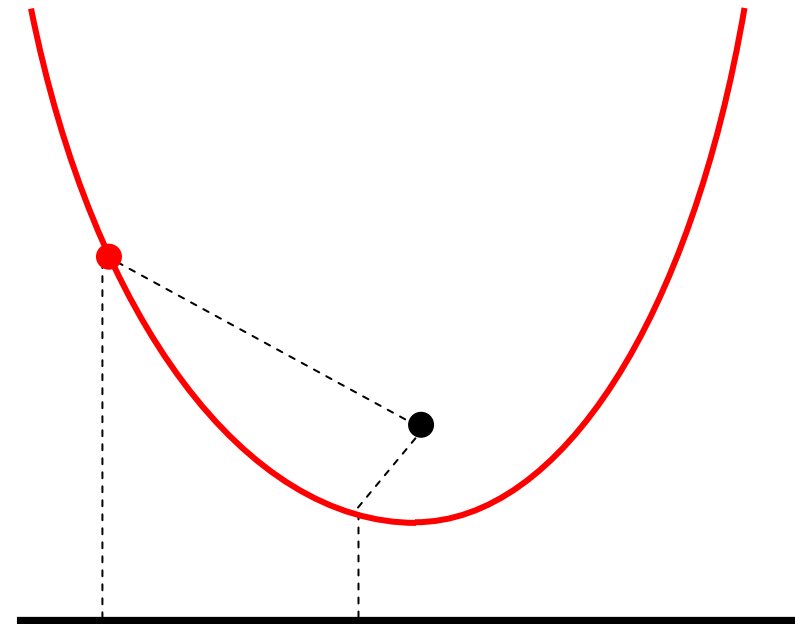
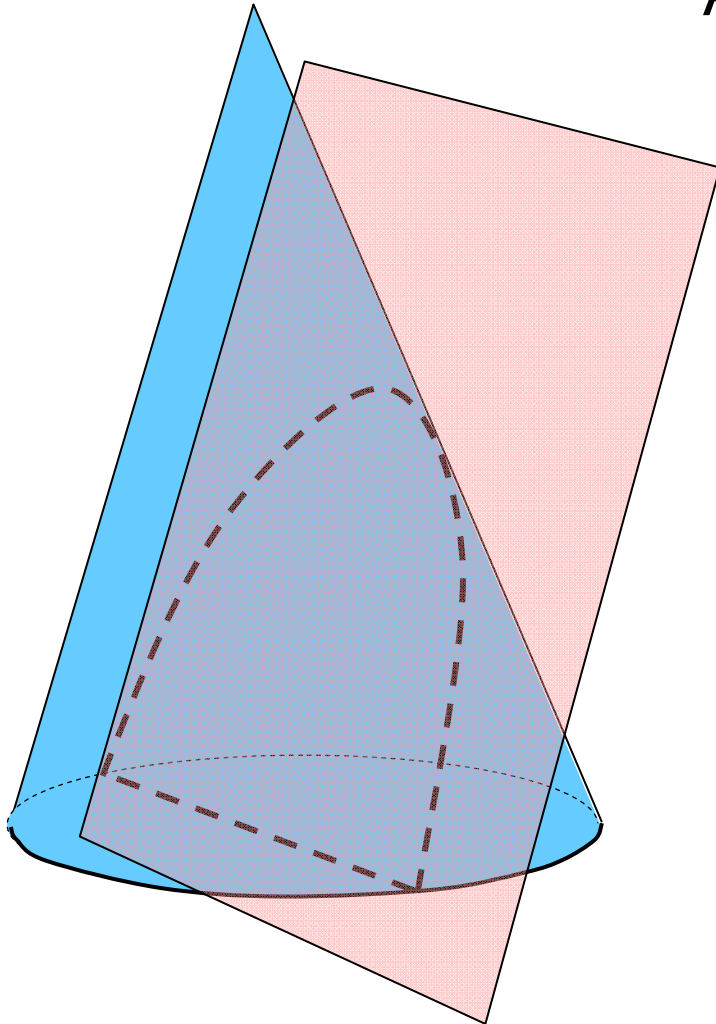
$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$



Kegelschnitt: Parabel

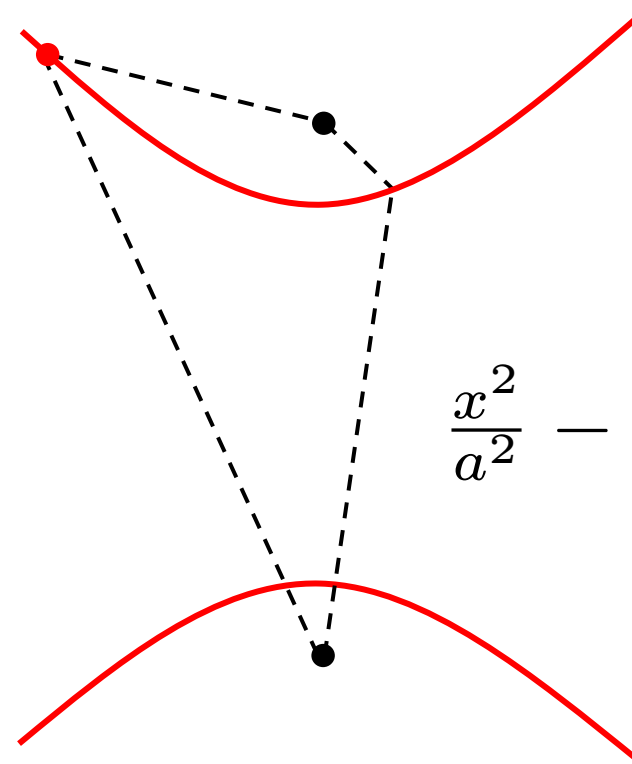
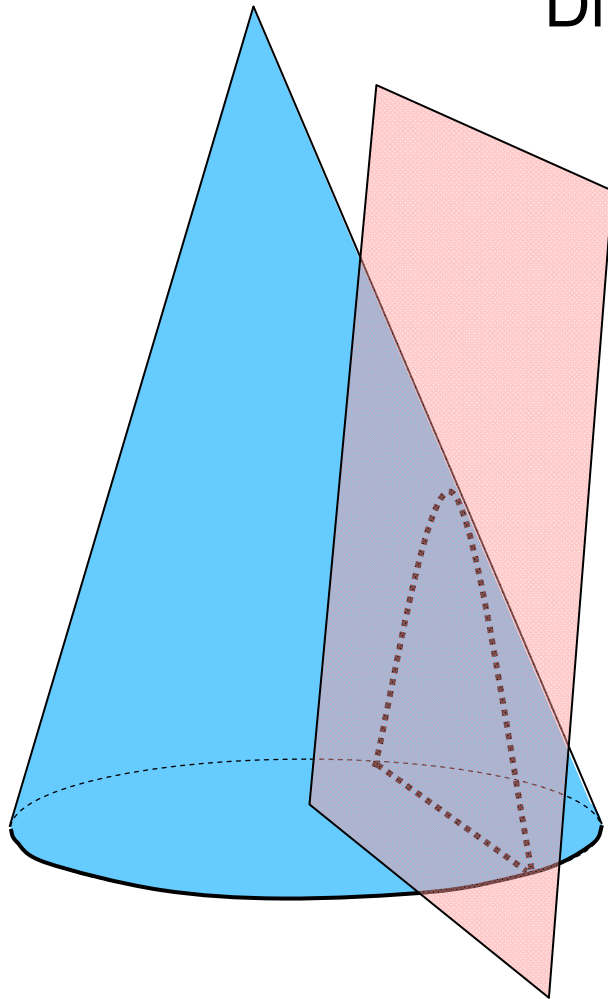
Abstand zu Punkt und Gerade
ist gleich

$$y = ax^2 + bx + c$$



Kegelschnitt: Hyperbel

Differenz der Abstände zu 2 Punkten
ist konstant



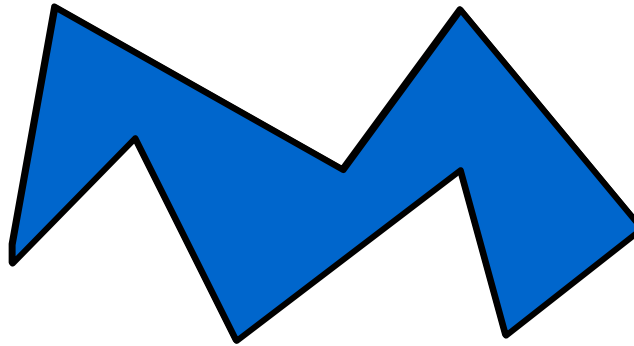
$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$$

