Bilgisayar Grafiklerinde DDA (Digital Differential Analyzer)

Çizgi Oluşturma Algoritması

DDA (Digital Differential Analyzer), doğrunun bilgisayar ekranına çizimi için kullanılan bir algoritmadır.

Herhangi bir 2 Boyutlu düzlemde (x0, y0) ve (x1, y1) iki noktayı birleştirirsek, bir doğru parçası elde ederiz. Ancak bilgisayar grafikleri söz konusu olduğunda, herhangi iki koordinat noktasını doğrudan birleştiremeyiz, bunun için ara noktaların koordinatlarını hesaplamalı ve fonksiyonlar yardımıyla istenen renkteki her ara nokta için bir piksel koymalıyız.

Grafik fonksiyonlarını kullanmak için, sistem çıktı ekranımız, sol üst köşenin koordinatının (0, 0) olduğu bir koordinat sistemi olarak ele alınır ve aşağı doğru hareket ettikçe y-ordinat artışlarımız ve sağa hareket ettikçe x-apsislerimiz artar.

Şimdi, herhangi bir çizgi parçasını oluşturmak için ara noktalara ihtiyacımız var ve bunları hesaplamak için DDA (Dijital diferansiyel analizör) çizgi oluşturma algoritması olarak adlandırılan temel bir algoritma kullanabiliriz .

DDA Algoritması :
Doğrunun bir noktasını (X0,Y0), ikinci noktasını (X1,Y1) olarak kabul edin.

```
// dx , dy'yi hesapla
dx = X1 - X0;
dy = Y1 - Y0;

// dx & dy'nin mutlak değerine bağlı olarak
// pikseli koymak için adım sayısını seçin
// adimlar = abs(dx) > abs(dy) ? abs(dx): abs(dy)
adimlar = abs(dx) > abs(dy) ? abs(dx): abs(dy);
```

```
// her adım için x ve y cinsinden artışı hesapla
Xinc = dx / (float) adımları;
Yinc = dy / (float) adımları;

// Her adım için piksel koy
X = X0;
Y = Y0;
for (int i = 0; i <= adımlar; i++)
{
   putpixel (yuvarlak(X),yuvarlak(Y),BEYAZ);
   X += Xinc;
   Y += Yinc;
}
```

Avantajlar:

- Algoritmayı uygulamak basit ve kolaydır.
- Yüksek zaman karmaşıklığına sahip çoklu işlemleri kullanmaktan kaçınır.
- Herhangi bir kayan nokta çarpımı kullanmadığı ve doğru üzerindeki noktaları hesapladığı için doğru denkleminin doğrudan kullanımından daha hızlıdır.

Dezavantajları :

- Yuvarlama işlemi ve kayan nokta aritmetiği ile ilgilenir, bu nedenle yüksek zaman karmaşıklığına sahiptir.
- Oryantasyona bağlı olduğundan uç nokta doğruluğu zayıftır.

- Kayan nokta gösterimindeki sınırlı kesinlik nedeniyle kümülatif hata üretir.

Bresenham Çizgi Üretimi Algoritması (Bresenham Doğru Algoritması)

A(x1, y1) ve B(x2, y2) iki noktasının koordinatları verildi. Piksellerin bilgisayar ekranında AB çizgisinin çizilmesi için gerekli tüm ara noktaları bulma görevi. Her pikselin tamsayı koordinatlarına sahip olduğunu unutmayın.

Örnekler:

Giriş : A(0,0), B(4,4)

Çıktı : (0,0), (1,1), (2,2), (3,3), (4,4)

Giriş : A(0,0), B(4,2)

Çıktı : (0,0), (1,0), (2,1), (3,1), (4,2)

Aşağıda algoritmayı basit tutmak için bazı varsayımlar bulunmaktadır.

