

GÖRÜNTÜ İŞLEME - (3.Hafta)

GEOMETRİK DÖNÜŞÜMLER

Geometrik dönüşümler resim üzerindeki her pikselin bir konumdan (x_1, y_1) başka bir konuma (x_2, y_2) haritalanmasıdır. Bununla ilgili olarak aşağıdaki işlemler kullanılabilir.

- Taşıma (Move): Resim içeriğinin konumunu değiştirme. Kanvas (tuval) üzerinde pikseller hareket ettirilir.
- Aynalama (Mirror): Resmin aynadaki görüntüsüne benzer simetrisini alma, görüntüyü takla attırma.
- Eğme ve Kaydırma (Skew): Resmin bir kenarından tutup kaydırma işlemidir.
- Ölçekleme (Zoom/Scale): Resim boyutunu değiştirme yada yaklaştırma uzaklaştırma.
- Döndürme (Rotate): Resmi çevirme.

Bu temel operatörler için aşağıdaki genel fonksiyonu kullanabiliriz. Bu formülde A matrisi ölçekleme, döndürme ve aynalama işlemlerini formülize ederken, B matrisi taşıma için kullanılır. Perspektif dönüşümlerde ise her iki matris de kullanılır.

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = [A] \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + [B]$$

Geometrik dönüşümler görüntünün belli bir bölgesini yaklaştırma, yada belli bir seri resmin birleştirme için kullanılabilir. Örneğin sırayla çekilmiş resimleri tek bir resim halinde panoramik (geniş açılı resimler) halin getirmek için kullanılabilir.

TAŞIMA (Translate)

Taşıma operatörü, giriş resmindeki her pikseli, çıkış resmindeki yeni bir konuma taşıma işlemidir. Orjinal resimdeki (x_1, y_1) koordinatındaki her piksel belli bir öteleme mesafesi (β_x, β_y) boyunca taşınarak yeni konumu olan (x_2, y_2) koordinatına yerleştirilir. Taşıma işlemi görüntü üzerinde iyileştirmeler yaparken yada başka uygulamaların alt işlemleri olarak kullanılabilir.

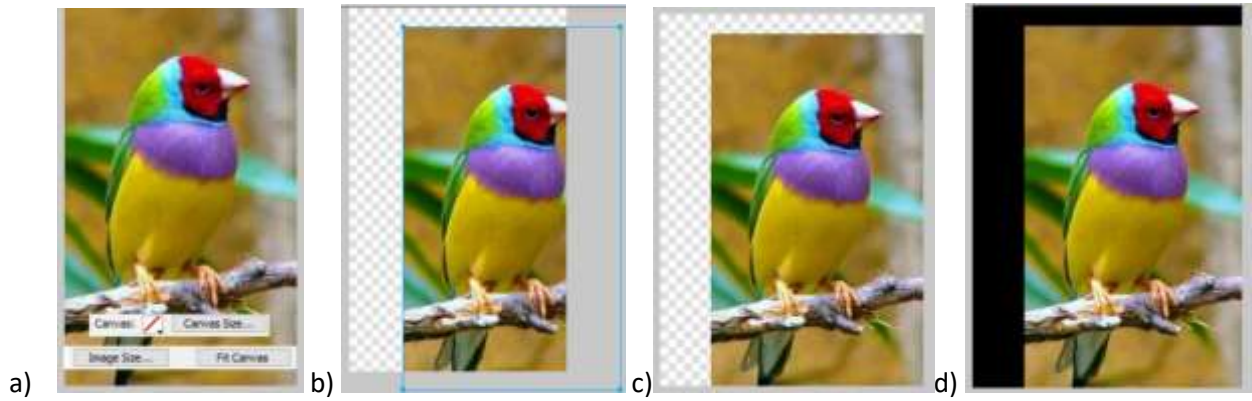
Taşıma operatörü aşağıdaki formüller kullanılarak gerçekleştirilir.

$$\begin{aligned} x_2 &= x_1 + T_x \\ y_2 &= y_1 + T_y \end{aligned}$$

Matris formatında

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}$$

Yeni oluşan koordinatlar (x_2, y_2) resmin sınırları dışına çıktıysa ya yok sayılır veya sınırlar genişletilerek, geride bırakılan boşluk alanları doldurulur. Bu durumda iki tane alandan bahsedebiliriz. Bir tanesi görüntünün bulunduğu resim (image) diğeri ise üzerinde kaydırıldığı zemin yani tuval (canvas) (bkz resim a). Burada resim kaydırıldığında altında kanvas da görünecektir (resim b). Kanvas yeni sınırları da kapsayacak şekilde büyütülürse resmin dışarı çıkan kısımları tekrar görünecektir fakat zeminde kanvas da görünecektir (resim c). Kanvas istenen renge boyanabilir yada renksiz düşünülebilir (resim d).



Programlama



```
public Bitmap Tasima()
{
    Color OkunanRenk;
    Bitmap GirisResmi, CikisResmi;
    GirisResmi = new Bitmap(pictureBox1.Image);
    int ResimGenisligi = GirisResmi.Width;
    int ResimYuksekligi = GirisResmi.Height;
    CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi);

    double x2 = 0, y2 = 0;

    //Taşıma mesafelerini atıyor.
    int Tx = 100;
    int Ty = 50;

    for (int x1 = 0; x1 < (ResimGenisligi); x1++)
    {
        for (int y1 = 0; y1 < (ResimYuksekligi); y1++)
        {
            OkunanRenk = GirisResmi.GetPixel(x1, y1);

            x2 = x1 + Tx;
            y2 = y1 + Ty;

            if (x2 > 0 && x2 < ResimGenisligi && y2 > 0 && y2 < ResimYuksekligi)
                CikisResmi.SetPixel((int)x2, (int)y2, OkunanRenk);
        }
    }
    pictureBox2.Image = CikisResmi;
}
```

AYNALAMA (Mirror/Reflect/Flip)

Yansıtma işlemi, görüntüyü orijinal resimdeki (x_1, y_1) konumundan alarak, belirlenen eksen veya bir nokta etrafında yansıtarak yeni bir konuma (x_2, y_2) yerleştirilmesidir.

