BİLGİSAYAR GRAFİKLERİ (BTS307)

2018-2019 Güz Dönemi Ödevi

# ÇOKGEN DOLDURMA

KASIM BÖLÜCÜ

16-701-005

MUHAMMED İŞBAKAN

16-701-026

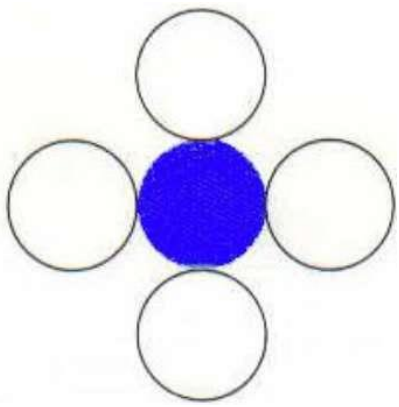
# Sınır Doldurma Algoritması (Boundary Fill Algorithm)

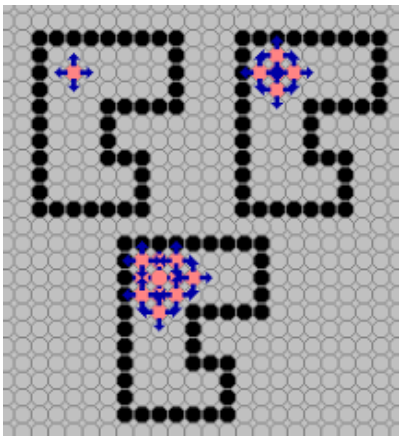
Bu bir alan dolum algoritmasıdır. İç mekan noktalarının kolayca seçildiği bilgisayar grafiğinde interaktif bir resim yapmak zorunda olduğumuz yerde kullanılır.Sınır Doldurma Algoritması doldurulacak poligonun içindeki bir pikselde başlar ve iç tarafa doğru ve dışa doğru ilerler. Bu algoritma, yalnızca bölgenin doldurulması gereken renk ve bölgenin sınırının rengi farklıysa çalışır. Sınır tek bir renkten oluşuyorsa, bu yaklaşım bölgenin sınırına ulaşana kadar pikselden piksele doğru ilerler.

Sınır Doldurma Algoritması tekrarlayıcıdır. Girdi olarak bir iç nokta (x, y), bir dolgu rengi ve bir sınır rengi alır. Algoritma, (x, y) rengini kontrol ederek başlar. Renk, dolgu rengine ve sınır rengine eşit değilse, dolgu rengi ile boyanır ve (x, y) 'nin tüm komşuları için işlev çağrılır. Bir nokta dolgu rengi veya sınır renginde bulunursa, işlev komşularını çağırmaz ve döner. Bu süreç, bölge için sınır rengine kadar olan tüm noktalar test edilene kadar tekrarlanır ve devam eder.

Sınır doldurma algoritması 4 bağlantılı piksel veya 8 bağlantılı piksel olmak üzere 2 şekilde uygulanabilir.

- **4 bağlantılı piksel (**4-connected pixels **):**





Bu teknikte 4 bağlantılı pikseller şekilde gösterildiği gibi kullanılır.Pikselleri şu anki piksellerin üstüne, sağına, altına ve sol tarafına koyuyoruz ve bu işlem, farklı renklere sahip bir sınır bulana kadar devam edecek.

**Algoritma :**

Adım 1 - Tohum noktası (seedx, tohumlu), fcolor ve dcol değerini sıfırlayın.

Adım 2 - Poligonun sınır değerlerini tanımlayın.

Adım 3 - Mevcut tohum noktasının varsayılan renk olup olmadığını kontrol edin, sonra sınır piksellerine ulaşıncaya kadar 4. ve 5. adımları tekrarlayın.

If getpixel(x, y) = dcol then repeat step 4 and 5

Adım 4 - Varsayılan rengi tohum noktasında dolgu rengiyle değiştirin.

setPixel(seedx, seedy, fcol)

Adım 5 - Prosedürü dört mahalle noktası ile tekrarlayın.