Computergrafik SS 2016

Oliver Vornberger

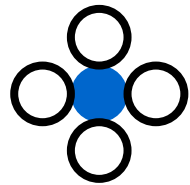
Kapitel 4: 2D-Füllen

Füllverfahren

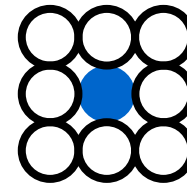


- Universelle Füllverfahren
(Zusammenhangseigenschaften)
- Scan-Line-Verfahren
(Geometrie)

Universelle Füllverfahren



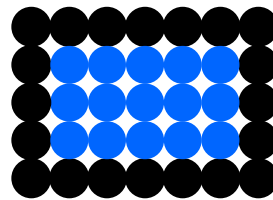
4-way-stepping



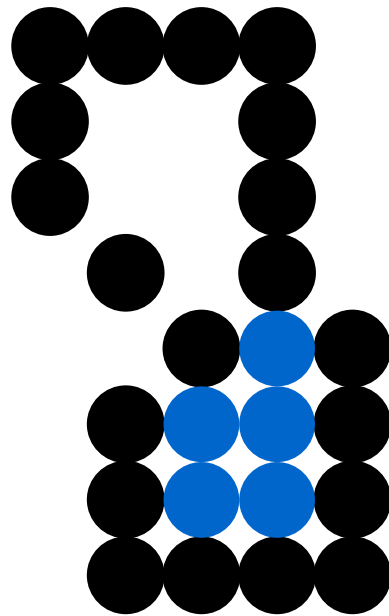
8-way-stepping

Beginnend beim Saatpixel:

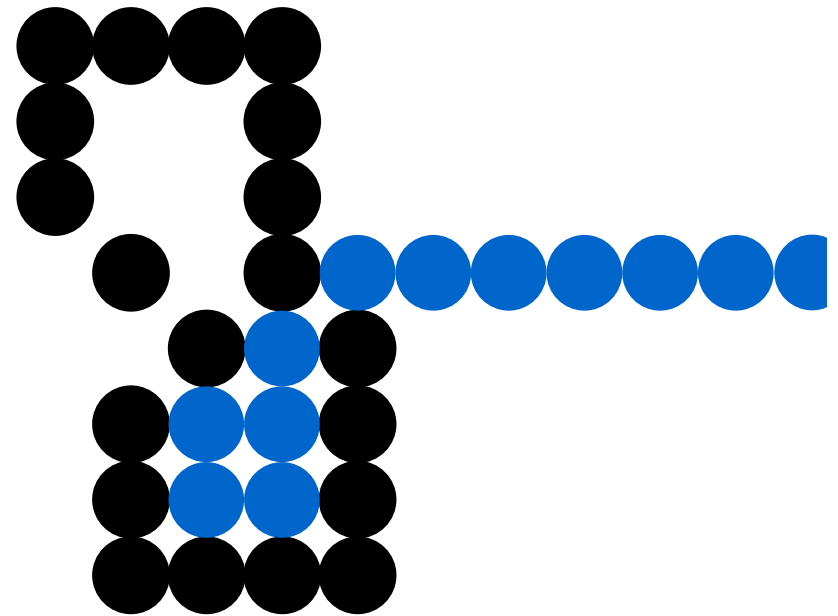
färbe alle Nachbarn, bis Umgrenzung erreicht ist.



Probleme beim universellen Füllen



4-way-stepping



8-way-stepping

Rekursives Füllen

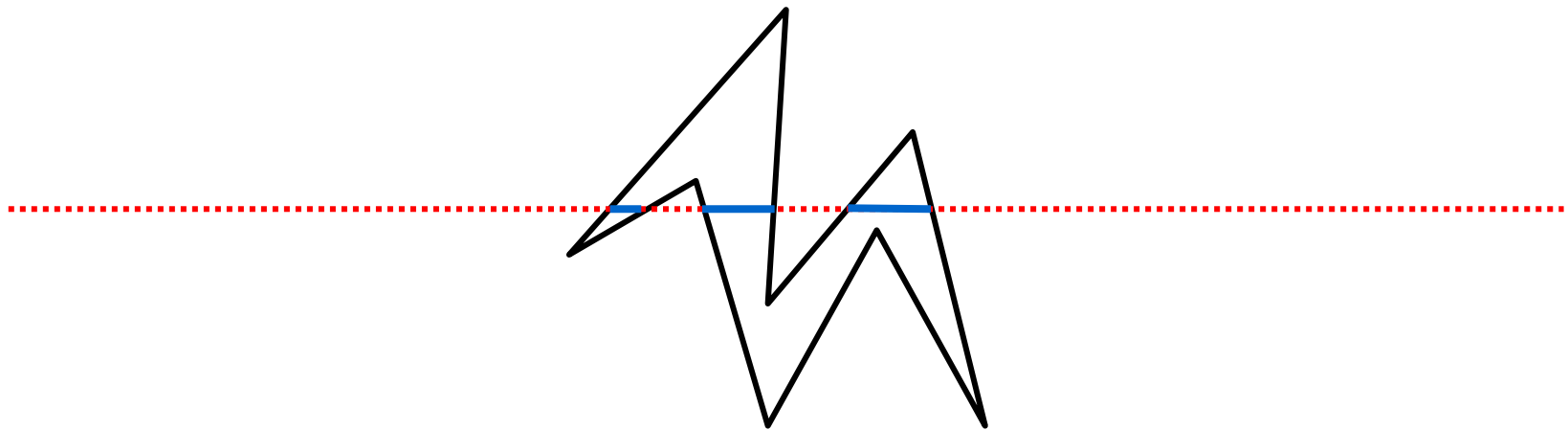
boolean rangeOK(x,y) true, falls Punkt x,y
innerhalb des Bildbereichs

boolean getPixel(x,y) true, falls Vordergrundfarbe an Punkt x,y

void setPixel(x,y) setze Vordergrundfarbe an Punkt x,y

```
public void boundaryFill(int x, int y){  
    if (rangeOk(x,y) && !getPixel(x,y)){  
        setPixel(x,y);  
        boundaryFill(x+1,y);  
        boundaryFill(x, y+1);  
        boundaryFill(x-1,y);  
        boundaryFill(x, y-1);  
    }  
}
```

