# Praktikum 1: Linien

Kopieren Sie Ihre Klasse `int2` aus dem Praktikum 0 in den Code-Ordner dieser Aufgabe!

## Aufgabe 1 Ursprungslinien

a) In `A1Lines.pde` finden Sie die Methode `draw()`. Stellen Sie sicher, dass nur

```
test_x_fast_a_at_origin_dx_positive_dy_positive()
```

ausgeführt wird!

b) Implementieren Sie die Methode

```
drawLine(int[] framebuffer, int w, int2 b, int col)
```

in der Klasse `LineRasterizer`. Diese soll eine Ursprungslinie, die an der Stelle `b` endet, zeichnen. Die $x$-Richtung soll dabei die *schnelle* Richtung sein, das heißt, $x$ wächst schneller als $y$ entlang der Linie, bzw.

$$|m| = \frac{|d_y|}{|d_x|} = \frac{|b_y - a_y|}{|b_x - a_x|} \leq 1.$$

c) Rufen Sie die eben implementierte Methode in der Methode `public static final void drawLine(int[] framebuffer, int w, int h, int2 a, int2 b, int col)` auf!

d) Vergleichen Sie Ihre Ausgabe mit nachstehendem Referenzbild!

*Hinweis:* Mit `framebuffer[y * w + x] = col` setzt die Farbe col an die Pixel-Stelle $[x, y]^\top$ im `framebuffer` mit Zeilenlänge $w$ Pixel.

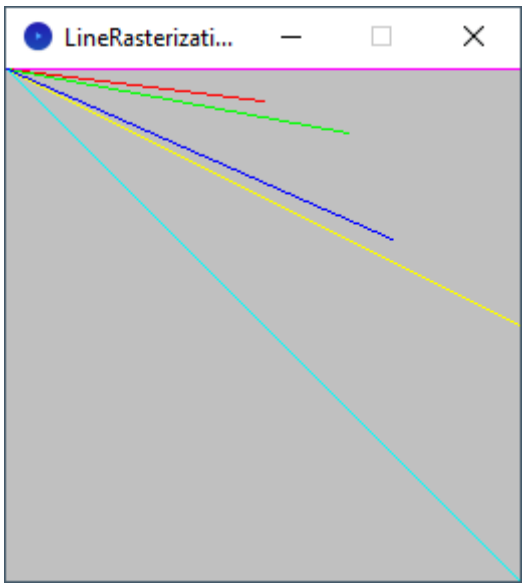


**Abbildung 1:** Ursprungslinen in einem Koordinatensystem dessen Ursprung sich oben rechts befindet.

## Aufgabe 2 Linien im 1. Oktanten

a) Deaktivieren Sie nun die Test-Methode

```
test_x_fast_a_at_origin_dx_positive_dy_positive()
```

und aktivieren Sie die Test-Methode `test_x_fast_dx_pos_dy_pos()`.

b) Erweitern Sie Ihren Algorithmus zum Linienzeichen nun so, dass Linien bei einem Punkt $a$ beginnen und einem Punkt $b$ enden! Nennen Sie dazu

```
drawLine(int[] framebuffer, int w, int h, int2 a, int2 b, int col)
```

in

```
drawLine_x_fast(int[] framebuffer, int w, int2 a, int2 b, int col)
```

um!

c) Testen Sie Ihr Programm und stellen Sie sicher, dass Sie folgende Ausgabe erhalten.


**Abbildung 2:** Ergebnisbild im 1. Oktanten

## Aufgabe 3 Linie im 5. Oktanten

Erweitern Sie nun die Fähigkeit Ihres Algorithmus um Linien im 5. Oktanten zu zeichnen. Ändern Sie dazu `drawLine_x_fast` geeignet. Als Test-Methode dient `test_x_fast_dx_neg_dy_pos`, welches nachstehendes Bild erzeugt.