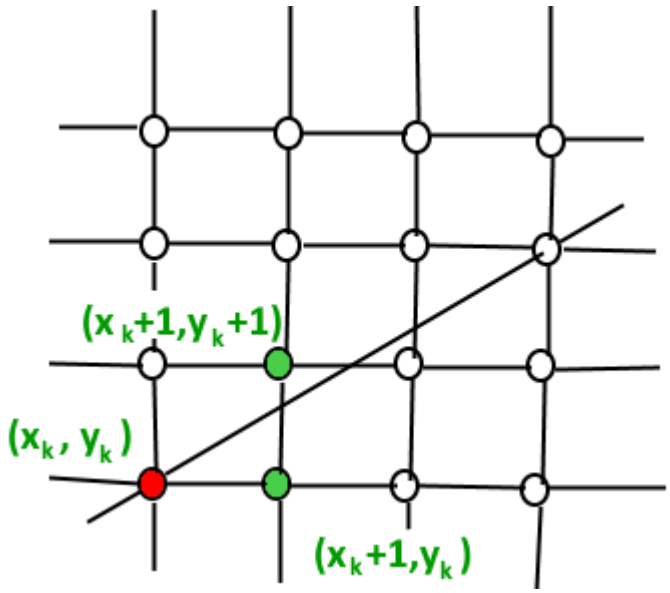
1. Soldan sağa doğru çiziyoruz.
2. x1 < x2 ve y1< y2
3. Çizginin eğimi 0 ile 1 arasındadır. Sol alttan sağ üste doğru bir çizgi çiziyoruz.

```
// Doğru çiziminin temeli
void naiveDrawLine(x1, x2, y1, y2)
{
  m = (y2 - y1)/(x2 - x1)
  for (x  = x1; x <= x2; x++)
```

```
  {
    // Assuming that the round function finds
    // closest integer to a given float.
    y = round(mx + c);
    print(x, y);
  }
}
```

Bresenham'ın algoritmasının fikri, kayan nokta çarpımını ve mx + c'yi hesaplamak için toplamayı önlemek ve ardından her adımda (mx + c) yuvarlak değerini hesaplamaktır. Bresenham'ın algoritmasında x ekseni boyunca birim aralıklarla hareket ediyoruz.

1. x'i her zaman 1 artırırız ve bir sonraki y'yi seçeriz, y+1'e gitmemiz mi yoksa y'de kalmamız mı gerekeceğini buluruz. Başka bir deyişle, herhangi bir konumdan örneğin; (X_k , Y_k)'da isek (X_k + 1, Y_k) ve (X_k + 1, Y_k + 1) arasında seçim yapmamız gerekir .



Orijinal doğruya daha yakın bir noktaya karşılık gelen y değerini ($Y_k + 1$ ve Y_k arasında) seçmek istiyoruz .

Bir sonraki nokta olarak $Y_k + 1$ 'i mi yoksa Y_k' yı mı seçeceğimize karar vermek için bir karar parametresine ihtiyacımız var. Biz Y almaya karar vermek bir karar parametresine ihtiyaç $Y_k + 1$ veya Y_k sonraki noktası olarak. Buradaki fikir, önceki artıştan y'ye kadar olan eğim hatasını takip etmektir. Eğim hatası 0,5'ten büyük olursa, çizginin bir piksel yukarı hareket ettiğini ve y koordinatımızı artırmamız ve hatayı yeni pikselin tepesinden olan mesafeyi temsil edecek şekilde yeniden ayarlamamız gerektiğini biliyoruz - bu, hatadan bir çıkarılarak yapılır.

```
// Modifying the naive way to use a parameter
// to decide next y.
void withDecisionParameter(x1, x2, y1, y2)
{
  m = (y2 - y1)/(x2 - x1)
  slope_error = [Some Initial Value]
  for (x = x1, y = y1; x = 0.5)
  {
    y++;
    slope_error  -= 1.0;
  }
}
```

Kayan nokta aritmetiğinden nasıl kaçınılır

Yukarıdaki algoritma hala kayan nokta aritmetiği içerir. Kayan nokta aritmetiğinden kaçınmak için, m değerinin altındaki değeri göz önünde bulundurun.

