ISTANBUL TECHNICAL UNIVERSITY ★ INSTITUTE OF SCIENCE AND TECHNOLOGY

A NEW IMAGE PRINTING APPROACH

M.S. THESIS

Ayhan ÖZTURK B.S.

Date of submission : June 21, 1993

Date of Approval : July 29, 1993

Examination Committee

Supervisor : Doç. Dr. Melih PAZARCI

Member : Prof. Dr. Hakan KUNTMAN

Member : Yar. Doç. Dr. Ertuğrul ÜNLEBİ

JULY 1993

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

YENİ BİR GÖRÜNTÜ BASMA YAKLAŞIMI

YÜKSEK LİSANS TEZİ

Müh. Ayhan ÖZTÜRK

Tezin Enstitüye Verildiği Tarih : 21 Haziran 1993

Tezin Savunulduğu Tarih : 29 Temmuz 1993

Tez Danışmanı : Doç. Dr. Melih PAZARCI

Diğer Jüri Üyeleri : Prof. Dr. Hakan KUNTMAN

Yar. Doç. Dr. Ertuğrul ÇELEBİ

TEMMUZ 1993

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my thesis supervisor Doç. Dr. Melih PAZARCI for his encouragement, help and sympathy.

TABLE OF CONTENTS

ABSTRACT

This thesis is intended to improve the printing quality of a digitized image. Input information source is a digitized and processed (if required) image file. Destination device is a dot matrix or laser printer.

The image is represented by on nxm dimensional matrix. Each element of the matrix corresponds to 8-bit pixel intensity. This means that one of 256 different intensity values is possible for the elements of the matrix. Each image pixel is presented by a 16x16 dot matrix on the paper. Different pixel intensities correspond to a different number of dots in the 16x16-dot matrices. This means that 256 different pixel levels are presented by 256 grey levels on the paper. So, 256 different grey tones are possible for any image printout.

The problem which this work mainly deals with is how these different grey levels can be represented on a paper by the known dot matrix or laser printer. The more precise the printer, the better is the hardcopy.

In this thesis a new image printing algorithm has been developed. The algorithm is applied to base Halftone printing technique to determine the place of dots. The algorithm manages the placement in relation to intensity of neighbors dots. With that result, soft crossing between the different grey levels can be achieved. The algorithm is based on the static electrical charge interaction analogy.

In addition to the new algorithm, some known image processing techniques which are segmentation, regional approach, edge detection and so on.. can be applied to improve the image printing quality.

ÖZET

YENİ BİR GÖRÜNTÜ BASMA YAKLAŞIMI

Günümüzde hızla gelişen ve gittikçe genişleyen uygulama alanıyla görüntü işleme, bilgisayar teknolojisine paralel ilerlemektedir. Daha detaylı görüntü işleme, daha büyük miktarda bellek ve daha yoğun işlem gerektirdiğinden, bilgisayar teknolojisindeki hız ve kapasite artışı otomatik olarak bu alana ivme kazandırmaktadır.

Amaca uygun işlenmiş bir görüntünün başka bir uygulama ortamına aktarılması veya sözkonusu ortam tarafından kullanılması ile, yeni, ortama özgü teknolojik sınırlamalar ortaya çıkar. Buna belirgin bir örnek olarak işlenmiş bir görüntünün kağıda basma aşamasında yazıcı teknolojisinin getirdiği sınırlamalar verilebilir. Bir görüntüyü günümüz monitor teknolojisiyle, 8-bit örnekleyip gözlemek mümkün iken, çok yaygın olarak kullanılan yazıcılarla aynı duyarlılıkta basmak mümkün değildir. Son zamanlarda lazer yazıcılardaki gelişmeler bu konuda oldukça umut verici durumdadır.

Bu çalışma ile, sayısallaştırılmış ve işlenmiş bir görüntünün kağıda aktarılması aşamasında katkıda bulunmak amacıyla, sadece kağıda basma tekniği üzerinde yoğunlaşılmıştır. Bu amaçla yeni bir görüntü basma algoritması geliştirilmiş ve denenmiştir.

Basılacak görüntü siyah-beyaz olup, nxm boyutlarında bir matris ile temsil edilmektedir. Genelde 512x512, 256x256 ve 128x128 gibi matris boyutları çok sık kullanılmakla birlikte, istenilen herhangi bir matris boyutu da kullanılabilir. Her bir matris elemanı 8-bitlik bir sayıdan oluşmaktadır. Bunun anlamı ise herhangi bir matris elemanı için 0 ile 255 gri seviyeleri arasında yanlızca bir gri seviyesi olasıdır. Böylece herhangi bir görüntü için, maksimum 256 tane gri seviyesi olasıdır. Görüntü matrisini yazıcıya aktarmada yine çok kullanılan yarıtonlama tekniği kullanılmıştır. Negatif görüntü için, bu tekniğe göre, görüntüdeki her bir gri seviyesi, yazıcıda belirli ve sabit bir fiziksel alanda aynı sayıda nokta yoğunluğu ile temsil edilmektedir. Pozitif görüntü basmak için ise, sözkonusu gri seviyesini 255 den çıkardıktan sonra, karşı düşen nokta yoğunluğu ile temsil edilir. Ya da, negatif görüntüde sözkonusu gri seviyesine karşı düşen nokta yerleşiminin tersi alınarak

basılabilir. Yazıcıda 256 tane farklı gri seviyesi basabilmek için birim fiziksel alan 16x16 kare nokta matris seçilmiştir. Matrisin tümden boş olması en düşük gri seviyesi olan, sıfır gri seviyesine karşı gelmektedir. Matrisin bir nokta hariç tüm noktalarının dolu olması ise, en koyu gri seviyesi olan 255'e karşı gelmektedir. Burada kağıt üzerindeki birim 16x16 nokta matrisin fiziksel alanı, basılan görüntünün kalitesini direkt belirleyen en önemli faktördür. Bir yazıcı ne kadar küçük bir alna birim matrisi sığdırabiliyorsa, o oranda kaliteli görüntü basacak demektir. Diğer bir deyimle bir yazıcının nokta çözünürlüğü ne kadar büyükse o oranda iyi görüntü basacak demektir. Yaygın olarak kullanılan nokta matris yazıcıların çözünürlüğü, inç başına en fazla 240 nokta cıvarındadır. Son yıllarda gelişmeye başlıyan lazer yazıcıların çözünürlüğü inç başına 300 noktayla başlayıp bugün 1200 nokta yoğunluğuna ulaşmıştır. Hatta, renkli yazıcılarda çözünürlük inç başına 2400 nokta yoğunluğuna ulaşmıştır. Bütün bu gelişmeler bize yakın gelecekte fotoğraf kalitesinde görüntü basabileceğimizi işaret etmektedir. Basılacak görüntü için, kabul edilebilir birim fiziksel alan belirli olsun. Bu alana, çözünürlüğü yüksek olan bir yazıcının çok daha fazla nokta yerleştirebileceği açıktır. Bu da direkt, olarak çözünürlüğü yüksek olan bir yazıcıyla, çok daha fazla sayıda gri tonlamanın yapılabileceğine, dolayısıyla çok daha iyi kalitede görüntü basılabilineceği sonucuna götürür. Buna belirgin bir örnek olarak, 512x512 boyutlarındaki bir görüntü 8 inçlik standart A4 kağıdına basmak istiyelim. Eğer görüntü 256 tane farklı tonlamayla basılacaksa, bu durumda kulanılacak yazıcının çözünürlüğünün inç başına 1024 nokta olması gerektiği kolayca hesaplanabilir. Yine aynı görüntüyü, aynı alana, inç başına 300 nokta çözünürlüğe sahip bir yazıcıyla basmak istersek, bu durumda 4x4 nokta birim yerleşim matrisleriyle, ancak 16 gri seviyesiyle basabiliriz. Yani 256 gri seviyesine sahip bir görüntüyü, kağıtta 16 gri seviyesiyle temsil edebilmekteyiz. Buda basılan görüntüde önemli bilgi kayıbına neden olmaktadır. Yine basılan görüntünün kalitesi gözönüne alınmadan, 512x512 boyutlarındaki bir görüntü 256 gri seviyesi ile basılmak istenirse, yine aynı yazıcıyla, 27"x27" (69cm x 69cm) boyutlarında bir alana yerleşebilmektedir. Sonuç olarak herhangi bir yazıcı teknolojisi seçildiğinde, basılan görüntünün kalitesi gözönüne alınmadığında, gri seviye sayısı ile fiziksel alan, ters orantılıdır. Birini artırmak istediğimizde, diğerini küçültmemiz gerekir. Diğer taraftan, basılan görüntü seçilen yazıcı teknolojisine göre büyüyebilmekte. Ya buna göz yumulur, yada bire-bir dönüşüm için gerekeli olan gri seviyesi saptanıp buna göre görüntü basılır. Ikincinin seçilmesi durumunda basılan görüntüdeki ayrıntılar kaybolur. Kabul edilebilir bir görüntü basmak istendiğinde, bunun en az bire-bir boyutları sağlıyacak minimum çözünürlüğe sahip bir yazıcı teknolojisi ile basılmasıyla amaca ulaşılmış olacaktır.

Bu çalışmada, basılacak görüntü boyutları çok fazla önemsenmeden daha çok basılacak birim matrislerin içindeki noktaların yerleşiminin nasıl olması gerektiği sorusuna cevap aranmış ve yerleşim mekanizması kurulmaya çalışılmıştır. Bu amaçla yeni bir yerleşim algoritması geliştirilmiştir. Yarıtonlama tekniği ilk kullanılmaya başlandığında, her bir gri seviyesine karşılık daha önce oluşturulmuş hazır yerleşim fontları kullanılarak görüntü basılırdı. Bunun, basılan görüntü üzerinde yapay desenler oluşturmasından dolayı yeni yöntemlere yönelindi. Buna bir çözüm olarak, yarıtonlama yapılmış görüntüye gürültü katıldı. Bu da basılan görüntü üzerinde billurlaşan gürültüye neden oldu. Diğer bir yöntemde matris içindeki noktaların rasgele konumlandırılmasıyla yapay desenlerin oluşumu ortadan kaldırılmaya çalışıldı. Yine, buda, görüntü üzerinde billurlaşan gürültüye neden oldu. Halen pratikte kullanılmakta olan yöntemler temelde yarıtonlama tekniğine dayanmakta ve değişik eşikleme teknikleri kullanmaktalar. Bunlara, Floyd-Steinberg, Knuth ve nokta difüzyonu gibi algoritmalar örnek verilebilir. Yeni geliştirilen algoritma, diğer algoritmalar gibi yarı tonlama tekniğine dayanmaktadır. Algoritma, ne gürültü ilavesi, ne de eşikleme tekniklerini kullanmamaktadır. Algoritma görüntü üzerinde hiç bir ilave işlem yapmadan tüm doğal yapısını olduğu gibi korumaktadır. Bu nedenle basılan görüntünün de doğal yapısına çok daha yakın olması beklenmektedir. Algoritma basılacak görüntüye ait birim matristeki noktaların yerleşimini yapmaktadır. Bunu yaparken doğu, batı, kuzey ve güney komşularına bakarak çok yoğun komşulara yakın bölgelerde, daha çok nokta yığılmasını sağlıyacak şekilde bir mekanizma ile yerleşimi yönetmektedir. Zaten, bir resim gözönüne alındığında, resmin noktaları arasında bütünleştirici bir özilişki vardır. Yani noktalar yalnız başına pek bir anlam oluşturmazlar. Fakat noktalar, aralarındaki özilişki ile bir araya getirildiğinde bir anlam oluştururlar. Yönetmi yapacak algoritma statik elektrik etkileşim analojisine dayanmaktadır. Kurulan analojiye göre yerleşimi yapılacak görüntü noktasının doğu, batı, kuzey ve güney komşularının gri seviyelerine aynı sayıda statik pozitif elektrik yükü karşı getirilmektedir. Yerleştirilecek noktanın gri seviyesi de, aynı sayıda negatif yükle temsil edilmektedir. Komşuların yük eşdeğeri yerleşim matrisinin kenarlarının ortasına konulmakta, yerleşimi yapılacak gri seviyesinin tek bir negatif yük karşılığı olan nokta, matrisin ortasına konur. Statik elektrik etkileşiminden bilindiği gibi, negatif yük bileşke kuvvet doğrultusunda hareket edecektir. Böylece birinci noktanın yerleşim yeri kabaca belirlenmiş olur. Kaba yerleşimi yapılan nokta, yerleştiği bölgenin komşularıyla nötürleşir. Bu da, herbir nokta yerleştikten sonra bileşke kuvvetin yön değiştirmesini sağlamaktadır. Bu işlem sözkonusu birim matrisin gri seviyesinin bütün noktaları yerleştirilinceye kadar devam edilir. Aynı işlemler bütün bir görüntüyü basmak için tekrarlanır.

Yerleşimi yapılmış bir gri seviyesi 16x16 boyutlarında bir matris olup, gri seviyesi kadar 1, gerikalan elemanları sıfır olan bir görünüm sunar. Sözkonusu birim matris 32-bayt bir data ile temsil edilmektedir. Her bir baytta, eğer bir bit 1 ise yazıcıda ilgili pozisyonuna bir nokta konur. Eğer sözkonusu bit sıfır ise, yazıcıda ilgili pozisyon boş bırakılır. Yerleşimi yapılmış birim matristeki noktalar daha çok gri seviyesi yüksek olan kenarlarda toplanmaktadır. Eğer yerleşimi yapılan noktanın gri seviyesi komşularına göre daha yüksek ise, birim matris içindeki noktalar matris merkezine doğru yoğunlaşır. Yok eğer, komşuları daha yoğun bir seviyede ise, noktaların yoğunluğu kenarlara doğru artacaktır. Bu da zaten bir görüntüdeki surekliliğin bir sonucudur. Sonuçta bir görüntünün her bir gri seviyesi için 32 baytlık data oluşturulur. 128x128 boyutlarındaki bir görüntü için 524,288 baytlık bir data oluşur. Bu datanın yazıcıya aktarılması, ortalama 15 dakika zaman gerektirmektedir. Algoritmanın en büyük dezavantajı zaman alıcı olmasıdır. Ancak bilgisayar teknolojisindeki hız ve kapasite artışı bunu problem olmaktan çıkaracaktır. Hatta algoritmanın yazıcıya yerleştirilmesi bu problemi büyük ölçüde ortadan kaldıracaktır. Burada asıl üzerinde durulması gereken durum yazıcının nokta çözünürlüğüdür. Yazıcı çözünürlüğünün düşük olması, yapay desenlerin oluşmasına neden olmaktadır. Bu nedenle 256 gri seviyesiyle basılacak bir görüntü için, inç başına 1024 nokta çözünürlük, kabul edilebilir bir görüntü basabilir. inç başına 1200 nokta basan lazer yazıcılar henüz yeni piyasaya sunulduklarından, algoritma bu tür yazıcılarda denenmemiştir. Ayrıca algoritmanın, şu an yaygın olarak kullanılan görüntü basma algoritmalarıyla karşılaştırması, yapılamamıştır. Henüz gelişme aşamasında olan algoritma nokta yerleştirme sırasında zaman zaman zorlanmaktadır. Bu durumları düzeltecek yeni çözümler üreterek, tam performansı gözlenebilir. Şu an algoritma kabaca çalışmaktadır. Geçici çözüm getirilmiş bazı yerleşim durumları mevcuttur. Ayrıca görüntü işlemede uygulanan bir çok yöntem, (kenar yakalama, bölgesel işlemler... gibi) algoritmaya kılavuzluk ederek, görüntünün bazı özelikleri ön plana çıkarılabilir. Yine bazı özelikleri de bastırılarak kağıda aktarılablir. Ayrıca bütün bu teknikler basma kalitesini de iyileştirebilirler. Algoritmanın diğer bir özelliği de komşu gri seviyeleri arasında yumuşak geçişi sağlamasıdır. Bu durum çok keskin kenarların önemli olduğu bir görüntü için istenmez. Ancak böyle durumlarda kenar yakalama tekniği algoritmaya kılavuzluk ederek, yumuşak geçişler, bölgesel olarak ortadan kaldırılabilir. Algoritma kolaylıkla renkli görüntü basabilecek şekilde yeniden düzenlenebilir. Ancak bu durumda görüntü yerleştirme ve yazıcıda basma zamanının üç katına çıkacağı unutulmamalıdır.

Sayısallaştırılmış bir resmin kağıt üstüne aktarılması, özellikle masa-üstü yayıncılıkta yoğun olarak kullanılmaktadır. Burada ulaşılan kalite yazıcı teknolojisinin sunduğu olanaklarla sınırlıdır. Görüntü basmanın, yakın gelecekte yoğun ilgi bulacağı diğer bir uygulama alanı da, yaygınlaşması beklenilen elektronik gazeteciliktir. Herhangi bir belgenin veya haberin anında kopyasının tutulması için kağıt ortamı son derece pratik ve ucuz olacağından, çok yaygınlaşması beklenmektedir. Ayrıca çıplak gözle ulaşma imkanı sağlamasıyla da diğer görüntü saklama yöntemlerine oranla önemli avantajlar sağlamaktadır. Yine yakın gelecekte görüntü işlemenin yoğun olarak uygulandığı tıp alanında, görüntünün negatif film yerine kağıda aktarılması çok daha pratik ve ucuz olacaktır. Fotoğraf makinalarından alınan görüntünün, kimyasal bir işlem gerktirmeden, fotoğraf kalitesinde, anında kağıda aktarılması yakın zamanda gerçekleşmesi beklenmektedir.

Bu çalışmanın sonunda yerleşim algoritmasını içeren bir görüntü basma programı yazılmıştır. Program herhangi bir dosya formatındaki görüntüyü, verilen bir pencereyle ekranda kaba hatlarıyla göstermektedir. Bu pencereyi, verilen bir dosya ismiyle saklayabilmektedir. Pencere boyutları dikkate alınarak, yeni algoritma ile yerleşim yapıldıktan sonra, PLACE.DAT isimli dosyada yerleşim bilgisi tutulur. Daha sonra seçilen bir yazıcı tipi ile yazıcıya aktarılır. Program EPSON LX 400, HP LASER JET III ve JET IV yazıcılarını desteklemektedir. Program ayrıca bazı görüntü işleme tekniklerini opsiyon olarak içermektedir. Bu tekniklere ilişkin alt programlar yazılmamıştır.

CHAPTER 1 INTRODUCTION

An image is constructed from energy reflected from three dimensional visible or nonvisible objects or scene. Detection of an image can be done by human eyes or electronic sensors, such as camera.
An electronic sensor converts three-dimensional light energy to electrical signal, as intensity of the image. After detection of the image, processing procedure starts. First the image is sampled with convenient sampling level, then quantized with convenient level. Both sampling rate and quantization levels are the vital points of image processing success. At the result quantized image is saved as digital image.
Digitized image can be processed for different purpose.

Edge detection, noise reduction, artificial intelligence are some of the processing purpose.
Assumed that, related image has been digitized and processed, if required. Printing that image file is the subject of this thesis.
In this thesis 512x512 dimensional and 8 bits quantized an image is processed.

1.1 Digital Image Representation

Both sampling interval and quantization level are vital during capturing the image. Sampling interval should satisfy Nyquist sampling criteria. According to criteria, sampling rate should be two times greater then the highest frequency of the image. If this not satisfied distortion of spectrum (aliasing) occurs[1]. According to CCIR A accepted bandwidth for a video signal is 5.5 M Hz. So sampling rate should be at least 11 MHz. For digital image 512x512 dimensional sampling of an image is accepted. For that reason, it can be shown a digitizer with 100 ns sampling interval is suitable for that propose.
It has been seen that 5 or 6 bits quantization of intensity of an image is acceptable. More than that quantization level results better quality image. Nowadays 32 bits quantization is in use.
In this thesis 512x512 dimensional and 8 bits quantized image is processed.

1.2 Halftone Printing Technique

Halftone technique which is in use only technique today's environment. Two type imagery is present.
First, continuous tone imagery is whose intensity quantization involves multiple grey level with no perceptible quantization. Example are television images and photographs. Halftone imagery is imagery which appears to have multiple grey levels due to halftone dots that vary locally over the area of coverage. Examples are newspaper magazine, and book images[2].

An digitized image is modeled as nxm dimensional matrix. Each element of matrix is the intensity of image at related point as binary number.

Mathematical model of the this can be expressed as follows :

$$0 \leq f(x,y) \leq 2^{l}-1 \qquad (1.1)$$
$$1 \leq x \leq n$$
$$1 \leq y \leq m$$

n : number of image columns
m : number of image rows
l : number of image intensity bits

In this work, l equals to 8. So $2^8=256$ different grey levels are used for processing of image.
An image can be represented as following matrix :

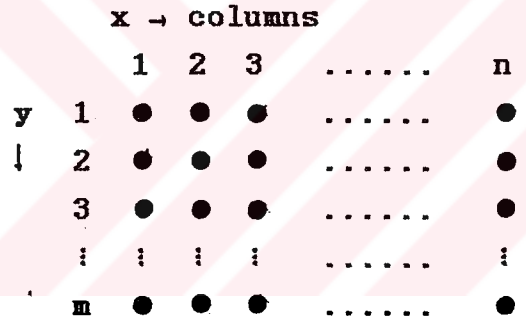


Figure 1.1 Matrix presentation of an image.

Printing an image file is the transformation from intensity value (i.e.,a binary number) to physical area on the paper via printing device.
The basic process of the transformation has been shown.

Figure 1.2 Image transformation scheme.

Transformation should be done point by point between the image matrix and printer.Varying image intensities in the matrix are correspond to varying dots density in the printing area. Intensities smaller than the heaviest dots can be printed in many different placement in the placement area. This means one-to-many transformation has been encountered. In this situation decision given an algorithm required.

Let us focus on the 8 bits grey level image transformation and take a point from image matrix :
assume that related position intensity is

$$f(2,2)=1 \qquad x=2\ ,\ y=2$$

Let us transform the intensity of image to the printer. Assume that printer font is 2x2 dots. Since transformation means to placement of that single dot in the 2x2 matrix.
Number of different placement combination is

$$\binom{4}{1}=\frac{4!}{(4-1)!\ 1!}=4$$

4 different placement are possible.

Placement as follows :

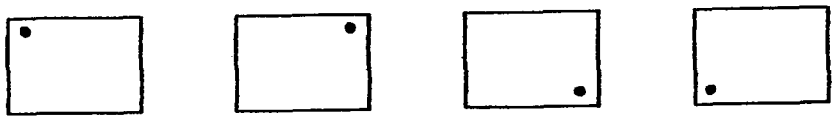


Figure 1.3 Different placement of a dot.

If image intensity correspond to 2, i.e., f(2,2)=2 then 6 different placement are possible.

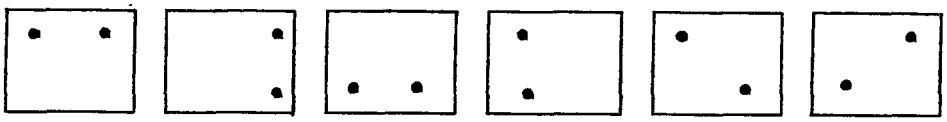


Figure 1.4 Different placement of two dots.

If f(2,2)= 3 , then 4 different placement are possible.

Figure 1.5 Different placement of three dots.

If f(2,2)= 4 , then 1 placement is possible.

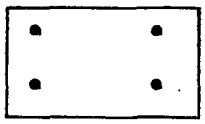


Figure 1.6 Placement of four dots.

In the present Halftoning strategy, two type transformation is in use.

1. Creation grey-scale fonts for pxp pixel reagion, are used to map individual input image grey-scale intencities into suitable pxp binary regions or characters in the output image. This yealds a conceptually and practically simple implementation scheme, but the appearance of false countours often precludes the use of this process alone.

2 Psedorandom thresholding and the aplication of the dither.

The visual appearance of false countours generated in haftone process is reduced by introduction of random noise to the halftone image. The perceptual effect of the introduction of this noise is dicreased concentration on high-frequency image information .

This spatially disturbuted noise couses the observer to locally integrate the intencity in a local region.

In practice, noise may be added to the input grey level prior to conversation to a bilevel image. Additionally, optimization of this process is possible. Since the process of pseudorandom noise addition followed by pixel thresholding is equivalent to the thresholding of the signal with a pseudorandom threshold. This is the base of dither matrix[2,3].

CHAPTER 2 NEW PLACEMENT ALGORITHM

With new implemented algorithm, placement of any image pixel in related position has been done with respect to its neighborhood intensities. Placement algorithm has been based on the static electrical charge-effect behavior. So, some assumptions have been done.

Assumption 1: All positive numbers which are corresponded to neighbors dots, represent same number of positive charges.

Assumption 2: All negative numbers which are corresponded to neighbors dots represent same number of negative charges.

Assumption 3: The dots number which is going to be placed always represents same number of negative charges.

Assumption 4: As in electricity, same type dots repel each other and different type dots attract each other.

Assumption 5: If the same number of opposite types of dots come close together, this results in neutralization. This means that, the resultant charge can not affect any other charges, and can not be affected either.

Assumption 6: The pixel which is going to be placed has square dots structure. So, west and east neighbors representative dots have been installed on related

directional line in same distance from the center of the square. And also north and south neighbors dots have been installed on related directional line in same distance from the center of the square.

Assumption 7: Center of the square and west, east, north and south directional lines are forbidden placement areas of the square. So, any dot can not be placed in these areas.

A pixel should be placed according to these assumptions. The same process is repeated for placement of whole image.

2.1 Review of Static Charge Interaction

Because static charge interaction analogy has been used for placement process, it has been found useful to review charge interaction for better understanding.
Lets get a positive and a negative charge, then place them apart from each other with a distance r. As shown in Figure 2.1 (a), the positive charge attracts the negative charge with known the force which is toward the positive charge. Because of the negative charge, same amount of force in opposite direction attracts the positive charge.