FloodFill (seedx – 1, seedy, fcol, dcol)

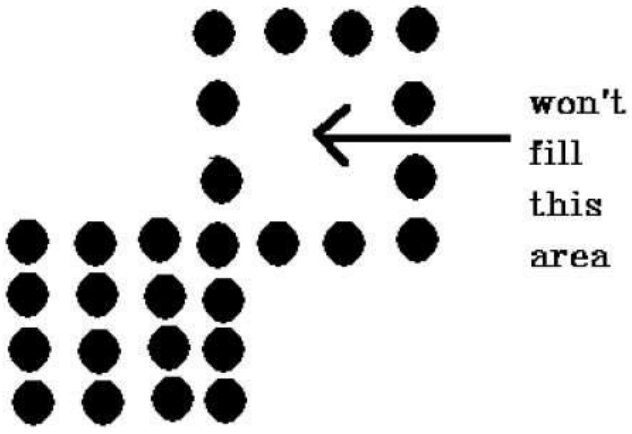
FloodFill (seedx + 1, seedy, fcol, dcol)

FloodFill (seedx, seedy - 1, fcol, dcol)

FloodFill (seedx – 1, seedy + 1, fcol, dcol)

6. Adım - Çıkış

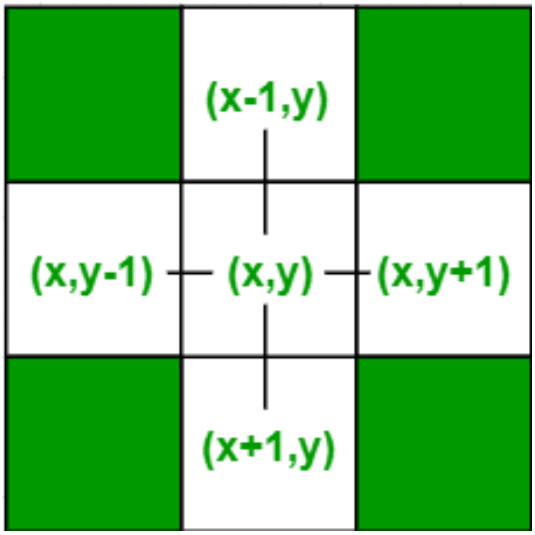
Bu teknikte bir problem var. Tüm bölgeyi doldurmaya çalıştığımız aşağıdaki gibi davayı düşünün. Burada görüntü sadece kısmen doldurulur. Bu gibi durumlarda 4 bağlantılı piksel tekniği kullanılamaz.



**Algoritma 2:**

```
void boundaryFill4(int x, int y, int fill_color,int boundary_color)
{
   if(getpixel(x, y) != boundary_color &&
      getpixel(x, y) != fill_color)
   {
      putpixel(x, y, fill_color);
      boundaryFill4(x + 1, y, fill_color, boundary_color);
      boundaryFill4(x, y + 1, fill_color, boundary_color);
      boundaryFill4(x - 1, y, fill_color, boundary_color);
      boundaryFill4(x, y - 1, fill_color, boundary_color);
   }}
```

Bir piksel çizdikten sonra, dört komşu nokta için işlev çağrılır. Bunlar mevcut pikselin sağ, sol, üstünde ve altında olan piksel konumlardır.

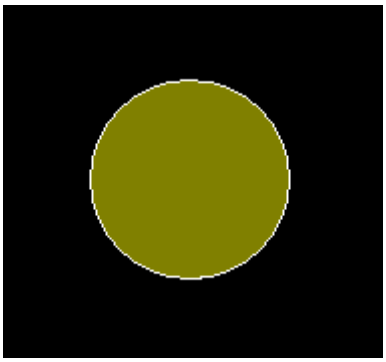



Algoritmanın uygulanması aşağıdadır :

# Sınır Doldurma Algoritması // C Uygulaması
## (4 bağlantılı piksel (4-connected pixels ) fonksiyonu)

```
#include <graphics.h>
void boundaryFill4(int x, int y, int fill_color,int boundary_color)
{
   if(getpixel(x, y) != boundary_color &&
      getpixel(x, y) != fill_color)
   {
      putpixel(x, y, fill_color);
      boundaryFill4(x + 1, y, fill_color, boundary_color);
      boundaryFill4(x, y + 1, fill_color, boundary_color);
      boundaryFill4(x - 1, y, fill_color, boundary_color);
      boundaryFill4(x, y - 1, fill_color, boundary_color);
   }
}

int main()
{

   initgraph(&gd, &gm, "");

   int x = 250, y = 200, radius = 50;

 // Daire fonksiyonu
   circle(x, y, radius);

 // Çağrı fonksiyonu
   boundaryFill4(x, y, 6, 15);

   delay(10000);

   getch();
// Closegraph işlevi, grafik modunu kapatır ve grafik sistemi tarafından tahsis edilen tüm belleği serbest bırakır.
closegraph();

   return 0;
}
```

**Çıktı:**



- **8 Bağlı Poligon(**8-connected pixels**):**

Bu teknikte, 8 bağlantılı pikseller şekilde gösterildiği gibi kullanılır. 4-bağlantılı teknikte yaptığımız gibi mevcut piksellerin sağ, sol ve sağ taraflarının üstüne pikseller koyuyoruz.