Scan-Line-Verfahren

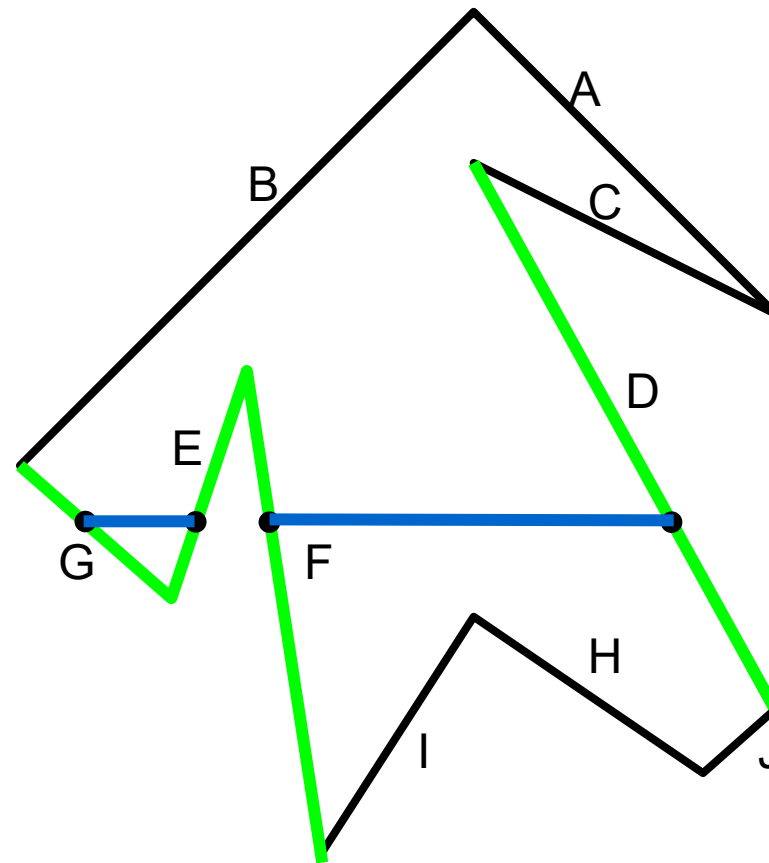


Bewege waagerechte Scan-Line von oben nach unten über das Polygon und färbe entsprechende Abschnittsgeraden

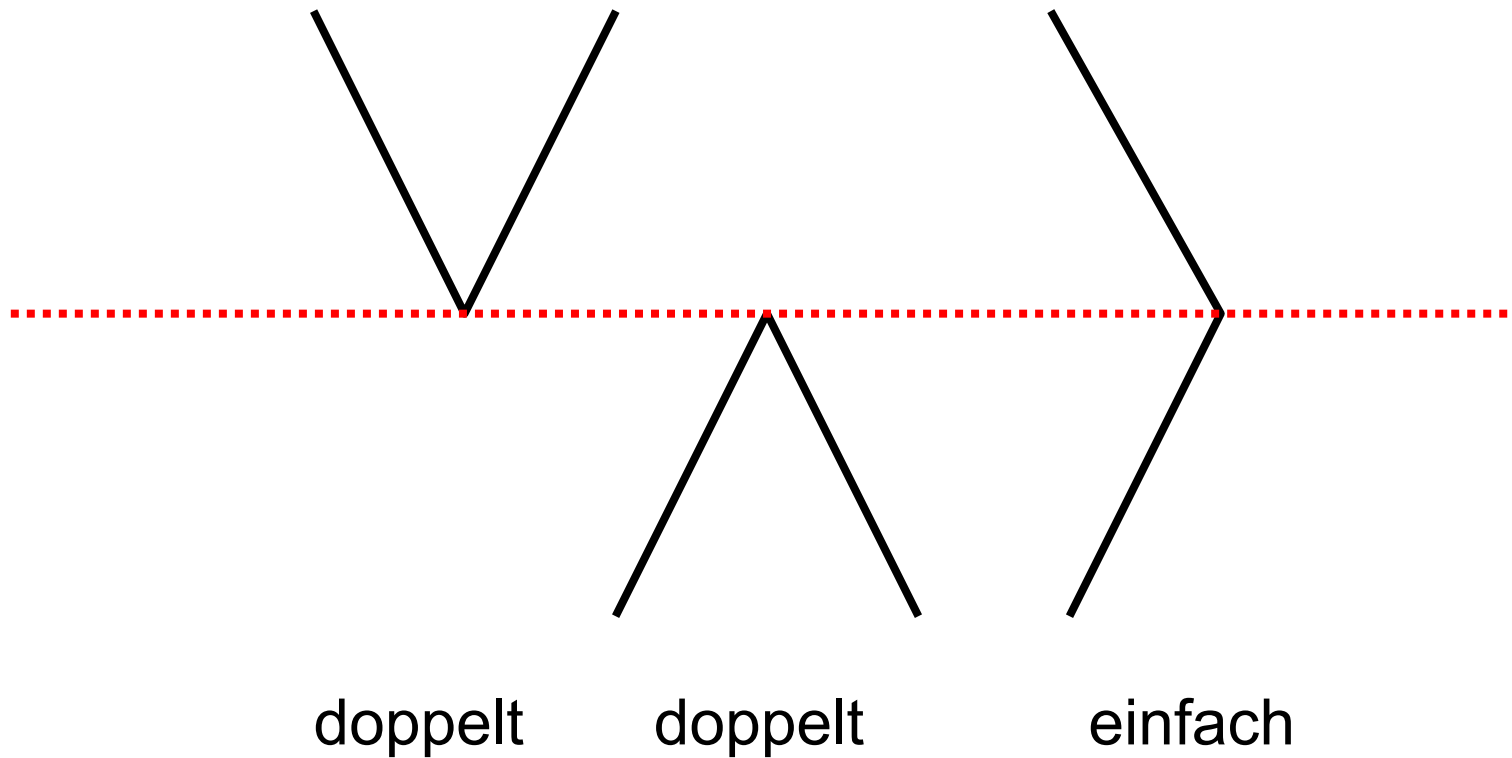
Scan-Line-Verfahren: Detail

1. Sortiere Kanten nach größtem y-Wert
2. Bewege Scan-Line von oben nach unten
3. für jede Position der Scan-Line:
 - ermittle aktive Kanten
 - berechne Schnittpunkte mit Scan-Line
 - sortiere die Schnittpunkte nach x Werten
 - färbe abwechselnd zwischen Schnittpunkten