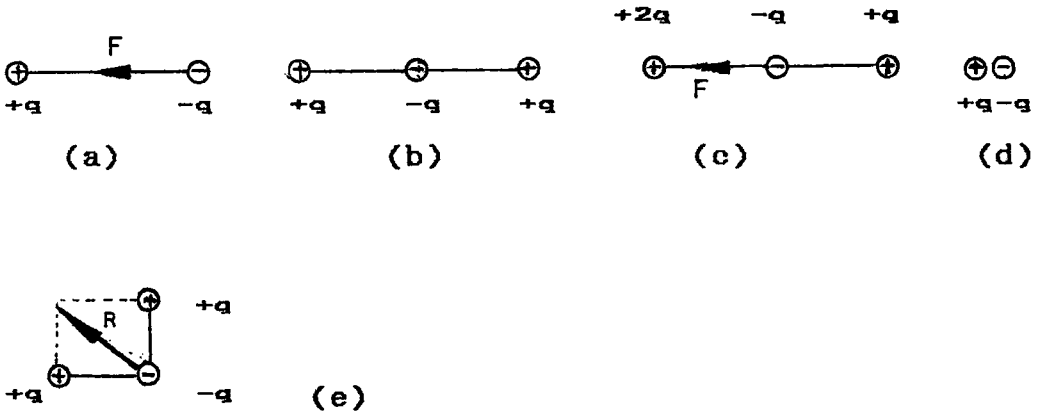


Figure 2.1 Static charge interaction

As known in electricity, attracting force is as follows:

$$F = \frac{1}{4\pi\epsilon}\frac{q^2}{r^2} \qquad (2.1)$$

F[newton]

$q = 1.6\ 10^{-19}$ C

$\epsilon = 8.85\ 10^{-12}$ (F/m)

In analogy, force magnitude is not important. The motion direction is only used result of the charge interaction. So, we do not deal with the magnitude of resultant force.

As can been seen in Figure 2.1 (b), a negative charge between the two positive charges in the same distance to each, can not move in any direction, remaining stationary.

In Figure 2.1 (c) a negative charge between the +2q and +q in the same distance from each, moves toward the +2q.

In Figure 2.1 (d), same number of different type charges close together neutralize each other. There is no any charge effect in result.

In Figure 2.1 (e) Different number of positive charges have been placed in perpendicular direction move a negative charge which is placed in intersection point, in direction of resultant force.

Assume that each neighbor of pixel has been placed at middle of each side of placement square, in same distance from center of the square. All possible movement directions have been shown in Figure 2.2.

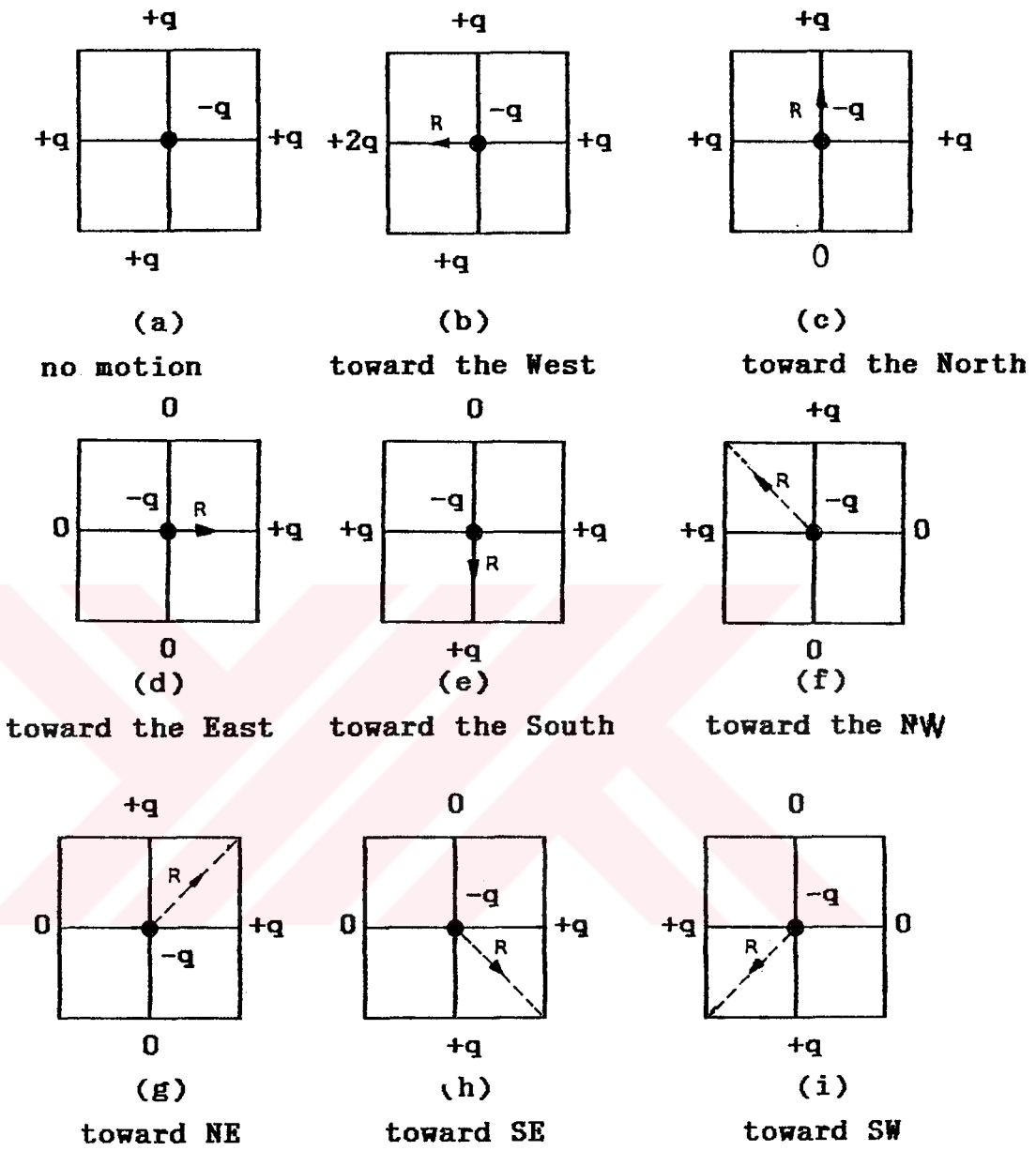


Figure 2.2 Motion direction of negative charge under west, east, north and south neighbor charges.

2.2 Presentation Limits of Pixels

As previously mentioned, each pixel of image has been presented as 16x16 dots matrix on the paper. Limits of presentation as follows:

Limit 1 : Each pixel which is going to be placed has been presented with zero as minimum dots number and 255 as

maximum dots number. Mathematically say,

$$0 \leq \text{intensity of pixel} \leq 2^8 - 1 \qquad (2.2)$$

counterpart to

$$0 \leq \text{dots number} \leq 2^8 - 1 \qquad (2.3)$$

Limit 2 : Any intensity of a pixel is presented as same number of dots for negative image. For positive image, each pixel should be presented as (255 - intensity) number of dots.

negative image :

$$0 \leq (\text{dots number} = \text{intensity}) \leq 2^8 - 1 \qquad (2.4)$$

positive image :

$$0 \leq (255 - (\text{dots number} = \text{intensity})) \leq 2^8 - 1 \qquad (2.5)$$

Limit 3 : An eight bit pixel is presented as 16x16 square matrix on the paper. So, zero intensity pixel correspond to full 16x16 square matrix, and 255 intensity pixel correspond to empty 16x16 square matrix for positive image. An other words, zero matrix correspond to white, full filled matrix correspond to maximum black shade. Any pixel intensity lies between the black and white corresponds to related shading.

Limit 4 : All dots which are corresponded to pixels intensity are placed. Some degenerative placement events can take place for particular neighbors.

2.3 Static Charge Like Dots Interaction

All of dot placement algorithm has been based on static charge like interaction. In certain placement situations dot which is going to be placed is under the diagonal resultant force. So, dots will move toward the cross of square. All other directions of movement are not certain. So, degenerative situations have been encountered. These kind of problems have been solved with redirecting the placement process. Some mechanism

has been implemented for that propose.

2.4 Simple Placement statements

if the resultant force over the dot which is going to be placed has diagonal direction, placement of dot is simple and certain.
All possible simple and certain placement situations are as follows :

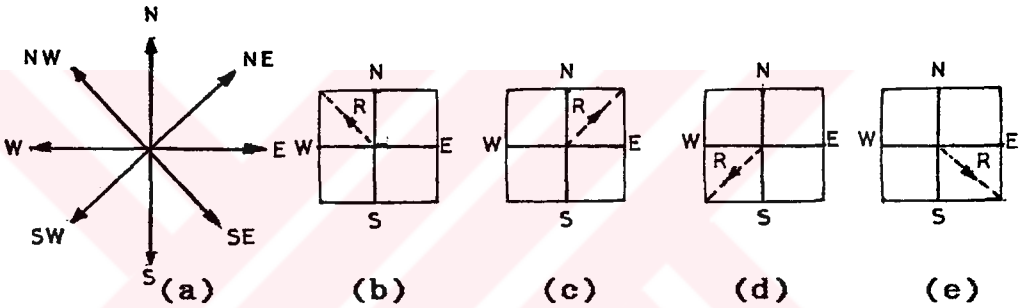


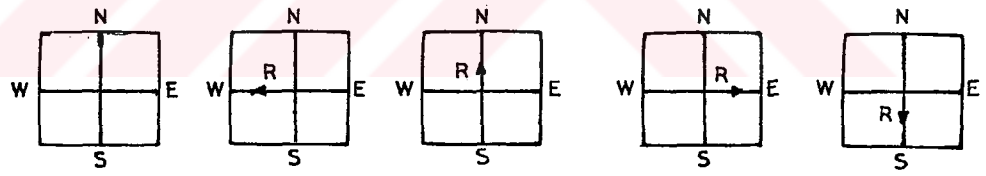
Figure 2.3 (a) All possible directions.
(b) North-west direction of resultant force.
(c) North-east direction of resultant force.
(d) South-west direction of resultant force.
(e) South-east direction of resultant.
force.

2.5 Forbidden Degenerative Placement Statements

Because of both the on-line and the center of square are forbidden regions of placement areas, any dot can not be placed in these areas. This is the cause of degenerative situations. Because of all dots which are correspond to image intensity should be placed, any

mechanism should be found to place these dots.
Because of idea of algorithm, it is meaningless to place a dot on the line or the center of the square. These regions are not defined.
Lets get a regular image which has same intensity neighbors. Lets try to place a single dot. Because the resultant force is zero, dot can not move to any direction and also center is forbidden region, a mechanism should found to place that dot. This statement will be discussed later on.
Again lets get west directed resultant force and try to place a dot. Two possible areas are present which are north-west and south-west. How this dot can be placed is the problem should be solved. Some mechanism should be found to place this kind of dots.

Forbidden degenerative statements are as follows :



(a) (b) (c) (d) (e)

Figure 2.4 (a) Zero resultant force statement.
(b) West directional resultant force.
(c) North directional resultant force.
(d) East directional resultant force.
(e) South directional resultant force.

2.6 Partition Maximum Dots Number

This definition comes from regular placement and limits over the placement area. It is impossible to place 255 dots in 8x8 matrix. The 8x8 matrix can take only 64 dots as maximum. As mentioned before, each matrix's dot corresponds to a certain physical area. It

is impossible to place a dot in not defined area. This kind of problems occur when try to place a dot with very dense one or two neighbors with respect to other neighbors.

The maximum dots number is one of dots number between 1, 4, 16, and 64 dots. It varies as power of 4.

If the number of dots which is going to be placed is smaller or equal to 4, maximum dots number is 1 dot.

If the number of dots which is going to be placed is smaller then 17 and grater than 4, the maximum dots number is 4 dots. If the number of dots which is going to be placed is grater then 16 and smaller then 65, the maximum dots number is 16 dots. If the number of dots which is going to be placed is grater or equal to 65, then maximum dots number is 64.

For placement of single pixel all of maximum dots number can be used because of subdivision process. The intensity of pixel is direct related parameter of maximum dots number. If the any partition of placement has raised to maximum dots number, it can not accept any more dots even if resultant force is directed to that partition. This is the job of maximum dots number mechanism.

2.7 Hidden Degenerative Statements

These degenerative statements occur during the execution of algorithm. The situations can not predicted from static charge analogy. This is why, it is called so.

During execution of algorithm, sequential three partition of placement area may be turned off, because of raising its maximum dots number. But still, neighbors direct the dots to that field with one of east, west, north or south related forbidden degenerative statement. During the execution of forbidden degenerative statement algorithm, decrease the dots number of neighbors to change direction of placement. But if the partition which is going to be

placed is turned off, and new direction is also turned off,then the process is continued to redirect placement to previous off partition causes problem. In this statement algorithm can not converge. So, new redirecting mechanism should be found for to place related dot. Placement should be directed to enabled partition. This means to the algorithm is forced to behave degenerative. An other degenerative behave can occur if the placement direction is diagonal directional to the disabled placement partition. In this placement situation, the algorithm decrease both diagonal neighbors by one, for to redirect the placement. And also, bidirectional placement can be evaluated as a degenerative statement. If the placement direction is bidirectional (i.e. line directional), algorithm first looks for polarity direction and enable statement of both partitions. If the both properties are satisfied for a partition, placement is realized. If the both partitions have only enable statement, the placement can be done in the one of these partitions. This causes to occurrence of pattern on the image for sequential same placement statement. A solution to that problem, dynamic partition switching mechanism has been appended to algorithm.

2.8 Polarizing Direction

Lets take a square has been divided its four partitions. And also, lets each partition has the subsquare which is going to be divided its four partitions. Lets main square has zero resultant force statement and take four dots to place. Without polarity direction, each dot will be placed in north-west subpartitions. This causes asymmetry problem. For to solve this problem, polarity direction should be defined for each subpartitions. This means to for north-west subpartition, polarizing direction should be north-west, for north-east subpartition, polarizing direction should be north-east, for south-west subpartition, polarizing direction should be south-west.

Polarizing direction enforces dot to be placed in this direction at first, if the placement is allowed.
Placement of four dots without polarizing direction and with polarizing direction has been shown as follows :

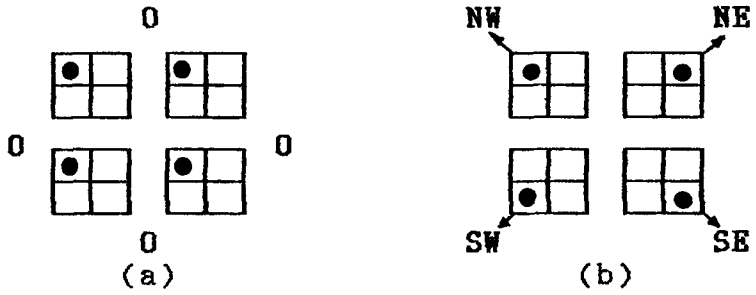


Figure 2.5 (a) Placement of 4 dots without polarizing direction.
(b) Placement of 4 dots with polarizing direction.

2.9 Escape Mechanisms of Degenerative Statements

1. Placement in zero resultant statement :

During the placement of a pixel, if the west and east neighbors have the same dots number and at same time, the north and south neighbors also have the same dots number, both may have negative numbers either, cause this statement. In this situation, algorithm looks for partition which has minimum dots number and places dot in this partition. If all partitions have same number of dots, then placement is done in first privilege partition. The partition which has placement privilege is determined by polarizing direction. If the partition is turned off statement, then placement is redirected toward the next privilege partition.

2. Placement in forbidden degenerative statements :

If dot has been directed toward the west, east, north or south line direction by related neighbors, cause to this problem. In this situation, placement is done according to the given polarity direction. Because two partitions are possible for placement, which one will be selected first, should be determined by the privilege information. If first selected partition has been turned to off statement, placement should be done in next passible partition. If the both are turned off statement, last partition neighbors should be decreased by one, to redirect the placement progress to other partitions.

3. Placement in the hidden degenerative statements :

If the placement has been directed toward the off statement partitions, cause the problem. If this is the result of diagonal direction of dot movement, both neighbors of related partition are decreased by one until dot has been redirected to other partitions. Then the placement is done.
If the problem is result of line directed statements, the algorithm first looks for first privilege partition. if it is turned off statement, off course it should be, then it looks for second privilege partition. If the second partition is also turned off statement, algorithm will decrease both neighbors of last partition, until directed to the different partition for placement.
If the next directed partition is also turned off statement, and resulting to non convergence statement, three neighbors of last partition are decreased by one. This selection has been done to encourage the neighbors to the zero degenerative statement. And then, the dot is placed according to the zero degenerative statement.

2.10 Result of Placement Process

At the result of placement of single pixel, the pixel pattern is changing with different neighbors set. If the neighbors intensities are denser then pixel itself, then dots of pixel will disturb around the side of square. If the pixel itself is denser then neighbors, then dots will disturb around the center of the square. And also, the placed dots distribution is denser around the side of square which has denser neighbors.
At the paper side of event, all these features result to very soft crossing between the image pixels is expected.

2.11 Inside of The Algorithm

Algorithm get a pixel which is going to be placed and its four neighbors. Do the placement of the pixel in 16x16 square dot matrix. Then, return 32 bytes as placement data. As can be predicted, huge placement data amount will be obtained at the result of whole image.
Lets make some calculations.
If the image which is going to be placed has
128x128 dimensions, placement data amount is

128*128*32 = 524,288 bytes

256x256 dimensions, placement data amount is

256*256*32 = 2,097,152 bytes

512x512 dimensions, placement data amount is

512*521*32 = 8,388,608 bytes.

And also, as can be predicted this amount of data needs huge storage memory and little placement time. This amount of storage, placement time are not too much today's computer environment. But original problem comes from printing devices which are have limit of 600 dots per inch. As mentioned before, to print out the 512x512 dimensional image to 8 inch paper 1024 dots per inch printer is required. This is the problem. We should wait for a while to see actual result of algorithm.

General algorithm flowchart is as follows :

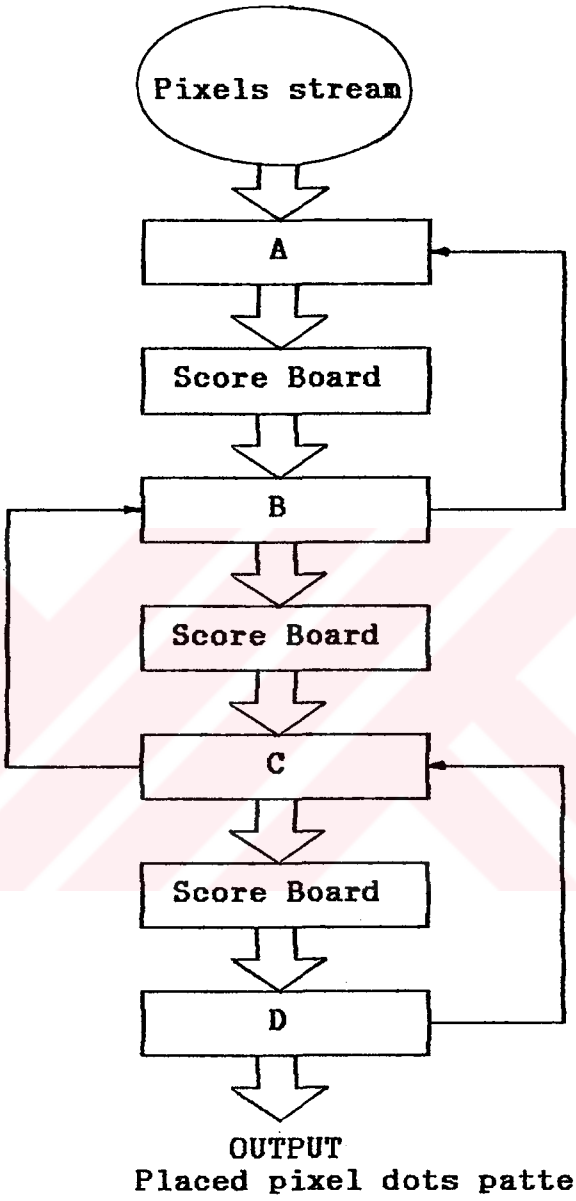


Figure 2.6 General flowchart of placement algorithm.

Single step division of whole pixel dot matrix to its submatrices as follows :

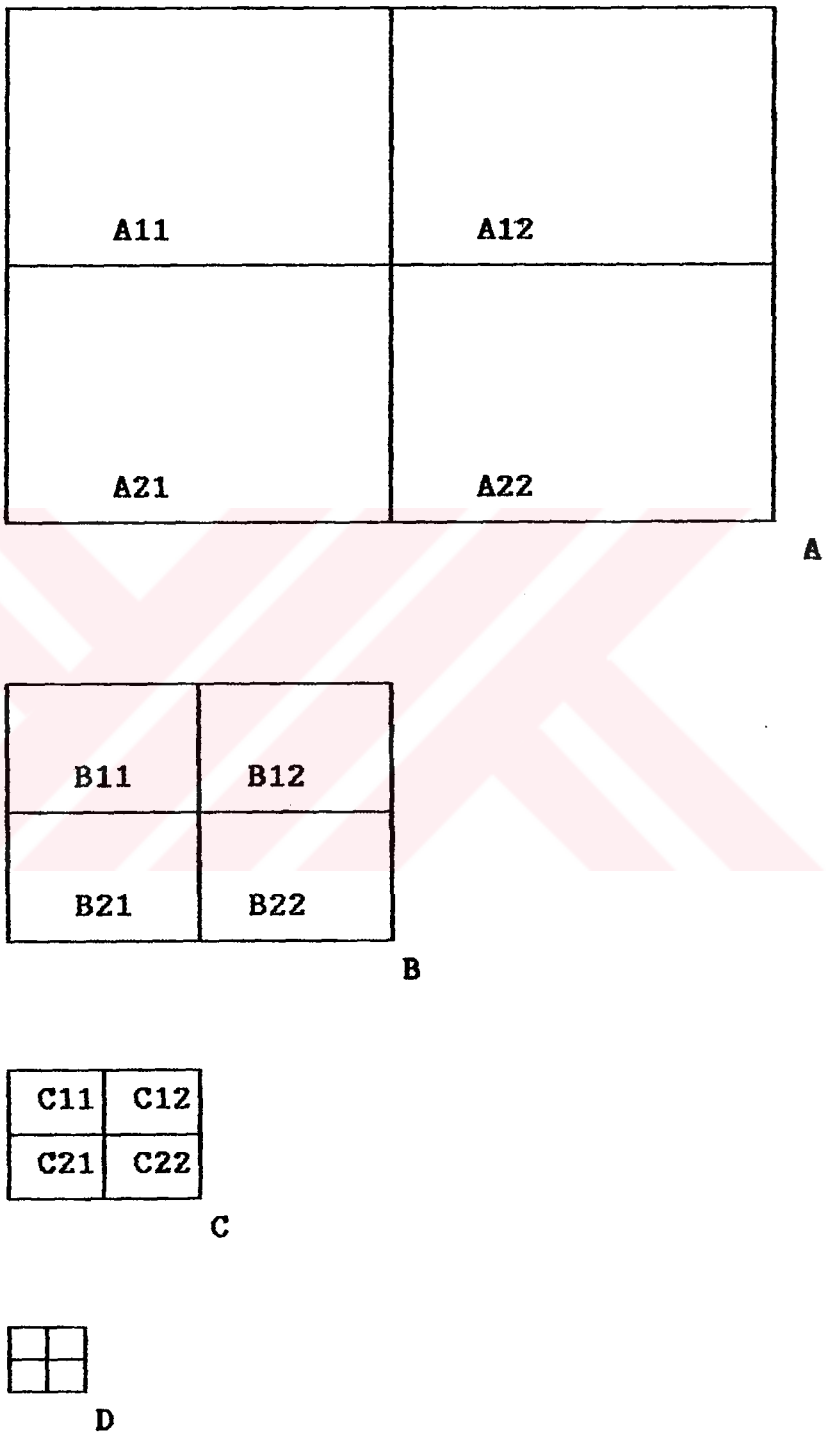


Figure 2.7 Single step division of the pixel dot matrix.

Lets analyze the single step through the algorithm flowchart.
Algorithm starts with getting pixel which is going to be placed and its four neighbors from image data stream, then tries to place the whole pixel dot matrix.
First call score board function to divide A matrix to its four partitions as A11, A12, A21, A22 with the taken neighbors. Then, each partition of A matrix is assigned to B submatrices. First get A11 partition and assign to B matrix, then call score board function to divide B matrix to its four subpartitions as B11, B12, B21, B22, with neighbors of A11. Calculation of neighbors for subdivision matrixes is vital point of algorithm. As can been seen following Figure 2.8, submatrixes of divisions are neighbors of each other for next division. This is the interactive property of the algorithm. So, placement is getting complicated.

Lets determine the neighbor of A11 :
West neighbor is (west neighbor of A matrix / 4).
East neighbor is A12 submatrix.
North neighbor is (north neighbor of A matrix / 4).
South neighbor is A21 submatrix.

Other neighbors of submatrices are determined in similar method. Because the pixel which is going to be placed, divided four partitions, the neighbor pixel intensity is also should be divide by four, for subdivision matrices. This is why mother matrix neighbors has been divided by four, for submatrix neighbors. As going to the down inside of flowchart, B matrix has been taken as mother matrix, then score board function has been called with related neighbors. Division partitions has been assigned to C matrices. C submatrices neighbors can be determined as above. Then single C matrix has been taken as mother matrix, and score board function has been called for division of C matrix. Returned partitions now, in bit level in D matrix. At the end of this process placement of four dots has been completed.

Placement will progress over the feedbacks to get the remained partitions of matrices.
Division of all mother matrices and their neighbors are as follows :

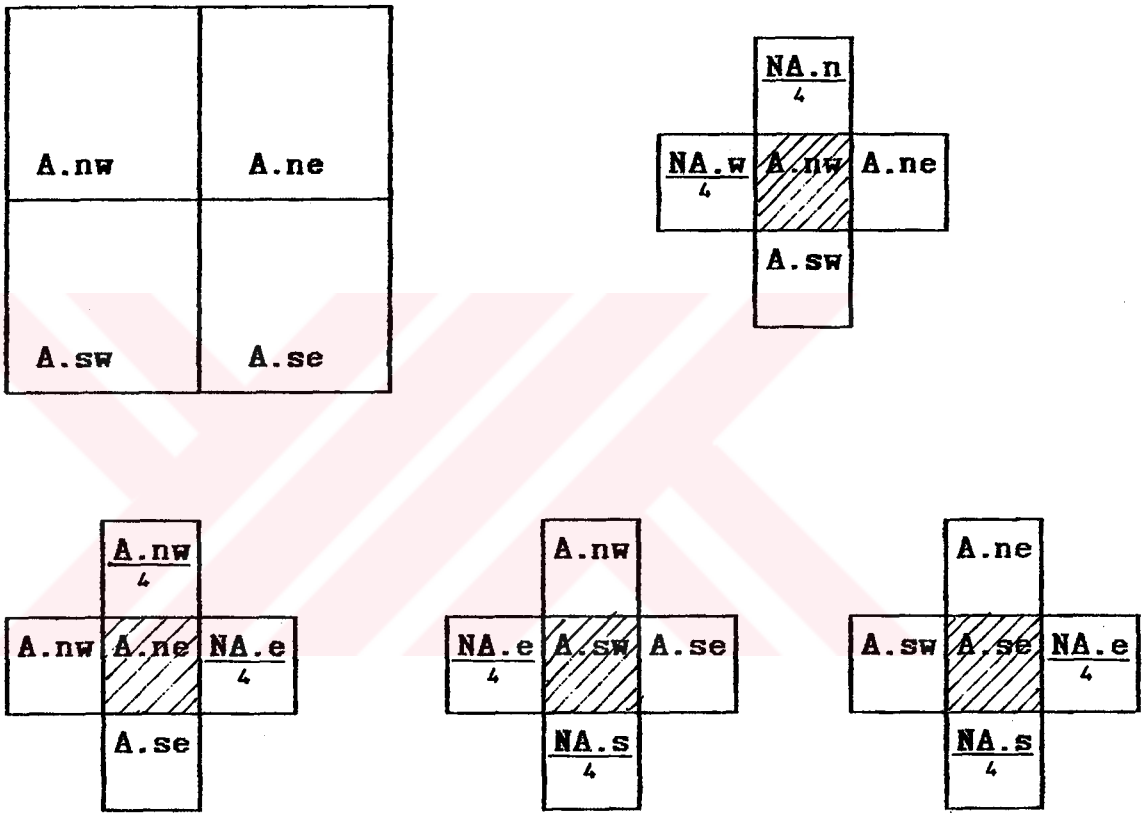


Figure 2.8 The partitions of A matrix and their related neighbors.

