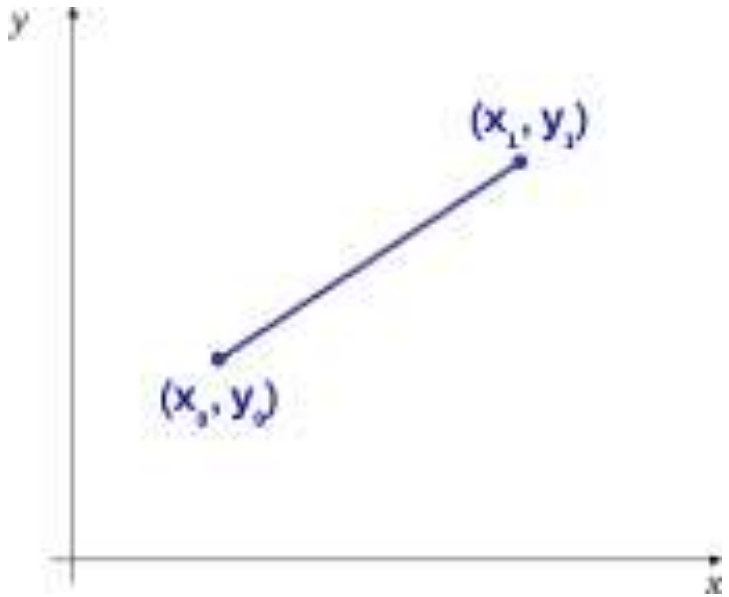
Rasterization

- Elimizde vektörel grafik bilgileri var
- Ancak ekran piksel bilgileriyle çalışıyor
- Vektörel grafik bilgilerini piksel bilgilerine çevirmemiz gerekiyor
- Bu işleme rasterization deniliyor (Pikselleştirme olarak çevrilebilir)

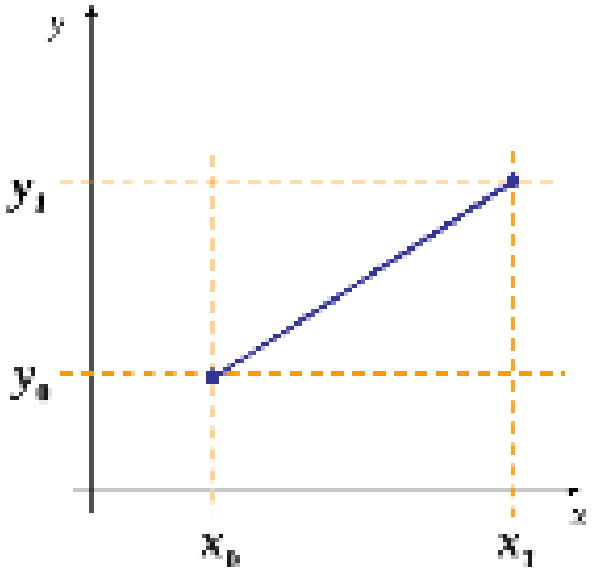
Doğru çizme

- Doğru bilgisayarda başlangıç ve bitiş noktalarıyla temsil edilmektedir.
- Bu noktaların koordinat bilgileri (x, y) şeklinde saklanır.