Buna ek olarak, mevcut pikselin tüm alanı kaplanacak şekilde pikselleri de köşegenlere koyuyoruz. Farklı renklere sahip bir sınır bulana kadar bu süreç devam edecek.

**Algoritma:**

Adım 1 - Tohum noktası (seedx, tohumlu), fcolor ve dcol değerini sıfırlayın.

Adım 2 - Poligonun sınır değerlerini tanımlayın.

Adım 3 - Geçerli tohum noktasının varsayılan renk olup olmadığını kontrol edin ve sınır piksellerine ulaşıncaya kadar 4. ve 5. adımları tekrarlayın.

If getpixel(x,y) = dcol then repeat step 4 and 5

Adım 4 - Varsayılan rengi tohum noktasında dolgu rengiyle değiştirin.

setPixel(seedx, seedy, fcol)

Adım 5 - Dört mahalle puanı ile prosedürü tekrar tekrar izleyin

FloodFill (seedx – 1, seedy, fcol, dcol)

FloodFill (seedx + 1, seedy, fcol, dcol)

FloodFill (seedx, seedy - 1, fcol, dcol)

FloodFill (seedx, seedy + 1, fcol, dcol)

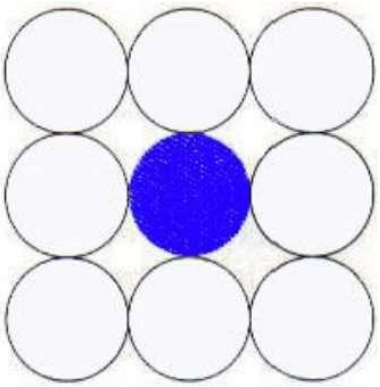
FloodFill (seedx – 1, seedy + 1, fcol, dcol)

FloodFill (seedx + 1, seedy + 1, fcol, dcol)

FloodFill (seedx + 1, seedy - 1, fcol, dcol)

FloodFill (seedx – 1, seedy - 1, fcol, dcol)

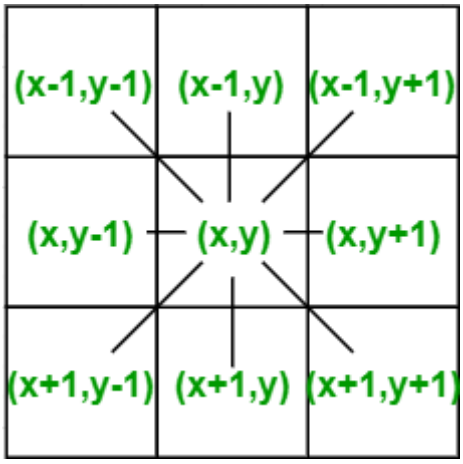
6. Adım – Çıkış



**Algoritma 2:**

```
void boundaryFill8 (int x, int y, int fill_color, int sınır_color)
{
   eğer (getpixel (x, y)! = boundary_color &&
     getpixel (x, y)! = fill_color)
   {
      putpixel (x, y, fill_color);
      boundaryFill8 (x + 1, y, fill_color, boundary_color);
      boundaryFill8 (x, y + 1, fill_color, boundary_color);
      boundaryFill8 (x - 1, y, fill_color, boundary_color);
      boundaryFill8 (x, y - 1, fill_color, boundary_color);
      boundaryFill8 (x - 1, y - 1, fill_color, boundary_color);
      boundaryFill8 (x - 1, y + 1, fill_color, boundary_color);
      boundaryFill8 (x + 1, y - 1, fill_color, boundary_color);
      boundaryFill8 (x + 1, y + 1, fill_color, boundary_color);  } }
```

Test edilecek pikseller, 8 komşu piksel, sağdaki piksel, sol, yukarıda, aşağıda ve 4 diyagonal pikseldir.