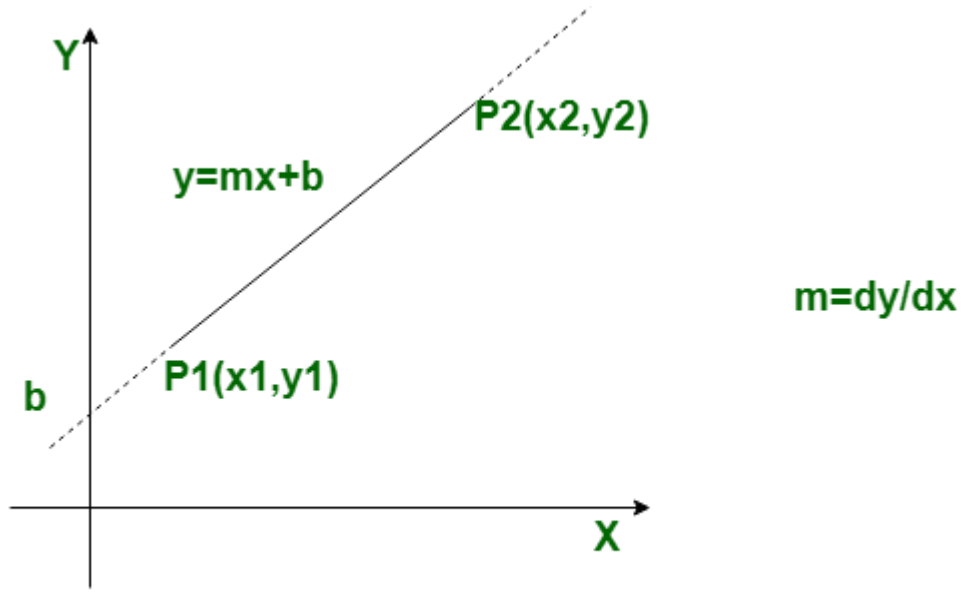
$m = (y2 – y1)/(x2 – x1)$

Her iki tarafı da $(x2 – x1)$ ile

çarpıyoruz. Ayrıca eğim_hatasını eğim_hatası * (x2 – x1) olarak değiştiriyoruz. 0,5 ile karşılaştırmayı önlemek için, bunu ayrıca eğim_hatası * (x2 – x1) * 2 olarak değiştiriyoruz.
Ayrıca, genellikle 1 yerine 0 ile karşılaştırmak tercih edilir.

DDA ve Bresenham Line Drawing algoritması arasındaki karşılaştırmalar

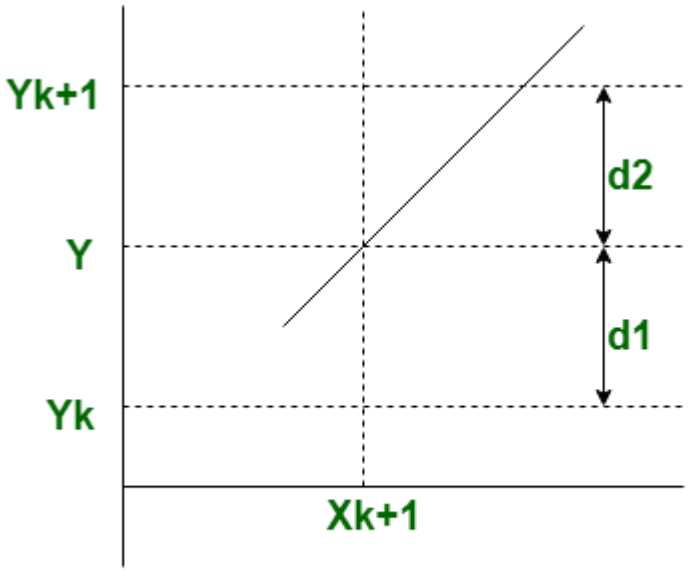
Aydınlatma hilelerinde, ekran üzerinde bir çizgi çizmek için kullanılan 2 algoritmik kural vardır; DDA (**Digital Differential Analyzer**) algoritmik kuralı ve Bresenham çizgi algoritması .



Scan Converting line Using DDA Algo.

DDA algoritması ile Bresenham çizgi algoritması arasındaki temel fark, DDA algoritmik kuralının kayan amaçlı değerleri kullanması, Bresenham'da küresel kapalı fonksiyonların kullanılmasıdır.

DDA algoritmik kuralı çarpma ve bölmeyi içerirken, bresenham algoritmik kuralında toplama ve çıkarma en çok gerçekleştirilen işlemlerdir.



Distance b/w pixels in Bresenham's algo

DDA algoritması ile Bresenham çizgi çizme algoritması arasındaki farkı görelim:

S.NO	DDA Çizgi Algoritması	Bresenham hattı Algoritması
1.	DDA algoritması, Bresenham çizgi algoritmasından daha az verimlidir.	DDA algoritmasından daha verimlidir.
2.	DDA algoritmasının hesaplama hızı Bresenham line algoritmasına göre daha düşüktür.	Bresenham çizgi algoritmasının hesaplama hızı DDA algoritmasından daha hızlıdır.
3.	DDA algoritması, Bresenham çizgi algoritmasından daha maliyetlidir.	Bresenham çizgi algoritması DDA algoritmasından daha ucuzdur.