Bununla ilgili yapılabilecek dönüşümler şu şekildedir.

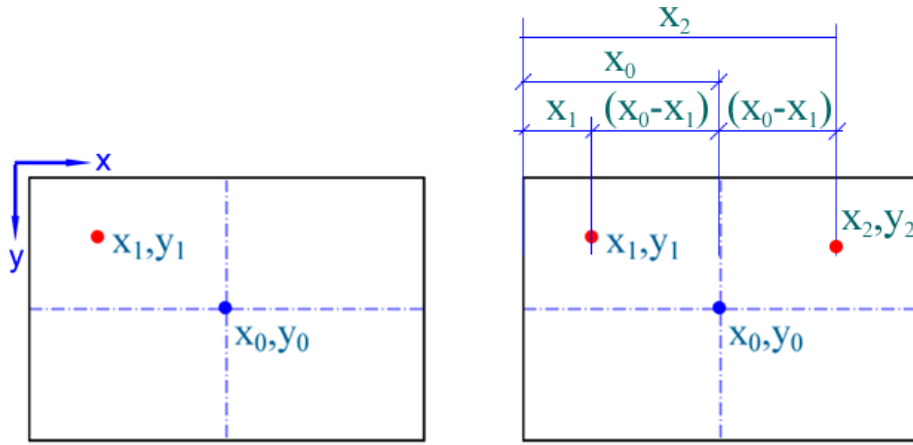
a) Ekranın orta noktasına (x_0, y_0) göre aynalama

Bunun için x_0 mesafesinden x_1 mesafesini çıkarırsak birinci koordinatın eksene uzaklığını buluruz. Bunun iki katını alıp x_1 mesafesini de eklersek x_2 koordinatının mesafesini bulmuş oluruz. y koordinatları değişmeyecektir. Ekranın ortası yerine tıklanan noktayı eksen kabul ederek aynalama da benzer şekilde yapılabilir.

$$x_2 = x_1 + 2(x_0 - x_1)$$

$$x_2 = -x_1 + 2x_0$$

$$y_2 = y_1$$



b) y_0 noktasından geçen yatay eksen etrafında aynalama

Bunun için de benzer mantık kullanılabilir. Bu durumda formüller aşağıdaki gibi olacaktır.

$$x_2 = x_1$$

$$y_2 = -y_1 + 2y_0$$

c) (x_0, y_0) noktasından geçen herhangi bir θ açısına sahip bir eksen etrafında aynalama.

$$\Delta = (x_1 - x_0) \sin\theta - (y_1 - y_0) \cos\theta$$

olmak üzere

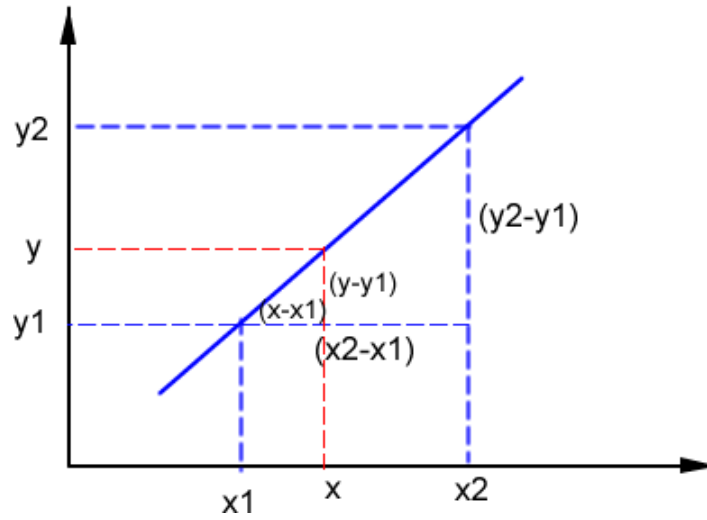
$$x_2 = x_1 + 2 \Delta (-\sin\theta) \text{ yeni x koordinatını verir.}$$

$$y_2 = y_1 + 2 \Delta (\cos\theta) \text{ yeni y koordinatını verir.}$$

Eğer (x_0, y_0) noktası resmin merkezi olmazsa, yansıyan görüntü sınırların dışına çıkacaktır.

d) İki noktası verilen eksen etrafında aynalama: (DÜZENLE-KENDİ TARZINLA ANLAT!)

İki noktası bilinen bir doğrunun şekli aşağıda görülmektedir



Şekil. İki noktası bilinen doğrunun eğimi karşı kenarın komşu kenara oranı ile belirtilir.

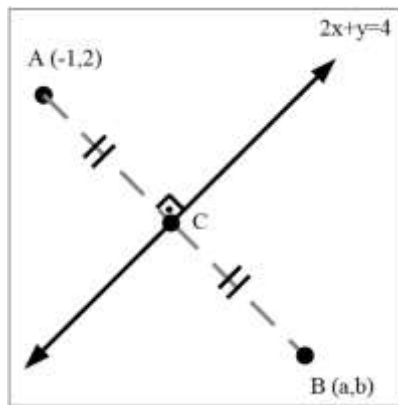
İki noktası bilinen doğrunun eğimin den hareket edersek x ve y değerleri arasındaki bağıntı bu doğrunun eğiminden bulunur. denklemini ifade eder.