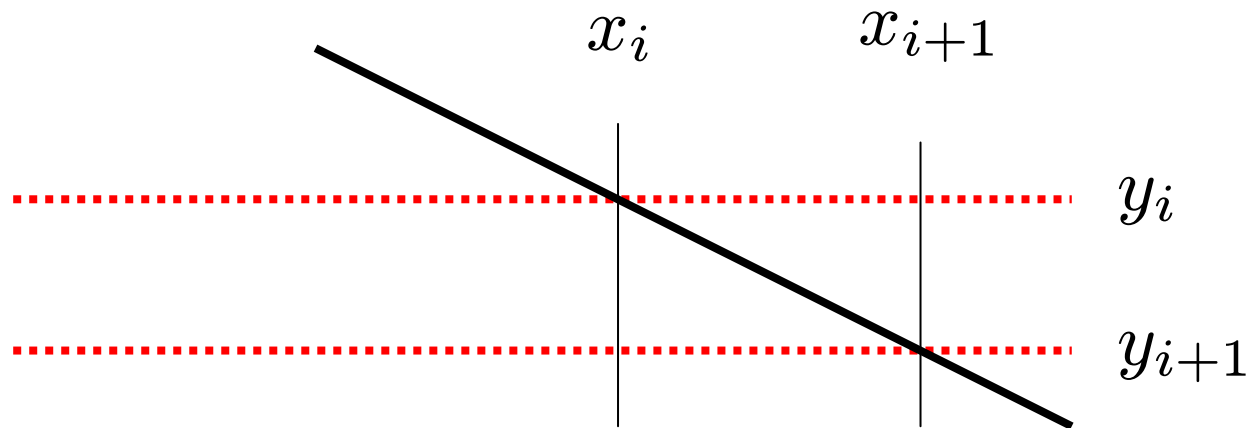
Scan-Line-Verfahren: Beispiel



Problemfälle



Schnittpunkte fortschreiben



$$s = \frac{y_i - y_{i+1}}{x_i - x_{i+1}}$$

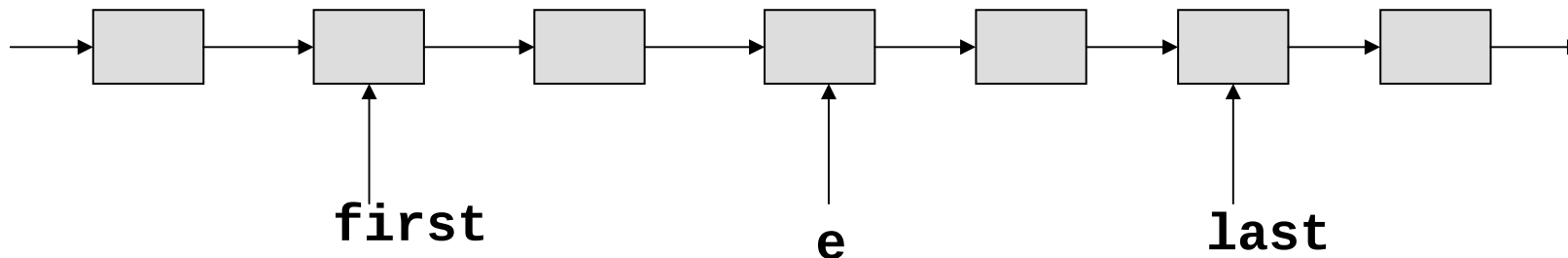
$$y_i - y_{i+1} = 1$$

$$x_i - x_{i+1} = \frac{y_i - y_{i+1}}{s}$$

$$x_{i+1} = x_i - \frac{1}{s}$$

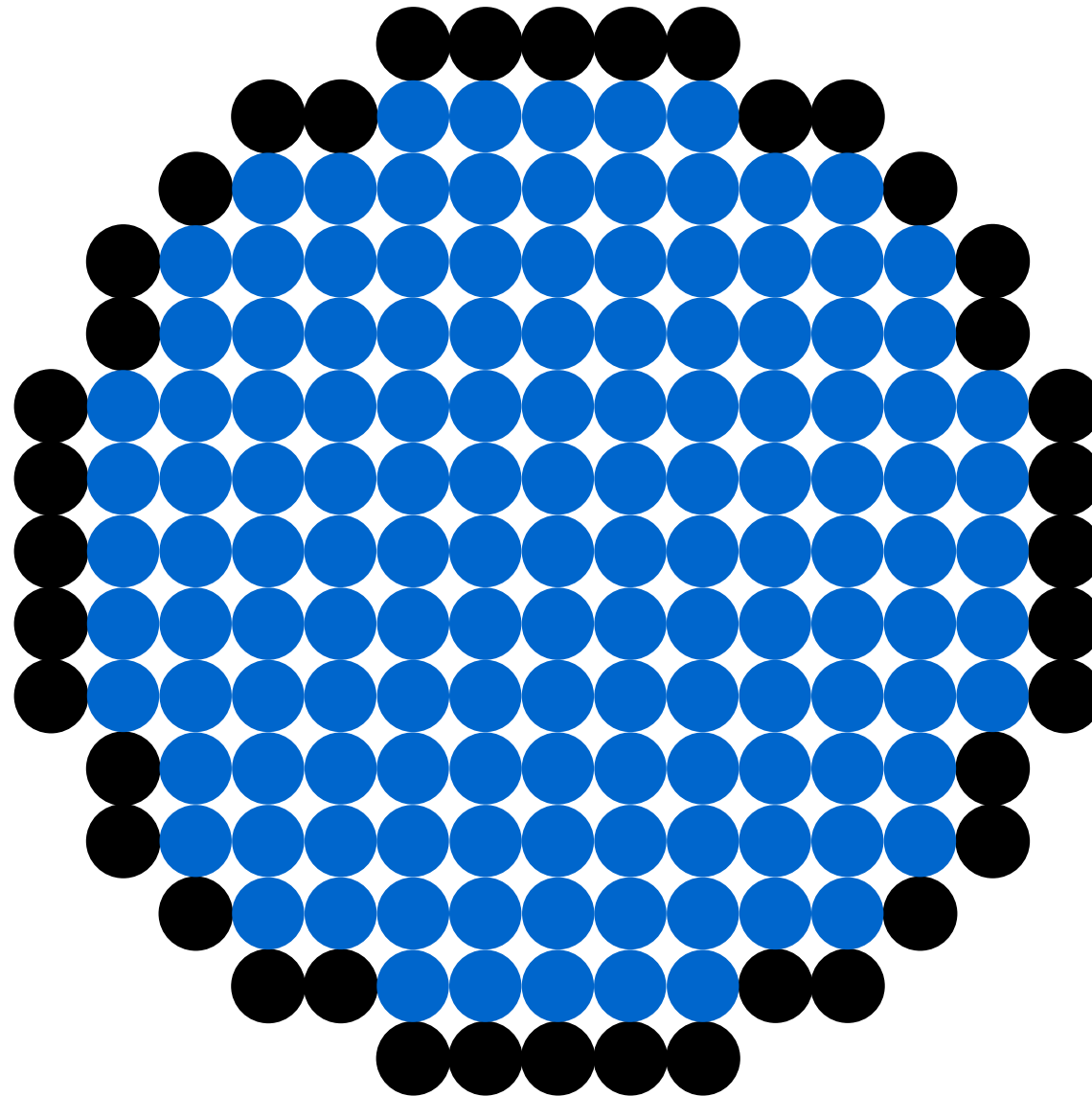
Datenstruktur für Kante

```
public class Edge {  
    int y_top;           // groesster y-Wert  
    int delta_y;         // Ausdehnung in y-Richtung  
    double delta_x;      // inverse Steigung  
    double x_int;        // errechneter Schnittpunkt  
    Edge next;           // Verweis auf naechste Kante  
}
```

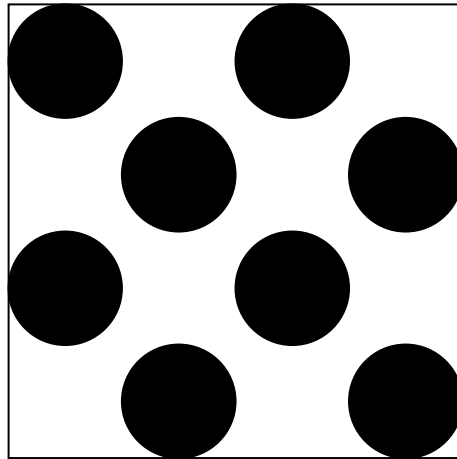


```
if (e.delta_y) > 0) {  
    e.delta_y--;  
    e.x_int = e.x_int - e.delta_x;  
    e = e.next;  
}
```

Scan-Line-Verfahren für Kreis



Dither-Matrix: Definition

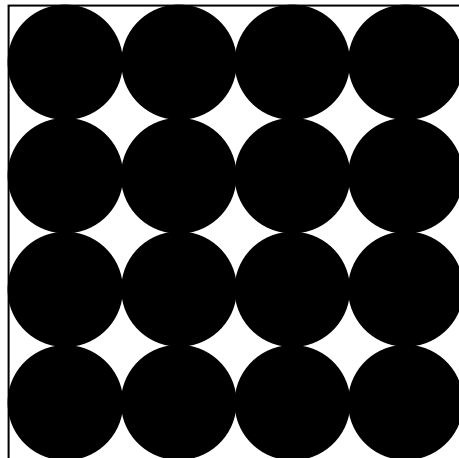


0	8	2	10
12	4	14	6
3	11	1	9
15	7	13	5

Eine $n \times n$ Dithermatrix enthält gleichmäßig verteilt alle Zahlen aus dem Intervall $[0..n^2 - 1]$