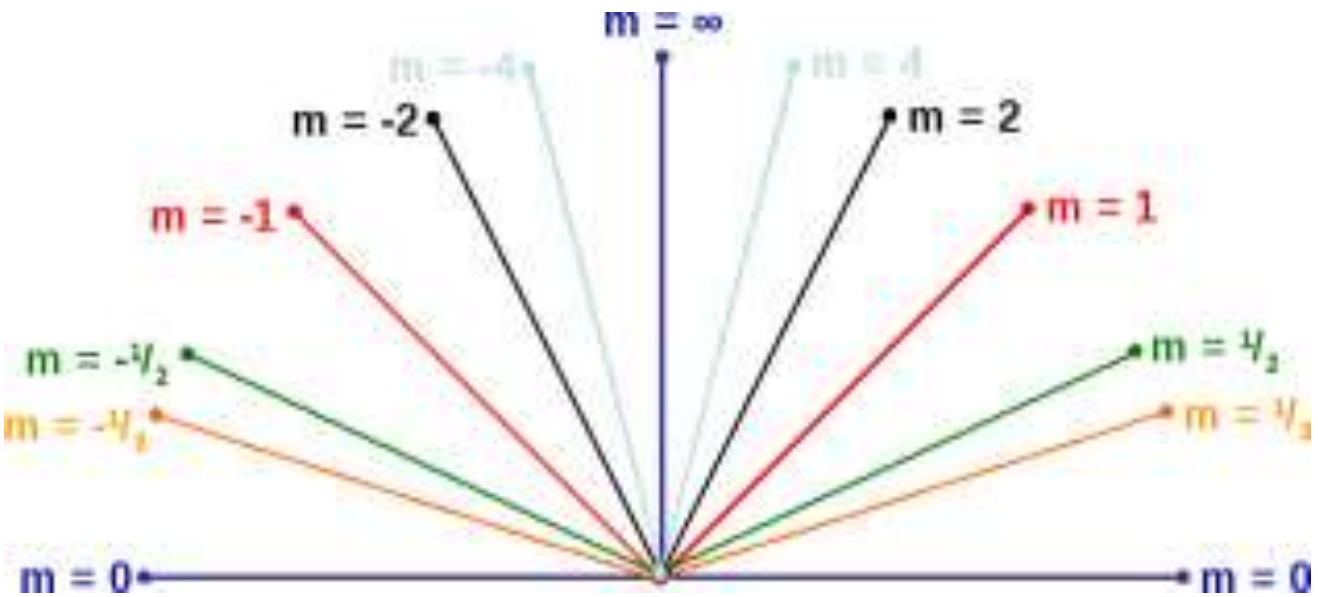
Doğru denklemleri

- Genel doğru denklemimiz
  $y = m.x + b$
- Doğrunun başlangıç ve bitiş noktaları: (x_0, y_0), (x_1, y_1)
- Eğimin bulunması $m = \dfrac{y_1 - y_0}{x_1 - x_0}$
- b'nin bulunması $b = y_0 - m.x_0$



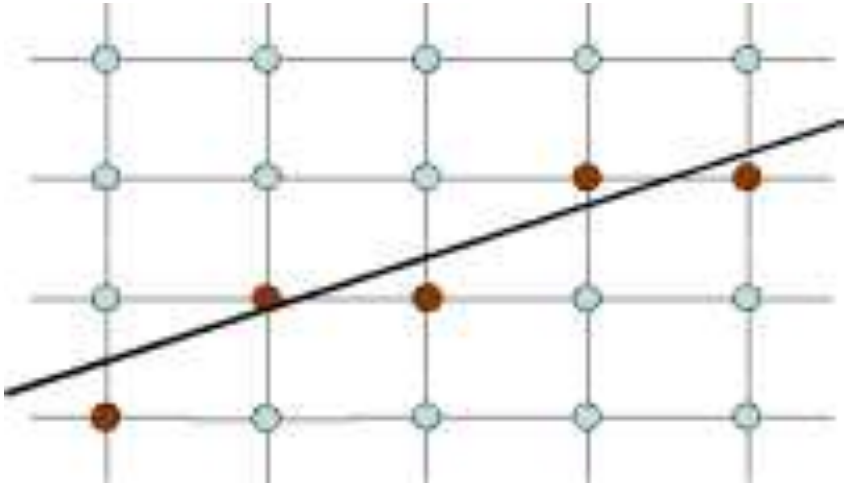
Doğru eğimi

- Doğru eğimi *m*
- Aşağıdaki resimde farklı eğimli doğrular görüyorsunuz



Doğru çizmek neden zordur?

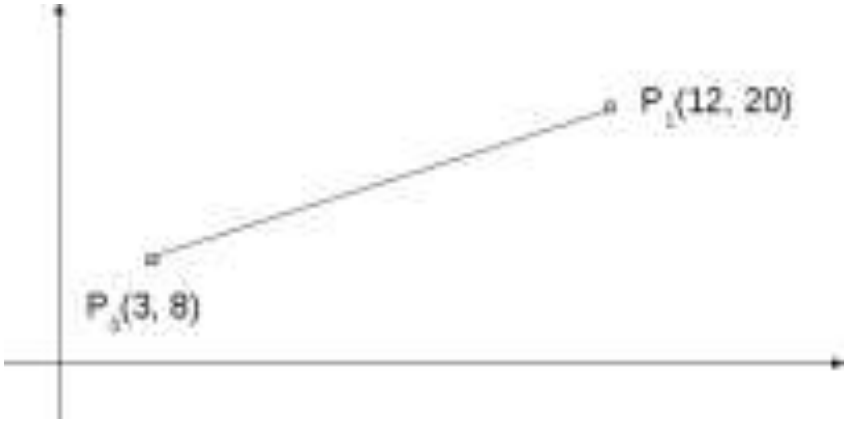
- Ekran piksellerden oluşmaktadır
- Pikseller tamsayıdır
- Elimizdeki doğru bilgilerini rasterize edip piksellere çevirmemiz sonucunda tamsayı sonuçlar elde ederiz