Algoritmanın uygulanması aşağıdadır :

## Sınır Doldurma Algoritması // C Uygulaması

**(8 bağlantılı piksel (8-connected pixels ) fonksiyonu)**

```
#include <graphics.h>
  void boundaryFill8(int x, int y, int fill_color,int boundary_color)
{
    if(getpixel(x, y) != boundary_color &&
       getpixel(x, y) != fill_color)
    {
        putpixel(x, y, fill_color);
        boundaryFill8(x + 1, y, fill_color, boundary_color);
        boundaryFill8(x, y + 1, fill_color, boundary_color);
        boundaryFill8(x - 1, y, fill_color, boundary_color);
        boundaryFill8(x, y - 1, fill_color, boundary_color);
        boundaryFill8(x - 1, y - 1, fill_color, boundary_color);
        boundaryFill8(x - 1, y + 1, fill_color, boundary_color);
        boundaryFill8(x + 1, y - 1, fill_color, boundary_color);
        boundaryFill8(x + 1, y + 1, fill_color, boundary_color);
    }
}



int main()
{
```

**//Gm, piksel kullanarak görüntü oluşturan bir bilgisayar görüntüleme modu olan Grafik modudur. DETECT, "graphics.h" başlık dosyasında tanımlanan bir makrodur.**

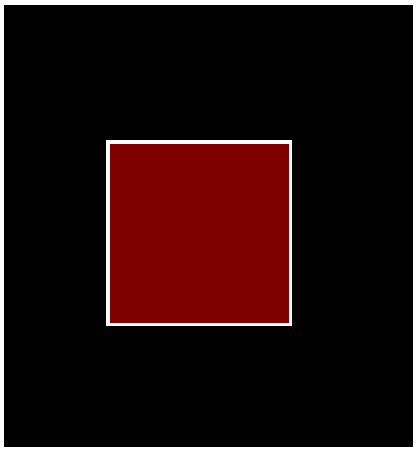
```
    int gd = DETECT, gm;
```

**//İnitgraph, diskten bir grafik sürücüsü yükleyerek grafik sistemini başlatır**

```
    initgraph(&gd, &gm, "");
```

```
// Dikdörtgen fonksyonu
    rectangle(50, 50, 100, 100);

// Fonksiyon çağrısı
    boundaryFill8(55, 55, 4, 15);

    delay(10000);

    getch();

// Closegraph işlevi, grafik modunu kapatır ve grafik sistemi tarafından tahsis edilen tüm belleği serbest bırakır.

    closegraph();

    return 0;
}
```

## Çıktı:



## Sınır dolgu ile ilgili problemler

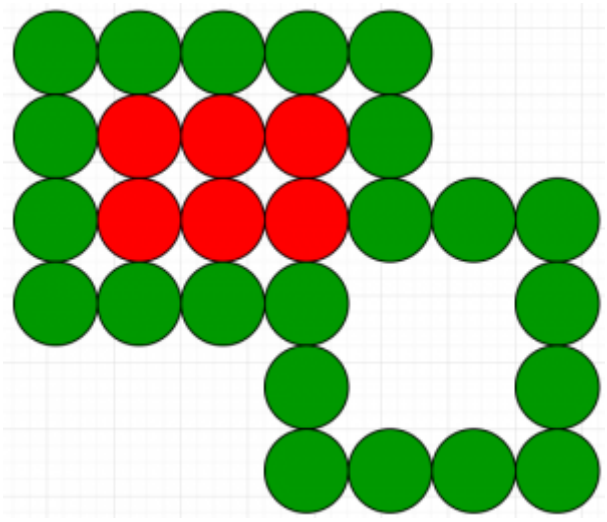
- Aynı iç pikseller de dolgu renginde de görüntüleniyorsa, bölgeleri doğru şekilde doldurmayabilir.
- 4 bağlantılı piksel yönteminde bir problem var. Bazen, verilen pikselin yalnızca bitişik konumunu kontrol ettiği için köşe pikselini doldurmaz.

## Taşırma dolgusu ile Sınır doldurma algoritması arasındaki farklar

Hem **Flood dolgusu** hem de **Sınır doldurma algoritmaları** belirli bir rakamı seçilen bir renkle renklendirse de, bir yönden farklıdır. Flood dolgusunda, seçilen rengin tüm bağlı pikselleri bir dolgu rengi ile değiştirilir. Öte yandan, Sınır doldurmasında, belirli bir renk sınırı bulunduğunda program durur.