Für Grauwert $0 \leq k \leq n^2$
schwärze alle Pixel mit Eintrag $< k$

Dither-Matrix: Beispiel



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

Dither-Matrix: Konstruktion

$$D_0 = (0)$$

$U_n = n \times n$ - Matrix, besetzt mit 1

$$D_n = \begin{pmatrix} 4 \cdot D_{n-1} + 0 \cdot U_{n-1} & 4 \cdot D_{n-1} + 2 \cdot U_{n-1} \\ 4 \cdot D_{n-1} + 3 \cdot U_{n-1} & 4 \cdot D_{n-1} + 1 \cdot U_{n-1} \end{pmatrix}$$

$$D_1 = \begin{pmatrix} 0 & 2 \\ 3 & 1 \end{pmatrix}$$

$$D_2 = \begin{pmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{pmatrix}$$

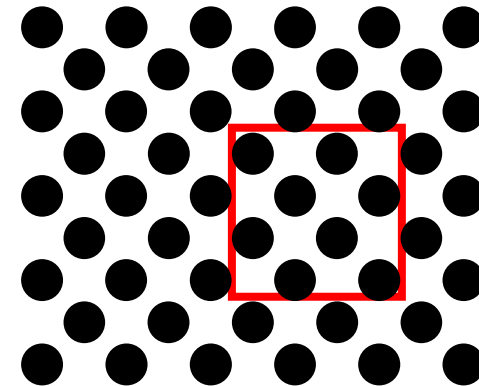
$$\begin{pmatrix} 0 & 8 & 0 & 8 \\ 12 & 4 & 12 & 4 \\ 0 & 8 & 0 & 8 \\ 12 & 4 & 12 & 4 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 \\ 3 & 3 & 1 & 1 \\ 3 & 3 & 1 & 1 \end{pmatrix}$$

Dither-Matrix: Aufruf

Gegeben $N \times N$ Dithermatrix D .

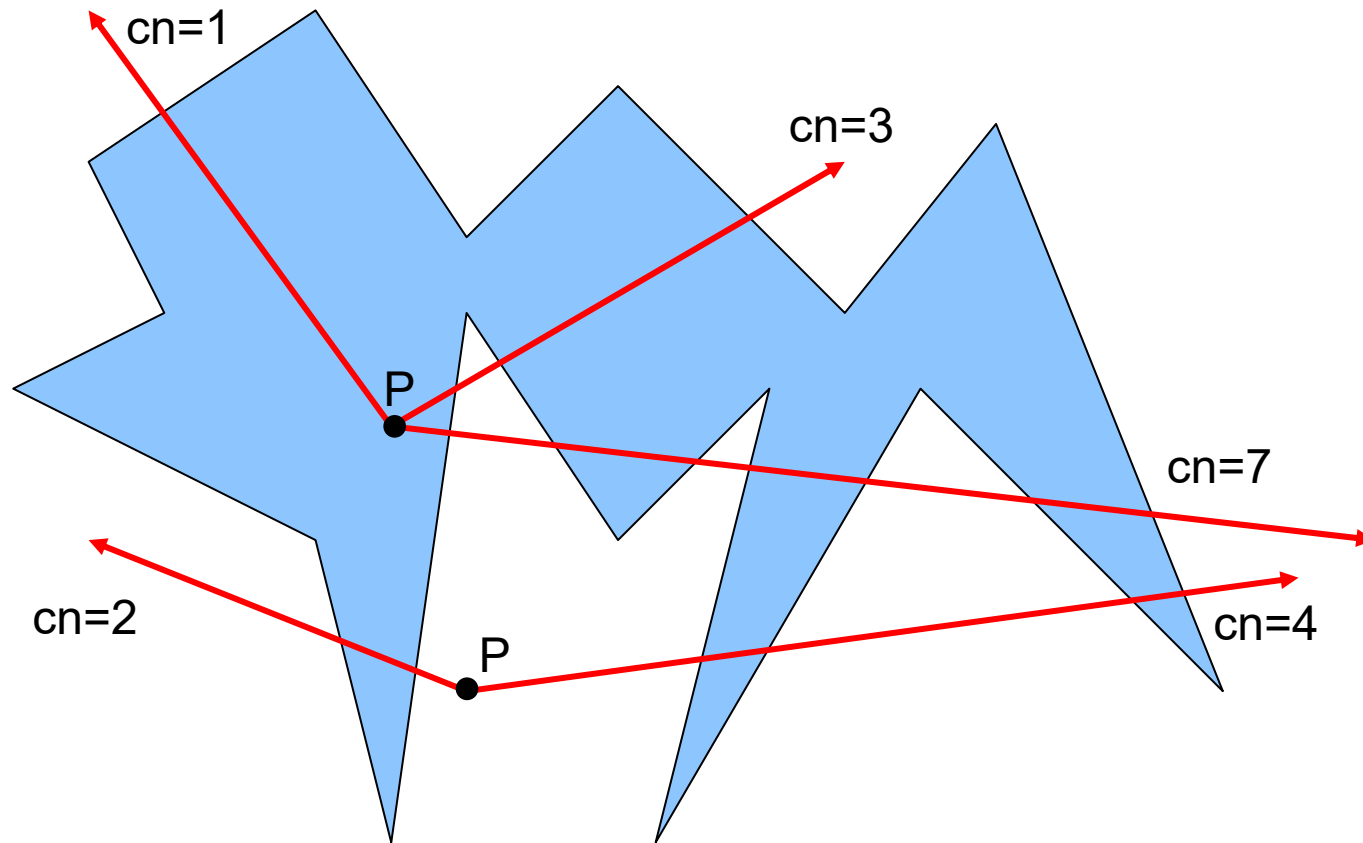
Einfärben an Position (x,y) mit Grauwert k :

```
if (D[x%N][y%N] < k)
    setPixel(x,y);
else
    delPixel(x,y);
```

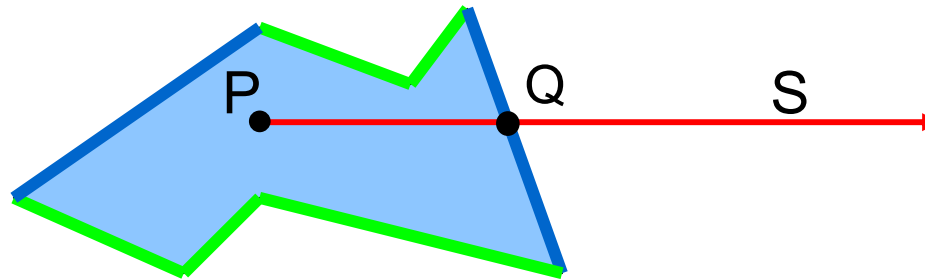


[~cg/2016/skript/Applets/2D-basic/App.html](http://cg/2016/skript/Applets/2D-basic/App.html)

Punkt in Polygon



Kreuzungszahl berechnen



Sei S der von P nach rechts gehende Strahl

Für jede Polygonkante von P_1 nach P_2 :

- falls P_1 und P_2 oberhalb: kein Schnittpunkt

- falls P_1 und P_2 unterhalb: kein Schnittpunkt

- falls P_1 und P_2 auf verschiedenen Seiten:

 - berechne Schnittpunkt Q mit S

 - falls rechts von P : erhöhe Kreuzungszahl

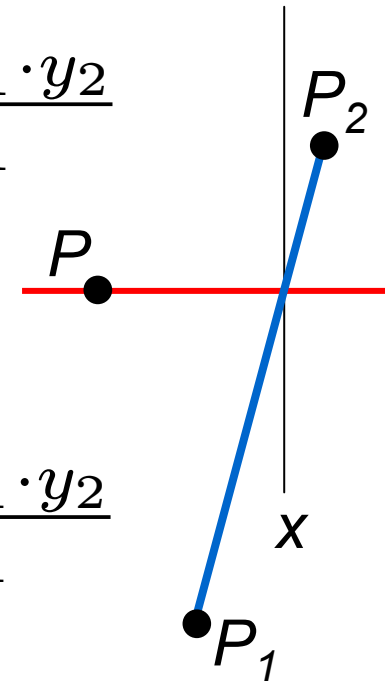
Schnittpunkt berechnen

$$f(x) = \frac{y_2 - y_1}{x_2 - x_1} \cdot x + \frac{x_2 \cdot y_1 - x_1 \cdot y_2}{x_2 - x_1}$$

$$f(x) = y$$

$$y = \frac{y_2 - y_1}{x_2 - x_1} \cdot x + \frac{x_2 \cdot y_1 - x_1 \cdot y_2}{x_2 - x_1}$$

$$x = \frac{y \cdot (x_2 - x_1) - x_2 \cdot y_1 + x_1 \cdot y_2}{y_2 - y_1}$$



public boolean contains (int x, int y)

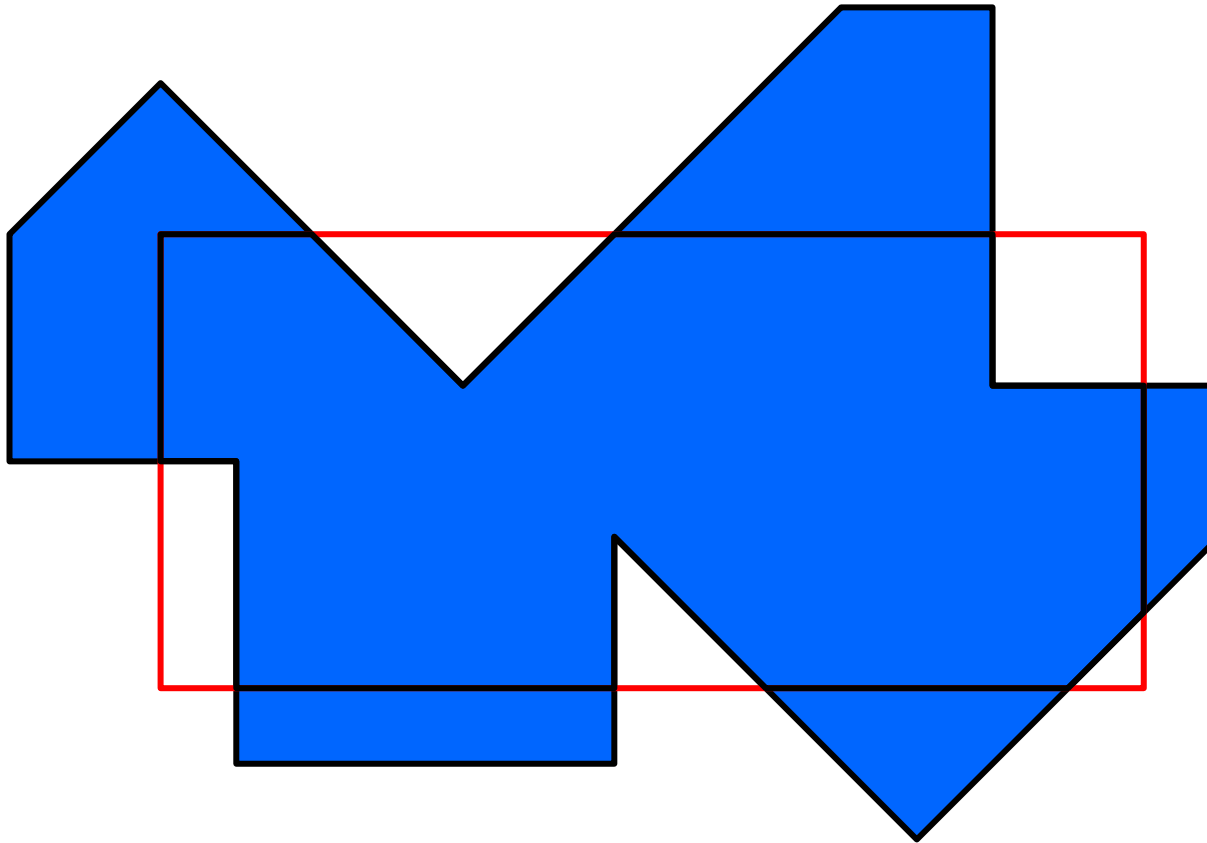
```
x1 = xpoints[n-1]; y1 = ypoints[n-1];
x2 = xpoints[0];   y2 = ypoints[0];
boolean inside = false;
boolean startUeber = y1 >= y ? true : false;
for (i=1; i<n; i++) {
    boolean endUeber = y2 >= y ? true : false;
    if ((startUeber != endUeber &&
        (double)(y*(x2-x1)- x2*y1 + x1*y2)/(y2-y1)>=x))
        inside = !inside;
    startUeber = endUeber;
    y1=y2; x1=x2; x2=xpoints[i]; y2=ypoints[i];
}
return inside;
```