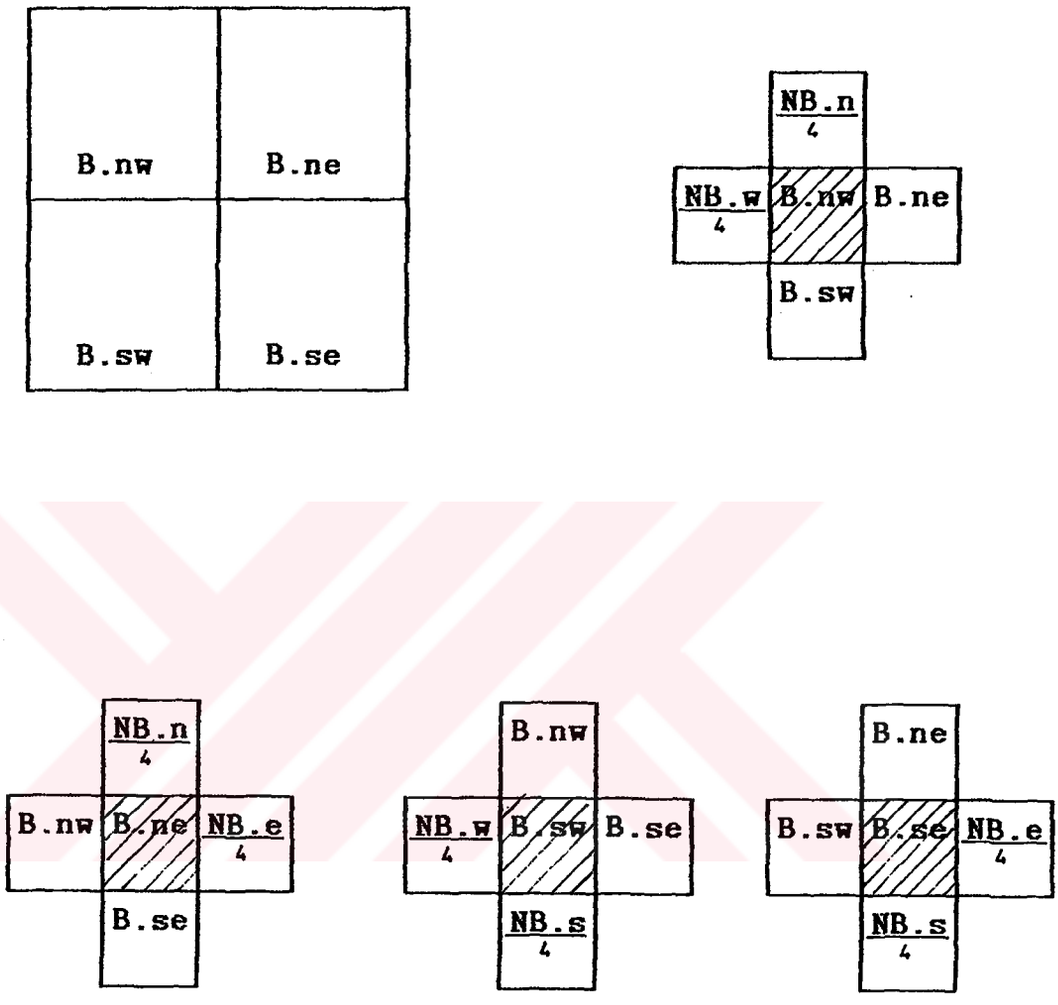


Figure 2.9 The partitions of B matrix and their related neighbors.

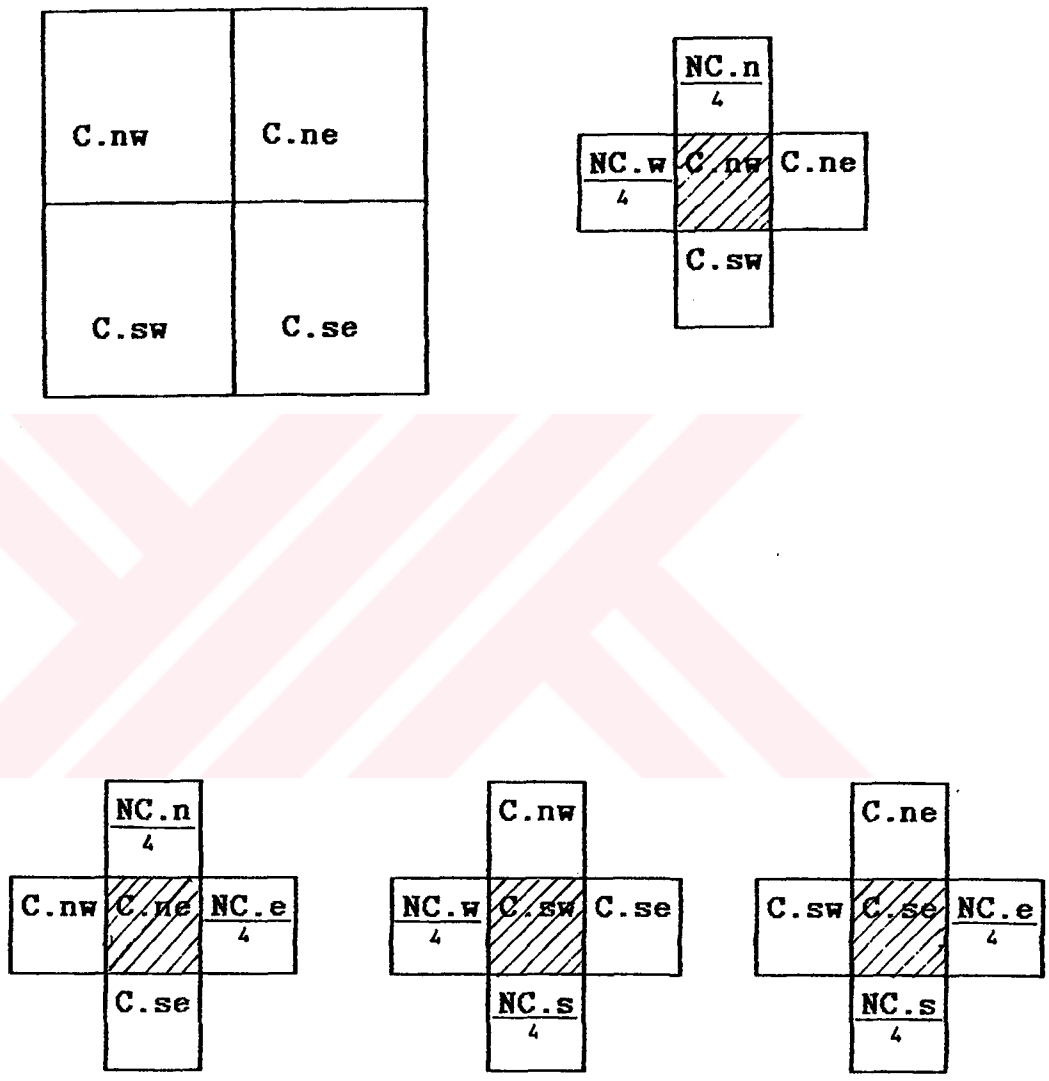


Figure 2.10 The partition of C matrix and its related neighbors.

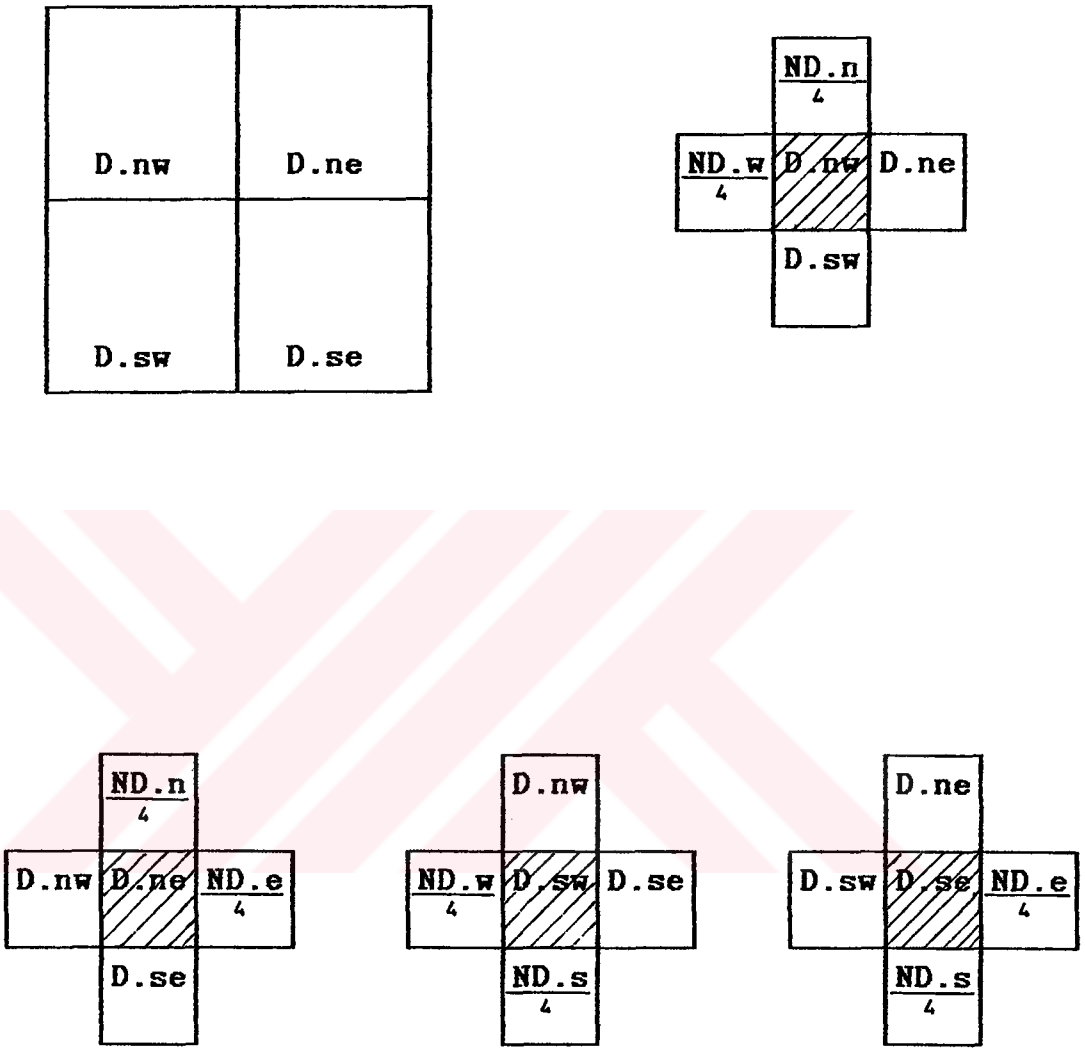


Figure 2.11 The partitions of D matrix and its related neighbors.

Same division is realized for remaining partitions of mother and child matrices. At the end the placement process 32 bytes has been saved for each pixel. And placement is progressed for whole image pixels.

In the placement matrix each bit correspond to a single

dot. So, if the bit is set, the printer place a dot on the paper. If not, then printer passes related place.

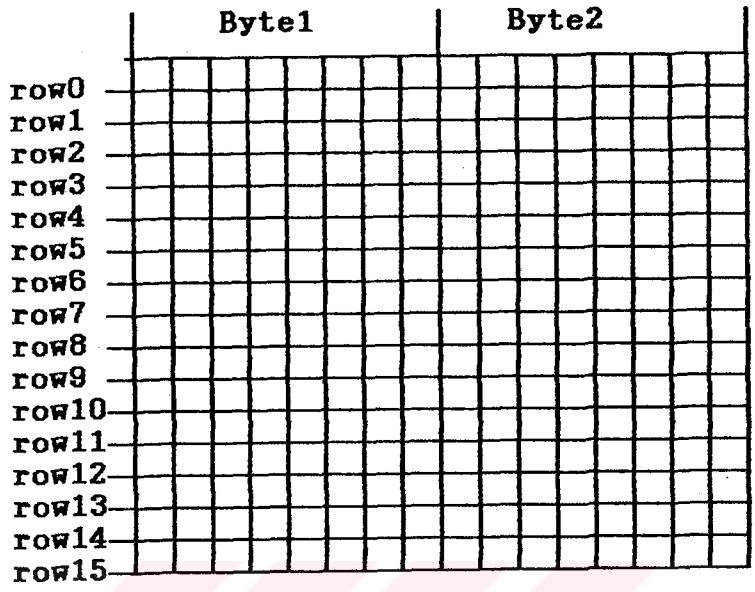


Figure 2.12 Appearance of the placed pixel in byte level.

2.12 Inside of Score Board Function

Score board function is the decision given part of algorithm. So, it is the heart of algorithm. The function gets number of dots and its related neighbors with polarizing direction, then return four partition of dots. This process is same for all division of mothers and their child matrices.

Some short hand words are used in the function as follows:

nw : stand for north-west partition
ne : stand for north-east partition
sw : stand for south-west partition
se : stand for south-east partition
west : stand for west neighbor
east : stand for east neighbor
north : stand for north neighbor

south : stand for south neighbor

During placement, the function gives the decision according to simple and degenerative statements. Use the all possible placement procedure as mentioned if required.
Lets do placement of 4 dots with all zero neighbors.
Maximum number of dots partition is 1. And polarity direction is nw.
First, the function calculates west-east and north-south differences, then interpret results for encouragement of dot. If placement partition has been determined then, decrease the remained dots number and both neighbors by one. Then recalculate neighbors difference for placement of next dot. The process is continued until all dots has been placed.

Table 2.1 Division Table

Dots Number	West	East	North	South	nw	ne	sw	se
4	-1	0	-1	0	1	0	0	0
3	-1	-1	-1	-1	off	0	0	1
2	-1	-2	-2	-1	off	1	0	off
1	-2	-2	-2	-2	off	off	1	off

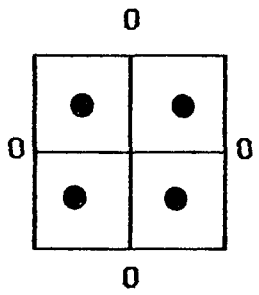
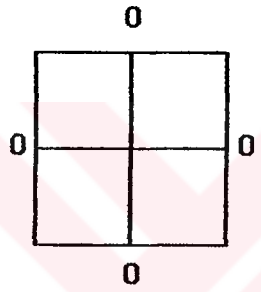


Figure 2.13 Division result dots matrix.

2.13 An Example of Placement Procedure

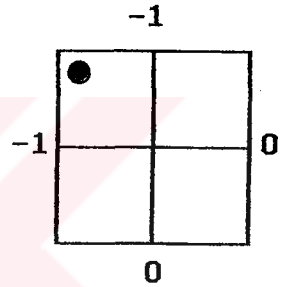
Lets take 8 dots which are going to be placed with all zero neighbors in three levels. This means that, placement will be started with the B matrix in the algorithm flowchart. This selection has been done because of simplifying the execution of the algorithm.

First level division results have been given as follows : Because the number of dots equals to 8, maximum dots number for each partitions should be 4 for the first level division.



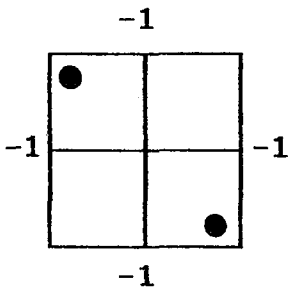
remained dots = 8
west - east = 0
north - south = 0

(a)



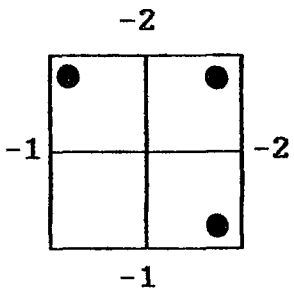
remained dots = 7
west - east = -1
north - south = -1

(b)



remained dots = 6
west - east = 0
north - south = 0

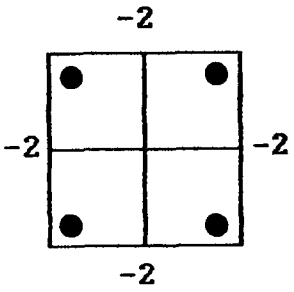
(c)



remained dots = 5
west - east = 1
north - south = -1

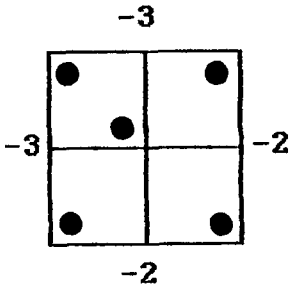
(d)

Figure 2.14 First level division.



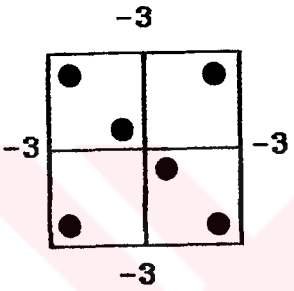
remained dots = 4
west - east = 0
north - south = 0

(e)



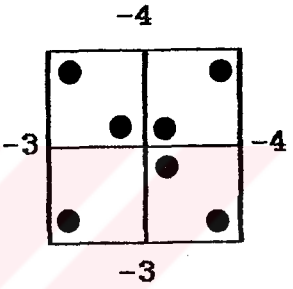
remained dots = 3
west - east = -1
north - south = -1

(f)



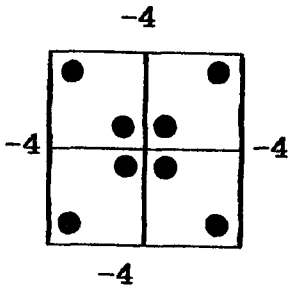
remained dots = 2
west - east = 0
north - south = 0

(g)



remained dots = 1
west - east = 1
north - south = -1

(k)



remained dots = 0
west - east = 0
north - south = 0

(l)

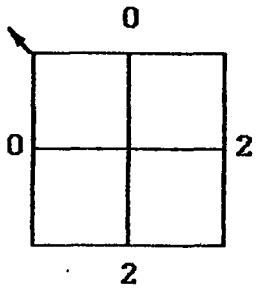
Figure 2.15 First level division is continued.

At the end of first division neighbors are in

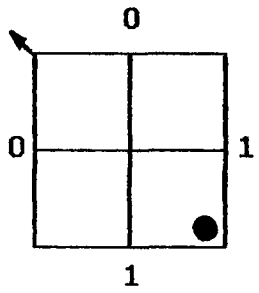
west = -4 east = -4 north = -4 south = -4
statement. These neighbors do not affect next divisions.
The first level division results of each partition dots number are as follows :
nw = 2 dots sw = 2 dots
ne = 2 dots se = 2 dots

For second level division, first level partitions dots number have been assigned to mother matrices dots number. Then, their child matrices allocation will be done as second level division.

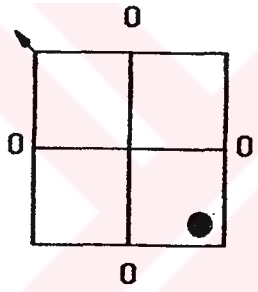
Division of nw partition :
Number of dots is 2.
Neighbors of first level nw are
west = (0/4) = 0 east = 2
north = (0/4) = 0 south = 2
Polarizing direction is north-west.
Maximum partition dots number is 1.



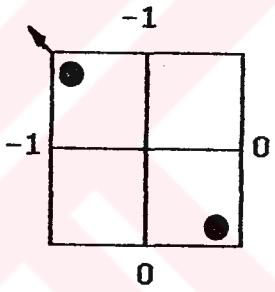
remained dots = 2
west - east = -2
north - south = -2
(a)



remained dots = 1
west - east = -1
north - south = -1
(b)



remained dots = 1
west - east = 0
north - south = 0
(c)



remained dots = 0
west - east = -1
north - south = -1
(d)

Figure 2.16 Division of the first level nw partition.

Division results :

nw = 1 ne = 0 sw = 0 se = 1

west = -1 east = 0 north = -1 south = 0

Lets do the division of first level ne mother partition to its second level division :

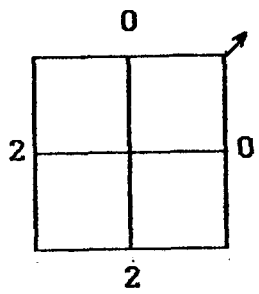
Number of dots is 2.

The maximum dots number is 1.

First level ne partition neighbors are

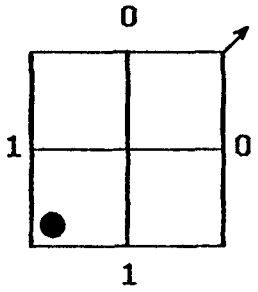
west = 2 east = (0/4) = 0 north = (0/4) =0 south = 2

Polarity direction is north-east.



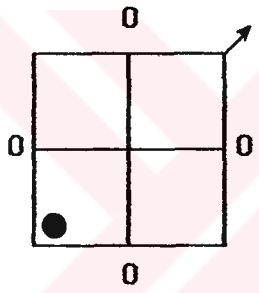
remained dots = 2
west - east = 2
north - south = -2

(a)



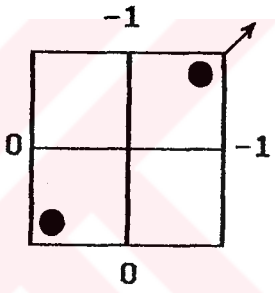
remained dots = 1
west - east = 1
north - south = -1

(b)



remained dots = 1
west - east = 0
north - south = 0

(c)



remained dots = 0
west - east = 1
north - south = -1

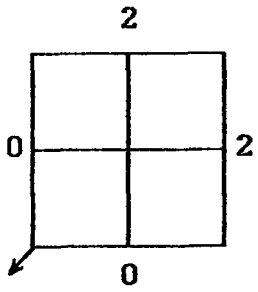
(d)

Figure 2.17 Division of the first level ne partition.

At the and of division, second partitions are
nw = 0 ne = 1 sw = 1 se = 0
And statement of neighbors are as follows :
west = 0 east = -1 north = -1 south = 0

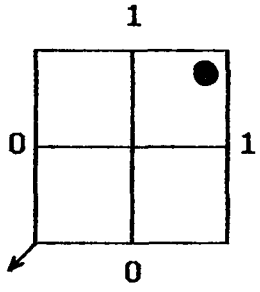
Lets do division of first level sw partition
Dots number is 2.
Maximum partition dots number is 1.
Polarity direction is sw.
Partition neighbors are

west = (0/4) = 0 east = 2 north = 2 south = (0/4) = 0



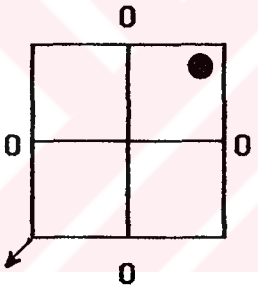
remained dots = 2
west - east = -2
north - south = 2

(a)



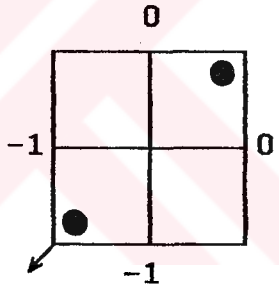
remained dots = 1
west - east = -1
north - south = 1

(b)



remained dots = 1
west - east = 0
north - south = 0

(c)



remained dots = 0
west - east = -1
north - south = 1

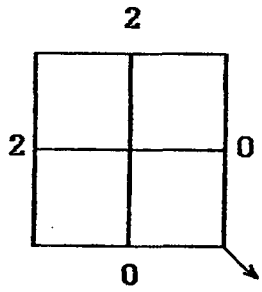
(d)

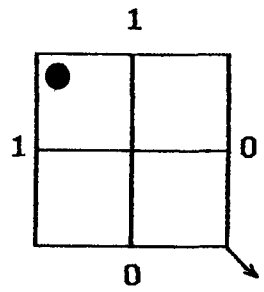
Figure 2.18 Division of the first level sw partition.

Division results :
nw = 0 ne = 1 sw = 1 se = 0
west = -1 east = 0 north = 0 south = -1

Lets do the division of first level se mother partition to its four child partitions.
Dots number which is going to be divided is 2.
Maximum partition dots number is 1.
Polarizing direction is south-east.

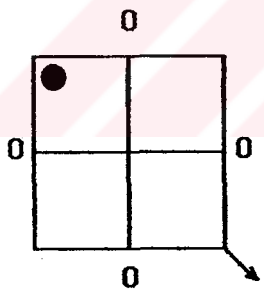
Partition neighbors are
west = 2 east = (0/4) = 0 north = 2 south = (0/4) = 0

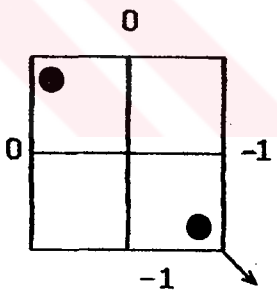




remained dots = 2
west - east = 2
north - south = 2
(a)

remained dots = 1
west - east = 1
north - south = 1
(b)





remained dots = 1
west - east = 0
north - south = 0
(c)

remained dots = 0
west - east = -1
north - south = 1
(d)

Figure 2.19 Division of the first level se partition.

Division results :
nw = 1 ne = 0 sw = 0 se = 1
west = 0 east = -1 north = 0 south = -1
At the end of second division submatrices have been formed, but still bit level assignment has not been done.