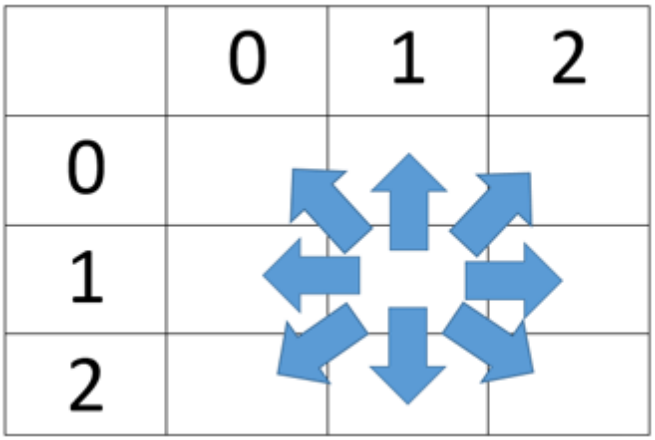
## 4 bağlantılı piksel Vs 8 bağlantılı piksel

Sınır rengini YEŞİL ve dolgu rengi KIRMIZI olarak alalım. 4 bağlantılı yöntem bu rakamı tamamen doldurmuyor. Bu şekil 8 bağlantılı teknik kullanılarak verimli bir şekilde doldurulacaktır.



# Taşırma Dolgu Algoritması (Flood Fill Algorithm)

Bu algoritma ile, tek bir renk sınırı içinde tanımlanmamış bir alanı yeniden renklendirebiliriz. Bu şekilde, sınır renk değeri aramak yerine bir rengi değiştirerek bu alanları boyayabiliriz. Bu bütün yaklaşım, taşkın dolgusu algoritması olarak adlandırılmaktadır. Bu prosedür ayrıca, piksel açıklıklarını doldurarak yığının depolama gereksinimini azaltmak için de kullanılabilir.

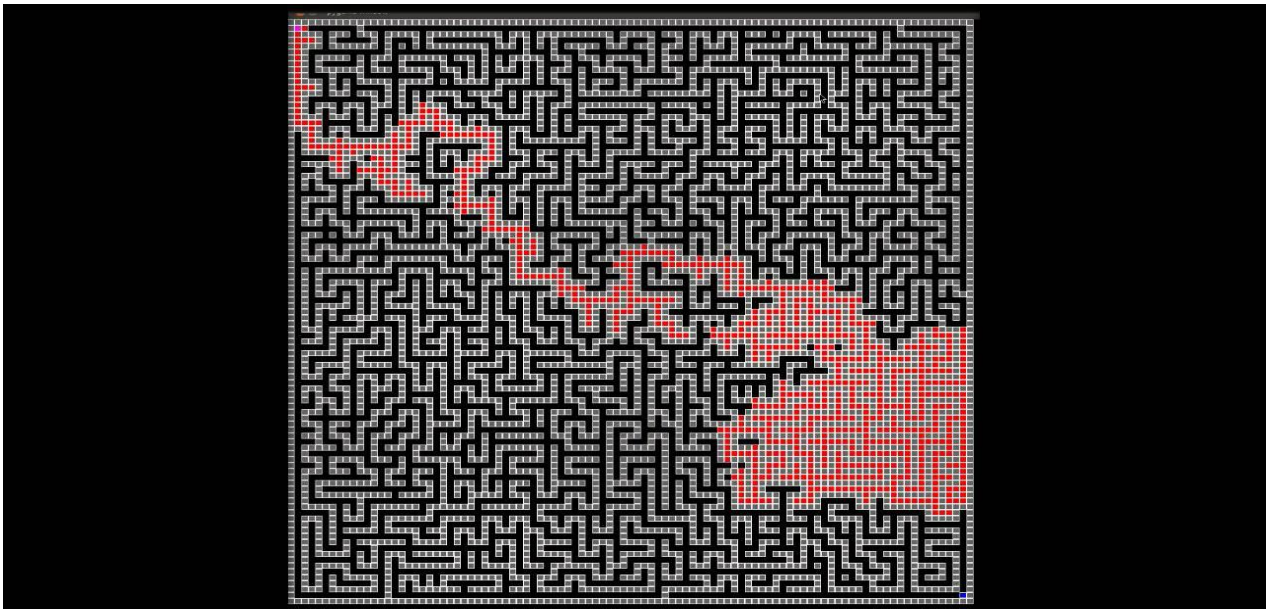



**Algoritma :**

```
floodfill4 (x, y, fillcolor, oldcolor: integer)
begin
   if getpixel (x, y) = old color then
begin
      setpixel (x ,y, fillcolor)
      floodfill4 (x+1, y, fillcolor, oldcolor)
      floodfill4 (x-1, y, fillcolor, oldcolor)
      floodfill4 (x, y+1, fillcolor, oldcolor)
      floodfill4 (x, y-1, fillcolor, oldcolor)
end.
```

## Bir Labirenti Çözme Örneği:

Bir başlangıç noktası olan bir matris ve bazı engellerin bulunduğu bir hedef verildiğinde, bu algoritma kaynaktan hedefe giden yolu bulmaya yardımcı olur.