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{y - y_1}{x - x_1}$$

İçler dışlar çarpımı yaparak x değerini girdi y değerini çıktı olarak verirsek buna göre doğrunun denklemini aşağıdaki şekilde olacaktır. Bu denklemde sadece x ve y değerleri bilinmiyor. Diğerlerinin hepsi bilinen sayısal bir değerdir.

$$y = \frac{(y_2 - y_1)(x - x_1)}{(x_2 - x_1)} + y_1$$

Örnek: Noktanın doğruya göre simetrisinin bulunması için aşağıdaki şekilde verildiği şekliyle somut bir örnek üzerinde görelim. Buradaki doğrunun denklemini ve A noktasının koordinatları bilinmektedir. Bilinenler ışığında B noktasının koordinatlarını bulmaya çalışalım.



Bir noktanın doğruya göre simetrisinin görünümü

Doğru denkleminde doğrunun eğimi bulunmaktadır. Denklemde y yalnız bırakılır ve x'in katsayısı eğimi ifade etmektedir. Doğrunun eğimi aşağıda verilmiştir.

$$y = 4 - 2x$$

$$m_d = -2$$

A noktasının eğime olan dik uzaklığı ile B noktasının eğime olan dik uzaklığı birbirine eşittir. Bu eşitlik yardımı ile koordinat bilgisine ulaşılabacaktır. A noktası ile B noktası arasındaki uzaklığın orta noktası yani C noktasının koordinatları aşağıda bulunmaktadır.

$$C\left(\frac{a-1}{2}, \frac{b+2}{2}\right)$$

C noktasının x ve y noktaları doğrunun denkleminde yerine yazılırsa aşağıdaki eşitlikler elde edilir.

$$2 \cdot \left(\frac{a-1}{2}\right) + \frac{b+2}{2} = 4$$

$$\frac{2a-2+b+2}{2} = 4$$

$$2a+b=8$$

Birbirine dik doğruların eğimleri çarpımı 1 eşittir. Bu eşitlik ile bir denklem daha türetilcektir. Öncelikle AB doğrusunun eğimi aşağıdaki gibi ifade edilmektedir.

$$m_{AB} = \frac{b-2}{a+2}$$

Doğrunun eğimi ve AB doğrusunun eğimlerinin çarpım ifadesinden aşağıdaki denklem elde edilir.

$$\frac{b-2}{a+2} \cdot -2 = -1$$

$$-2b+4 = -a-1$$

$$a-2b = -5$$

Yukarıdaki ($2a+b=8$) denklemi ile ($a-2b=-5$) eşitliği alt alta toplanarak denklemdaki bilinmeyen (a, b) değerleri bulunur (iki bilinmeyen için iki denklem gerekir).

$$4a+2b=16$$

$$a-2b=-5$$

$$5a=11$$

$$a = \frac{11}{5} \rightarrow b = \frac{18}{5}$$

Bu noktalarda zaten aynalamanın yapılmış olduğu noktalar olmuş olur.

Programlama



Orta dikey eksen etrafında aynalama



Orta yatay eksen etrafında aynalama



45 derecelik eksen etrafında aynalama

```

public Bitmap Aynalama()
{
    Color OkunanRenk;
    Bitmap GirisResmi, CikisResmi;
    GirisResmi = new Bitmap(pictureBox1.Image);

    int ResimGenisligi = GirisResmi.Width;
    int ResimYuksekligi = GirisResmi.Height;

    CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi);

    double Aci = Convert.ToDouble(textBox1.Text);
    double RadyanAci = Aci * 2 * Math.PI / 360;

    double x2 = 0, y2 = 0;

    //Resim merkezini buluyor. Resim merkezi etrafında döndürecek.
    int x0 = ResimGenisligi / 2;
    int y0 = ResimYuksekligi / 2;

    for (int x1 = 0; x1 < (ResimGenisligi); x1++)
    {
        for (int y1 = 0; y1 < (ResimYuksekligi); y1++)
        {
            OkunanRenk = GirisResmi.GetPixel(x1, y1);

            //----A-Orta dikey eksen etrafında aynalama -----
            //x2 = Convert.ToInt16(-x1 + 2 * x0);
            //y2 = Convert.ToInt16(y1);

            //----B-Orta yatay eksen etrafında aynalama -----
            //x2 = Convert.ToInt16(x1);
            //y2 = Convert.ToInt16(-y1 + 2 * y0);

            //----C-Ortadan geçen 45 açılı çizgi etrafında aynalama-----
            double Delta = (x1 - x0) * Math.Sin(RadyanAci) - (y1 - y0) * Math.Cos(RadyanAci);

            x2 = Convert.ToInt16(x1 + 2 * Delta * (-Math.Sin(RadyanAci)));
            y2 = Convert.ToInt16(y1 + 2 * Delta * (Math.Cos(RadyanAci)));

            if (x2 > 0 && x2 < ResimGenisligi && y2 > 0 && y2 < ResimYuksekligi)
                CikisResmi.SetPixel((int)x2, (int)y2, OkunanRenk);
        }
    }
}

```

```

    pictureBox2.Image = CikisResmi;
}

```

EĞME-KAYDIRMA (Shearing)

Resmin bir tarafı sabit dururken diğer tarafının x eksenini ya da y eksenini doğrultusunda kaydırmak için aşağıdaki matrisi kullanabiliriz.

x-ekseni doğrultusunda kaydırmak için;

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & b \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \quad \begin{aligned} x_2 &= x_1 + b y_1 \\ y_2 &= y_1 \end{aligned}$$



$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & -b \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \quad \begin{aligned} x_2 &= x_1 - b y_1 \\ y_2 &= y_1 \end{aligned}$$



y-ekseni doğrultusunda kaydırmak için;

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ b & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \quad \begin{aligned} x_2 &= x_1 \\ y_2 &= b x_1 + y_1 \end{aligned}$$



$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -b & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \quad \begin{aligned} x_2 &= x_1 \\ y_2 &= -b x_1 + y_1 \end{aligned}$$



Ödev: Üst taraftan diğer yönlerde verilmeyen eğilme formüllerini çıkarın.