To do so, a third level division should be realized. After that third level division actual place of each dot will be determined with the bit assignment. Even if the dots number raise to single dot, it can not been known where is the actual place of a dot, unless the D submatrices are formed.

Lets do the third level division of second level partitions.
Second level division results :
for first level nw partition,

nw = 1 ne = 0 sw = 0 se = 1

for first level ne partition,

nw = 0 ne = 1 sw = 1 se = 0

for first level sw partition,

nw = 0 ne = 1 sw = 1 se = 0

for first level se partition,

nw = 1 ne = 0 sw = 0 se = 1

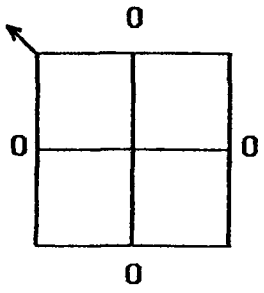
Since the partition which has zero dots number, always gives the zero subpartitions at the bit level, we can not pursue to calculate the submatrices of these type partitions.

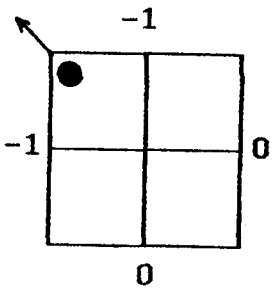
Lets do the division of nw1/se2 partition.
The number which is close to directions (i.e. nw1,ne2...) denotes the division level in following division process.

Dots number is 1.
Polarity direction is nw.
The maximum dots number is 1.
The neighbors are

west = 0 east = integer(2/4) = 0

north = integer(2/4) = 0 south = 0





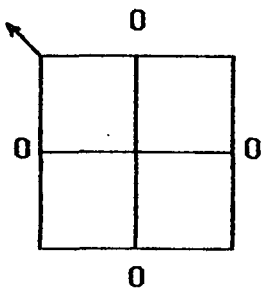
remained dots = 1
west - east = 0
north - south = 0
(a)

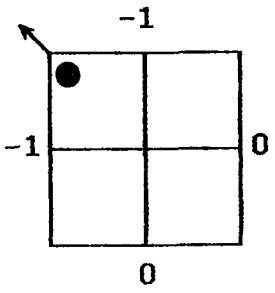
remained dots = 0
west - east = -1
north - south = -1
(b)

Figure 2.20 Division of the nw1/se2 partition.

Division results are
nw = 1 ne = 0 sw = 0 se = 0
D11 = 1 D12 = 0 D21 = 0 D22 = 0 assignments should be done at the bit level.

Lets do the division of nw1/nw2 partition.
Dots number is 1.
Maximum dots number is 1.
Polarity direction is nw.
Neighbors are
west = 0 east = 0 north = 0 south = 0





remained dots = 1
west - east = 0
north - south = 0
(a)

remained dots = 0
west - east = -1
north - south = -1
(b)

Figure 2.21 Division of the nw1/nw2 partition.

Division results are

nw = 1 ne = 0 sw = 0 se = 0

This means that, D11 = 1, D12 = 0, D21 = 0, D22 = 0 assignments should be done at the bit level.

Lets do the division of ne1/sw2 partition:

Dots number is 1.

Polarizing direction is ne.

Maximum dots number is 1.

Neighbors are

west = integer(2/4) = 0 east = 0

north = 0 south = integer(2/4) = 0

remained dots = 1
west - east = 0
north - south = 0

(a)

remained dots = 0
west - east = 1
north - south = -1

(b)

Figure 2.22 Division of the ne1/sw2 partition.

Division results are

nw = 0 ne = 1 sw = 0 se = 0

D11 = 0 D12 = 1 D21 = 0 D22 = 0 assignments should be done at the bit level.

Lets do the division of ne1/ne2 partition.

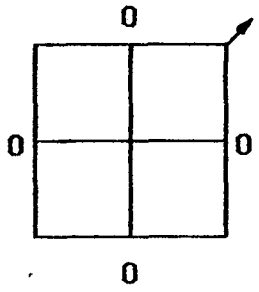
Dots number is 1.

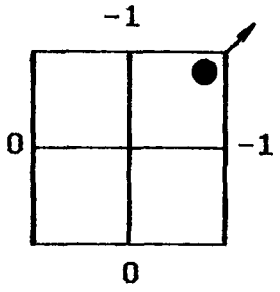
Maximum dots number is 1.

Polarity direction is ne.

Neighbors are

west = 0 east = 0 north = 0 south = 0





remained dots = 1
west - east = 0
north - south = 0
(a)

remained dots = 0
west - east = 1
north - south = -1
(b)

Figure 2.23 Division of the ne1/ne2 partition.

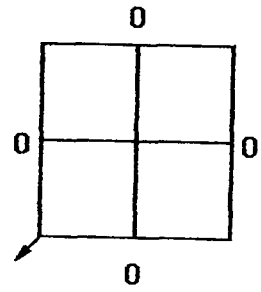
Division results are

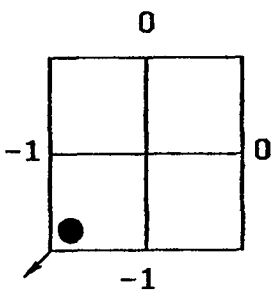
nw = 0 ne = 1 sw = 0 se = 0

This means that, D11 = 0, D12 = 1, D21 = 0, D22 = 0
assignment should be done at the bit level.

Lets do the division sw1/ne2 partition.
Dots number is 1.
polarity direction is sw.
Maximum dots number is 1.
Neighbors are

west = 0 east = integer(2/4) = 0
north = integer(2/4) = 0 south = 0





remained dots = 1
west - east = 0
north - south = 0
(a)

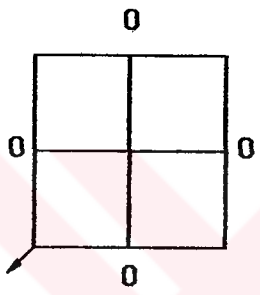
remained dots = 0
west - east = -1
north - south = 1
(b)

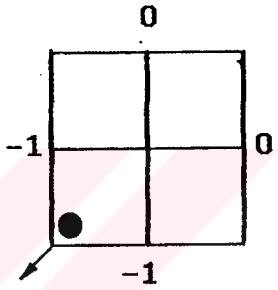
Figure 2.24 Division of the sw1/ne2 partition.

Division results are

nw = 0 ne = 0 sw = 1 se = 0
D11 = 0 D12 = 0 D21 = 1 D22 = 0 assignments should be done at bit level.

Lets do the division of sw1/sw2 partition division.
Dots number is 1.
Maximum dots number is 1.
Polarity direction is sw.
Neighbors are
west = 0 east = 0 north = 0 south = 0





remained dots = 1
west - east = 0
north - south = 0
(a)

remained dots = 0
west - east = -1
north - south = 1
(b)

Figure 2.25 Division of the sw1/sw2 partition.

Division results are
nw = 0 ne = 0 sw = 1 se = 0
This means that, D11 = 0, D12 = 0, D21 = 1, D22 = 0 assignments should be done at the bit level.

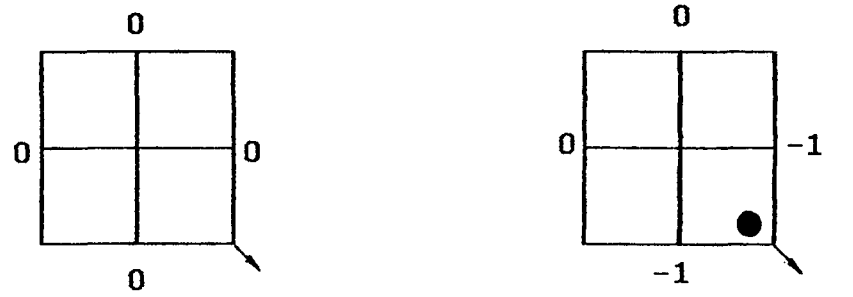
Lets do the division of se1/nw2 partition.
Dots number is 1.
Polarity direction is se.
Maximum dots number is 1.
Neighbors are
west = integer(2/4) = 0 east = 0
north =integer(2/4) = 0 south = 0

remained dots = 1 remained dots = 0
west - east = 0 west - east = 1
north - south = 0 north - south = 1
(a) (b)

Figure 2.26 Division of the se1/nw2 partition.

Division results are
nw = 0 ne = 0 sw = 0 se = 1
D11 = 0 D12 = 0 D21 = 0 D22 = 1 assignment should be done at the bit level.

Lets do the division of se1/se2 partition.
Dots number is 1.
Maximum dots number is 1.
Polarity direction is se.
Neighbors are
west = 0 east = 0 north = 0 south = 0



remained dots = 1 remained dots = 0
west - east = 0 west - east = 1
north - south = 0 north - south = 1
(a) (b)

Figure 2.27 Division of the se1/se2 partition.

Division results are

nw = 0 ne = 0 sw = 0 se = 1

This means that, D11 = 1, D12 = 0, D21 = 0, D22 = 1 assignments should be done at the bit level.

After the bit level assignments, pixel placement data is now ready for the printing.

Whole placement of pixel is as follows :

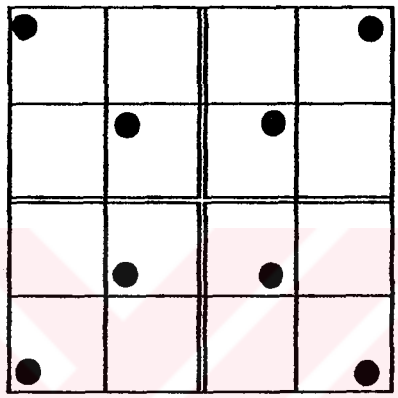


Figure 2.28 Placement pattern of 8 intensity pixel.

2.14 Program Structure

An image handle tool has been written in C language with in Microsoft C 6.0 Compiler environment. Mainly tool has file handle, some image processing techniques including placement process and printing jobs. The program has been designed as user friendly menu driver tool. So, during executing program very limited information is needed.

The program has been developed with in IBM PC compatible ACER 1120SX computer. Program needs conventional memory and VGA display driver. Because placement data can be in huge amount, magnetic storage driver should have enough memory to keep the data. Data amount is

(window row dimension * window column dimension * 32) bytes.

With this tool, an image file, up to 512x512 dimensions,

can be displayed whit any window of image with a displacement offset. Then selected a window can be saved with 512 bytes displacement. Saved file has 512 bytes from beginning of the file as header may be used for future usage for some useful information about the image file.

Process menu has placement and histogram equalization processes. Edge detection and regional processes are included for optional future processes. They are not present at this moment. In placement process, after placement has been done saves 32 bytes of placed pixel, first byte1 of rows then byte2 of rows in place.dat file. Do this process for hole given image window.

In printer menu a placed data can be printed. So any image file which is going to be placed, first should be processed with placement process, then can be printed. Two types printer has been allowed. Dot matrix or Laser printer can be selected. Program supports Epson dot matrix and HP Laser printers. The care should be take especially row dimensions of printing for 80 columns printer, it should not accede the 128 rows of image.
Same thing for laser printer is 150 rows of image.
Printer menu prints out the image in sideways (landscape mode). So, start to printing from right side of image column from top to bottom. First get 16 byte2 of pixel rows of hole column, then print it out, and then get next 16 bytes1 of pixel rows.

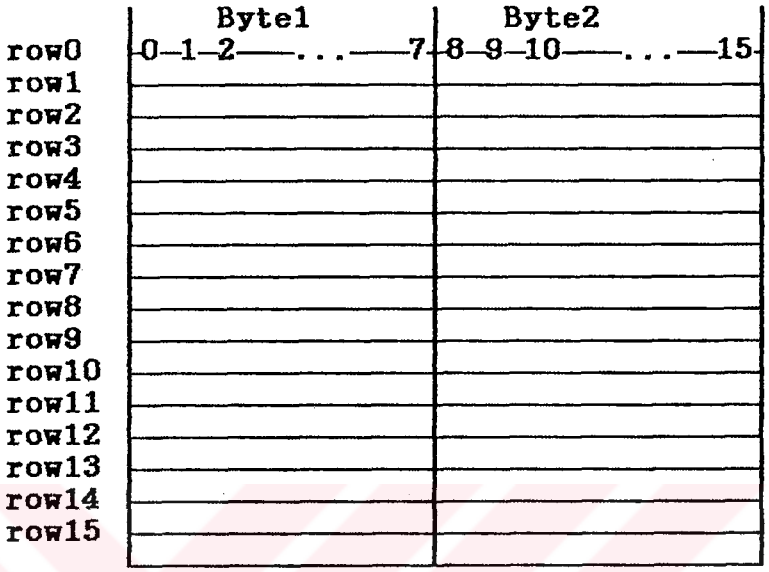


Figure 2.29 Row structure of a single pixel in byte.

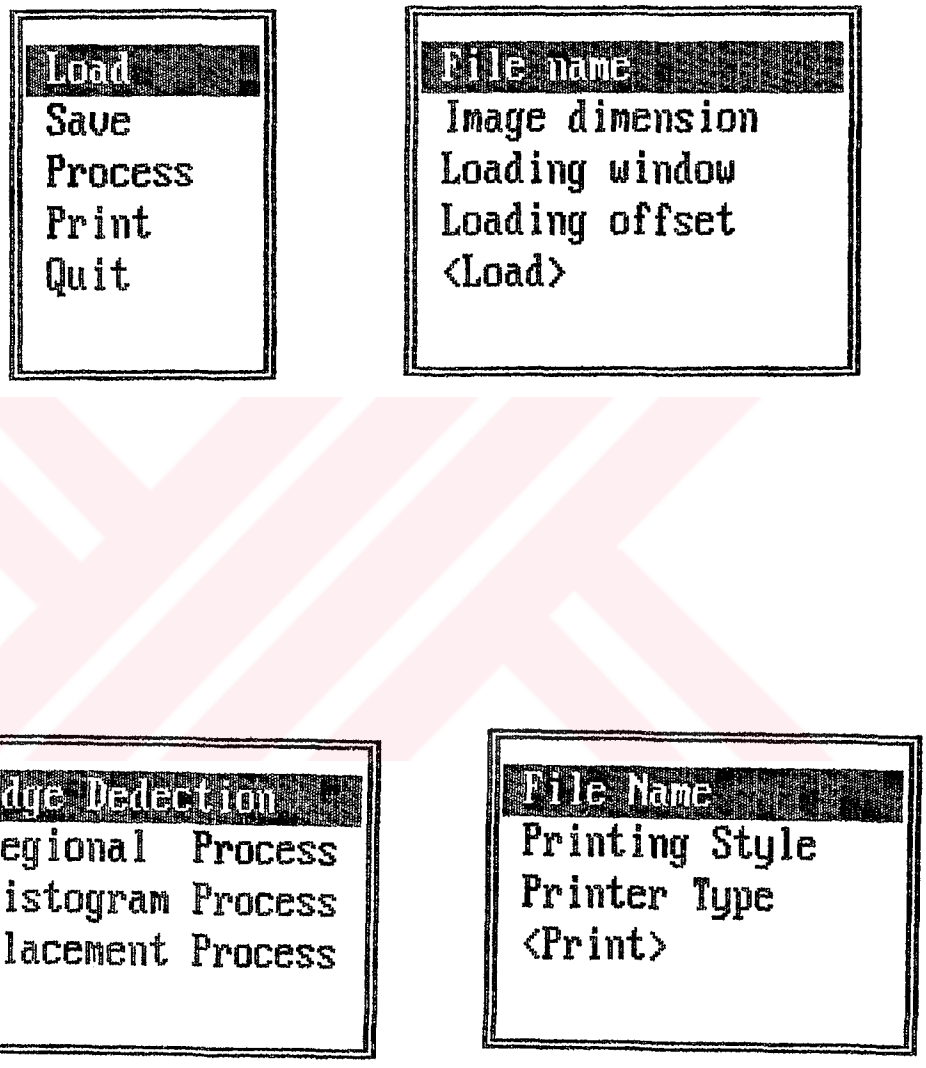


Figure 2.30 Appearance of program menus.

2.15 Program outputs

Figure 2.31 120x120 dimensional an image print out in 300 dpi laser printer.

Figure 2.32. 256x256 dimensional an image print out in 600 dpi laser printer.

RESULTS AND CONCLUSION

As a result, in this work present half tone printing technique's problems has been shown. The solution to that problem has been given with the new implemented algorithm. With the new placement algorithm, efficiency of present printers has been improved. But exact solution the problem is still depend on the printer technology. In nowadays, new 1200 dpi laser printer and 2400 dpi color printer have been announced in the market. With these printer exact result of algorithm can be seen. But unfortunately, we have not got the time and access possibility to those printer, results of the algorithm in those resolutions could not been given in this thesis. Instead, algorithm performance has been given in the 300 dpi and 600 dpi printers. Comparison of the algorithm result with present printing techniques has not been done. The best printing technique should be presented in the software environment, then comparison should be done.

Because of the soft crossing between the grey levels, very close grey levels can not be distinguished from each other during the printing. This may be problem if the edge of the image is very important for us. But as explained before, with some image processing techniques, this problem can be solved in the image processing environment. And the algorithm can be enriched with those techniques.

The algorithm can be easily evaluated for color printing. Again, better printing quality is expected. And also, algorithm can be applied to other fields which have the similar problem.

At the end of this work an image printing computer program has been written with the new implemented algorithm. And results have been given in the 300 and 600 dpi printers.

REFERENCES

[1] LIM, Jea S., "Two Dimensional Signal and Image Processing", Prentice-Hall Inc., (1990).

[2] SCHALKOFF, Robert J., "Digital Image processing and Computer Vision", John Wiley & Sons Inc., (1989).

[3] BÜTÜN, Bilgin, "An Image Processing System", M.S. Thesis, Institute of Science and Technology, I.T.U., (1989).