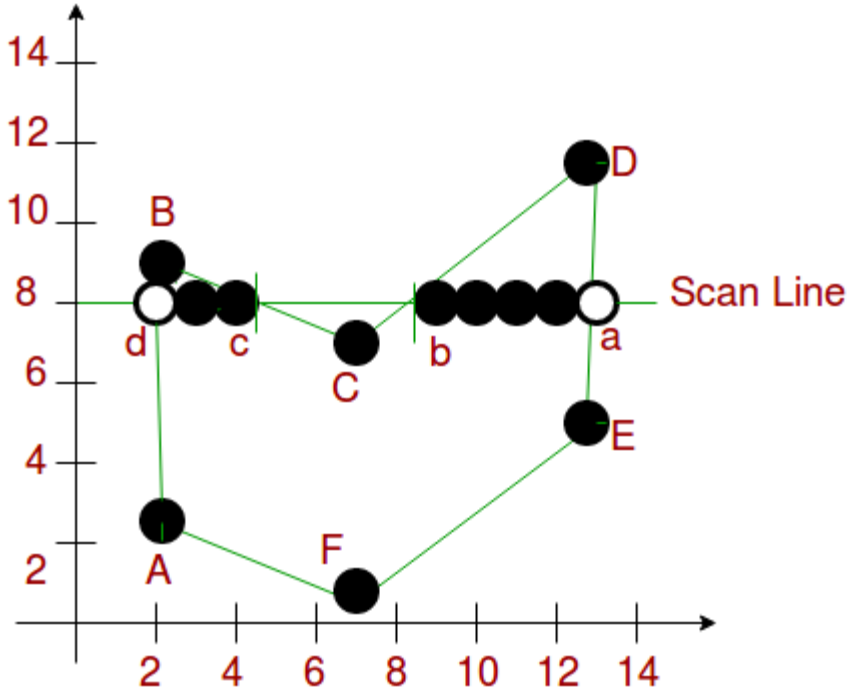


# Scanline Çokgen Doldurma Algoritması

**OPENGL kullanarak C-line Poligon dolum**
Bilgisayar ekranındaki şekiller çokgenler kullanılarak çizilebilir. Bu rakamları renkle doldurmak için bazı algoritmalar geliştirmemiz gerekir. Bu amaçla 3 algoritma vardır: Sınır doldurma ,Scanline ve Taşırma algoritmaları.

Sınır doldurma işlemi çok fazla işlem gerektirir ve böylece gerçek zamanlı olarak çok az sorunla karşılaşır. Böylelikle uygun alternatif, doğada çok sağlam olduğu için tarama dolgusu yapmaktır. Bu makalede, bir görüntüdeki renkleri doldurmak için Scanline dolgu algoritmasının nasıl kullanılacağı anlatılmaktadır.

**Scanline Çokgen doldurma Algoritması**
Scanline doldurma temel olarak yatay çizgiler veya tarama hatları kullanılarak çokgenlerin doldurulmasıdır. SLPF algoritmasının amacı, şeklin yalnızca köşeleri verilen bir poligonun iç piksellerini doldurmak (renklendirmek). Scanline'ı anlamak için, soldan başlayarak, sağa doğru devam eden tek bir kalemle çizilen görüntüyü düşünün, resimde bir nokta bulunan noktaların çizilmesi ve çizgi tamamlandığında, bir sonraki satırdan başlar ve devam eder.
Bu algoritma, tarama çizgisinin çokgen kenarlarıyla kesişerek çalışır ve çokgenleri kesişim çiftleri arasında doldurur.




**Özel poligon köşeleri:**

1) Köşede kesişen iki çizgi de tarama hattının aynı tarafındaysa, bunu iki nokta olarak düşünün.

2) Köşede kesişen çizgiler tarama hattının zıt taraflarındaysa, bunu sadece bir nokta olarak düşünün.