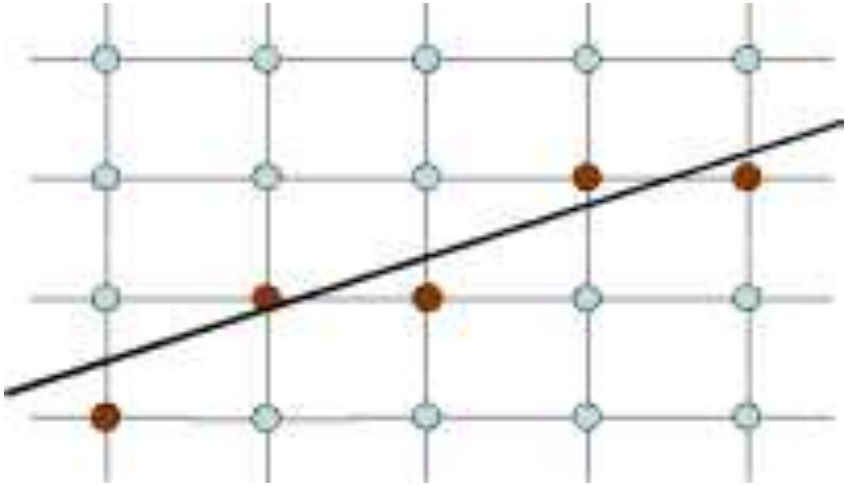


Eğimden yararlanarak çizme

- Eğim bilgisini kullanarak başlangıç ve bitiş noktaları arasındaki ara noktaları bulabiliriz
- $m = \frac{y_1 - y_0}{x_1 - x_0}$ x değerini sabit olarak 1 arttırken y değerini hesaplarız
- $\Delta x = 1$
- $\Delta y = m\Delta x$
- $x_{n+1} = x_n + \Delta x$
- $y_{n+1} = y_n + \Delta y$
- Bu hesaplamalar yeterli midir?

Eğimden yararlanarak çizme





DDA (Digital Differential Analyzer)

- Arttırımsal bir yöntemdir
- Bir sonraki değeri o anki değere göre hesaplar
- Temelinde eğimden yararlanma yatar
- $|m| < 1$
  - $x_{n+1} = x_n + 1$
  - $y_{n+1} = y_n + m$
- $|m| > 1$
  - $x_{n+1} = x_n + \frac{1}{m}$
  - $y_{n+1} = y_n + 1$

DDA (Digital Differential Analyzer)

- Doğrumuzun başlangıç ve bitiş noktalarıyla başlıyoruz (x_0, y_0), (x_1, y_1)
- Ilk pikseli boyuyoruz (başlangıç noktasına en yakın tamsayı)
- $|m| < 1$ varsayalım $(x_1 - x_0 > y_1 - y_0)$

  $x_1 - x_0$ adım olacaktır
- $x = x_0$ ve $y = y_0$ atamalarını yapıyoruz
- Her adımda
  - x'i $\frac{(x_1 - x_0)}{adimsayisi}$ kadar arttırıyoruz
  - y'i $\frac{(y_1 - y_0)}{adimsayisi}$ kadar arttırıyoruz
- Her adımda x ve y değerlerini en yakın tamsayıya yuvarlayarak boyuyoruz

DDA (Digital Differential Analyzer)

```
1   #include <stdlib.h>
2   #include <math.h>
3
4   inline int round (const float a) { return int (a + 0.5); }
5
6   void lineDDA(int x0, int y0, int xEnd, int yEnd){
7       int dx = xEnd - x0, dy = yEnd - y0, steps, k;
8       float xIncrement, yIncrement, x = x0, y = y0;
9
10      if (fabs(dx) > fabs(dy))
11          steps = fabs (dx);
12      else
13          steps = fabs (dy);
14
15      xIncrement = float (dx) / float (steps);
16      yIncrement = float (dy) / float (steps);
17
18      setPixel ( round (x), round (y));
19      for (k = 0; k < steps ; k++) {
20          x += xIncrement;
21          y += yIncrement;
22          setPixel (round (x), round (y));
23      }
24  }
```

- Her adımda 4 kayan noktalı işlem var
- Her adımda 2 tamsayı işlemi var
- Unutmayın kayan noktalı işlemler ve yuvarlama işlemleri maliyetlidir!