APPENDIX A SOURCE PROGRAM LIST

```
/* general.h includes general declarations. */

#define TRUE     1
#define FALSE    0
#define XORIT    0x80
#define BS       0x08
#define CR       0x0D
#define ENTER      CR
#define RETURN     CR
#define NULLCHAR '\0'
#define ESC      0x1B
#define HOME    0x147
#define UP       0x48
#define DOWN     0x50
#define LEFT     0x4B
#define RIGHT    0x4D
#define LF      0x00A
#define M_MENU      0
#define L_MENU      1
#define PSS_MENU    2
#define P_MENU      3
#define STL_MENU    4
#define DRV_MENU    5
#define OFF         0
#define ON          1
```

```
/* Color.h file contains possible color codes in VGA
** 16 color mode.
*/

#define  BLACK     0x00
#define  BLUE      0x01
#define  GREEN     0x02
#define  CYAN      0x03
#define  RED       0x04
#define  MAGENTA   0x05
#define  BROWN     0x06
#define  WHITE     0x07
#define  DGRAY     0x08
#define  LBLUE     0x09
#define  LGREEN    0x0A
#define  LCYAN     0x0B
#define  LRED      0X0C
#define  LMAGENTA  0x0d
#define  YELLOW    0x0E
#define  IWHITE    0x0F
```

```
/* Manifest.h file declarations are about menus, warnings
** and title which are used in the main and subprograms.
*/

#define X_MAX  639
#define Y_MAX  479
#define FALSE   0
#define ON      1
#define OFF     0
int in_menu   = FALSE;
int new_style = ON;
int css_style = OFF;

int main_select  =   1;
int load_select  =   1;
int pss_select   =   1;
int prt_select   =   1;
int stl_select   =   1;
int drv_select   =   1;

char file_name [61];
char save_name [61];
char image_rdim [6];
char image_cdim [6];
char wind_roffset [6];
char wind_coffset [6];
char image_offset [6];
char driver [15];

char load_message [] = {"Loading..."};
char save_message [] = {"Saving..."};
char print_message [] = {"Printing..."};

char  *m_menu []={

       "IMMMMMMMMMMM;",
       ": Load      :",
       ": Save      :",
       ": Process   :",
       ": Print     :",
       ": Quit      :",
       ":           :",
       "HMMMMMMMMMMM<",
       ""};


char  *l_menu []={

       "IMMMMMMMMMMMMMMMMMMM;",
       ": File name          :",
       ": Image dimension    :",
       ": Loading window     :",
       ": Loading offset     :",
       ": <Load>             :",
       ":                    :",
       "HMMMMMMMMMMMMMMMMMMM<",
       ""};
```

```
char  *s_menu []={
                    "IMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM \
               MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM;",
              ": Save as...:
                                                       :",
              ":
                                                        :",
                    "HMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM \
             MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM<",
                                                   ""};


char  *pss_menu []={

     "IMMMMMMMMMMMMMMMMMMMM;",
     ": Edge Dedection    :",
     ": Regional  Process :",
     ": Histogram Process :",
     ": Placement Process :",
     ":                   :",
     "HMMMMMMMMMMMMMMMMMMMM<",
     ""};

char  *p_menu []={
     "IMMMMMMMMMMMMMMMMMMMM;",
     ": File Name         :",
     ": Printing Style    :",
     ": Printer Type      :",
     ": <Print>           :",
     ":                   :",
     "HMMMMMMMMMMMMMMMMMMMM<",
     ""};

char  *stl_menu []={
     "IMMMMMMMMMM;",
     ": New      :",
     ": Classic  :",
     ":          :",
     "HMMMMMMMMMM<",
     ""};

char  *drv_menu []={
     "IMMMMMMMMMMMMMMMMMM;",
     ": DOT  MATRIX     :",
     ": LASER           :",
     ":                 :",
     "HMMMMMMMMMMMMMMMMMM<",
     ""};

char *f_menu [] ={
"IMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
MMMMMMMMMMMMMM;",
": File  name :
            :",
"HMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
MMMMMMMMMMMMMMM<",
""};
```

```
char *offset_menu []={
                "IMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM;",
                ": row :              column :          :",
                "HMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM<",
                                                    ""};

char *foffset_menu []={
                    "IMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM;",
                    ": Image displacement :           :",
                    "HMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM<",
                    ""};
```

```
/* place.h is a header file of dot placement of image
** printing.
*/
struct ROW
  {
   union
    {
     struct
      {
       unsigned char bit7 : 1;
       unsigned char bit6 : 1;
       unsigned char bit5 : 1;
       unsigned char bit4 : 1;
       unsigned char bit3 : 1;
       unsigned char bit2 : 1;
       unsigned char bit1 : 1;
       unsigned char bit0 : 1;
       }bits1;
      struct
       {
     unsigned char holebyte1;
     }bt1;
       }byte1;
   union
       {
     struct
      {
       unsigned char bit15 : 1;
       unsigned char bit14 : 1;
       unsigned char bit13 : 1;
       unsigned char bit12 : 1;
       unsigned char bit11 : 1;
       unsigned char bit10 : 1;
       unsigned char bit9  : 1;
       unsigned char bit8  : 1;
       }bits2;
     struct
      {
       unsigned char holebyte2;
       }bt2;
     }byte2;
    };
 struct ELEMENT
   {
    float  nw;
    float  ne;
    float  sw;
    float  se;
    };
 struct NEIGHBOR
   {
    float  west;
    float  east;
    float  north;
    float  south;
    };
```

```
/* This program has been devaloped for to print an image
** file. For that reason, new implemented algorithm has
** has been applied in printing process.  IP.C is the
** main program of over all.
*/

#include  <dos.h>
#include  <stdio.h>
#include  <graph.h>
#include  <ctype.h>
#include  <conio.h>
#include  <string.h>
#include  <stdlib.h>
#include  "manifest.h"
#include  "color.h"
#include  "general.h"


#define   XORIT  0x80

union REGS  regs ;

extern void disp_curs(int,int,int,int);
extern int  loadf(char *);
extern int  placef(char *);
extern int  savef(char *);
extern int  prnf(char *,int);
extern int  lprnf(char *);
extern int menu_curs(int,int,int,int,int,int);
extern int retkey(void);
int  write_menu(char *[],int,int,int);
void load_menu(int);
void save_menu(int);
void process_menu(int);
void print_menu(int);
void write_char(char,int,int,int,int);
void set_cursor(int,int);
void write_grchar(char,int,int,int,int);
void get_str(char *,int,int,int,int);
void write_str(char *,int,int,int);
```

```
void main ()
      {
       int depth,inchar;

       _setvideomode(_VRES16COLOR);
       do{
          if(!in_menu)
           {
            in_menu=TRUE;
            depth = write_menu(m_menu,0,65,IWHITE);
           }
          inchar=menu_curs(1,66,(depth-4),11,IWHITE, \
                                                M_MENU);

          if(inchar == ESC)
            {
             _setvideomode(_TEXTC80);
             exit(0);
             }
          switch(main_select)
            {
            case 1:
                load_menu(IWHITE);
                break;
            case 2:
                save_menu(IWHITE);
                break;

            case 3:
                process_menu(IWHITE);
                break;
            case 4:
                print_menu(IWHITE);
                break;
            case 5:
               _setvideomode(_TEXTC80);
                exit(0);
            }
          }while(inchar != ESC);


       _setvideomode(_TEXTC80);
       }/* end of main program */
```

```
void  write_array(string,row,col,color)
      char  *string;
      int   row,col,color;
       {
        unsigned int i =0;
        if( strlen(string) !=0 )
          do {
          ++i;
          ++col;
          write_char(string[i],row,col,color,XORIT);
          }while(i < strlen(string));
         else;
       }


void write_str(string,row,col,color)
    char *string;
    int   row,col,color;
      {
       unsigned int j,newcol;
       for (j=0, newcol=col; j < strlen(string); j++,  \
                                                 newcol++)
       write_char(string[j],row,newcol,color,XORIT);
       }

void write_char(character,row,col,color,orxor)
     char character;
     int row,col,color,orxor;
       {
       set_cursor(row,col);
       regs.h.ah=9;
       regs.x.cx=0x01;
       regs.h.al= character;
       regs.h.bl= (unsigned char)(color | orxor);
       int86(0x10,&regs,&regs);
       }

void read_char(character,row1,col1)
    char *character;
    int   row1,col1;
     {
      set_cursor(row1,col1);
      regs.h.ah=8;
      regs.h.bh=0;
      int86(0x10,&regs,&regs);
      }

void set_cursor(int row,int col)
      {
       union REGS regs;
       regs.h.ah=2;
       regs.h.bh=0;
       regs.h.dh= (unsigned char) row;
       regs.h.dl= (unsigned char) col;
       int86(0x10,&regs,&regs);
       }
```

```
int write_menu(string,row,col,color)
char *string [];
int row,col,color;
  {
   int i,row_width,col_width, newcol,col_offset;
   unsigned int j;
   row_width=strlen(string[0]);
   col_offset=col+row_width-1;
   for (i=0; 0 < strlen(string[i]);i++)
     ;
   col_width=i;
   i=0;

   for(i=0,newcol=col; strlen(string[i]) !=0 ;++i,++row)
       {
       if((i==0) || (i==(col_width-1)))
       {
       for(j=0,newcol=col; j<  strlen(string[i]); \
                                    ++j,++newcol)
       write_grchar(string[i][j],  row, newcol, color, \
                                                XORIT);
       }

       else if (0<i< (col_width-1))
          {
           write_grchar(string[i][0], row,col,color, \
                                                XORIT);
           write_grchar(string[i][(row_width-1)],row, \
                              col_offset,color,XORIT);


          for(j=1,newcol=(col+1);j<(strlen(string[i])-1);
                                              ++j, ++newcol)
           write_char(string[i][j], row, newcol,  color, \
                                                XORIT);
           }
     }
      return(col_width);
   } /* end of write_menu fanction */


void  write_grchar(character, row, col, color, orxor)
      char character;
      int row, col, color, orxor;
     {
      set_cursor(row,col);
      regs.h.ah=9;
      regs.x.cx=1;
      regs.h.al= character;
      regs.h.al |=0x80;
      regs.h.bl=(unsigned char) (color | orxor);
      int86(0x10,&regs, &regs);
      }
```

```
void main_menu(color)
     int color ;
      {
       int depth,inchar;
       depth=write_menu(m_menu,0,65,color);
       inchar = menu_curs(1,66,(depth-4),11,  \
                                  color,M_MENU);
       write_menu(m_menu,0,65,color);
       }

void load_menu(color)
     int color;
       {
     int inchar,count;
     int row,col,curr_row,curr_col;
     int depth,curr_depth;
     curr_row = row = 20;
     curr_col = col = 5;
     depth=write_menu(l_menu,10,10,color);
     curr_depth = depth;
     do{
        inchar = menu_curs(11,11,(depth-4),20, \
                                  color,L_MENU);
        if(inchar == RETURN)
          {
           switch(load_select)
          {
            case 1:
              count = 60;
              row = curr_row;
              col = curr_col;
              write_menu(f_menu,row,col,color);
              write_str(file_name,(row + 1), \
                              (col+14),color);
              disp_curs((row +1),(col + 14),(count  -5), \
                                                  color);
              get_str(file_name,(row + 1),(col+14),  \
                                    color,count);
              disp_curs((row +1),(col +14),(count -5),  \
                                                 color);
              write_str(file_name,(row + 1),(col+14), \
                                                 color);
              write_menu(f_menu,row,col,color);
              inchar = 'a';
              break;

           case 2:
             count = 6;
             row = curr_row;
             col = curr_col;
             write_menu(offset_menu,row,col,color);
             write_str(image_rdim,(row + 1),(col+8), \
                                               color);
             write_str(image_cdim,(row +1),(col+27), \
                                               color);
             disp_curs((row +1),(col +8),count,color);
             get_str(image_rdim,(row + 1), (col+8), \
                                    color,count);
```

```
    disp_curs((row +1),(col +8),count,color);
    disp_curs((row +1),(col +27),count,color);
    get_str(image_cdim,(row + 1), (col+27), \
                                    color,count);
    disp_curs((row +1),(col +27),count,color);
    write_str(image_rdim,(row + 1),(col+8),  \
                                        color);
    write_str(image_cdim,(row + 1),(col+27), \
                                        color);
    write_menu(offset_menu,row,col,color);
    inchar = 'a';
    break;

  case 3:
    count = 6;
    row = curr_row;
    col = curr_col;
    write_menu(offset_menu,row,col,color);
    write_str(wind_roffset,(row + 1),(col+8),  \
                                          color);
    write_str(wind_coffset,(row  + 1),(col+27), \
                                           color);
    disp_curs((row +1),(col +8), count,color);
    get_str(wind_roffset,(row + 1),  (col+8), \
                                    color,count);
    disp_curs((row +1),(col +8),count,color);
    disp_curs((row +1),(col +27),count,color);
    get_str(wind_coffset,(row + 1), (col+27),  \
                                    color, count);
    disp_curs((row +1),(col +27),count,color);
    write_str(wind_roffset,(row + 1),(col+8),  \
                                          color);
    write_str(wind_coffset,(row  + 1),(col+27), \
                                           color);
    write_menu(offset_menu,row,col,color);
    inchar = 'a';
    break;

  case 4:
    count = 6;
    row = curr_row;
    col = curr_col;
    write_menu(foffset_menu,row,col,color);
    write_str(image_offset,(row + 1),(col+23),  \
                                        color);
    disp_curs((row +1),(col +23),count,color);
    get_str(image_offset,(row + 1), (col+23),  \
                                    color, count);
    disp_curs((row +1),(col +23),count,color);
    write_str(image_offset,(row + 1),(col+23),  \
                                        color);
    write_menu(foffset_menu,row,col,color);
    inchar = 'a'; /*for menu correction*/
    break;
```

```
            case 5:
              in_menu = FALSE;
              _clearscreen(_GCLEARSCREEN);
              write_str(load_message,(row +8),col,color);
              loadf(file_name);
              write_str(load_message,(row +8),col,color);
              getch();
              inchar = 'a'; /* for menu correction.*/
              break;

            }
       }
      if(in_menu == FALSE) break;
     }while(inchar != ESC);
     if (in_menu != FALSE)
     write_menu(1_menu,10,10,color);
     else ;
 }

void save_menu(color)
     int color;
      {
       int count = 60;
       write_menu(s_menu,20,0,color);
       write_str(save_name,21,14,color);
       disp_curs(21,14,(count -4),color);
       get_str(save_name, 21,14,color,count);
       disp_curs(21,14,(count -4),color);
       write_str(save_name, 21, 14, color);
       write_menu(s_menu,20,0,color);
       write_str(save_message,25,5,color);
       savef(save_name);
       write_str(save_message,25,5,color);
       }

void process_menu(color)
    int color;
     {
      int depth,inchar;
      int row,col;
      row = 23;
      col = 5;
      depth=write_menu(pss_menu,20,20,color);
      do
       {
       inchar = menu_curs(21,21,(depth-4),19,color, \
                                              PSS_MENU);
       if(inchar == RETURN)
        {
         switch(pss_select)
         {
          case 1:
          inchar = 'a';
          break;
```

```
            case 2:
            inchar = 'a';
            break;
            case 3:
            inchar = 'a';
            break;
            case 4:
              _clearscreen(_GCLEARSCREEN);
              in_menu = OFF;
              write_str("placement in  progress...",(row+5)\
                                                  col,LGREEN);
              placef(file_name);
              printf("\a");
              write_str("placement  inprogress...",(row+5),\
                                                  col,LGREEN);
              inchar = 'a';
              return ;
            }
          };
        }while(inchar != ESC);
          write_menu(pss_menu,20,20,color);
         }

void print_menu(color)
     int color;
      {
       int depth,inchar,count;
       int curr_row,curr_col,curr_depth,width;
       int row, col;
       int double_strike;
       curr_row= row = 10;
       curr_col= col = 5;
       count = 60;
       width = 19;
       depth = write_menu(p_menu,row,col,color);
       curr_depth = depth;
       do{
      inchar = menu_curs((row + 1),(col + 1),(depth-4),  \
                                      width,color,P_MENU);
 if (inchar == RETURN)
   {
    switch(prt_select)
    {
     case 1:
     row = curr_row + 10;
     col = curr_col;
     write_menu(f_menu,row,col,color);
     write_str(file_name,(row + 1),(col +14),color);
     disp_curs((row + 1),(col +14),(count -5),color);
     get_str(file_name,(row +1),(col + 14),color,count);
     write_str(file_name,(row +1),(col + 14),color);
     disp_curs((row + 1),(col + 14),(count -5),color);
     write_menu(f_menu,row,col,color);
     inchar = 'a';
     break;
```

```
case 2:
 width = 10;
 row = curr_row + 3;
 col = curr_col + 22;
 depth = write_menu(stl_menu,row ,col ,color);
 do{
    inchar = menu_curs((row +1),(col  +1),(depth-4), \
                                  width,color,STL_MENU);
    if(inchar == RETURN);
     {
      switch(stl_select)
      {
       case 1:
        new_style = ON;
        css_style = OFF;
        break;

       case 2:
        new_style = OFF;
        css_style = ON;
        break;
      }
     }
    }while ((inchar != ESC) && (inchar != RETURN));
 write_menu(stl_menu,row,col,color);
 inchar = 'a';
 break;

 case 3:
   width =17;
   row = curr_row + 4;
   col = curr_col + 22;
   depth = write_menu(drv_menu,row,col,color);
   do{
  inchar =  menu_curs((row +1),(col +1),(depth-4), \
                                 width,color,DRV_MENU);
   if (inchar == RETURN)
     {
      switch(drv_select)
       {
        case 1:
       strcpy(driver,"DOT MATRIX");
       break;
        case 2:
       strcpy(driver,"LASER");
       break;
      }
      }
    }while((inchar != ESC) && (inchar !=RETURN));
    write_menu(drv_menu,row,col,color);
    inchar = 'a';
    break;
```

```
        case 4:
          row = curr_row + 10;
          col = curr_col;
          double_strike = OFF;
          write_menu(p_menu, curr_row, curr_col, color);
          write_str(print_message,row,col,color);
          /*print the image file */
          if(! strcmp(driver,"DOT MATRIX"))
            {
             prnf(file_name,double_strike);
             };
          if(! strcmp(driver,"LASER"))
             {
              lprnf(file_name);
              };
          write_str(print_message,row ,col,color);
          inchar = 'a';
          return;
       }
      }
     row = curr_row;
     col = curr_col;
     width = 18;
     depth = curr_depth;
     }while(inchar != ESC);
     row = curr_row;
     col = curr_col;
     write_menu(p_menu,row,col,color);
  }


void get_str(string,row,col,color,count)
     char *string;
     int row,col,color,count;
      {
       int curr_col,position ;
       char oldchar,inchar;
       curr_col = col;
       position = 0;
       set_cursor(row,col);
       if((inchar = (char) retkey()) != RETURN )
     {
     write_str(string,row,col,color);
     strcpy(string,"");
     write_char(inchar,row,col,color,XORIT);
     write_char('_',row,++col,color,XORIT);
     *(string + position) = inchar;
     *(string + (++position)) = NULLCHAR;

     while( (inchar = (char) retkey() ) != RETURN )
      {
      switch (inchar)
        {
         case BS :
          if (position > 0)
              {
               oldchar = *(string +(--position));
               *(string + position) = NULLCHAR;
```

```
                write_char('_',row,col,color,XORIT);
                write_char(oldchar, row, --col, \
                                    color,XORIT);
                write_char('_', row, col,color,XORIT);
                }
              else ;
             break;
           default :
            if(col < (curr_col + count))
            {
             write_char('_', row, col, color, XORIT);
             write_char(inchar, row, col, color, XORIT);
             write_char('_', row,++col,color,XORIT);
             *(string + position) = inchar;
             *(string + (++position)) = NULLCHAR;
             }
            else ;
           break;
          }
       }
       write_char('_',row,col,color,XORIT);
    }
  else ;
  } /* end of get_str function */
```

```
/* menu_curs.c swaps  cursor around the  any  passed menu.
*/
#include  <stdio.h>
#include  <conio.h>
#include  <dos.h>
#include  "general.h"
#define   FALSE  0

menu_curs(row,col,depth,width,color,menu_key)
int  row,col,depth,width,color,menu_key;
{
 int inchar,select;
 extern int main_select;
 extern int load_select;
 extern int pss_select;
 extern int prt_select;
 extern int stl_select;
 extern int drv_select;

 switch(menu_key)
    {
     case M_MENU :
       select=main_select;
       break;

     case L_MENU :
       select=load_select;
       break;

     case PSS_MENU  :
       select = pss_select;
       break;

     case P_MENU :
       select=prt_select;
       break;

     case DRV_MENU :
       select = drv_select;
       break;

     case STL_MENU :
       select = stl_select;
       break;

     }


row +=select-1;
disp_curs(row,col,width,color);
```

```
do {
   inchar=retkey();
   switch(inchar)
     {
     case DOWN :
       if(select <= depth)
         {
          ++select;
          disp_curs(row,col,width,color);
          disp_curs(++row,col,width,color);
         }
         break;

      case  UP :
       if(select > 1)
        {
         --select;
         disp_curs(row,col,width,color);
         disp_curs(--row,col,width,color);
         }
         break;
      case ESC :
         break;
     }
   }while((inchar != RETURN) && (inchar != ESC));

    switch(menu_key)
     {
      case M_MENU :
           main_select=select;
           break;
      case L_MENU :
           load_select=select;
           break;
      case PSS_MENU :
           pss_select= select;
           break;
      case P_MENU :
           prt_select = select;
           break;

      case DRV_MENU :
           drv_select = select;
           break;
      case STL_MENU :
           stl_select = select;
           break;

      }
disp_curs(row,col,width,color);
return(inchar);
}/*end of menu_curs */
```

```
disp_curs(row,col,width,color)
int row,col,width,color ;
{
 int i;
 for( i=0; i<width;++i)
 write_char(219,row,col+i,color,XORIT);
}


int retkey()
  {
  int inchar;
  inkey(&inchar);
  return (inchar);
  }

int inkey(ascii)
int *ascii;
{
 union REGS regs;
 regs.x.ax = 0;
 int86(0x16,&regs,&regs);
 if(regs.h.al == 0)
   *ascii=regs.h.ah;
 else *ascii = regs.h.al;
 }
```

```
/* loadf.c loads related image file to the screen with
** the given window parameters.
*/

#include  <dos.h>
#include  <stdlib.h>
#include  <math.h>
#include  <stdio.h>
#include  <ctype.h>
#include  <string.h>
#include  <malloc.h>
#include  <graph.h>
#include  <conio.h>
#include  "color.h"

#define  COLUMNLENGHT  512

extern    void write_str(char *,int, int, int);
extern    int write_menu(char *[], int, int, int);
extern    void disp_curs(int, int, int, int);
extern    void get_str(char *, int, int, int, int);
void write_dot(unsigned char,int,int);

extern    char image_offset[];
extern    char image_rdim[];
extern    char image_cdim[];
extern    char wind_roffset[];
extern    char wind_coffset[];
extern    char *f_menu[];

unsigned char fbuffer[COLUMNLENGHT];
fpos_t     *pos;
char fread_warr [] =  {"cannot open the  file... press any
key"};

int loadf(file_name)
char *file_name;
 {
  FILE    *loadstream;
  int i,j;
  long im_offset;
  int im_rdim,im_cdim;
  int win_rdim, win_cdim;
  int count = 60;
  int row  = 20;
  int col  = 5;

  im_offset = atol(image_offset);
  im_rdim   = atoi(image_rdim);
  im_cdim   = atoi(image_cdim);
  win_rdim  = atoi(wind_roffset);
  win_cdim  = atoi(wind_coffset);
```

```
  if (!(strcmp(file_name,"")))
    {
     write_menu(f_menu,row,col,IWHITE);
     write_str(file_name,(row + 1),(col + 14),IWHITE);
     disp_curs((row +1),(col +14),(count -5),IWHITE);
     get_str(file_name,(row +1),(col +14),IWHITE,count);
     disp_curs((row +1),(col +14),(count -5),IWHITE);
     write_str(file_name,(row +1),(col +14),IWHITE);
     write_menu(f_menu,row,col,IWHITE);
     }
    else
     ;
     if ((loadstream = fopen(file_name,"rb")) == NULL)
       {
     write_str(fread_warr,(row +3),col,RED);
     getch();
     write_str(fread_warr,(row +3),col,RED);
     return (1);
     }
      else
       ;
      fseek(loadstream,im_offset,SEEK_SET);

      /*fgetpos(loadstream,pos);*/
      for( i = 0; i< win_rdim; i++)
     {
      fread(&fbuffer[0],(sizeof(char) * win_cdim), \
                                      1,loadstream);
      fseek(loadstream,(long)(im_cdim - win_cdim), \
                                        SEEK_CUR);
      for(j = 0; j < win_cdim; ++j)
        {
         if(fbuffer[j] > 96)  _setpixel(j,i);
         };
      }

      fclose(loadstream);
      /* load the buffered image to display. */
       /*write_dot(dot,i,j);*/
} /* end of loadf.c function. */

void  write_dot(l_data,x,y)
      unsigned char l_data;
      int x,y;
       {
        union REGS regs;

        regs.h.ah = 0x0c;
        regs.h.al = l_data;
        regs.h.bh = 0;
        regs.x.dx = x;
        regs.x.cx = y;
        int86(0x10,&regs,&regs);
        }
```

```
/* savef.c function save the current loaded file.*/

#include  <stdio.h>
#include  <string.h>
#include  <conio.h>
#include  "color.h"
#include  <malloc.h>
#include  <stdlib.h>

#define   SAVEOFFSET 512

extern void write_str(char *, int,int,int);
extern char image_offset[];

extern char file_name [];
extern char image_offset [];
extern char image_cdim [];
extern char wind_roffset [];
extern char wind_coffset [];

unsigned char fbuffer[512];

int savef(save_name)
char *save_name;
 {
  int i;
  int row = 24;
  int col = 5;
  int win_rdim, win_cdim;
  int im_cdim;
  long im_offset;
  char *blank;
  char character;
  FILE    *savestream;
  FILE    *filestream;

  im_offset = atol(image_offset);
  im_cdim   = atoi(image_cdim);
  win_rdim  = atoi(wind_roffset);
  win_cdim  = atoi(wind_coffset);


  if( (blank = (char *) calloc((size_t)SAVEOFFSET, \
                               sizeof(char))) == NULL)
    {
     write_str("calloc failed",row,col,RED);
     getch();
     write_str("calloc failed",row,col,RED);
     return (1);
     }
  blank = realloc(blank,(size_t)SAVEOFFSET);
```

```
if((savestream = fopen(save_name,"rb")) != NULL)
 {
  write_str("over write ?(y/n)", row,col,RED);
  character = (char) getch();
  write_str("over write ?(y/n)",row,col,RED);
  if((character == 'y') || (character == 'Y'))
   {
   fclose(savestream);
   if((savestream = fopen(save_name,"wb")) == NULL)
    {
     write_str("cannot save the file !",row,col,RED);
     getch();
     write_str("cannot save the file !",row,col,RED);
     free(blank);
     return(1);
     };

    if((filestream = fopen(file_name,"rb")) == NULL)
     {
    write_str("cannot open the  image file!", \
                                  row,col,RED);
     getch();
     write_str("cannot  open the image file!", \
                                  row,col,RED);
     fclose(savestream);
     free(blank);
     return(1);
     };

     fwrite(&blank[0],(sizeof(char) * SAVEOFFSET),  \
                                    1,savestream);
     fseek(filestream,(long)(im_offset),SEEK_SET);

     for(i = 0; i < win_rdim; i++)
      {
       fread(&fbuffer[0],(sizeof(char) * win_cdim),  \
                                     1,filestream);
    fseek(filestream,(long)(im_cdim - win_cdim),  \
                                      SEEK_CUR);
    fwrite(&fbuffer[0],(sizeof(char) * win_cdim), \
                                    1,savestream);
    };

      fflush(savestream);
      fclose(savestream);
      fclose(filestream);
      free(blank);
      return(0);
   }
   else  return (1);
  }
```

```
    if ((savestream = fopen(save_name,"wb")) == NULL)
      {
       write_str("cannot open new file !",row,col,RED);
       getch();
       write_str("cannot open new file !",row,col,RED);
       return (1);
       }

    if((filestream = fopen(file_name,"rb")) == NULL)
      {
        write_str("cannot open the image file!", \
                                        row,col,RED);
    getch();
    write_str("cannot open the image file!", \
                                      row,col,RED);
    fclose(savestream);
    free(blank);
    return(1);
    };

     fwrite(&blank,(sizeof(char) * (int)SAVEOFFSET), \
                                            1,savestream);
     fseek(filestream,(long)(im_offset),SEEK_SET);
     for(i = 0; i < win_cdim; i++)
    {
     fread(&fbuffer[0],(sizeof(char) * win_cdim), \
                                        1,filestream);
     fseek(filestream,(long)(im_cdim - win_cdim), \
                                          SEEK_CUR);
     fwrite(&fbuffer[0],(sizeof(char) * win_cdim),  \
                                        1,savestream);
     };

     fflush(savestream);
     fclose(savestream);
     fclose(filestream);
     free(blank);
     return (0);
}
```

```
/* palcef.c does the  placement of whole image which is
** currently loaded.
*/

#include  <stdio.h>
#include  <string.h>
#include  <conio.h>
#include  <ctype.h>
#include  <malloc.h>
#include  <stdlib.h>
#include  "color.h"

extern    int place(FILE *,int,int,int,int,int,int,int);
extern    int write_str(char *,int,int,int);
extern    char wind_roffset[];
extern    char wind_coffset[];
extern    char image_offset[];
extern    char image_rdim[];
extern    char image_cdim[];

int placef(char *file_name)
 {
  int i, j, k, row, col;
  int win_rdim, win_cdim;
  int im_rdim,im_cdim;
  long im_offset;
  unsigned char *r1, *r2, *r3, *temp;
  FILE *placestream;
  FILE *filestream;

  row = 23;
  col = 5;
  im_offset = atol(image_offset);
  im_rdim   = atoi(image_rdim);
  im_cdim   = atoi(image_cdim);
  win_rdim  = atoi(wind_roffset);
  win_cdim  = atoi(wind_coffset);

  if(! (strcmp(file_name,"")))
    {
     write_str("load file first!", row, col, RED);
     getch();
     write_str("load file first!", row, col, RED);
     return (1);
     };

  if((placestream = fopen("place.dat","wb")) == NULL)
    {
     write_str("cannot open the file !", row, col, RED);
     getch();
     write_str("cannot open the file !", row, col, RED);
     return (1);
     };
```

```
if((filestream = fopen(file_name,"rb")) == NULL)
   {
   write_str("cannot open the image file!", \
                                  row,col,RED);
   getch();
   write_str("cannot open the image file!", \
                                  row,col,RED);
    };

if((r1 = (unsigned char *) calloc(
         (size_t)(win_cdim +  2),(size_t)sizeof(char)))\
                                                 == NULL)
  {
   write_str("no enough memory !", row, col, RED);
   getch();
   write_str("no enough memory !", row, col, RED);
   };

if((r2 = (unsigned char *) calloc(
         (size_t)(win_cdim +  2),(size_t)sizeof(char)))\
                                                 == NULL)
  {
   write_str("no enough memory !", row, col, RED);
   getch();
   write_str("no enough memory !", row, col, RED);
   return (1);
   };

if((r3 = (unsigned char *) calloc(
         (size_t)(win_cdim +  2),(size_t)sizeof(char)))\
                                                 == NULL)
  {
   write_str("no enough memory !", row, col, RED);
   getch();
   write_str("no enough memory !", row, col, RED);
   return (1);
   };

if((temp = (unsigned char *) calloc(
           (size_t)(win_cdim  +2),(size_t)sizeof(char)))
                                                 == NULL)
   {
    write_str("no enough memory !",row,col,RED);
    getch();
    write_str("no enough memory !",row,col,RED);
    return(1);
   };
/* initialize the image rows for the processing. */
/* first row is zero for upper boundary. */
fseek(filestream,(long)(im_offset),SEEK_SET);

fread(&r2[1],(sizeof(char) * win_cdim),1,filestream);
fseek(filestream,(long)(im_cdim - win_cdim),SEEK_CUR);

fread(&r3[1],(sizeof(char) * win_cdim),1,filestream);
fseek(filestream,(long)(im_cdim - win_cdim),SEEK_CUR);
```

```
/* send first initialized rows to place.c function */
for(j = 0; j < (win_cdim); j++)
   {
    place(placestream,(int)r2[j + 1],(int)r2[j], \
                    (int)r2[j + 2],(int)r1[j +1],\
                    (int)r3[j+1],0,(j*16));
    };

/* send next rows to place.c function */
for(i = 1; i < (win_rdim - 1); i++)
   {
    temp = r1;
    r1 = r2;
    r2 = r3;
    r3 = temp;
    fread(&r3[1],(sizeof(char) * win_cdim), 1, \
                                    filestream);
    fseek(filestream,(long)(im_cdim - win_cdim), \
                                      SEEK_CUR);
    for(j = 0; j < (win_cdim); j++)
    {
     place(placestream,(int)r2[j+1],(int)r2[j], \
                     (int)r2[j+2],(int)r1[j+1], \
                    (int)r3[j+1],(i*16),(j*16));
     };
    };

  /* initialize last row */
  temp = r1;
  r1 = r2;
  r2 = r3;
  r3 = temp;
  for(k = 0; k < win_cdim; k++)
     {
      r3[k + 1] = 0;
      };
  for(j = 0; j < (win_cdim); j++)
     {
   place(placestream,(int)r2[j+1],(int)r2[j], \
        (int)r2[j+2],(int)r1[j+1],(int)r3[j+1],  \
                    ((win_rdim - 1) * 16),(j*16));
    };

  fflush(placestream);
  fclose(placestream);
  fclose(filestream);
  free(r1);
  free(r2);
  free(r3);
  free(temp);
} /* end of placef.c function */
```

```
/*  placement.c does placement of printing  an image file.
*/

#include  <stdio.h>
#include  <graph.h>
#include  <conio.h>
#include  <stdlib.h>
#include  "place.h"

#define   NW    1
#define   NE    2
#define   SW    3
#define   SE    4


int   maxdotsf(float);
void  row_setf(int, struct ROW, int,int);
int   pol_dirf(float,float);

extern struct ELEMENT s_boardf(float dots_number,  \
             int pol_dir,int maxdots, struct NEIGHBOR N);

struct    NEIGHBOR  NA0, NA[4], NAB[16], NABC[64], \
                                          neighbor;
struct    ELEMENT   A, AB[4], ABC[16], ABCD[64], buffer;
struct    ROW    row0, row1, row2, row3, row4, row5, \
                                          row6, row7;
struct    ROW    row8, row9, row10, row11,  row12, row13,\
                                          row14, row15;

int place(placestream,dots_numberin,west,east, \
                    north,south,roffset,coffset)
 int dots_numberin;
 int west,east,north,south;
 int roffset, coffset;
 FILE  *placestream;
 {
  int i,x;
  int     maxdots,pol_dir;
  float   w_e, n_s;
  float dots_number;


 /* initialization  of A matrix's neighbor. */
  NA0.west  = (float)west;
  NA0.east  = (float)east;
  NA0.north = (float)north;
  NA0.south = (float)south;

  w_e = west - east;
  n_s = north - south;
  dots_number = (float)(dots_numberin);

  /* divide A matrix to its our part */
  maxdots = maxdotsf(dots_number);
  pol_dir = pol_dirf(w_e, n_s);
  A = s_boardf(dots_number,pol_dir,maxdots, NA0);
```

```
/* initialize the neighbor of  A11, A12, A21, A22
** submatrices */
NA[0].west  = (NA0.west / 4);  /* neighbors of A11 */
NA[0].east  = A.ne;
NA[0].north = (NA0.north / 4);
NA[0].south = A.sw;

NA[1].west  = A.nw;            /* neighbors of A12 */
NA[1].east  = (NA0.east / 4);
NA[1].north = (NA0.north / 4);
NA[1].south = A.se;

NA[2].west  = (NA0.west / 4);   /* neighbors of A21 */
NA[2].east  = A.se;
NA[2].north = A.nw;
NA[2].south = (NA0.south / 4);

NA[3].west  = A.sw;            /* neighbors of A22 */
NA[3].east  = (NA0.east / 4);