Programlama



```

public Bitmap Egme_Kaydirma()
{
    Color OkunanRenk;
    Bitmap GirisResmi, CikisResmi;
    GirisResmi = new Bitmap(pictureBox1.Image);

    int ResimGenisligi = GirisResmi.Width;
    int ResimYuksekligi = GirisResmi.Height;

    CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi);

    //Taşıma mesafelerini atıyor.
    double EgmeKatsayisi = 0.2;
    double x2 = 0, y2 = 0;

    for (int x1 = 0; x1 < (ResimGenisligi); x1++)
    {
        for (int y1 = 0; y1 < (ResimYuksekligi); y1++)
        {
            OkunanRenk = GirisResmi.GetPixel(x1, y1);

            // +X eksenini yönünde
            //x2 = x1 + EgmeKatsayisi * y1;
            //y2 = y1;

            // -X eksenini yönünde
            //x2 = x1 - EgmeKatsayisi * y1;
            //y2 = y1;

            // +Y eksenini yönünde
            //x2 = x1;
            //y2 = EgmeKatsayisi * x1 + y1;

            // -Y eksenini yönünde
            x2 = x1;
            y2 = -EgmeKatsayisi * x1 + y1;

            if (x2 > 0 && x2 < ResimGenisligi && y2 > 0 && y2 < ResimYuksekligi)
                CikisResmi.SetPixel((int)x2, (int)y2, OkunanRenk);
        }
    }
    pictureBox2.Image = CikisResmi;
}

```


ÖLÇEKLEME (Scaling)

Ölçekleme işlemi bir görüntünün boyutunu veya bir bölümünü küçültme yada büyültmek için kullanılır. Bu işlem görüntü üzerinde alınan bir grup matrisin pikselin değerini daha küçük yada büyük matrise dönüştürme ile gerçekleştirilir. Böylece matris büyüklüğüne göre ya pikseller çoğaltılır (yaklaştırma), yada azaltılır (uzaklaştırma).

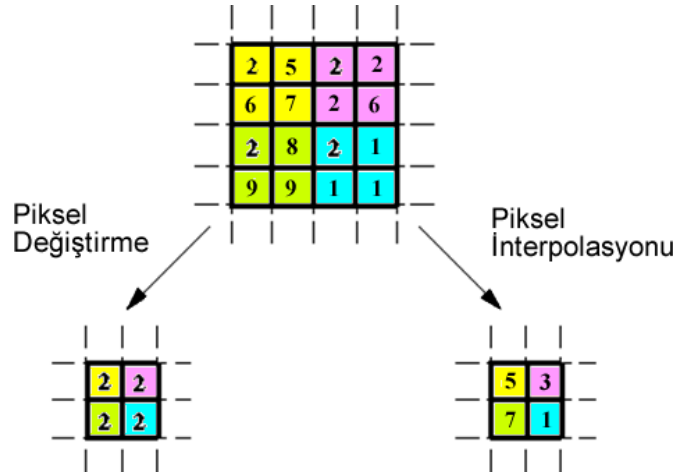
Ölçeklenen resimler büyük resimlerin küçültülerek daha küçük hafızalarda saklanması, yada büyütürken belli bir bölgeye dikkat çekmek için kullanılabilir.

Uzaklaştırma (Küçültme)(Zoom out)

Yeni oluşturulacak görüntüdeki pikselin değeri için iki farklı teknik kullanılabilir.

a) **Piksel değiştirme:** Bir grup komşu pikselin değeri, tek bir piksel değerine dönüştürülürken içlerinden birinin değerine yada rastgele herhangi birinin değerine dönüştürülebilir. Bu metod işlemler açısından daha basittir fakat örneklenen pikseller arasındaki farklılık çok fazla ise zayıf sonuçlara yol açabilir.

b) **Piksel interpolasyonu:** İkinci metod ise komşu pikseller arasında istatistiksel bir örnekleme yaparak (örneğin ortalamasını alarak) oluşturulacak pikselin değerini belirler.



Şekil. Piksel değiştirmede dörtlü çerçevelerin sol üst köşelerindeki değerler kullanılmıştır. İnterpolasyonda ise dört tane pikselin ortalama değeri kullanılmıştır.