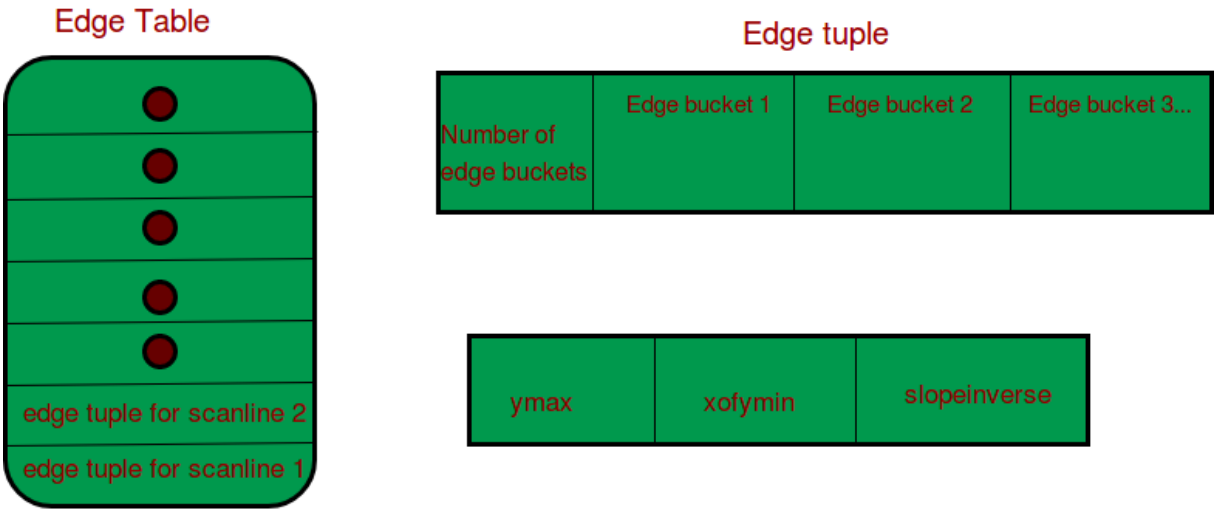
**Poligon dolgusu bileşenleri:**

1) Kenar Bucketları: Bir kenarın bilgilerini içerir. Kenar bucket girişleri, kullandığınız veri yapısına göre değişir. Aşağıda ele aldığımız örnekte üç kenar kepçesi vardır: ymax, xofymin, slopeinverse.

2) Kenar Tablosu: Birkaç kenar listesi içerir -> şeklin tamamını oluşturan tüm kenarları tutar. Kenarlar oluştururken, kenarların köşe noktalarının soldan sağa doğru sıralanması gerekir ve kenarları artan yMin düzeninde korunur. Tüm kenarlar ET(Edge Table)'den çıkarıldığında doldurma tamamlanır.

3) Aktif Liste: Çokgen doldurmak için kullanılan mevcut kenarları korur. Bir kenarın yMin'i işlenmekte olan mevcut tarama hattına eşit olduğunda, Kenar Tablosundan AL(Aktif Liste)'a itilen iğneler sayesinde her geçişten sonra Aktif Liste yeniden sıralanacaktır.

**Veri yapısı:**




ymax - max y - coordinate of edge
xofymin - x-coordinate of lowest edge point, updated at every scanline while scanline filling
slopeinverse - inverse of edge slope (horizontal lines are not stored)

## Scanline Filling

**Algoritma:**

1. Kenardan sonra çokgen kenarı işleyeceğiz ve kenar Tablosunda saklayacağız.
2. Kaydetme kenarı, aynı tarama çizgisi kenarındaki gibi kenetlenerek yapılır. En alt noktanın kenarına y koordinatı değeri atanır.
3. Bir kenar tuple'ında herhangi bir kenarın eklenmesinden sonra, tuple, xofymin değerine göre ekleme yapılarak sıralanır.
4. Tüm çokgen kenar tablosuna eklendikten sonra, şekil doldurulur.
5. Doldurma, alttaki ilk tarama hattından başlatılır ve en üste kadar devam edilir.
6.Aktif kenar tablosu alınır ve her bir tarama hattı için aşağıdaki şeyler tekrarlanır:
   a. Belirlenen tarama hattının tüm kenar bucketları etkin kenar tuplerine kopyalayın
   b. Xofymin değerlerine göre bir ekleme, sıralama gerçekleştirin
   c. Ymax, tarama çizgisine eşit veya daha büyük olan tüm kenar bucketları kaldırın.
   d. Herhangi bir tepe noktası varsa, aktif tuple kenarlarının Fillup çiftleri, şu talimatları için adımları izleyin:
      - Köşede kesişen iki çizgi de tarama hattının aynı tarafındaysa, bunu iki nokta olarak düşünün.