S.NO	DDA Çizgi Algoritması	Bresenham hattı Algoritması
4.	DDA algoritmasının kesinliği veya doğruluğu daha azdır.	Daha hassas veya doğru olmasına rağmen.
5.	DDA algoritmasında hesaplamanın karmaşıklığı daha karmaşıktır.	Hesaplamanın karmaşıklığı basittir.
6.	DDA algoritmasında optimizasyon sağlanmaz.	Optimizasyon sağlanır.

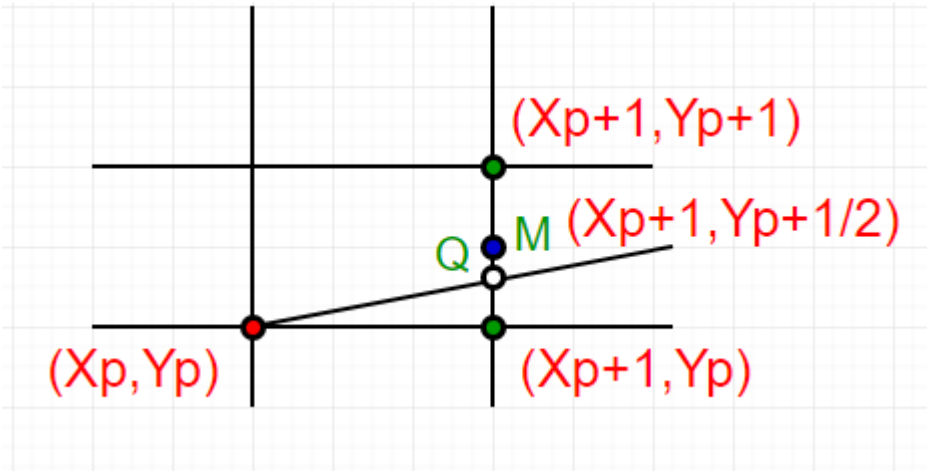
Orta Nokta Çizgi Oluşturma Algoritması

x1 < x2 ve y1 < y2 olacak şekilde A(x1, y1) ve B(x2, y2) iki noktasının koordinatları verildi. Piksellerin bilgisayar ekranında AB çizgisinin çizilmesi için gerekli tüm ara noktaları bulma görevi. Her pikselin tamsayı koordinatlarına sahip olduğunu unutmayın.
Bu görev için aşağıdaki algoritmaları tartıştık.

1. Çizgi çizme için DDA algoritması
2. Bresenhams'ın çizgi çizme algoritmasına giriş .

Bu yazıda, Bresenham'ın bir önceki gönderide tanıtılan algoritmasını temsil etmenin farklı bir yolu olan Orta Nokta Çizgisi çizme algoritması tartışılmaktadır. Herhangi bir verilen/hesaplanan önceki piksel için $P(X_p, Y_p)$, çizgiye en yakın sonraki piksel için iki aday vardır, $E(X_p+1, Y_p)$ ve $NE(X_p+1, Y_p+1)$ (**E** , Doğu ve **NE** , Kuzey-Doğu anlamına gelir). Orta Nokta algoritmasında aşağıdakileri yapıyoruz.

1. Sonraki iki olası noktanın ortasını bulun. $E(X_p+1, Y_p)$ ve $NE(X_p+1, Y_p+1)$'nin ortası $M(X_p+1, Y_p+1/2)$'dir.
2. M çizginin üzerindeyse, sonraki nokta olarak E'yi seçin.
3. M çizginin altındaysa, sonraki nokta olarak NE'yi seçin.



Bir noktanın çizginin üstünde mi yoksa çizginin altında mı olduğu nasıl bulunur?
Aşağıda algoritmayı basit tutmak için bazı varsayımlar bulunmaktadır.

1. Soldan sağa doğru çiziyoruz.
2. x1 < x2 ve y1< y2
3. Çizginin eğimi 0 ile 1 arasındadır. Sol alttan sağ üste doğru bir çizgi çiziyoruz.

Yukarıdaki varsayımların dışındaki durumlar yansıma kullanılarak ele alınabilir.

Kaynaklar:

https://www.geeksforgeeks.org/dda-line-generation-algorithm-computer-graphics/

https://www.geeksforgeeks.org/bresenhams-line-generation-algorithm/

https://www.geeksforgeeks.org/comparions-between-dda-and-bresenham-line-drawing-algorithm/?ref=rp

https://www.geeksforgeeks.org/mid-point-line-generation-algorithm/