Programlama (Küçültme-Piksel Değiştirme Metodu)

Giriş resmini x ve y yönünde birer piksel atlayarak okursak ve okunan değerleri yeni resimde sırayla yerleştirirsek $1/2x$ lik bir küçültme sağlamış oluruz. Döngüdeki x ve y artış değeri küçültme katsayısı olacaktır.



```

public Bitmap Kucultme_DegistirmeMetodu()
{
    Color OkunanRenk;
    Bitmap GirisResmi, CikisResmi;
    GirisResmi = new Bitmap(pictureBox1.Image);

    int ResimGenisligi = GirisResmi.Width;
    int ResimYuksekligi = GirisResmi.Height;
    CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi);

    int x2= 0, y2 = 0; //Çıkış resminin x ve y si olacak.
    int KucultmeKatsayisi = 2;
    for (int x1 = 0; x1 < ResimGenisligi; x1=x1+KucultmeKatsayisi)
    {
        y2 = 0;
        for (int y1 = 0; y1 < ResimYuksekligi; y1 = y1 + KucultmeKatsayisi)
        {
            OkunanRenk = GirisResmi.GetPixel(x1, y1);

            CikisResmi.SetPixel(x2, y2, OkunanRenk);
            y2++;
        }
        x2++;
    }
    pictureBox2.Image = CikisResmi;
}

```

Programlama (Küçültme-İnterpolasyon Metodu)

Giriş resmini x ve y yönünde küçültme katsayısı kadar atlayarak okursak ve her atladığında aradaki piksellerin ortalama değerini alırsak ve bu ortalama değerleri çıkış resmine aktarırsak 1/x lik bir küçültme sağlamış oluruz.



```

public Bitmap Kucultme_InterpolasyonMetodu()
{
    Color OkunanRenk, DonusenRenk;
    Bitmap GirisResmi, CikisResmi;
    int R = 0, G = 0, B = 0;

    GirisResmi = new Bitmap(pictureBox1.Image);
    int ResimGenisligi = GirisResmi.Width; //GirisResmi global tanımlandı.
    int ResimYuksekligi = GirisResmi.Height;

    CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi); //Cikis resminin boyutları
}

```

```

int x2 = 0, y2 = 0; //Çıkış resminin x ve y si olacak.
int KucultmeKatsayisi = 2;

for (int x1 = 0; x1 < ResimGenisligi; x1 = x1 + KucultmeKatsayisi)
{
    y2 = 0;
    for (int y1 = 0; y1 < ResimYuksekligi; y1 = y1 + KucultmeKatsayisi)
    {
        //x ve y de ilerlerken her atlanan pikselleri okuyacak ve ortalama değerini alacak.
        R = 0; G = 0; B = 0;
        try //resim sınırının dışına çıkıldığında hata vermesin diye
        {
            for (int i = 0; i < KucultmeKatsayisi; i++)
            {
                for (int j = 0; j < KucultmeKatsayisi; j++)
                {
                    OkunanRenk = GirisResmi.GetPixel(x1 + i, y1 + j);

                    R = R + OkunanRenk.R;
                    G = G + OkunanRenk.G;
                    B = B + OkunanRenk.B;
                }
            }
        }
        catch { }

        //Renk kanallarının ortalamasını alıyor
        R = R / (KucultmeKatsayisi * KucultmeKatsayisi);
        G = G / (KucultmeKatsayisi * KucultmeKatsayisi);
        B = B / (KucultmeKatsayisi * KucultmeKatsayisi);

        DonusenRenk = Color.FromArgb(R, G, B);
        CikisResmi.SetPixel(x2, y2, DonusenRenk);
        y2++;
    }
    x2++;
}

return CikisResmi;
}

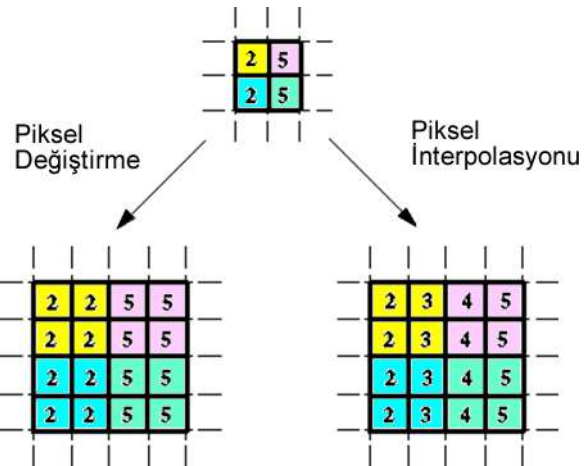
```

Yaklaştırma (Büyültme)(Zoom in)

Bir görüntünün tamamı yada belli bir bölgesi büyütülürken benzer bir işlemle, tek piksel çoğaltma yada interpolasyon olarak çoğaltma kullanılabilir.

a) Piksel Değiştirme: Bu yöntemde ele alınan pikselin değeri değiştirilmeden istenen matris büyüklüğüne göre çoğaltılır. Matris büyüklüğü burada Ölçekleme skalasına bağlı olarak değişir.

b) Piksel İnterpolasyonu: Komşu piksellerin değerleri arasında fark, oluşturulacak piksel sayısına bağlı olarak kademeli şekilde belirlenebilir (interpolasyon işlemi). Böylece piksel geçişleri daha az bir keskinliğe sahip olmuş olur.



Şekil. Yakınlaştırma yöntemleri. a) Piksel değeri ile çoğaltma, b) İnterpolasyon ile çoğaltma.