- Köşede kesişen çizgiler tarama hattının zıt taraflarındaysa, bunu sadece bir nokta olarak düşünün.

e. Her bir bucket için eğim ekleyerek xofymin'i güncelleyin.

```
// C ++ 'da Tarama Hattı Doldurma Algoritmasını Kullanarak Çokgen Doldurma Programı.
#include <conio.h>
#include <iostream>
#include <graphics.h>
#include <stdlib.h>
using namespace std;
// Sınıf noktası beyanıclass point
{
  public:
  int x,y;
};
class poly
{
  private:
    point p[20];
    int inter[20],x,y;
    int v,xmin,ymin,xmax,ymax;
  public:
    int c;
    void read();
    void calcs();
    void display();
    void ints(float);
    void sort(int);
};
void poly::read()
{
  int i;
  cout<<"\n\t SCAN_FILL ALGORITHM";
  cout<<"\n Enter the no of vertices of polygon:";
  cin>>v;
  if(v>2)
  {
    for(i=0;i<v; i++) //ACCEPT THE VERTICES
    {
      cout<<"\nEnter the co-ordinate no.- "<<i+1<<" : ";
      cout<<"\n\tx"<<(i+1)<<"=";
      cin>>p[i].x;
      cout<<"\n\ty"<<(i+1)<<"=";
      cin>>p[i].y;
    }
    p[i].x=p[0].x;
    p[i].y=p[0].y;
    xmin=xmax=p[0].x;
    ymin=ymax=p[0].y;
  }
  else
    cout<<"\n Enter valid no. of vertices.";
}
// BULMA İÇİN FONKSİYON
void poly::calcs()
{ //MAX,MIN
  for(int i=0;i<v;i++)
```

```
  {
    if(xmin>p[i].x)
    xmin=p[i].x;
    if(xmax<p[i].x)
    xmax=p[i].x;
    if(ymin>p[i].y)
    ymin=p[i].y;
    if(ymax<p[i].y)
    ymax=p[i].y;
  }
}
// EKRAN FONKSİYONU
void poly::display()
{
  int ch1;
  char ch='y';
  float s,s2;
  do
  {
    cout<<"\n\nMENU:";
    cout<<"\n\n\t1 . Scan line Fill ";
    cout<<"\n\n\t2 . Exit ";
    cout<<"\n\nEnter your choice:";
    cin>>ch1;
    switch(ch1)
    {
      case 1:
        s=ymin+0.01;
        delay(100);
        cleardevice();
        while(s<=ymax)
        {
          ints(s);
          sort(s);
          s++;
        }
        break;
      case 2:
        exit(0);
    }
    cout<<"Do you want to continue?: ";
    cin>>ch;
  }while(ch=='y' || ch=='Y');
}
void poly::ints(float z) //Tanımlama Fonksiyonları
{
  int x1,x2,y1,y2,temp;
  c=0;
  for(int i=0;i<v;i++)
  {
    x1=p[i].x;
    y1=p[i].y;
    x2=p[i+1].x;
    y2=p[i+1].y;
    if(y2<y1)
    {
      temp=x1;
```

```
        x1=x2;
        x2=temp;
        temp=y1;
        y1=y2;
        y2=temp;
      }
      if(z<=y2&&z>=y1)
      {
        if((y1-y2)==0)
        x=x1;
        else
// Belirli bir uzaklıktan sonra poligonumuzu doldurabilmek için x'de değişiklikler yapmak için kullanılır.
    {
          x=((x2-x1)*(z-y1))/(y2-y1);
          x=x+x1;
        }
        if(x<=xmax && x>=xmin)
        inter[c++]=x;
      }
    }
}

void poly::sort(int z) //SIRALAMA FONKSİYONU
{
   int temp,j,i;

      for(i=0;i<v;i++)
      {
        line(p[i].x,p[i].y,p[i+1].x,p[i+1].y); // Bir çokgenin içi boş ana hatlarını yapmak için kullanılır.
      }
      delay(100);
      for(i=0; i<c;i+=2)
      {
        delay(100);
        line(inter[i],z,inter[i+1],z);  // Çokgen doldurmak için kullanılır ....
      }
}

int main()
// ANA BAŞLANGIÇ
{
   int cl;
   initwindow(500,600);
   cleardevice();
   poly x;
   x.read();
   x.calcs();
   cleardevice();
   cout<<"\n\tEnter the colour u want:(0-15)->"; //Renk seçimi
   cin>>cl;
   setcolor(cl);
   x.display();
   closegraph(); //Graphtan çıkış
  getch();
   return 0}
```

**KAYNAKÇA:**


https://www.includehelp.com/basics/boundary-fill-and-flood-fill-algorithm.aspx
https://www.geeksforgeeks.org/boundary-fill-algorithm/
https://www.geeksforgeeks.org/scan-line-polygon-filling-using-opengl-c/
https://www.tutorialspoint.com/computer_graphics/polygon_filling_algorithm.htm
http://mycurlycode.blogspot.com/2016/02/fill-polygon-using-scan-line-fill.html