NA[3].north = A.ne;
NA[3].south = (NA0.south / 4);

/* initialize the AB[] element */
/* assignement of A11's B11,B12,B21,B22 */
dots_number = A.nw;
maxdots = maxdotsf(dots_number);
AB[0] = s_boardf(A.nw, NW, maxdots, NA[0]);

/* assignement of A12's B11,B12,B21,B22 */
dots_number = A.ne;
maxdots = maxdotsf(dots_number);
AB[1] = s_boardf(A.ne, NE, maxdots, NA[1]);

/*assignement of A21's B11,B12,B21,B22 */
dots_number = A.sw;
maxdots = maxdotsf(dots_number);
AB[2] = s_boardf(A.sw,SW, maxdots, NA[2]);

/* assignement of A22's B11,B12,B21,B22 */
dots_number = A.se;
maxdots = maxdotsf(dots_number);
AB[3] = s_boardf(A.se,SE, maxdots, NA[3]);
```

```
/* initialize neighbor of B11, B12,B21, B22 matrices */

for(i = 0, x = 0; i < 4; i++, x += 4)
  {
   NAB[x].west  = (NA[i].west / 4);    /* neighbors of
                                       ** A[i,j]B11 */
   NAB[x].east  = AB[i].ne;
   NAB[x].north = (NA[i].north / 4);
   NAB[x].south = AB[i].sw;

   NAB[x+1].west  = AB[i].nw;     /* neighbors of
                                  ** A[i,j]B12 */
   NAB[x+1].east  = (NA[i].east / 4);
   NAB[x+1].north = (NA[i].north / 4);
   NAB[x+1].south = AB[i].se;

   NAB[x+2].west   = (NA[i].west / 4); /* neighbors of
                                       ** A[i,j]B21 */
   NAB[x+2].east  = AB[i].se;
   NAB[x+2].north = AB[i].nw;
   NAB[x+2].south = (NA[i].south / 4);

   NAB[x+3].west  = AB[i].sw;     /* neighbors of
                                  ** A[i,j]B22 */
   NAB[x+3].east  = (NA[i].east / 4);
   NAB[x+3].north = AB[i].ne;
   NAB[x+3].south = (NA[i].south / 4);
   };

/* division of A[i,j]B[i,j] to C[i,j] */
for( i = 0, x = 0; i < 4; i++, x +=4)
  {
   /* division of A[i,j]B11 to C[i,j] */
   dots_number = AB[i].nw;
   maxdots = maxdotsf(dots_number);
   ABC[x]  = s_boardf(AB[i].nw,(i + 1), maxdots,\
                                        NAB[x]);

   /* division of A[i,j]B12 to C[i,j] */
   dots_number =  AB[i].ne;
   maxdots = maxdotsf(dots_number);
   ABC[x+1] =  s_boardf(AB[i].ne,(i + 1),maxdots,\
                                        NAB[x+1]);

   /* division of A[i,j]B21 to C[i,j] */
   dots_number = AB[i].sw;
   maxdots = maxdotsf(dots_number);
   ABC[x+2] =  s_boardf(AB[i].sw,(i + 1),maxdots,\
                                        NAB[x+2]);

   /* division of A[i,j]B22 to C[i,j] */
   dots_number = AB[i].se;
   maxdots = maxdotsf(dots_number);
   ABC[x+3] =  s_boardf(AB[i].se,(i + 1),maxdots,\
                                        NAB[x+3]);
   };
```

```
for( i = 0, x = 0; i < 16; i++, x += 4)
  {
   /* initialize neigbors of A[i,j]B[i,j]C11 */
   NABC[x].west  = (NAB[i].west / 4);
   NABC[x].east  = ABC[i].ne;
   NABC[x].north = (NAB[i].north / 4);
   NABC[x].south = ABC[i].sw;

   /* initialize neighbors of A[i,j]B[i,j]C12 */
   NABC[x+1].west  = ABC[i].nw;
   NABC[x+1].east  = (NAB[i].east / 4);
   NABC[x+1].north = (NAB[i].north / 4);
   NABC[x+1].south = ABC[i].se;

   /* initialize neighbors of A[i,j]B[i,j]C21 */
   NABC[x+2].west  = (NAB[i].west / 4);
   NABC[x+2].east  = ABC[i].se;
   NABC[x+2].north = ABC[i].nw;
   NABC[x+2].south = (NAB[i].south / 4);

   /* initialize neighbors of A[i,j]B[i,j]C22 */
   NABC[x+3].west  = ABC[i].sw;
   NABC[x+3].east  = (NAB[i].east / 4);
   NABC[x+3].north = ABC[i].ne;
   NABC[x+3].south = (NAB[i].south / 4);
   };

for( i = 0, x = 0; i < 16 ; i++, x += 4)
  {
  pol_dir =(int)(i/4) + 1;
  /* division of A[i,j]B[i,j]C11 to D[i,j] */
  dots_number = ABC[i].nw;
  maxdots = maxdotsf(dots_number);
  ABCD[x] = s_boardf(ABC[i].nw,pol_dir,maxdots,\
                                        NABC[x]);

  /* division of A[i,j]B[i,j]C12 to D[i,j] */
  dots_number = ABC[i].ne;
  maxdots = maxdotsf(dots_number);
  ABCD[x+1]  = s_boardf(ABC[i].ne,pol_dir,maxdots,\
                                        NABC[x+1]);

  /* division of A[i,j]B[i,j]C21 to D[i,j] */
  dots_number = ABC[i].sw;
  maxdots = maxdotsf(dots_number);
  ABCD[x+2] = s_boardf(ABC[i].sw, pol_dir,maxdots,\
                                        NABC[x+2]);

  /* division of A[i,j]B[i,j] to D[i,j] */
  dots_number = ABC[i].se;
  maxdots = maxdotsf(dots_number);
  ABCD[x+3] = s_boardf(ABC[i].se, pol_dir,maxdots,\
                                        NABC[x+3]);
  };
```

```
/* bit manuplation */

/* first byte of row0 */
row0.byte1.bits1.bit0 =  (unsigned char)((int)\
                                    ABCD[0].nw);
row0.byte1.bits1.bit1 =  (unsigned char)((int)\
                                    ABCD[0].ne);
row0.byte1.bits1.bit2 =  (unsigned char)((int)\
                                    ABCD[1].nw);
row0.byte1.bits1.bit3 =  (unsigned char)((int)\
                                    ABCD[1].ne);
row0.byte1.bits1.bit4 =  (unsigned char)((int)\
                                    ABCD[4].nw);
row0.byte1.bits1.bit5 =  (unsigned char)((int)\
                                    ABCD[4].ne);
row0.byte1.bits1.bit6 =  (unsigned char)((int)\
                                    ABCD[5].nw);
row0.byte1.bits1.bit7 =  (unsigned char)((int)\
                                    ABCD[5].ne);

/* second byte of row0 */
row0.byte2.bits2.bit8 =  (unsigned char)((int)\
                                   ABCD[16].nw);
row0.byte2.bits2.bit9 =  (unsigned char)((int)\
                                   ABCD[16].ne);
row0.byte2.bits2.bit10 = (unsigned char)((int)\
                                    ABCD[17].nw);
row0.byte2.bits2.bit11  = (unsigned char)((int)\
                                    ABCD[17].ne);
row0.byte2.bits2.bit12 = (unsigned char)((int)\
                                    ABCD[20].nw);
row0.byte2.bits2.bit13 = (unsigned char)((int)\
                                    ABCD[20].ne);
row0.byte2.bits2.bit14 = (unsigned char)((int)\
                                    ABCD[21].nw);
row0.byte2.bits2.bit15 = (unsigned char)((int)\
                                    ABCD[21].ne);

/* first byte of row1 */
row1.byte1.bits1.bit0 =  (unsigned char)((int)\
                                    ABCD[0].sw);
row1.byte1.bits1.bit1 =  (unsigned char)((int)\
                                    ABCD[0].se);
row1.byte1.bits1.bit2 =  (unsigned char)((int)\
                                    ABCD[1].sw);
row1.byte1.bits1.bit3 =  (unsigned char)((int)\
                                    ABCD[1].se);
row1.byte1.bits1.bit4 =  (unsigned char)((int)\
                                    ABCD[4].sw);
row1.byte1.bits1.bit5 =  (unsigned char)((int)\
                                    ABCD[4].se);
row1.byte1.bits1.bit6 =  (unsigned char)((int)\
                                    ABCD[5].sw);
row1.byte1.bits1.bit7 =  (unsigned char)((int)\
                                    ABCD[5].se);
```

```
/* 2nd byte of row1 */
row1.byte2.bits2.bit8   = (unsigned char)((int)\
                                ABCD[16].sw);
row1.byte2.bits2.bit9   = (unsigned char)((int)\
                                ABCD[16].se);
row1.byte2.bits2.bit10 =  (unsigned char)((int)\
                                ABCD[17].sw);
row1.byte2.bits2.bit11 = (unsigned char)((int)\
                                ABCD[17].se);
row1.byte2.bits2.bit12  = (unsigned char)((int)\
                                ABCD[20].sw);
row1.byte2.bits2.bit13 = (unsigned char)((int)\
                                ABCD[20].se);
row1.byte2.bits2.bit14 = (unsigned char)((int)\
                                ABCD[21].sw);
row1.byte2.bits2.bit15 = (unsigned char)((int)\
                                ABCD[21].se);

/* first byte of row2 */
row2.byte1.bits1.bit0 =  (unsigned char)((int)\
                                ABCD[2].nw);
row2.byte1.bits1.bit1 =  (unsigned char)((int)\
                                ABCD[2].ne);
row2.byte1.bits1.bit2 =  (unsigned char)((int)\
                                ABCD[3].nw);
row2.byte1.bits1.bit3 =  (unsigned char)((int)\
                                ABCD[3].ne);
row2.byte1.bits1.bit4 =  (unsigned char)((int)\
                                ABCD[6].nw);
row2.byte1.bits1.bit5 =  (unsigned char)((int)\
                                ABCD[6].ne);
row2.byte1.bits1.bit6 =  (unsigned char)((int)\
                                ABCD[7].nw);
row2.byte1.bits1.bit7 =  (unsigned char)((int)\
                                ABCD[7].ne);

/*second byte of row2 */
row2.byte2.bits2.bit8 =  (unsigned char)((int)\
                               ABCD[18].nw);
row2.byte2.bits2.bit9 =  (unsigned char)((int)\
                               ABCD[18].ne);
row2.byte2.bits2.bit10 =  (unsigned char)((int)\
                                ABCD[19].nw);
row2.byte2.bits2.bit11 = (unsigned char)((int)\
                                ABCD[19].ne);
row2.byte2.bits2.bit12  = (unsigned char)((int)\
                                ABCD[22].nw);
row2.byte2.bits2.bit13 = (unsigned char)((int)\
                                ABCD[22].ne);
row2.byte2.bits2.bit14 = (unsigned char)((int)\
                                ABCD[23].nw);
row2.byte2.bits2.bit15 = (unsigned char)((int)\
                                ABCD[23].ne);
```

```
/* 1st byte of row3 */
row3.byte1.bits1.bit0 =  (unsigned char)((int)\
                                ABCD[2].sw);
row3.byte1.bits1.bit1 =  (unsigned char)((int)\
                                ABCD[2].se);
row3.byte1.bits1.bit2 =  (unsigned char)((int)\
                                ABCD[3].sw);
row3.byte1.bits1.bit3 =  (unsigned char)((int)\
                                ABCD[3].se);
row3.byte1.bits1.bit4 =  (unsigned char)((int)\
                                ABCD[6].sw);
row3.byte1.bits1.bit5 =  (unsigned char)((int)\
                                ABCD[6].se);
row3.byte1.bits1.bit6 =  (unsigned char)((int)\
                                ABCD[7].sw);
row3.byte1.bits1.bit7 =  (unsigned char)((int)\
                                ABCD[7].se);

/* 2nd byte of row3 */
row3.byte2.bits2.bit8   = (unsigned char)((int)\
                                ABCD[18].sw);
row3.byte2.bits2.bit9   = (unsigned char)((int)\
                                ABCD[18].se);
row3.byte2.bits2.bit10 =  (unsigned char)((int)\
                                ABCD[19].sw);
row3.byte2.bits2.bit11 = (unsigned char)((int)\
                                ABCD[19].se);
row3.byte2.bits2.bit12  = (unsigned char)((int)\
                                ABCD[22].sw);
row3.byte2.bits2.bit13 = (unsigned char)((int)\
                                ABCD[22].se);
row3.byte2.bits2.bit14 = (unsigned char)((int)\
                                ABCD[23].sw);
row3.byte2.bits2.bit15 = (unsigned char)((int)\
                                ABCD[23].se);

/* first byte of row4 */
row4.byte1.bits1.bit0 =  (unsigned char)((int)\
                                ABCD[8].nw);
row4.byte1.bits1.bit1 =  (unsigned char)((int)\
                                ABCD[8].ne);
row4.byte1.bits1.bit2 =  (unsigned char)((int)\
                                ABCD[9].nw);
row4.byte1.bits1.bit3 =  (unsigned char)((int)\
                                ABCD[9].ne);
row4.byte1.bits1.bit4 =  (unsigned char)((int)\
                               ABCD[12].nw);
row4.byte1.bits1.bit5 =  (unsigned char)((int)\
                               ABCD[12].ne);
row4.byte1.bits1.bit6 =  (unsigned char)((int)\
                               ABCD[13].nw);
row4.byte1.bits1.bit7 =  (unsigned char)((int)\
                               ABCD[13].ne);
```

```
/* second  byte of row4 */
row4.byte2.bits2.bit8   = (unsigned char)((int)\
                                    ABCD[24].nw);
row4.byte2.bits2.bit9   = (unsigned char)((int)\
                                    ABCD[24].ne);
row4.byte2.bits2.bit10 =  (unsigned char)((int)\
                                    ABCD[25].nw);
row4.byte2.bits2.bit11 = (unsigned char)((int)\
                                    ABCD[25].ne);
row4.byte2.bits2.bit12  = (unsigned char)((int)\
                                    ABCD[28].nw);
row4.byte2.bits2.bit13 = (unsigned char)((int)\
                                    ABCD[28].ne);
row4.byte2.bits2.bit14 = (unsigned char)((int)\
                                    ABCD[29].nw);
row4.byte2.bits2.bit15 = (unsigned char)((int)\
                                    ABCD[29].ne);

/* first byte of row5 */
row5.byte1.bits1.bit0 =  (unsigned char)((int)\
                                    ABCD[8].sw);
row5.byte1.bits1.bit1 =  (unsigned char)((int)\
                                    ABCD[8].se);
row5.byte1.bits1.bit2 =  (unsigned char)((int)\
                                    ABCD[9].sw);
row5.byte1.bits1.bit3 =  (unsigned char)((int)\
                                    ABCD[9].se);
row5.byte1.bits1.bit4 =  (unsigned char)((int)\
                                   ABCD[12].sw);
row5.byte1.bits1.bit5 =  (unsigned char)((int)\
                                   ABCD[12].se);
row5.byte1.bits1.bit6 =  (unsigned char)((int)\
                                   ABCD[13].sw);
row5.byte1.bits1.bit7 =  (unsigned char)((int)\
                                   ABCD[13].se);

/* second byte of row5 */
row5.byte2.bits2.bit8   = (unsigned char)((int)\
                                    ABCD[24].sw);
row5.byte2.bits2.bit9   = (unsigned char)((int)\
                                    ABCD[24].se);
row5.byte2.bits2.bit10 =  (unsigned char)((int)\
                                    ABCD[25].sw);
row5.byte2.bits2.bit11 = (unsigned char)((int)\
                                    ABCD[25].se);
row5.byte2.bits2.bit12  = (unsigned char)((int)\
                                    ABCD[28].sw);
row5.byte2.bits2.bit13 = (unsigned char)((int)\
                                    ABCD[28].se);
row5.byte2.bits2.bit14 = (unsigned char)((int)\
                                    ABCD[29].sw);
row5.byte2.bits2.bit15 = (unsigned char)((int)\
                                    ABCD[29].se);
```

```
/*first byte of row6 */
row6.byte1.bits1.bit0 =  (unsigned char)((int)\
                                ABCD[10].nw);
row6.byte1.bits1.bit1 =  (unsigned char)((int)\
                                ABCD[10].ne);
row6.byte1.bits1.bit2 =  (unsigned char)((int)\
                                ABCD[11].nw);
row6.byte1.bits1.bit3 =  (unsigned char)((int)\
                                ABCD[11].ne);
row6.byte1.bits1.bit4 =  (unsigned char)((int)\
                                ABCD[14].nw);
row6.byte1.bits1.bit5 =  (unsigned char)((int)\
                                ABCD[14].ne);
row6.byte1.bits1.bit6 =  (unsigned char)((int)\
                                ABCD[15].nw);
row6.byte1.bits1.bit7 =  (unsigned char)((int)\
                                ABCD[15].ne);

/* second byte of row6 */
row6.byte2.bits2.bit8 =  (unsigned char)((int)\
                                ABCD[26].nw);
row6.byte2.bits2.bit9 =  (unsigned char)((int)\
                                ABCD[26].ne);
row6.byte2.bits2.bit10 =  (unsigned char)((int)\
                                ABCD[27].nw);
row6.byte2.bits2.bit11 = (unsigned char)((int)\
                                ABCD[27].ne);
row6.byte2.bits2.bit12  = (unsigned char)((int)\
                                ABCD[30].nw);
row6.byte2.bits2.bit13 = (unsigned char)((int)\
                                ABCD[30].ne);
row6.byte2.bits2.bit14 = (unsigned char)((int)\
                                ABCD[31].nw);
row6.byte2.bits2.bit15 = (unsigned char)((int)\
                                ABCD[31].ne);

/* first byte of row7 */
row7.byte1.bits1.bit0 =  (unsigned char)((int)\
                                ABCD[10].sw);
row7.byte1.bits1.bit1 =  (unsigned char)((int)\
                                ABCD[10].se);
row7.byte1.bits1.bit2 =  (unsigned char)((int)\
                                ABCD[11].sw);
row7.byte1.bits1.bit3 =  (unsigned char)((int)\
                                ABCD[11].se);
row7.byte1.bits1.bit4 =  (unsigned char)((int)\
                                ABCD[14].sw);
row7.byte1.bits1.bit5 =  (unsigned char)((int)\
                                ABCD[14].se);
row7.byte1.bits1.bit6 =  (unsigned char)((int)\
                                ABCD[15].sw);
row7.byte1.bits1.bit7 =  (unsigned char)((int)\
                                ABCD[15].se);
```

```
/* second byte of row7 */
row7.byte2.bits2.bit8 =  (unsigned char)((int)\
                                    ABCD[26].sw);
row7.byte2.bits2.bit9 =  (unsigned char)((int)\
                                    ABCD[26].se);
row7.byte2.bits2.bit10 =  (unsigned char)((int)\
                                    ABCD[27].sw);
row7.byte2.bits2.bit11 = (unsigned char)((int)\
                                    ABCD[27].se);
row7.byte2.bits2.bit12  = (unsigned char)((int)\
                                    ABCD[30].sw);
row7.byte2.bits2.bit13 = (unsigned char)((int)\
                                    ABCD[30].se);
row7.byte2.bits2.bit14 = (unsigned char)((int)\
                                    ABCD[31].sw);
row7.byte2.bits2.bit15 = (unsigned char)((int)\
                                    ABCD[31].se);

/* first byte of row8 */
row8.byte1.bits1.bit0 =  (unsigned char)((int)\
                                    ABCD[32].nw);
row8.byte1.bits1.bit1 =  (unsigned char)((int)\
                                    ABCD[32].ne);
row8.byte1.bits1.bit2 =  (unsigned char)((int)\
                                    ABCD[33].nw);
row8.byte1.bits1.bit3 =  (unsigned char)((int)\
                                    ABCD[33].ne);
row8.byte1.bits1.bit4 =  (unsigned char)((int)\
                                    ABCD[36].nw);
row8.byte1.bits1.bit5 =  (unsigned char)((int)\
                                    ABCD[36].ne);
row8.byte1.bits1.bit6 =  (unsigned char)((int)\
                                    ABCD[37].nw);
row8.byte1.bits1.bit7 =  (unsigned char)((int)\
                                    ABCD[37].ne);

/* second byte of row8 */
row8.byte2.bits2.bit8   = (unsigned char)((int)\
                                    ABCD[48].nw);
row8.byte2.bits2.bit9   = (unsigned char)((int)\
                                    ABCD[48].ne);
row8.byte2.bits2.bit10 =  (unsigned char)((int)\
                                    ABCD[49].nw);
row8.byte2.bits2.bit11 = (unsigned char)((int)\
                                    ABCD[49].ne);
row8.byte2.bits2.bit12  = (unsigned char)((int)\
                                    ABCD[52].nw);
row8.byte2.bits2.bit13 = (unsigned char)((int)\
                                    ABCD[52].ne);
row8.byte2.bits2.bit14 = (unsigned char)((int)\
                                    ABCD[53].nw);
row8.byte2.bits2.bit15 = (unsigned char)((int)\
                                    ABCD[53].ne);
```

```
/* first byte of row9 */
row9.byte1.bits1.bit0 =  (unsigned char)((int)\
                                    ABCD[32].sw);
row9.byte1.bits1.bit1 =  (unsigned char)((int)\
                                    ABCD[32].se);
row9.byte1.bits1.bit2 =  (unsigned char)((int)\
                                    ABCD[33].sw);
row9.byte1.bits1.bit3 =  (unsigned char)((int)\
                                    ABCD[33].se);
row9.byte1.bits1.bit4 =  (unsigned char)((int)\
                                    ABCD[36].sw);
row9.byte1.bits1.bit5 =  (unsigned char)((int)\
                                    ABCD[36].se);
row9.byte1.bits1.bit6 =  (unsigned char)((int)\
                                    ABCD[37].sw);
row9.byte1.bits1.bit7 =  (unsigned char)((int)\
                                    ABCD[37].se);

/* second byte of row9 */
row9.byte2.bits2.bit8   = (unsigned char)((int)\
                                    ABCD[48].sw);
row9.byte2.bits2.bit9   = (unsigned char)((int)\
                                    ABCD[48].se);
row9.byte2.bits2.bit10 =  (unsigned char)((int)\
                                    ABCD[49].sw);
row9.byte2.bits2.bit11 = (unsigned char)((int)\
                                    ABCD[49].se);
row9.byte2.bits2.bit12  = (unsigned char)((int)\
                                    ABCD[52].sw);
row9.byte2.bits2.bit13 = (unsigned char)((int)\
                                    ABCD[52].se);
row9.byte2.bits2.bit14 = (unsigned char)((int)\
                                    ABCD[53].sw);
row9.byte2.bits2.bit15 = (unsigned char)((int)\
                                    ABCD[53].se);

/* first byte of row10 */
row10.byte1.bits1.bit0 = (unsigned  char)((int)\
                                    ABCD[34].nw);
row10.byte1.bits1.bit1 = (unsigned char)((int)\
                                    ABCD[34].ne);
row10.byte1.bits1.bit2 =  (unsigned char)((int)\
                                    ABCD[35].nw);
row10.byte1.bits1.bit3 = (unsigned char)((int)\
                                    ABCD[35].ne);
row10.byte1.bits1.bit4  = (unsigned char)((int)\
                                    ABCD[38].nw);
row10.byte1.bits1.bit5 = (unsigned char)((int)\
                                    ABCD[38].ne);
row10.byte1.bits1.bit6 = (unsigned char)((int)\
                                    ABCD[39].nw);
row10.byte1.bits1.bit7 = (unsigned char)((int)\
                                    ABCD[39].ne);
```

```
/* second byte of row10 */
row10.byte2.bits2.bit8 = (unsigned  char)((int)\
                                    ABCD[50].nw);
row10.byte2.bits2.bit9 = (unsigned char)((int)\
                                    ABCD[50].ne);
row10.byte2.bits2.bit10 = (unsigned char)((int)\
                                    ABCD[51].nw);
row10.byte2.bits2.bit11 = (unsigned char)((int)\
                                    ABCD[51].ne);
row10.byte2.bits2.bit12 = (unsigned char)((int)\
                                    ABCD[54].nw);
row10.byte2.bits2.bit13 = (unsigned char)((int)\
                                    ABCD[54].ne);
row10.byte2.bits2.bit14 = (unsigned char)((int)\
                                    ABCD[55].nw);
row10.byte2.bits2.bit15 = (unsigned char)((int)\
                                    ABCD[55].ne);

/* first byte of     row11 */
row11.byte1.bits1.bit0 = (unsigned  char)((int)\
                                    ABCD[34].sw);
row11.byte1.bits1.bit1 = (unsigned char)((int)\
                                    ABCD[34].se);
row11.byte1.bits1.bit2 =  (unsigned char)((int)\
                                    ABCD[35].sw);
row11.byte1.bits1.bit3 = (unsigned char)((int)\
                                    ABCD[35].se);
row11.byte1.bits1.bit4  = (unsigned char)((int)\
                                    ABCD[38].sw);
row11.byte1.bits1.bit5 = (unsigned char)((int)\
                                    ABCD[38].se);
row11.byte1.bits1.bit6 = (unsigned char)((int)\
                                    ABCD[39].sw);
row11.byte1.bits1.bit7 = (unsigned char)((int)\
                                    ABCD[39].se);

/* second byte of row11 */
row11.byte2.bits2.bit8 = (unsigned  char)((int)\
                                    ABCD[50].sw);
row11.byte2.bits2.bit9 = (unsigned char)((int)\
                                    ABCD[50].se);
row11.byte2.bits2.bit10 = (unsigned char)((int)\
                                    ABCD[51].sw);
row11.byte2.bits2.bit11 = (unsigned char)((int)\
                                    ABCD[51].se);
row11.byte2.bits2.bit12 = (unsigned char)((int)\
                                    ABCD[54].sw);
row11.byte2.bits2.bit13 = (unsigned char)((int)\
                                    ABCD[54].se);
row11.byte2.bits2.bit14 = (unsigned char)((int)\
                                    ABCD[55].sw);
row11.byte2.bits2.bit15 = (unsigned char)((int)\
                                    ABCD[55].se);
```

```
/* first byte of row12 */
row12.byte1.bits1.bit0 = (unsigned  char)((int)\
                                    ABCD[40].nw);
row12.byte1.bits1.bit1 = (unsigned char)((int)\
                                    ABCD[40].ne);
row12.byte1.bits1.bit2 =  (unsigned char)((int)\
                                    ABCD[41].nw);
row12.byte1.bits1.bit3 = (unsigned char)((int)\
                                    ABCD[41].ne);
row12.byte1.bits1.bit4  = (unsigned char)((int)\
                                    ABCD[44].nw);
row12.byte1.bits1.bit5 = (unsigned char)((int)\
                                    ABCD[44].ne);
row12.byte1.bits1.bit6 = (unsigned char)((int)\
                                    ABCD[45].nw);
row12.byte1.bits1.bit7 = (unsigned char)((int)\
                                    ABCD[45].ne);

/* second byte of row12 */
row12.byte2.bits2.bit8  =  (unsigned char)((int)\
                                     ABCD[56].nw);
row12.byte2.bits2.bit9  = (unsigned char)((int)\
                                     ABCD[56].ne);
row12.byte2.bits2.bit10 = (unsigned char)((int)\
                                     ABCD[57].nw);
row12.byte2.bits2.bit11 = (unsigned char)((int)\
                                     ABCD[57].ne);
row12.byte2.bits2.bit12 = (unsigned char)((int)\
                                     ABCD[60].nw);
row12.byte2.bits2.bit13 = (unsigned char)((int)\
                                     ABCD[60].ne);
row12.byte2.bits2.bit14 = (unsigned char)((int)\
                                     ABCD[61].nw);
row12.byte2.bits2.bit15 = (unsigned char)((int)\
                                     ABCD[61].ne);

/* first byte of row13 */
row13.byte1.bits1.bit0 = (unsigned  char)((int)\
                                    ABCD[40].sw);
row13.byte1.bits1.bit1 = (unsigned char)((int)\
                                    ABCD[40].se);
row13.byte1.bits1.bit2 =  (unsigned char)((int)\
                                    ABCD[41].sw);
row13.byte1.bits1.bit3 = (unsigned char)((int)\
                                    ABCD[41].se);
row13.byte1.bits1.bit4  = (unsigned char)((int)\
                                    ABCD[44].sw);
row13.byte1.bits1.bit5 = (unsigned char)((int)\
                                    ABCD[44].se);
row13.byte1.bits1.bit6 = (unsigned char)((int)\
                                    ABCD[45].sw);
row13.byte1.bits1.bit7 = (unsigned char)((int)\
                                    ABCD[45].se);
```

```
/* second byte of row13 */
row13.byte2.bits2.bit8  =  (unsigned char)((int)\
                                    ABCD[56].sw);
row13.byte2.bits2.bit9  = (unsigned char)((int)\
                                    ABCD[56].se);
row13.byte2.bits2.bit10 = (unsigned char)((int)\
                                    ABCD[57].sw);
row13.byte2.bits2.bit11 = (unsigned char)((int)\
                                    ABCD[57].se);
row13.byte2.bits2.bit12 = (unsigned char)((int)\
                                    ABCD[60].sw);
row13.byte2.bits2.bit13 = (unsigned char)((int)\
                                    ABCD[60].se);
row13.byte2.bits2.bit14 = (unsigned char)((int)\
                                    ABCD[61].sw);
row13.byte2.bits2.bit15 = (unsigned char)((int)\
                                    ABCD[61].se);

/* first byte of row14 */
row14.byte1.bits1.bit0 = (unsigned  char)((int)\
                                   ABCD[42].nw);
row14.byte1.bits1.bit1 = (unsigned char)((int)\
                                   ABCD[42].ne);
row14.byte1.bits1.bit2 =  (unsigned char)((int)\
                                   ABCD[43].nw);
row14.byte1.bits1.bit3 = (unsigned char)((int)\
                                   ABCD[43].ne);
row14.byte1.bits1.bit4  = (unsigned char)((int)\
                                   ABCD[46].nw);
row14.byte1.bits1.bit5 = (unsigned char)((int)\
                                   ABCD[46].ne);
row14.byte1.bits1.bit6 = (unsigned char)((int)\
                                   ABCD[47].nw);
row14.byte1.bits1.bit7 = (unsigned char)((int)\
                                   ABCD[47].ne);

/* second byte of row14 */
row14.byte2.bits2.bit8  =  (unsigned char)((int)\
                                    ABCD[58].nw);
row14.byte2.bits2.bit9  = (unsigned char)((int)\
                                    ABCD[58].ne);
row14.byte2.bits2.bit10 = (unsigned char)((int)\
                                    ABCD[59].nw);
row14.byte2.bits2.bit11 = (unsigned char)((int)\
                                    ABCD[59].ne);
row14.byte2.bits2.bit12 = (unsigned char)((int)\
                                    ABCD[62].nw);
row14.byte2.bits2.bit13 = (unsigned char)((int)\
                                    ABCD[62].ne);
row14.byte2.bits2.bit14 = (unsigned char)((int)\
                                    ABCD[63].nw);
row14.byte2.bits2.bit15 = (unsigned char)((int)\
                                    ABCD[63].ne);
```

```
/* first byte of row15 */
row15.byte1.bits1.bit0 = (unsigned  char)((int)\
                                        ABCD[42].sw);
row15.byte1.bits1.bit1 = (unsigned char)((int)\
                                        ABCD[42].se);
row15.byte1.bits1.bit2 =  (unsigned char)((int)\
                                        ABCD[43].sw);
row15.byte1.bits1.bit3 = (unsigned char)((int)\
                                        ABCD[43].se);
row15.byte1.bits1.bit4  = (unsigned char)((int)\
                                        ABCD[46].sw);
row15.byte1.bits1.bit5 = (unsigned char)((int)\
                                        ABCD[46].se);
row15.byte1.bits1.bit6 = (unsigned char)((int)\
                                        ABCD[47].sw);
row15.byte1.bits1.bit7 = (unsigned char)((int)\
                                        ABCD[47].se);

/* second byte of row15 */
row15.byte2.bits2.bit8  =  (unsigned char)((int)\
                                          ABCD[58].sw);
row15.byte2.bits2.bit9  = (unsigned char)((int)\
                                          ABCD[58].se);
row15.byte2.bits2.bit10 = (unsigned char)((int)\
                                          ABCD[59].sw);
row15.byte2.bits2.bit11 = (unsigned char)((int)\
                                          ABCD[59].se);
row15.byte2.bits2.bit12 = (unsigned char)((int)\
                                          ABCD[62].sw);
row15.byte2.bits2.bit13 = (unsigned char)((int)\
                                          ABCD[62].se);
row15.byte2.bits2.bit14 = (unsigned char)((int)\
                                          ABCD[63].sw);
row15.byte2.bits2.bit15 = (unsigned char)((int)\
                                          ABCD[63].se);
/* end of bit manuplation */

/* revers placed data for positive image */
row0.byte1.bt1.holebyte1 = (unsigned char)
                  ~row0.byte1.bt1.holebyte1;
row1.byte1.bt1.holebyte1 = (unsigned  char)
                  ~row1.byte1.bt1.holebyte1;
row2.byte1.bt1.holebyte1 = (unsigned char)
                  ~row2.byte1.bt1.holebyte1;
row3.byte1.bt1.holebyte1 = (unsigned char)
                  ~row3.byte1.bt1.holebyte1;
row4.byte1.bt1.holebyte1 = (unsigned char)
                  ~row4.byte1.bt1.holebyte1;
row5.byte1.bt1.holebyte1 = (unsigned char)
                  ~row5.byte1.bt1.holebyte1;
row6.byte1.bt1.holebyte1  = (unsigned char)
                  ~row6.byte1.bt1.holebyte1;
row7.byte1.bt1.holebyte1 = (unsigned char)
                  ~row7.byte1.bt1.holebyte1;
row8.byte1.bt1.holebyte1 = (unsigned  char)
                  ~row8.byte1.bt1.holebyte1;
row9.byte1.bt1.holebyte1 = (unsigned char)
                  ~row9.byte1.bt1.holebyte1;
```

```
row10.byte1.bt1.holebyte1 =  (unsigned char)
                  ~row10.byte1.bt1.holebyte1;
row11.byte1.bt1.holebyte1 = (unsigned char)
                  ~row11.byte1.bt1.holebyte1;
row12.byte1.bt1.holebyte1 = (unsigned char)
                  ~row12.byte1.bt1.holebyte1;
row13.byte1.bt1.holebyte1 = (unsigned char)
                  ~row13.byte1.bt1.holebyte1;
row14.byte1.bt1.holebyte1 = (unsigned  char)
                  ~row14.byte1.bt1.holebyte1;
row15.byte1.bt1.holebyte1 = (unsigned char)
                  ~row15.byte1.bt1.holebyte1;

row0.byte2.bt2.holebyte2 = (unsigned char)
                  ~row0.byte2.bt2.holebyte2;
row1.byte2.bt2.holebyte2  = (unsigned char)
                  ~row1.byte2.bt2.holebyte2;
row2.byte2.bt2.holebyte2 = (unsigned char)
                  ~row2.byte2.bt2.holebyte2;
row3.byte2.bt2.holebyte2 = (unsigned  char)
                  ~row3.byte2.bt2.holebyte2;
row4.byte2.bt2.holebyte2 = (unsigned char)
                  ~row4.byte2.bt2.holebyte2;
row5.byte2.bt2.holebyte2 = (unsigned char)
                  ~row5.byte2.bt2.holebyte2;
row6.byte2.bt2.holebyte2 = (unsigned char)
                  ~row6.byte2.bt2.holebyte2;
row7.byte2.bt2.holebyte2 = (unsigned char)
                  ~row7.byte2.bt2.holebyte2;
row8.byte2.bt2.holebyte2  = (unsigned char)
                  ~row8.byte2.bt2.holebyte2;
row9.byte2.bt2.holebyte2 = (unsigned char)
                  ~row9.byte2.bt2.holebyte2;
row10.byte2.bt2.holebyte2 = (unsigned char)
                  ~row10.byte2.bt2.holebyte2;
row11.byte2.bt2.holebyte2 = (unsigned char)
                  ~row11.byte2.bt2.holebyte2;
row12.byte2.bt2.holebyte2 = (unsigned char)
                  ~row12.byte2.bt2.holebyte2;
row13.byte2.bt2.holebyte2 =  (unsigned char)
                  ~row13.byte2.bt2.holebyte2;
row14.byte2.bt2.holebyte2 = (unsigned char)
                  ~row14.byte2.bt2.holebyte2;
row15.byte2.bt2.holebyte2 = (unsigned char)
                  ~row15.byte2.bt2.holebyte2;
```

```
    /* save resault of placement in place.dat file */
    /* first of rows */
    fwrite(&row0.byte1.bt1.holebyte1,1,1, placestream);
    fwrite(&row1.byte1.bt1.holebyte1,1,1, placestream);
    fwrite(&row2.byte1.bt1.holebyte1,1,1, placestream);
    fwrite(&row3.byte1.bt1.holebyte1,1,1, placestream);
    fwrite(&row4.byte1.bt1.holebyte1,1,1, placestream);
    fwrite(&row5.byte1.bt1.holebyte1,1,1, placestream);
    fwrite(&row6.byte1.bt1.holebyte1,1,1, placestream);
    fwrite(&row7.byte1.bt1.holebyte1,1,1, placestream);
    fwrite(&row8.byte1.bt1.holebyte1,1,1, placestream);
    fwrite(&row9.byte1.bt1.holebyte1,1,1, placestream);
    fwrite(&row10.byte1.bt1.holebyte1,1,1, placestream);
    fwrite(&row11.byte1.bt1.holebyte1,1,1, placestream);
    fwrite(&row12.byte1.bt1.holebyte1,1,1, placestream);
    fwrite(&row13.byte1.bt1.holebyte1,1,1, placestream);
    fwrite(&row14.byte1.bt1.holebyte1,1,1, placestream);
    fwrite(&row15.byte1.bt1.holebyte1,1,1, placestream);

    /* second byte of rows */
    fwrite(&row0.byte2.bt2.holebyte2,1,1, placestream);
    fwrite(&row1.byte2.bt2.holebyte2,1,1, placestream);
    fwrite(&row2.byte2.bt2.holebyte2,1,1, placestream);
    fwrite(&row3.byte2.bt2.holebyte2,1,1, placestream);
    fwrite(&row4.byte2.bt2.holebyte2,1,1, placestream);
    fwrite(&row5.byte2.bt2.holebyte2,1,1, placestream);
    fwrite(&row6.byte2.bt2.holebyte2,1,1, placestream);
    fwrite(&row7.byte2.bt2.holebyte2,1,1, placestream);
    fwrite(&row8.byte2.bt2.holebyte2,1,1, placestream);
    fwrite(&row9.byte2.bt2.holebyte2,1,1, placestream);
    fwrite(&row10.byte2.bt2.holebyte2,1,1, placestream);
    fwrite(&row11.byte2.bt2.holebyte2,1,1, placestream);
    fwrite(&row12.byte2.bt2.holebyte2,1,1, placestream);
    fwrite(&row13.byte2.bt2.holebyte2,1,1, placestream);
    fwrite(&row14.byte2.bt2.holebyte2,1,1, placestream);
    fwrite(&row15.byte2.bt2.holebyte2,1,1, placestream);



    /* display row structurs */
     row_setf(0,row0,roffset,coffset);
     row_setf(1,row1,roffset,coffset);
     row_setf(2,row2,roffset,coffset);
     row_setf(3,row3,roffset,coffset);
     row_setf(4,row4,roffset,coffset);
     row_setf(5,row5,roffset,coffset);
     row_setf(6,row6,roffset,coffset);
     row_setf(7,row7,roffset,coffset);
     row_setf(8,row8,roffset,coffset);
     row_setf(9,row9,roffset,coffset);
     row_setf(10,row10,roffset,coffset);
     row_setf(11,row11,roffset,coffset);
     row_setf(12,row12,roffset,coffset);
     row_setf(13,row13,roffset,coffset);
     row_setf(14,row14,roffset,coffset);
     row_setf(15,row15,roffset,coffset);
     return (0);
} /* end of place.c */
```

```
/* row_set function display row structurs */

void row_setf(row_num,row,roffset,coffset)
     int row_num,roffset,coffset;
     struct ROW     row;
     {
     int j;
     unsigned char byte, mask;
     mask = 1;
     for(j = 0; j < 8; j++)
      {
       byte  = (unsigned char)
               (row.byte1.bt1.holebyte1 >> j);
       byte &= mask;
       if(byte)   _setpixel((row_num  + roffset),\
                            (j + coffset));
       };

     for(j = 0; j < 8; j++)
      {
       byte  = (unsigned char)\
               (row.byte2.bt2.holebyte2 >> j);
       byte &= mask;