Programlama



```
private void button1_Click(object sender, EventArgs e)
{
    Color OkunanRenk;
    Bitmap GirisResmi, CikisResmi;
    GirisResmi = new Bitmap(pictureBox1.Image);

    int ResimGenisligi = GirisResmi.Width;
    int ResimYuksekligi = GirisResmi.Height;

    CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi);

    int BuyutmeKatsayisi = Convert.ToInt32(textBox1.Text);

    int x2 = 0, y2 = 0;

    for (int x1 = 0; x1 < ResimGenisligi; x1++)
    {
        for (int y1 = 0; y1 < ResimYuksekligi; y1++)
        {
            OkunanRenk = GirisResmi.GetPixel(x1, y1);

            for (int i = 0; i < BuyutmeKatsayisi; i++)
            {
                for (int j = 0; j < BuyutmeKatsayisi; j++)
                {
                    x2 = x1 * BuyutmeKatsayisi + i;
                    y2 = y1 * BuyutmeKatsayisi + j;
                }
            }
        }
    }
}
```

```

        if (x2 > 0 && x2 < ResimGenisligi && y2 > 0 && y2 < ResimYuksekligi)
            CikisResmi.SetPixel(x2 , y2 , OkunanRenk);

    }

}

pictureBox2.Image = CikisResmi;

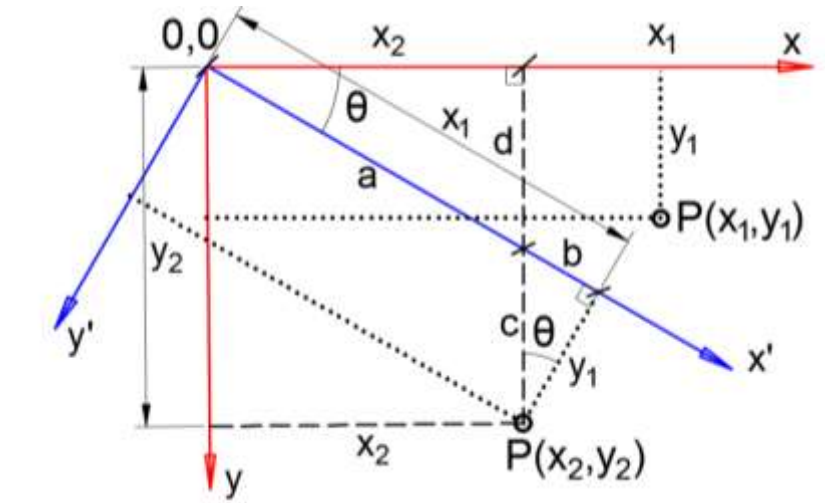
```

DÖNDÜRME (Rotation)

Döndürme işlemi bir nokta etrafında belli bir açı (θ) değerinde çevirerek giriş resmindeki (x_1, y_1) koordinatını çıkış resmindeki (x_2, y_2) noktasına taşıma işlemidir. Çoğu döndürme işleminde sınırların dışına çıkan kısımlar yok sayılır.

Döndürme işlemi çoğunlukla resmin görünümü daha iyi hale getirmek için kullanılabilir. Afin işlemlerinde (perspektif bakış) de kullanılan bir işlem olacaktır.

Döndürme işlemi için formüllerimizi çıkaralım. Bir orijin $(0,0)$ etrafında x - y eksen takımını saat yönünde θ açısı kadar döndürdüğümüzde şekildeki gibi olacaktır. Burada x_1 ve y_1 değerleri ile θ açısı bilenen değerlerdir. x_2 ve y_2 değerlerini bulmaya çalışalım. θ açısının bulunduğu üçgenlerin kenarlarına a, b, c, d sembollerini atarsak aşağıdaki formüllerimiz çıkacaktır.



$$\tan \theta = \frac{d}{x_2} \rightarrow d = x_2 \tan \theta \rightarrow d = x_2 \frac{\sin \theta}{\cos \theta}$$

$$\tan \theta = \frac{b}{y_1} \rightarrow b = y_1 \tan \theta \rightarrow b = y_1 \frac{\sin \theta}{\cos \theta}$$

$$\sin \theta = \frac{d}{a} \rightarrow a = \frac{d}{\sin \theta} \rightarrow a = \frac{(x_2 \frac{\sin \theta}{\cos \theta})}{\sin \theta} \rightarrow$$

$$a = x_2 \frac{1}{\cos \theta}$$

$$x_1 = a + b \rightarrow x_1 = x_2 \frac{1}{\cos \theta} + y_1 \frac{\sin \theta}{\cos \theta} \rightarrow$$

$$x_2 \frac{1}{\cos \theta} = x_1 - y_1 \frac{\sin \theta}{\cos \theta} \rightarrow$$

$$\boxed{x_2 = x_1 \cos \theta - y_1 \sin \theta}$$

$$y_2 = c + d \rightarrow y_2 = y_1 \frac{1}{\cos \theta} + x_2 \frac{\sin \theta}{\cos \theta} \rightarrow$$

$$\begin{aligned}
\sin\theta &= \frac{b}{c} \rightarrow c = \frac{b}{\sin\theta} \rightarrow c = \frac{(y_1 \frac{\sin\theta}{\cos\theta})}{\sin\theta} \rightarrow \\
c &= y_1 \frac{1}{\cos\theta}
\end{aligned}
\quad
\begin{aligned}
y_2 &= y_1 \frac{1}{\cos\theta} + (x_1 \cos\theta - y_1 \sin\theta) \frac{\sin\theta}{\cos\theta} \rightarrow \\
y_2 &= y_1 \frac{1}{\cos\theta} + x_1 \sin\theta - y_1 \frac{\sin^2\theta}{\cos\theta} \rightarrow \\
y_2 &= x_1 \sin\theta + y_1 \left(\frac{1}{\cos\theta} - \frac{\sin^2\theta}{\cos\theta} \right) \rightarrow \\
y_2 &= x_1 \sin\theta + y_1 \left(\frac{\cos^2\theta}{\cos\theta} \right) \rightarrow \\
\boxed{y_2} &= \boxed{x_1 \sin\theta + y_1 \cos\theta}
\end{aligned}$$

Bu iki denklemi matris formunda yazarsak, orijin etrafında (resmin sol üst köşesi) bir noktanın dönüşü şu şekilde olur.