Computergrafik SS 2015

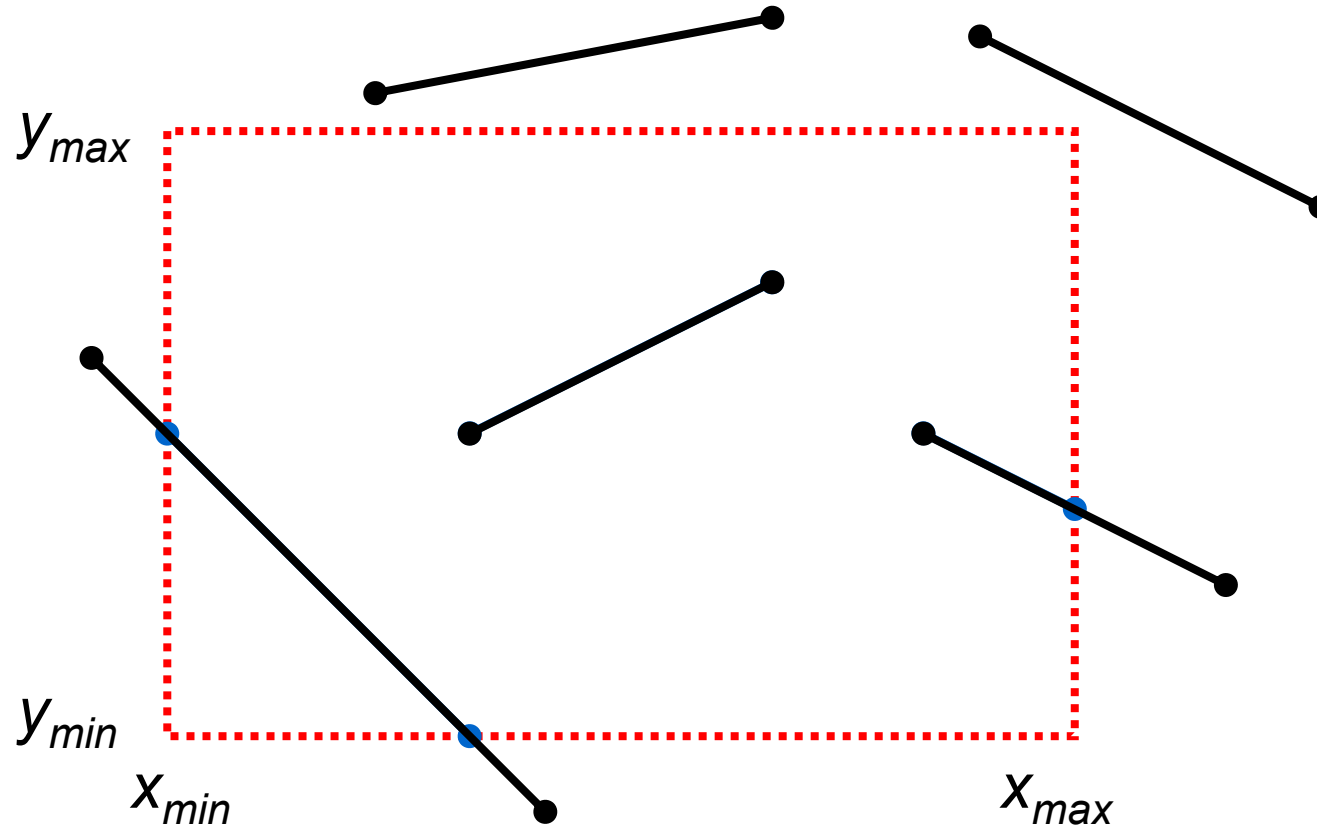
Oliver Vornberger

Kapitel 5: 2D-Clipping

2D-Clipping



Clipping von Linien



Region Code: Definition

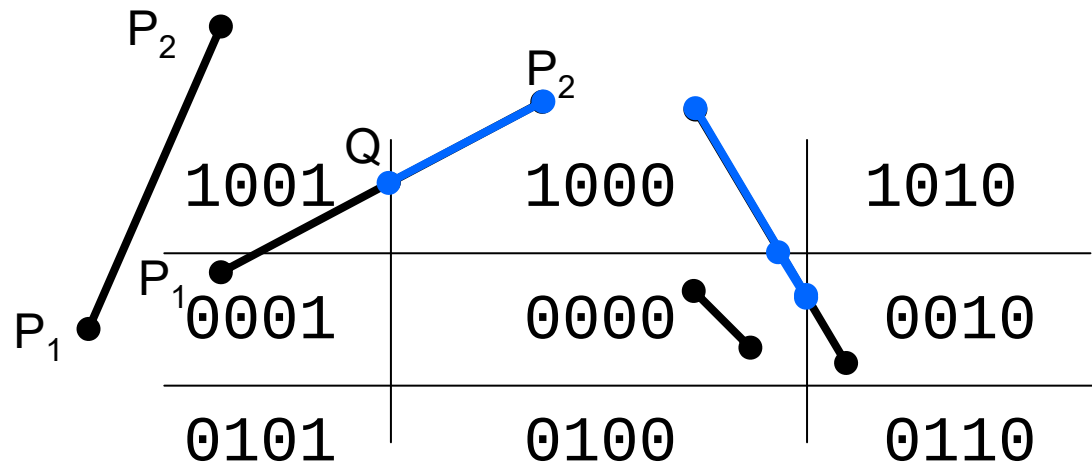
Bit 0: links Bit1:rechts Bit2: unten Bit3: oben

1001	1000	1010
y_{max}		
0001	0000	0010
y_{min}		
0101	0100	0110
x_{min}		x_{max}

Region Code: Berechnung

```
private static final byte CENTER = 0;
private static final byte LEFT   = 1;
private static final byte RIGHT  = 2;
private static final byte BOTTOM = 4;
private static final byte TOP    = 8;
public byte region_code (int x, int y) {
    byte c = CENTER;
    if (x < xmin) c = LEFT;
    if (x > xmax) c = RIGHT;
    if (y < ymin) c = c | BOTTOM;
    if (y > ymax) c = c | TOP;
    return c;
}
```

Cohen & Sutherland

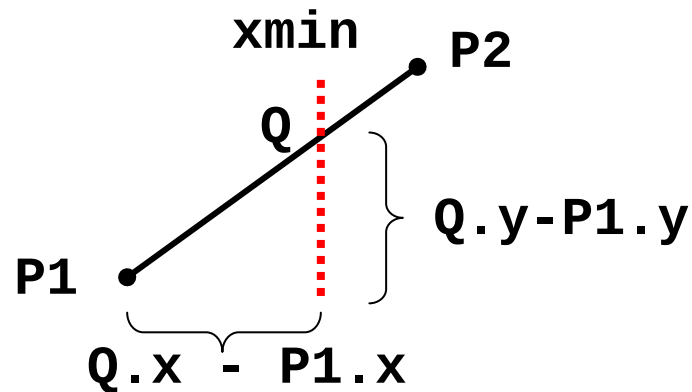


falls $\text{code}(P_1) \&\& \text{code}(P_2) \neq 0 \Rightarrow$ komplett auerhalb

falls $\text{code}(P_1) \parallel \text{code}(P_2) = 0 \Rightarrow$ komplett innerhalb

sonst: berechne Schnittpunkt Q und teste Restlinie erneut

Schnittpunkte



$$slope = \frac{Q.y - P1.y}{Q.x - P1.x}$$

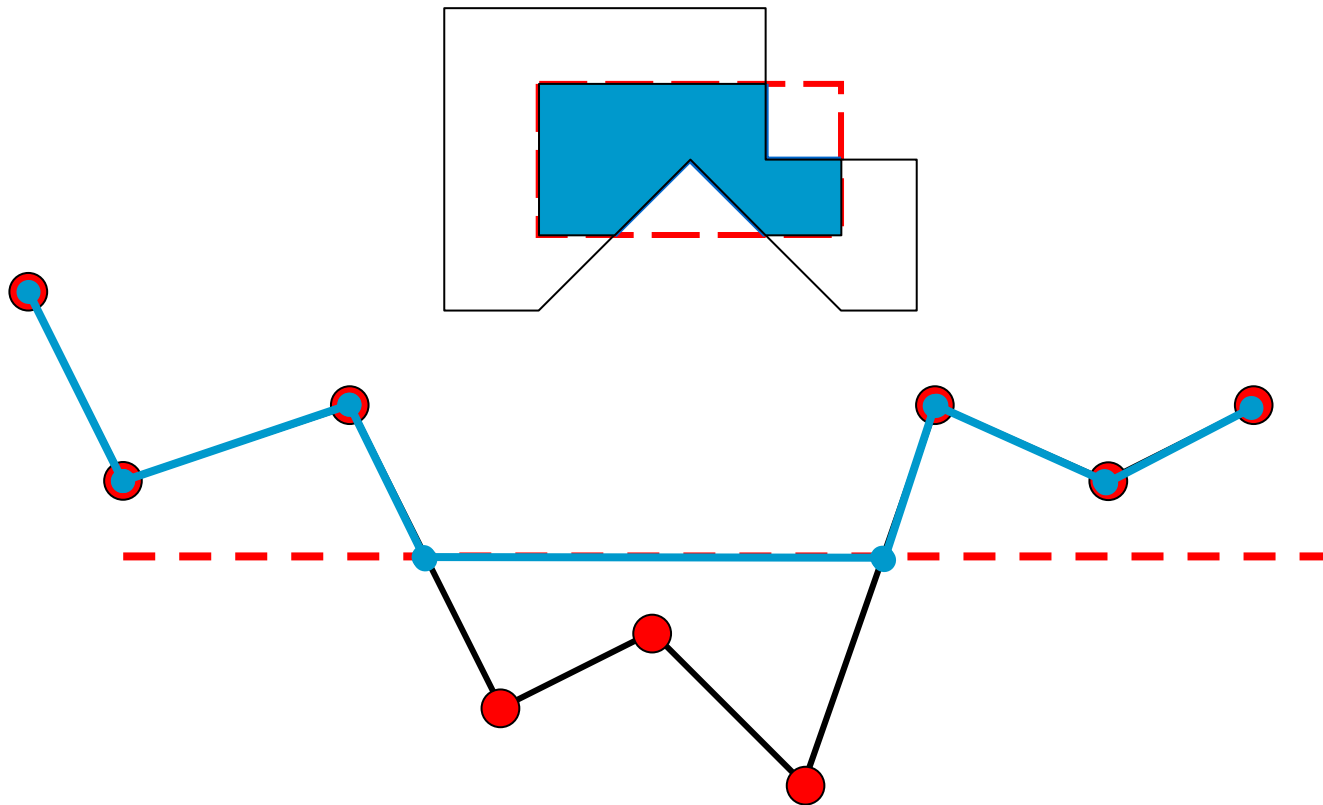
$$slope = \frac{P2.y - P1.y}{P2.x - P1.x}$$

```
slope = (double)(P2.y - P1.y)/(P2.x - P1.x);  
Q.x    = xmin  
Q.y    = (int)(Q.x - P1.x)*slope + P1.y
```