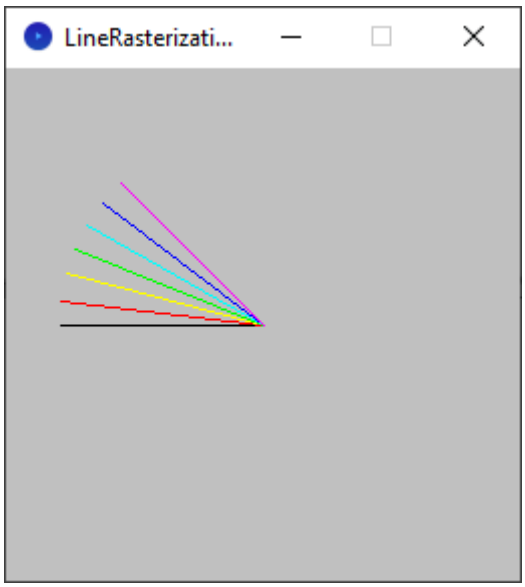


**Abbildung 3:** Ergebnisbild im 5. Oktanten

## Aufgabe 4 Linie im 4. Und 8. Oktanten

Um Linien im 4. Und 8. Oktanten zu testen, verwenden Sie die Methoden

`test_x_fast_dx_pos_dy_neg()`

und

`test_x_fast_dx_neg_dy_neg()`

um folgendes Bild zu erhalten:

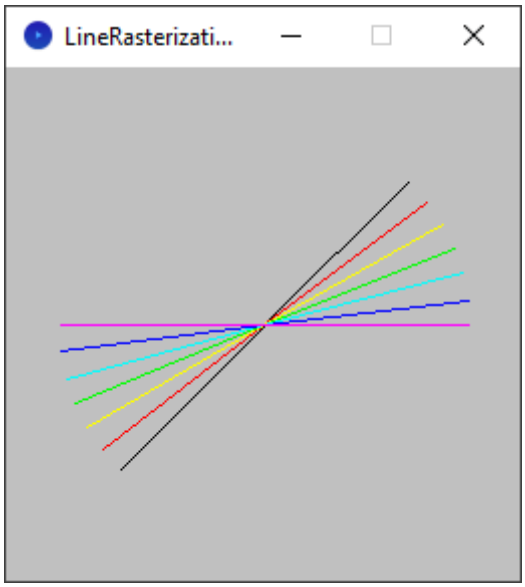


**Abbildung 4:** Ergebnis für den 4. und 8. Oktanten

Wahrscheinlich müssen Sie dazu zunächst die Methode `drawLine_x_fast` anpassen. Bisher wird nämlich $y$ nie dekrementiert. Linien im 4. und 8. Oktanten haben aber $d_y < 0$. Passen Sie Ihren Code an, um diesen Fall abzudecken!

## Aufgabe 5 Linie im 2., 3., 6. und 7. Oktanten

Für Linien aus den verbleibenden Oktanten gilt $|m| > 1$, also ist dort die y Richtung schneller als die $x$-Richtung. Implementieren Sie dazu eine neue private Methode `drawLine_y_fast`, welche diesen Fall behandelt. Rufen Sie in drawLine, die geeigneten Methode `drawLine_x_fast` bzw. `drawLine_y_fast` auf.

Wenn Sie die passenden Tests aktivieren, erhalten Sie folgendes Bild.

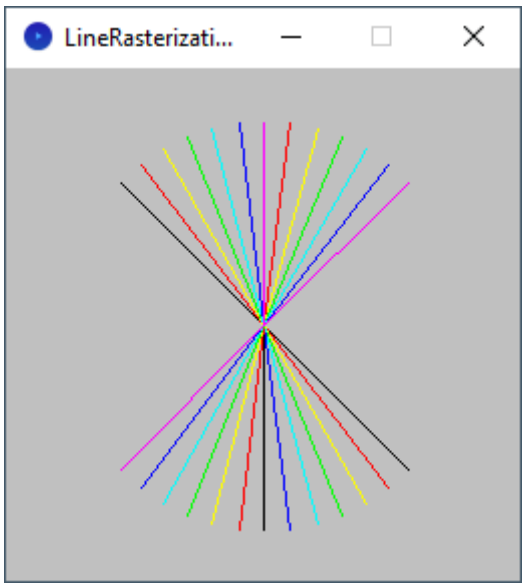


**Abbildung 5:** Ergebnisbild für den 2., 3., 6., und 7. Oktanten

## Aufgabe 6 Bonus

a) Eine Animation können Sie mit der Methode `test_animation_NoClip` betrachten.

b) In `LineRasterizerMeyer.java` finden Sie eine kompakte Version des Bresenham Algorithmus. Die innere `for`-Schleife von `drawLine(int[] framebuffer, int lu, int lv, int ru, int rv, int ofs, int col)` hat geringe arithmetische Dichte, benötigt keine `if`-Anweisungen und verzichtet gänzlich auf Multiplikation.