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

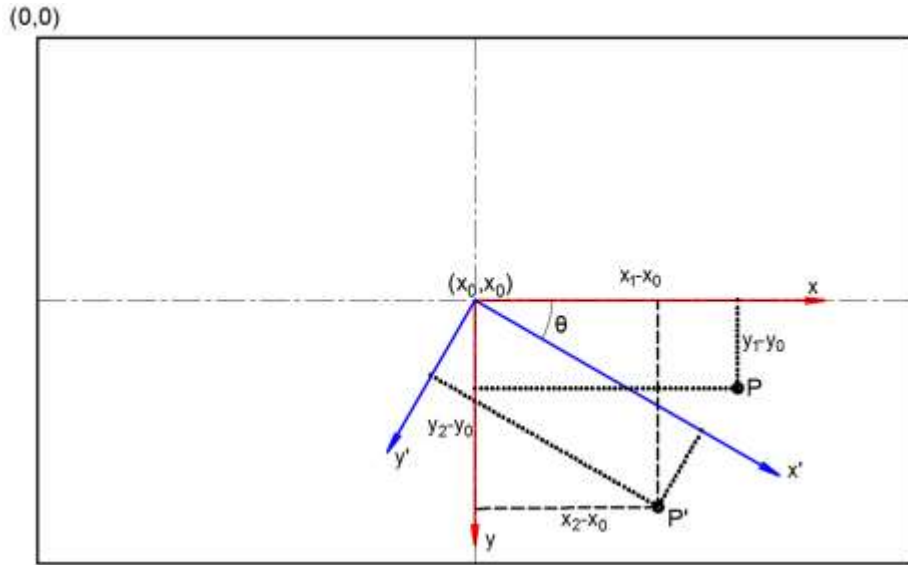
Pozitif θ açısı saatin yönünde dönmeyi ifade etmektedir. Resim alanı içerisindeki görüntü döndürülürken döndürme noktası sol üst köşe yerine resmin ortasını almak gerekir. Bu durumda koordinatlardan resmin orta noktasını gösteren x_0 ve y_0 değerlerini çıkarmak gerekir. Yani ilk işlemde orijin resmin ortasındaymış gibi düşünüp, dönme işlemi gerçekleştikten sonra, resmi sol üst köşeye taşırsak x_0 ve y_0 değerlerini çıkarırız. Resimlerde y koordinat eksenini aşağıya doğru bakmaktadır. Buna göre kullanacağımız formülleri çıkarırsak, şu şekilde olacaktır.

$$\begin{aligned}
x_2 &= x_1 \cos\theta - y_1 \sin\theta \\
(x_2 - x_0) &= (x_1 - x_0) \cos\theta - (y_1 - y_0) \sin\theta \\
\boxed{x_2} &= \boxed{(x_1 - x_0) \cos\theta - (y_1 - y_0) \sin\theta + x_0}
\end{aligned}$$

$$\begin{aligned}
y_2 &= x_1 \sin\theta + y_1 \cos\theta \\
(y_2 - y_0) &= (x_1 - x_0) \sin\theta + (y_1 - y_0) \cos\theta \\
\boxed{y_2} &= \boxed{(x_1 - x_0) \sin\theta + (y_1 - y_0) \cos\theta + y_0}
\end{aligned}$$

Bu iki denklemi matris formunda yazarsak, resmin ortasındaki (x_0, y_0) noktası etrafında, (x_1, y_1) noktasını çevirdiğimizde (x_2, y_2) yeni konumuna şu formülle döndürülür.

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} (x_1 - x_0) \\ (y_1 - y_0) \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$



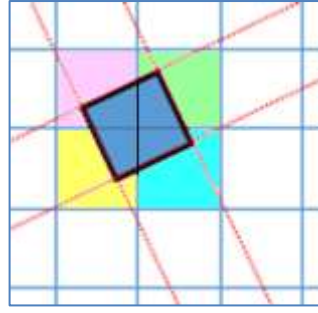
Döndürme işlemi nedeniyle resmin orjinal boyutunun dışına çıkan kısımlar yok sayılabilir yada bu kısımlar siyah renge dönüştürülebilir.

Bu formüller kullanıldığında x_2 ve y_2 koordinat değerleri, formül içindeki Sin ve Cos değerlerinden dolayı ondalık sayı olarak çıkacaktır. Bu durumda dönmüş olan piksel değeri, resim üzerindeki yatay ve dikey konumda olan piksel ızgarasının üzerine tam oturmayacaktır. Aynı anda birkaç pikselin üzerine basacaktır. Eğer en yakın tam sayı değerine yuvarlatılsa bile bazı noktalara renk ataması yapılamayacaktır. Bazı pikseller çift adreslenirken bazıları kaçırılacaktır. Bu durumda resim üzerinde boşluklar gözükcektir. Bu bozucu duruma **alias** (aliasing) denir.



Bu sorunu çözmek için değişik yöntemler kullanılabilir. Bunlardan üç tanesi şu şekildedir. Başka yöntemlerde geliştirilebilir.

- Kaynak resim üzerindeki pikseller aşırı örneklenerek büyütülebilir. Yani her piksel aynı renkte olmak üzere $n \times n$ lik küçük bir matris şeklinde (örn: 2×2) **büyütülebilir**. Bu haliyle resim döndürülür ve **ardından küçültülürse** aradaki boşluklar kaybolacaktır.
- İkinci yöntemde ise sorunu tersine çevirebiliriz. Hedef piksel üzerine basan 4 tane kaynak pikselin hangisi ağırlıkça en fazla yer işgal ediyorsa onun rengi atanabilir. Ayrıca bu algoritmaya üzerine basan 4 tane pikselin alan ağırlılığı ile doğru orantılı olacak şekilde ortalama bir renk değeri atanabilir. Bu durum daha pürüzsüz bir görüntünün oluşmasını sağlayacaktır fakat hesaplama zamanı artacaktır.