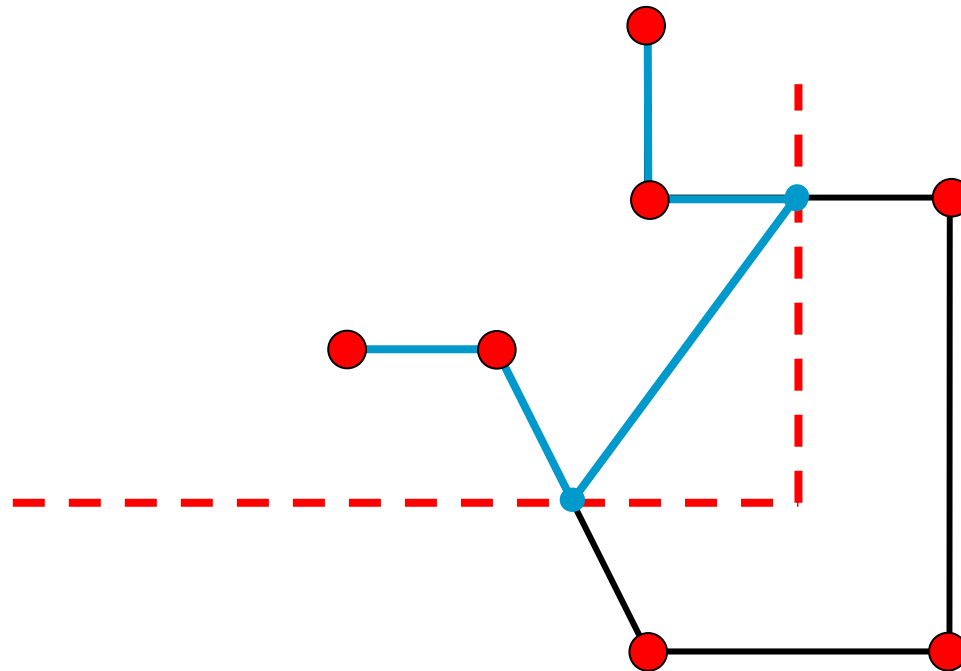
Cohen-Sutherland

```
boolean cohen_sutherland(Point P1,  
                        Point P2,  
                        Point Q1,  
                        Point Q2){  
    // clippt Gerade P1,P2 am Fenster  
    // liefert true, falls sichtbar  
    // liefert in Q1,Q2 den sichtbaren Teil  
    ...  
}
```

Clipping von Polygonen



Problem bei Clip-Fenster-Ecken



Sutherland & Hodgman

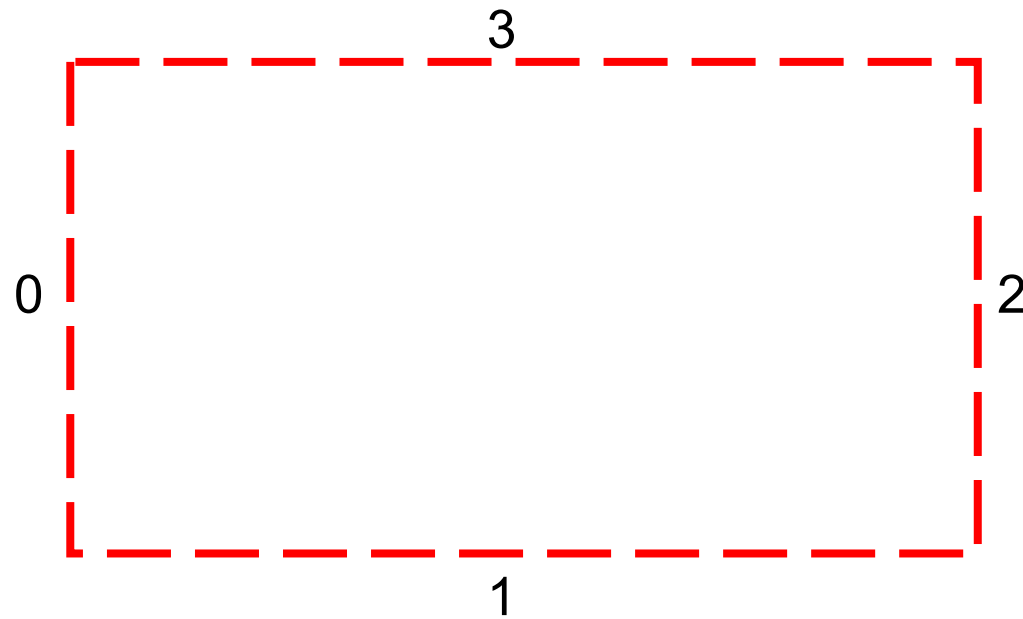
für eine Clipping-Gerade E

für jeden Polygonpunkt P_i :

falls P_i sichtbar: übernimm P_i

falls Kante von P_i zu P_{i+1} E schneidet:
übernimm Schnittpunkt

4 Clipping-Kanten



Sichtbarkeitstest

```
boolean On_Visible_Side(  
    Point P, int wert, int fall) {  
    switch (fall) {  
        case 0: return (P.x >= wert);  
        case 1: return (P.y >= wert);  
        case 2: return (P.x <= wert);  
        case 3: return (P.y <= wert);  
    }  
}
```

Schnittpunkt

```
boolean intersection(  
    Point P1, Point P2,  
    int wert, int fall, Point I) {  
    ...  
    P1_vis = On_Visible_Side(P1,wert,fall);  
    P2_vis = On_Visible_Side(P2,wert,fall);  
    ...  
    slope =(double)(P2.y-P1.y)/  
            (double)(P2.x-P1.x);  
    if (fall %2 == 0) {  
        I.x = (int) wert;  
        I.y = (int)(wert-P1.x)*slope + P1.y;  
    }  
    ...  
}
```

Aufruf von Sutherland_hodgman

```
int n;           // Zahl der Eckpunkte
Point[] points;  // Polygon

n = sutherland_hodgman(n, points, xmin, 0);
n = sutherland_hodgman(n, points, ymin, 1);
n = sutherland_hodgman(n, points, xmax, 2);
n = sutherland_hodgman(n, points, ymax, 3);
```

4 x Clippen