DDA (Digital Differential Analyzer)

Noktalar = (2,3) (12,8)
adimsayisi = 12 – 2 = 10
xIncrement = 10/10 = 1.0
yIncrement = 5/10 = 0.5

k	x	y	R(x)	R(y)
0	2	3	2	3
1	3	3.5	3	4
2	4	4	4	4
3	5	4.5	5	5
4	6	5	6	5
5	7	5.5	7	6
6	8	6	8	6
7	9	6.5	9	7
8	10	7	10	7
9	11	7.5	11	8
10	12	8	12	8

- (2, 3), (12, 8)
- (2, 2), (7, 5)
- (2, 7), (3, 2)

Bresenham doğru algoritması

- Sadece arttırımsal tamsayı işlemleri kullanır
- Hesaplanan ideal y değeri ile buna yakın piksellere uzaklık değerini kullanır ($|m| < 1$)
- Sağa doğru pikselleri boyarken, yukarıdaki veya aşağıdaki piksellerden uygun olanını seçmeye dayanır

Bresenham doğru algoritması

1. İki noktayı alır ve başlangıç noktasını (soldaki nokta) (x_0, y_0) olarak saklarız
2. Rengi ayarlar ve ilk pikseli $((x_0, y_0))$ boyarız
3. $\Delta x, \Delta y, 2\Delta y, 2\Delta y - 2\Delta x$ sabitlerini hesaplar, karar parametresini hesaplarız:
   - $p_0 = 2\Delta y - \Delta x$
4. Doğru üzerindeki her x_k ($k = 0$ ile başlıyor) değeri için aşağıdaki testi uygularız.
   - Eğer $p_k < 0$ ise, bir sonraki nokta $(x_k + 1, y_k)$ ve $p_{k+1} = p_k + 2\Delta y$
   - Değilse, sonraki nokta $(x_k + 1, y_k + 1)$ ve $p_{k+1} = p_k + 2\Delta y - 2\Delta x$
5. 4. adımı $\Delta x - 1$ kere tekrarlarız

Bresenham doğru algoritması

```
#include <stdlib.h>
#include <math.h>

/* Bresenham line-drawing procedure for |m| < 1.0 */

void lineBres(int x0, int y0, int xEnd, int yEnd){
    int dx = fabs (xEnd - x0), dy = fabs (yEnd - y0);
    int p = 2 * dy - dx;
    int twoDy = 2 * dy, twoDyMinusDx = 2 * (dy - dx);
    int x, y;

    /* Determine which end point to use as start position */
    if (x0 > xEnd) {
        x = xEnd;
        y = yEnd;
        xEnd = x0;
    }
    else {
        x = x0;
        y = y0;
    }
    setPixel(x, y);

    while (x < xEnd) {
        x++;
        if (p < 0)
            p += twoDy;
        else {
            y++;
            p += twoDyMinusDx;
        }
        setPixel(x, y);
    }
}
```

- Her adımda 0 kayan noktalı işlem var
- Her adımda 3 veya 4 tamsayı işlem var

Bresenham doğru algoritması

dx = 12 – 2 = 10
dy = 8 – 3 = 5
p0 = 2dy – dx = 0

2dy = 10
2dy – 2dx = -10

k	p	P(x)	P(y)
0	0	2	3
1	-10	3	4
2	0	4	4
3	-10	5	5
4	0	6	5
5	-10	7	6
6	0	8	6
7	-10	9	7
8	0	10	7
9	-10	11	8
10	0	12	8

- (2, 3), (12, 8)
- (2, 2), (7, 5)
- (2, 7), (3, 2)

OpenGL kodu

```
glLineWidth(2);
glBegin(GL_LINES);
    glVertex3f( 0.0f, 1.0f, -1.0f);
    glVertex3f(-6.0f,-6.0f, -1.0f);
glEnd();
```

Kaynaklar

- Tahir Emre KALAYCI, Doğru Çizim algoritmaları,
- https://tekrei.gitlab.io/presentations/2007-Dogru-Cizme.pdf

- Hearn and Baker, Computer Graphics with OpenGL

- http://www.comp.dit.ie/bmacnamee/materials/graphics/2006-2007Lectures/Graphics5-ScanConvertingLines.ppt

- http://www.cs.sjsu.edu/ teoh/teaching/previous/cs116afa07/lecture

Kaynaklar

- http://www.compapp.dcu.ie/_bmc-caul/CA433/raster graphics.ppt
- http://www-graphics.stanford.edu/courses/cs248-04/scan/scan1.html
- http://www.cs.sfu.ca/ torsten/Teaching/Cmpt361/LectureNotes/PD