       if(byte)  _setpixel((row_num + roffset),\
                           (j + coffset + 8));
       };
     }

/* maxdotsf() function gets dots number to be divided by
** four and retun maximum number of dots for each part
** dynamicly.
*/

 int  maxdotsf(dots_number)
      int dots_number;
       {
          if(dots_number > 64) return(64);
      else if((dots_number  > 16) && (dots_number < 65))
            return(16);
      else if((dots_number > 4 )  && (dots_number < 17))
             return( 4);
      else if((dots_number >= 1) && (dots_number < 5))
             return( 1);
      else return (0);
      }
```

```
/* s_boardf.c function gets number of dots to be divided
** to four part maximum number of dots in each division
** and neighbor structur.  Then function itself returns
** division structur.
*/

#include  <stdio.h>
#include  <math.h>
# include "place.h"
#include  <conio.h>
#define  ON    1
#define  OFF   0


struct ELEMENT s_boardf(float  dots_number,int pol_dir,\
                        int maxdots, struct NEIGHBOR N)
  {
   struct ELEMENT buffer;
   float west,east,north,south;
   float nw,ne,sw,se;
   float max_dots;
   float nw_trun, ne_trun, sw_trun, se_trun,trun_total;
   int nw_enable, ne_enable, sw_enable, se_enable;
   int temp;
   float w_e, n_s;

   float nw_max,ne_max,sw_max,se_max;
   float w_max, e_max, n_max, s_max;
   float nw_tmp, ne_tmp, sw_tmp, se_tmp;
   float total,r,net,percent;

   static int NWSEswitch, NESWswitch, switch2, switch3;

   max_dots = (float)(maxdots);

   nw = 0;
   ne = 0;
   sw = 0;
   se = 0;

   NWSEswitch = 2;
   NESWswitch = 3;
   switch2    = 1;
   switch3    = 4;

   nw_enable = ON;
   ne_enable = ON;
   sw_enable = ON;
   se_enable = ON;

   west  = N.west ;
   east  = N.east ;
   north = N.north;
   south = N.south;

   w_e = west - east;
   n_s = north - south;
```

```
if(dots_number)
  {
  w_max = (west + (2 * dots_number)) / 8;
  e_max = (east + (2 * dots_number)) / 8;
  n_max = (north + (2 * dots_number)) / 8;
  s_max = (south + (2 * dots_number)) / 8;

  nw_tmp = n_max + w_max;
  ne_tmp = n_max + e_max;
  sw_tmp = s_max + w_max;
  se_tmp = s_max + e_max;

  total  = nw_tmp + ne_tmp + sw_tmp + se_tmp;

  nw_max = (nw_tmp * dots_number) / total;
  ne_max = (ne_tmp * dots_number) / total;
  sw_max = (sw_tmp * dots_number) / total;
  se_max = (se_tmp * dots_number) / total;

  temp = (int)(nw_max);
  if((nw_max - (float)(temp)) > 0.1f)
    {
     ++temp;
     };
  nw_max = (float)(temp);

  temp = (int)(ne_max);
  if((ne_max - (float)(temp)) > 0.1f)
    {
     ++temp;
     };
  ne_max = (float)(temp);

  temp = (int)(sw_max);
  if((sw_max - (float)(temp)) > 0.1f)
    {
     ++temp;
     };
  sw_max = (float)(temp);

  temp = (int)(se_max);
  if((se_max - (float)(temp)) > 0.1f)
    {
     ++temp;
     };
  se_max = (float)(temp);


    do{
    r = (float)sqrt((double)((w_e * w_e)+(n_s * n_s)));
    net = r - dots_number;
    --dots_number;
```

```
if( (w_e > 0) && (n_s > 0) && (net >= 0) )
  {
       if(nw_enable)
      {
       ++nw;
       percent = n_s /(n_s + w_e);
       north -= percent;
       west -= (1 - percent);
       w_e = west - east;
       n_s = north - south;
       if(nw >= nw_max)
         {
          nw_enable = OFF;
          };
       }

   else {
       ++dots_number;
       --north;
       --west;
       w_e = west - east;
       n_s = north - south;
       };
    }

  /*********/
else if( (w_e > 0) && (n_s < 0) && (net >= 0))
       {
           if(sw_enable)
             {
              ++sw;
              percent = (- n_s) / (- n_s + w_e);
              south -= percent;
              west -= (1 - percent);
              w_e = west - east;
              n_s = north - south;
              if(sw >= sw_max)
                {
                 sw_enable = OFF;
                };
              }

        else {
              ++dots_number;
              --south;
              --west;
              w_e = west - east;
              n_s = north - south;
              };
        }
```

```
    /*********************/
else if( (w_e > 0) && (n_s == 0) && (net >= 0))
       {
          if((pol_dir == 1) && (nw_enable) &&  \
                                      (sw_enable))
            {
          nw += 0.50;
          sw += 0.50;
          --west;
          w_e = west - east;
          if(sw >= sw_max)
            {
              sw_enable = OFF;
              };
          if(nw >= nw_max)
            {
             nw_enable = OFF;
             };
          }

         else if((pol_dir == 3) && (sw_enable)  && \
                                          (nw_enable))
             {
              sw += 0.50;
              nw += 0.50;
              --west;
              w_e = west - east;
              n_s = north - south;
              if(nw >= nw_max)
                {
                 nw_enable = OFF;
                 };
              if(sw >= sw_max)
                {
                 sw_enable = OFF;
                 };
              }
         else if((nw_enable) && (sw_enable))
             {
              nw += 0.5;
              sw += 0.5;
              --west;
              w_e = west - east;
              if(sw >= sw_max)
                {
                 sw_enable = OFF;
                 };
              if(nw >= nw_max)
                {
                 nw_enable = OFF;
                 };
              }
         else {
              ++dots_number;
              --west;
              w_e = west - east;
              };
       }
```

```
      /**********************/
else if((w_e < 0) && (n_s > 0) && (net >= 0))
     {
          if(ne_enable)
         {
          ++ne;
          percent = n_s /(n_s - w_e);
          north -= percent;
          east -= (1 - percent);
          w_e = west - east;
          n_s = north - south;
          if(ne >= ne_max)
            {
             ne_enable = OFF;
          };
         }

       else {
             ++dots_number;
             --north;
             --east;
             w_e = west - east;
             n_s = north - south;
             };

       }

 /**********************/

else if( (w_e < 0) && (n_s < 0) && (net >= 0))
     {
          if(se_enable)
         {
          ++se;
          percent = n_s / (n_s + w_e);
          south -= percent;
          east -= (1-percent);
          w_e = west - east;
          n_s = north - south;
          if(se >= se_max)
            {
             se_enable = OFF;
             };
          }

       else {
             ++dots_number;
             --south;
             --east;
             w_e = west - east;
             n_s = north - south;
             };

       }
```

```
/******************/
else if((w_e < 0) && (n_s == 0) && (net >= 0))
     {
            if((pol_dir == 4) && (se_enable) &&\
                                        (ne_enable))
           {
            se += 0.50;
            ne += 0.50;
            --east;
            w_e = west - east;
            if(ne >= ne_max)
              {
               ne_enable = OFF;
               };
            if(se >= se_max)
              {
               se_enable = OFF;
               };
           }

       else if((pol_dir  == 2) && (ne_enable) && \
                                        (se_enable))
           {
            ne += 0.50;
            se += 0.50;
            --east;
            w_e = west - east;
            if(se >= se_max)
              {
               se_enable = OFF;
               };
            if(ne >= ne_max)
              {
               ne_enable = OFF;
               };
           }
       else if((se_enable) && (ne_enable))
           {
            se += 0.5;
            ne += 0.5;
            --east;
            w_e = west - east;
            if(ne >= ne_max)
              {
               ne_enable = OFF;
               };
            if(se >= se_max)
              {
               se_enable = OFF;
               };
           }
      else {
            ++dots_number;
            --east;
            w_e = west - east;
            };
      }
```

```
/*******************/
else if((w_e == 0) && (n_s > 0) && (net >= 0))
     {
          if((pol_dir ==  1) && (nw_enable) && \
                                    (ne_enable))
         {
          nw += 0.50;
          ne += 0.50;
          --north;
          n_s = north - south;
          if(ne >= ne_max)
            {
             ne_enable = OFF;
             };

          if(nw >= nw_max)
            {
             nw_enable = OFF;
             };
          }
        else if((pol_dir == 2) && (ne_enable) &&  \
                                   (nw_enable))
         {
          ne += 0.50;
          nw += 0.50;
          --north;
          n_s = north - south;
          if(nw >= nw_max)
            {
             nw_enable = OFF;
             };
          if(ne >= ne_max)
            {
             ne_enable = OFF;
             };
          }
        else if((ne_enable) && (nw_enable))
         {
          ne += 0.5;
          nw += 0.5;
          --north;
          n_s = north - south;
          if(nw >= nw_max)
            {
             nw_enable = OFF;
             };

          if(ne >= ne_max)
            {
             ne_enable = OFF;
             };
          }
        else {
              ++dots_number;
              --north;
              n_s = north - south;
              };
        }
```

```
/**********************/

else if((w_e == 0) && (n_s < 0) && (net >= 0))
    {
          if((pol_dir ==  3) && (sw_enable) && \
                                    (se_enable))
            {
             sw += 0.50;
             se += 0.50;
             --south;
             n_s = north - south;
             if( se >= se_max)
               {
                se_enable = OFF;
                };

             if(sw >= sw_max)
               {
                sw_enable = OFF;
                };
             }
      else if((pol_dir == 4) && (se_enable) && \
                                    (sw_enable))
          {
           se += 0.50;
           sw += 0.50;
           --south;
           n_s = north - south;
           if(sw >= sw_max)
             {
              sw_enable = OFF;
              };
           if(se >= se_max)
             {
              se_enable = OFF;
              };
           }
      else if((sw_enable) && (se_enable))
           {
            sw += 0.5;
            se += 0.5;
            --south;
            n_s = north - south;
            if(se >= se_max)
              {
               se_enable = OFF;
               };
            if(sw >= sw_max)
              {
               sw_enable = OFF;
               };
            }
      else {
            ++dots_number;
            --south;
            n_s = north - south;
            };
      }
```

```
/************************/
else if((w_e == 0) && (n_s == 0) || (net < 0))
    {
          if((nw_enable) &&  (ne_enable) && \
                (sw_enable) && (se_enable))
         {
          nw += 0.25;
          ne += 0.25;
          sw += 0.25;
          se += 0.25;
          if(nw >= nw_max)
            {
             nw_enable = OFF;
             };
          if(ne >= ne_max)
            {
             ne_enable = OFF;
             };
          if(sw >= sw_max)
            {
             sw_enable = OFF;
             };
          if(se >= se_max)
            {
             se_enable = OFF;
             };
         }
      else if((sw_enable) && (ne_enable))
        {
         sw += 0.50;
         ne += 0.50;
         if(sw >= sw_max)
           {
            sw_enable = OFF;
            };
         if(ne >= ne_max)
           {
            ne_enable = OFF;
            };
         }
      else if((nw_enable) && (se_enable))
         {
          nw += 0.50;
          se += 0.50;
          if(nw >= nw_max)
            {
             nw_enable = OFF;
             };
          if(se >= se_max)
            {
             se_enable = OFF;
             };
          }
```

```
else if((nw_enable) && (ne_enable))
    {
     nw += 0.50;
     ne += 0.50;
     north -= 0.50;
     n_s = north - south;
     if(nw >= nw_max)
       {
        nw_enable = OFF;
        };
     if(ne >= ne_max)
       {
        ne_enable = OFF;
        };
     }
else if((sw_enable) && (se_enable))
    {
     sw += 0.50;
     se += 0.50;
     south -= 0.50;
     n_s = north - south;
     if(sw >= sw_max)
       {
        sw_enable = OFF;
        };
     if(se >= se_max)
       {
        se_enable = OFF;
        };
     }
else if((ne_enable) && (se_enable))
       {
        ne += 0.50;
        se += 0.50;
        east -= 0.50;
        w_e = west - east;
        if(ne >= ne_max)
         {
          ne_enable = OFF;
          };
        if(se >= se_max)
          {
           se_enable = OFF;
           };
        }
```

```
else if((sw_enable) && (nw_enable))
        {
         sw += 0.50;
         nw += 0.50;
         west -= 0.50;
         w_e = west - east;
         if(sw >= sw_max)
           {
            sw_enable = OFF;
            };
        if(nw >= nw_max)
          {
           nw_enable = OFF;
           };
        }
else if(ne_enable)
        {
         ++ne;
         north -= 0.50;
         east -= 0.50;
         w_e  = west - east;
         n_s  = north - south;
         if(ne >= ne_max)
           {
            ne_enable = OFF;
            };
         }
else if(sw_enable)
        {
         ++sw;
         south -= 0.50;
         west -= 0.50;
         w_e  = west - east;
         n_s  = north - south;
         if(sw >= sw_max)
           {
            sw_enable = OFF;
            };
         }
else if(se_enable)
        {
         ++se;
         south -= 0.50;
         east -= 0.50;
         n_s  = north - south;
         w_e  = west - east;
         if(se >= se_max)
           {
            se_enable = OFF;
            };
         }
```

```
        else if(nw_enable)
               {
                ++nw;
                north -= 0.50;
                west -= 0.50;
                n_s  = north - south;
                w_e  = west  - east;
                if(nw >= nw_max)
                  {
                   nw_enable = OFF;
                   };
                }
        else;

        }

  else;

}while( dots_number );


nw_trun = nw - (int)(nw);
ne_trun = ne - (int)(ne);
sw_trun = sw - (int)(sw);
se_trun = se - (int)(se);
trun_total = nw_trun + ne_trun + sw_trun + se_trun;


if(trun_total)
 {
  if(trun_total == 1.0f)
    {
        if((pol_dir == 3) && (sw_enable))
          {
           ++sw;
           }
  else if((pol_dir == 2) && (ne_enable))
          {
           ++ne;
           }
  else if((pol_dir == 4) && (se_enable))
          {
           ++se;
           }
  else if((pol_dir == 1) && (nw_enable))
          {
           ++nw;
           }
  else if((pol_dir == 1) && (se_enable))
                              /*crossing*/
          {
           ++se;
           }
  else if((pol_dir == 4) && (nw_enable))
          {
           ++nw;
           }
```

```
else if((pol_dir == 2) && (sw_enable))
       {
        ++sw;
        }
else if((pol_dir == 3) && (ne_enable))
       {
        ++ne;
        }
else if((! nw_enable) && (! se_enable))
       {
          if((NWSEswitch == 2) && (ne_enable))
            {
             ++ne;
             NWSEswitch = 3;
             }
     else if((NWSEswitch == 3) && (sw_enable))
            {
             ++sw;
             NWSEswitch = 2;
             }
     else;
     }
else if((! ne_enable) && (! sw_enable))
       {
          if((NESWswitch == 1) && (nw_enable))
            {
             ++nw;
             NESWswitch = 4;
             }
     else if((NESWswitch == 4) && (se_enable))
            {
             ++se;
             NESWswitch = 1;
             }
     else;
     }

else;
};



if(trun_total == 2.0f)
  {
   do{
      --trun_total;
   if((nw_trun  == ne_trun) && (sw_trun == se_trun)
                            && (nw_trun == sw_trun))
     {
        if(((pol_dir == 2) || (pol_dir == 3)) &&
                     (ne_enable) && (sw_enable))
          {
           --trun_total;
           ++ne;
           ++sw;
           }
```

```
else if(((pol_dir  == 1) || (pol_dir == 4)) &&
                  (nw_enable) && (se_enable))
       {
        --trun_total;
       ++nw;
       ++se;
       }
else if(((pol_dir  == 1) || (pol_dir == 4)) &&
              (! nw_enable) && (! se_enable))
       {
        --trun_total;
        ++ne;
        ++sw;
        }
else if(((pol_dir  == 2) || (pol_dir == 3)) &&
              (! ne_enable) && (! sw_enable))
       {
        --trun_total;
        ++nw;
        ++se;
        }
else if(switch2 == 1)
       {
          if(nw_enable)
            {
             ++nw;
             nw_trun = 0.0f;
             ne_trun = 0.0f;
             if(nw >= nw_max)
               {
                nw_enable = OFF;
               };
             switch2 = 4;
             }
     else if(se_enable)
            {
             ++se;
             se_trun = 0.0f;
             sw_trun = 0.0f;
             if(se >= se_max)
               {
                se_enable = OFF;
               };
             switch2 = 1;
             }
     else if(ne_enable)
            {
             ++ne;
             ne_trun = 0.0f;
             nw_trun = 0.0f;
             if(ne >= ne_max)
               {
                ne_enable = OFF;
               };
             switch2 = 3;
             }
```

```
        else if(sw_enable)
               {
                ++sw;
                sw_trun = 0.0f;
                se_trun = 0.0f;
                if(sw >= sw_max)
                  {
                   sw_enable = OFF;
                   };
                switch2 = 2;
                }
        else {
              ++trun_total;
              switch2 = 4;
              };
        }
   else if(switch2 == 2)
          {
             if(ne_enable)
               {
                ++ne;
                ne_trun = 0.0f;
                nw_trun = 0.0f;
                if(ne >= ne_max)
                  {
                   ne_enable = OFF;
                   };
                switch2 = 3;
                }
        else if(sw_enable)
               {
                ++sw;
                sw_trun = 0.0f;
                se_trun = 0.0f;
                if(sw >= sw_max)
                  {
                   sw_enable = OFF;
                   };
                switch2 = 2;
                }
        else if(se_enable)
               {
                ++se;
                se_trun = 0.0f;
                sw_trun = 0.0f;
                if(se >= se_max)
                  {
                   se_enable = OFF;
                   };
                switch2 = 1;
                }
```

```
        else if(nw_enable)
                {
                 ++nw;
                 nw_trun = 0.0f;
                 ne_trun = 0.0f;
                 if(nw >= nw_max)
                   {
                    nw_enable = OFF;
                    };
                 switch2 = 4;
                 }
        else    {
                 ++trun_total;
                 switch2 = 3;
                 };
        }
else if(switch2 == 3)
       {
           if(sw_enable)
             {
              ++sw;
              sw_trun = 0.0f;
              se_trun = 0.0f;
              if(sw >= sw_max)
                {
                 sw_enable = OFF;
                 };
              switch2 = 2;
              }
        else if(ne_enable)
              {
               ++ne;
               ne_trun = 0.0f;
               nw_trun = 0.0f;
               if(ne >= ne_max)
                 {
                  ne_enable = OFF;
                  };
               switch2 = 2;
               }
        else if(sw_enable)
              {
               ++sw;
               sw_trun = 0.0f;
               se_trun = 0.0f;
               if(sw >= sw_max)
                 {
                  sw_enable = OFF;
                  };
               switch2 = 2;
               }
```

```
        else if(nw_enable)
               {
                ++nw;
                if(nw >= nw_max)
                  {
                   nw_enable = OFF;
                   };
                switch2 = 4;
                }
        else {
              ++trun_total;
              switch2 = 2;
              };
        }
  else if(switch2 == 4)
         {
             if(se_enable)
               {
                ++se;
                se_trun = 0.0f;
                sw_trun = 0.0f;
                if(se >= se_max)
                  {
                   se_enable = OFF;
                   };
                switch2 = 1;
                }
        else if(nw_enable)
               {
                ++nw;
                nw_trun = 0.0f;
                ne_trun = 0.0f;
                if(nw >= nw_max)
                  {
                   nw_enable = OFF;
                   };
                switch2 = 4;
                }
        else if(sw_enable)
               {
                ++sw;
                sw_trun = 0.0f;
                se_trun = 0.0f;
                if(sw >= sw_max)
                  {
                   sw_enable = OFF;
                   };
                switch2 = 2;
                }
```

```
      else if(ne_enable)
             {
              ++ne;
              ne_trun = 0.0f;
              nw_trun = 0.0f;
              if(ne >= ne_max)
                {
                 ne_enable = OFF;
                 };
              switch2 = 3;
              }
      else   {
              ++trun_total;
              switch2 = 1;
              };
      }
else;
}
else if((nw_enable) && (nw_trun == 0.75f))
       {
        ++nw;
        if(nw >= nw_max)
          {
           nw_enable = OFF;
           };
        }
else if((se_enable) && (se_trun == 0.75f))
       {
        ++se;
        if(se >= se_max)
          {
           se_enable = OFF;
           };
        }
else if((ne_enable) && (ne_trun == 0.75f))
       {
        ++ne;
        ne_trun = 0.0f;
        if(ne >= ne_max)
          {
           ne_enable = OFF;
           };
        }
else if((sw_enable) && (sw_trun == 0.75f))
       {
        ++sw;
        sw_trun = 0.0f;
        if(sw >= sw_max)
          {
           sw_enable = OFF;
           };
        }
```

```
        else if((sw_enable) && (sw_trun == 0.25f))
               {
                ++sw;
                sw_trun = 0.0f;
                if(sw >= sw_max)
                  {
                   sw_enable = OFF;
                   };
                }
        else if((ne_enable) && (ne_trun == 0.25f))
               {
                ++ne;
                ne_trun = 0.0f;
                if(ne >= ne_max)
                  {
                   ne_enable = OFF;
                   };
                }
        else if((se_enable) && (se_trun == 0.25f))
               {
                ++se;
                se_trun = 0.0f;
                if(se >= se_max)
                  {
                   se_enable = OFF;
                   };
                }
        else if((nw_enable) && (nw_trun == 0.25f))
               {
                ++nw;
                nw_trun = 0.0f;
                if(nw >= nw_max)
                  {
                   nw_enable = OFF;
                   };
                }
        else;

        }while(trun_total);/* end  of 2 truncation
                            ** dots*/
    };

if(trun_total == 3.0f)
  {
     if((pol_dir == 4) && (se_enable) && (ne_enable)
                                      && (sw_enable))
       {
        ++se;
        ++ne;
        ++sw;
        }
   else if((pol_dir == 1)  && (nw_enable) &&
                (ne_enable) && (sw_enable))
       {
        ++nw;
        ++ne;
        ++sw;
        }
```

```
else if((pol_dir == 2) && (ne_enable) &&
                (nw_enable) && (se_enable))
    {
     ++ne;
     ++nw;
     ++se;
     }
else if((pol_dir == 3)  && (sw_enable) &&
              (nw_enable) && (se_enable))
    {
     ++sw;
     ++nw;
     ++se;
     }
else if(switch3 == 3)
    {
       if((sw_enable) && (nw_enable) &&
                              (se_enable))
         {
          ++sw;
          ++nw;
          ++se;
          switch3 = 2;
          }
      else if((ne_enable) && (nw_enable) &&
                                  (se_enable))
         {
          ++ne;
          ++nw;
          ++se;
          switch3 = 3;
          }
      else  if((se_enable) && (ne_enable) &&
                                  (sw_enable))
         {
          ++se;
          ++ne;
          ++sw;
          switch3 = 1;
          }
      else if((nw_enable) && (ne_enable) &&
                                  (sw_enable))
         {
          ++nw;
          ++ne;
          ++sw;
          switch3 = 4;
          }
      else ;

      }
```

```
else if(switch3 == 2)
    {
       if((ne_enable) && (nw_enable) &&
                              (se_enable))
         {
          ++ne;
          ++nw;
          ++se;
          switch3 = 3;
          }
      else if((sw_enable) && (nw_enable) &&
                              (se_enable))
         {
          ++sw;
          ++nw;
          ++se;
          switch3 = 2;
          }
      else if((se_enable) && (ne_enable) &&
                              (sw_enable))
         {
          ++se;
          ++ne;
          ++sw;
          switch3 = 1;
          }
      else  if((nw_enable) && (ne_enable) &&
                              (sw_enable))
         {
          ++nw;
          ++ne;
          ++sw;
          switch3 = 4;
          }
      else;
      }
else if(switch3 == 4)
    {
       if((se_enable) && (ne_enable) &&
                              (sw_enable))
         {
          ++se;
          ++ne;
          ++sw;
          switch3 = 1;
          }
      else if((nw_enable) && (ne_enable) &&
                              (sw_enable))
         {
          ++nw;
          ++ne;
          ++sw;
          switch3 = 4;
          }
```

```
        else if((sw_enable) && (nw_enable) &&
                                    (se_enable))
            {
             ++sw;
             ++nw;
             ++se;
             switch3 = 2;
             }
        else  if((ne_enable) && (nw_enable) &&
                                    (se_enable))
            {
             ++ne;
             ++nw;
             ++se;
             switch3 = 3;
             }
        else;
        }
  else if(switch3 == 1)
      {
        if((nw_enable) && (ne_enable) &&
                                   (sw_enable))
           {
            ++nw;
            ++ne;
            ++sw;
            switch3 = 4;
            }
        else if((ne_enable) && (nw_enable) &&
                                    (se_enable))
            {
             ++ne;
             ++nw;
             ++se;
             switch3 = 3;
             }
        else if((sw_enable) && (nw_enable) &&
                                    (se_enable))
            {
             ++sw;
             ++nw;
             ++se;
             switch3 = 2;
             }
        else  if((se_enable) && (ne_enable) &&
                                    (sw_enable))
            {
             ++se;
             ++ne;
             ++sw;
             switch3 = 1;
             }
        else;

      }
else;
  };/* end of 3 truncation    dots if*/
}; /* end of whole truncation if*/
```

```
    };/* end of dots_number check if*/


     buffer.nw = (float)((int)nw);
     buffer.ne = (float)((int)ne);
     buffer.sw = (float)((int)sw);
     buffer.se = (float)((int)se);

     return buffer;
} /* end of scor_boardf.c function */
```

```
/* prnf.c function prints out a placed image file.
** to the dot matrix printer.
*/

#include  <stdio.h>
#include  <dos.h>
#include  <conio.h>
#include  <ctype.h>
#include  <stdlib.h>
#include  "color.h"

#define  ARRLENGHT  (512 * 16)
#define  ESC        0x1b

extern    void write_str(char *, int, int, int);
extern    char wind_roffset[];
extern    char wind_coffset[];
unsigned  char put_out(unsigned char);
unsigned  char status(void);
unsigned  char prn_buff[ARRLENGHT];

int prnf(char *file_name,int double_strike)
  {
   int i, j, m, n, k, row, col;
   int win_rdim, win_cdim;
   int remainder, division;
   long offset;
   FILE   *prnstream;
   unsigned char temp_buff[16];
   row = 23;
   col =  5;

   win_rdim = atoi(wind_roffset);
   win_cdim = atoi(wind_coffset);

   remainder = (win_rdim * 16) % 256;
   division  = (win_rdim * 16) / 256;

   if((win_rdim > 120) || (win_cdim > 180))
     {
      write_str("exceeding printer  dimensions!",\
                                    row,col,RED);
      getch();
      write_str("exceeding printer  dimensions!",\
                                    row,col,RED);
      return (-1);
      };

  if((prnstream = fopen("place.dat","rb")) == NULL)
    {
     write_str("cannot open the file !", row, col, RED);
     getch();
     write_str("cannot open the file !", row, col, RED);
     return (1);
     };
```

```
/* first get data, then send it to printer */
fprintf(stdprn,"\x01"); /* clear printer buffer */
fputs("\x1b\x41\x02",stdprn); /*set line feed */

for(j = 0; j < (2 * win_cdim); j++)
   {
    if(kbhit())
   {
    write_str("printing canceled !", row, col, RED);
    getch();
    write_str("printing canceled !", row, col, RED);
    fputs("\x1b\x41\x9",stdprn);
    fclose(prnstream);
    return (1);
    };

     for(i = (win_rdim - 1), m = 0; i >= 0; i--, m++)
     {
      offset = 32 * (long)win_cdim * i + j * 16;
      fseek(prnstream,offset,SEEK_SET);
      fread(&temp_buff[0],16,1,prnstream);
      for(k = 0; k < 16; k++)
         {
          prn_buff[(m * 16) + k] = temp_buff[15 - k];
          };
      };
      fputs("\x1b\x5a",stdprn);
      put_out((unsigned char) remainder);
      put_out((unsigned char) division);
      for(n = 0; n < (win_rdim * 16); n++)
         {
          put_out(prn_buff[n]);
          };
      if(double_strike)
        {
         put_out('\r');
         for(i = 0; i < (win_rdim * 16); i++)
          {
          put_out(prn_buff[i]);
          };
         };
       put_out('\r');
       put_out('\n');
    }; /* end of first for loop */
    fclose(prnstream);
    fflush(stdprn);
    fputs("\x1b\x41\x9",stdprn);
    return (0);
} /* end of prnf.c */
```

```
unsigned char put_out (unsigned char character)
            {
             union REGS regs;
             while(! status);
             regs.h.ah = 0;
             regs.h.al = character;
             regs.x.dx = 0;
             int86(0x17,&regs,&regs);
             return (regs.h.ah);
             }

unsigned char status (void)
            {
             union REGS regs;
             regs.h.ah = 2;
             regs.x.dx = 0;
             int86(0x17, &regs, &regs);
             return((regs.h.ah) & (unsigned char)(0x80));
             }
```

```
/* lprnf.c function prints out a placed image file to the
** hp laser printer.
*/

#include  <stdio.h>
#include  <dos.h>
#include  <conio.h>
#include  <ctype.h>
#include  <stdlib.h>
#include  "color.h"

#define  LARRLENGHT  (512 * 2)
#define  ESC         0x1b

extern    void write_str(char *,int,int,int);
extern    char wind_roffset[];
extern    char wind_coffset[];
extern    unsigned char put_out(unsigned char);
extern    unsigned char status(void);


int lprnf(char *file_name)
  {
   int i, j, row, col;
   int win_rdim, win_cdim;
   int k, n;
   long offset;
   FILE   *prnstream;
   unsigned char prn_buff[LARRLENGHT];
   row = 23;
   col =  5;

   win_rdim = atoi(wind_roffset);
   win_cdim = atoi(wind_coffset);

  if((win_rdim > 300) || (win_cdim > 375))
    {
     write_str("exceeding printer dimensions!",\
                                    row,col,RED);
     getch();
     write_str("exceeding printer dimensions!",\
                                    row,col,RED);
     return(-1);
     };


  if((prnstream = fopen("place.dat","rb")) == NULL)
    {
     write_str("cannot open the file !", row, col, RED);
     getch();
     write_str("cannot open the file !", row, col, RED);
     return (1);
     };
```

```
/* first get data, then send it to printer */
fprintf(stdprn,"\x01b\x45");  /* reset printer */
fprintf(stdprn,"\x1b*t300R"); /* set raster graphics
                              ** resulation */
fprintf(stdprn,"\x01b*r0F"); /* landscape printing */
fprintf(stdprn,"\x01b&l1O"); /* landscape orientation*/
fprintf(stdprn,"\x01b*r0A"); /* start raster at left
                             ** marge */


for(i = 0; i < win_rdim; i++)
   {
    if(kbhit())
   {
    write_str("printing canceled !", row, col, RED);
    getch();
    write_str("printing canceled !", row, col, RED);
    fputs("\x1b*rB",stdprn);/* end raster graphics */
    fclose(prnstream);
    return (1);
    };
    for(k = 0; k < 16; k++)
       {
        for(j = 0; j < (win_cdim * 2); j++)
         {
          offset = 32 *  (long)(win_cdim) * i\
                                 +(j * 16) + k;
          fseek(prnstream, offset, SEEK_SET);
          fread(&prn_buff[j],1,1,prnstream);
          };
          fprintf(stdprn,"\x1b*b%iW",(win_cdim * 2));
          for(n = 0; n < (win_cdim * 2); n++)
            {
             put_out(prn_buff[n]);
             };
        };

    }; /* end of first for loop */
    fclose(prnstream);
    fflush(stdprn);
    fputs("\x1b*rB",stdprn); /* end raster graphics */
    fputs("\x00c",stdprn);   /* send form feed */
    return (0);
 } /* end of lprnf.c */
```

```
/* lprnf600.c function prints out a placed image file to
** the 600 dpi hp laser printer.
*/

#include  <stdio.h>
#include  <dos.h>
#include  <conio.h>
#include  <ctype.h>
#include  <stdlib.h>
#include  "color.h"

#define  LARRLENGHT  (512 * 2)
#define  ESC         0x1b

extern    void write_str(char *,int,int,int);
extern    char wind_roffset[];
extern    char wind_coffset[];
extern    unsigned char put_out(unsigned char);
extern    unsigned char status(void);


int lprnf600(char *file_name)
  {
   int i, j, row, col;
   int win_rdim, win_cdim;
   int k, n;
   long offset;
   FILE   *prnstream;
   unsigned char prn_buff[LARRLENGHT];
   row = 23;
   col =  5;

   win_rdim = atoi(wind_roffset);
   win_cdim = atoi(wind_coffset);

  if((win_rdim > 300) || (win_cdim > 375))
    {
     write_str("exceeding printer dimensions!",\
                                    row,col,RED);
     getch();
     write_str("exceeding printer dimensions!",\
                                    row,col,RED);
     return(-1);
     };


  if((prnstream = fopen("place.dat","rb")) == NULL)
    {
     write_str("cannot open the file !", row, col, RED);
     getch();
     write_str("cannot open the file !", row, col, RED);
     return (1);
     };
```

```
/* first get data, then send it to printer */
fprintf(stdprn,"\x01b\x45");      /* reset printer */
fprintf(stdprn,"\01b%-12345X");/*set PJL language*/
fprintf(stdprn,"@PJL SET RESOLUTION=600 CR"); /* set
                                        ** 600dpi*/
fprintf(stdprn,"@PJL SET PAGE PROTECT=OFF CR");
fprintf(stdprn,"@PJL ENTER LANGUAGE=PCL CR"); /*in  PCL
                                       ** language*/
fprintf(stdprn,"\x01b*r0F"); /* landscape printing */
fprintf(stdprn,"\x01b&l1O"); /* landscape orientation*/
fprintf(stdprn,"\x01b*r0A"); /* start raster at left
                             ** marge */


for(i = 0; i < win_rdim; i++)
   {
    if(kbhit())
   {
    write_str("printing canceled !", row, col, RED);
    getch();
    write_str("printing canceled !", row, col, RED);
    fputs("\x1b*rB",stdprn);/* end raster graphics */
    fclose(prnstream);
    return (1);
    };
    for(k = 0; k < 16; k++)
       {
        for(j = 0; j < (win_cdim * 2); j++)
         {
          offset = 32 * (long)(win_cdim)  * i +(j * 16)\
                                                + k;
          fseek(prnstream, offset, SEEK_SET);
          fread(&prn_buff[j],1,1,prnstream);
          };
          fprintf(stdprn,"\x1b*b%iW",(win_cdim * 2));
          for(n = 0; n < (win_cdim * 2); n++)
         {
          put_out(prn_buff[n]);
         };
         };

    }; /* end of first for loop */
    fclose(prnstream);
    fflush(stdprn);
    fputs("\x1b*rB",stdprn); /* end raster graphics */
    fputs("\x00c",stdprn);   /* send form feed */
    return (0);
 } /* end of lprnf600.c */
```

APPENDIX B LAST ALGORITHM IMPLEMENTATION CHANGES

Last Algorithm Implementation Changes

During the implementation of algorithm, some changes have been done at the end. This changes have not be taken account in the given example for explanation simplicity. All changes have been done are as follows :

1. After the diagonal directed dot placement, each common neighbor of related partition has been decreased by the percentage of neighbor weight, instead of by 1.00. With this, placement of a diagonal dot is enforced toward the heavy neighbor.

```
neighbor1 = neighbor1-(neighbor1/(neighbor1 + neighbor2))
neighbor2 = neighbor2-(neighbor2/(neighbor1 + neighbor2))
```

2. If the placement direction is line directional, the placement of a dot has been done with 0.50 fraction of a dot to each of related two partitions. And only the neighbor which is the placement is directed to, decreased by 1.00, instead of decrement of each neighbor by 1.00. With this change, convergence problem vanishes.

3. For to take account the intensity of the pixel which is going to be placed, resultant force of neighbors have been calculated. And then, dots number is subtracted from resultant force dynamicly. Without this mechanism, some time, inside of placed pixel remains empty, even if soft crossing is succeeded.

```
r = sqrt((w_e)*(w_e) + (n_s)*(n_s))
net = r - dots_number
w_e = west - east
n_s = north - south
```

If the net is greater or equal to zero, placement is done according to neighbors encouragement. Other wise, the placement of dot is done with 0.25 fraction of a dot to all partitions.

4. At the end of each division session, some time, fraction part of dots occurs in the some partitions. Since the dot number is integer, summation of all these fractional parts is always an integer. At the end of division session, placement of these dots have been done. During the placement of the fractional part, symmetrical dots distribution is taken as base approach. So, diagonal placement has been taken as first placement process.

5. Maximum dots number of each partition is determined by the neighbors and dot_number variable dynamicly. The process is as follows :

First calculate the intensity of medial point of each side. This intensity is determined by dots_number variable and related neighbor.
west side intensity : w_sd = (west + dots_number)/ 2
east side intensity : e_sd = (east + dots_number)/ 2
north side intensity : n_sd = (north + dots_number)/ 2
south side intensity : s_sd = (south + dots_number)/ 2

Then, subtract the related neighbor's contribution. And calculate the half intensity of each partition :
w_max = (1/2)(w_sd - (west / 4))
e_max = (1/2)(e_sd - (east / 4))
n_max = (1/2)(n_sd - (north / 4))
s_max = (1/2)(s_sd - (south / 4))

Then, calculate the each partition intensity :
nw_tmp = w_max + n_max
ne_tmp = n_max + e_max
sw_tmp = s_max + w_max
se_tmp = s_max + e_max
total = nw_tmp + ne_tmp + sw_tmp + se_tmp

At result, calculate the each partition maximum dots number as percentage of dots number :
nw_max = (nw_tmp / total) * dots_number
ne_max = (ne_tmp / total) * dots_number
sw_max = (sw_tmp / total) * dots_number
se_max = (se_tmp / total) * dots_number

This maximum dots determining process is repeated for all levels of division.

CURRICULUM VITAE

Ayhan Öztürk was born in Varto, in 1962. He graduated from Varto Lisesi, as the best student in his school, in 1981. Then, he graduated from Electronics and Communications Engineering Department of Electrical-Electronics Engineering Faculty of Istanbul Technical University, in 1987. After working with several companies, he started his graduate program in 1990, in ITU. Since 1991, he was research assistant in ITU.