- c) Üçüncü yöntem ise daha ilginç bir yöntemdir. Bu yöntemde tüm pikseller önce yatay olarak sağa doğru kaydırılır. Daha sonra dikey olarak aşağı doğru kaydırılır. Sonra tekrar sağa doğru yatayda kaydırılırsa resim dönmüş olarak gözükcektir.



Sağa doğru kaydırma;

$$x_2 = (x_1 - x_0) - (y_1 - y_0) \tan (\theta / 2) + x_0$$

$$y_2 = (y_1 - y_0) + y_0$$

Aşağı kaydırma;

$$x_2 = (x_1 - x_0) + x_0$$

$$y_2 = (x_1 - x_0) \sin \theta + (y_1 - y_0) + y_0$$

Tekrar Sağa doğru kaydırma;

$$x_2 = (x_1 - x_0) - (y_1 - y_0) \tan (\theta / 2) + x_0$$

$$y_2 = (y_1 - y_0) + y_0$$

Bu işlemleri matris formatında gösterirsek topluca şu şekilde olacaktır.

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & -\tan (\theta / 2) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \sin \theta & 1 \end{bmatrix} \begin{bmatrix} 1 & -\tan (\theta / 2) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 - x_0 \\ y_1 - y_0 \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

Araştırma: Buradaki Kaydırma işlemi ile Döndürme işlemi arasındaki bağlantıların nasıl bulunduğunu çıkarın.

Programlama (Alias düzeltme yok)

Alias düzeltilmeden sadece resmi döndürmeye ait kodlar aşağıdadır. Resim üzerinde farklı açılardaki alias oluşumları gösterilmiştir.



```

public Bitmap Dondurme()
{
    Color OkunanRenk;
    Bitmap GirisResmi, CikisResmi;
    GirisResmi = new Bitmap(pictureBox1.Image);

    int ResimGenisligi = GirisResmi.Width;
    int ResimYuksekligi = GirisResmi.Height;

    CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi);

    int Aci = Convert.ToInt16(textBox1.Text);
    double RadyanAci = Aci * 2 * Math.PI / 360;

    double x2 = 0, y2 = 0;

    //Resim merkezini buluyor. Resim merkezi etrafında döndürecek.
    int x0 = ResimGenisligi / 2;
    int y0 = ResimYuksekligi / 2;

    for (int x1 = 0; x1 < (ResimGenisligi); x1++)
    {
        for (int y1 = 0; y1 < (ResimYuksekligi); y1++)
        {
            OkunanRenk = GirisResmi.GetPixel(x1, y1);

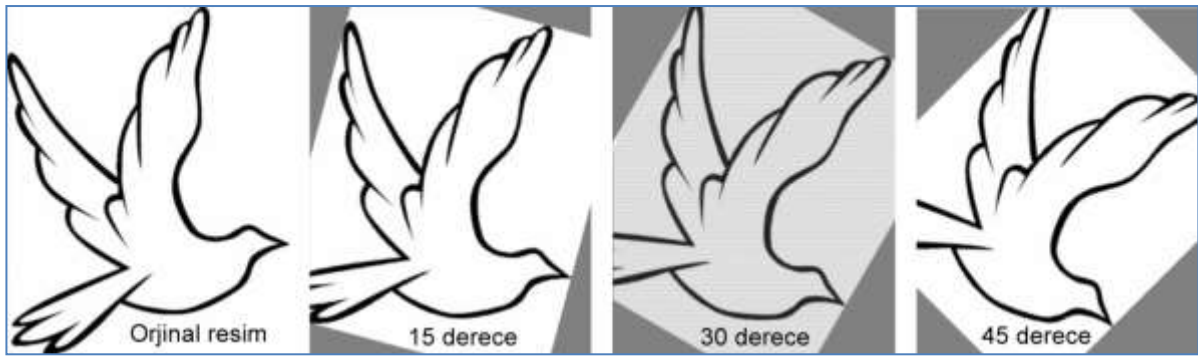
            //Döndürme Formülleri
            x2 = Math.Cos(RadyanAci) * (x1 - x0) - Math.Sin(RadyanAci) * (y1 - y0) + x0;
            y2 = Math.Sin(RadyanAci) * (x1 - x0) + Math.Cos(RadyanAci) * (y1 - y0) + y0;

            if(x2>0 && x2<ResimGenisligi && y2>0 && y2<ResimYuksekligi )
                CikisResmi.SetPixel((int)x2, (int)y2, OkunanRenk);
        }
    }
    pictureBox2.Image = CikisResmi;
}

```

Programlama (Alias düzeltme, kaydırma yöntemi ile)

Kaydırma yöntemi alias hatası düzelse de 30 derecede hala sorun devam etmektedir. Ondalık sayıların tam sayıya çevrilmesi bu derecede hataya neden olmaktadır.



```

public Bitmap Dondurme_Alias()
{
    Color OkunanRenk;
    Bitmap GirisResmi, CikisResmi;
    GirisResmi = new Bitmap(pictureBox1.Image);

    int ResimGenisligi = GirisResmi.Width;
    int ResimYuksekligi = GirisResmi.Height;

    CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi);

    int Aci = Convert.ToInt16(textBox1.Text);
    double RadyanAci = Aci * 2 * Math.PI / 360;

    double x2 = 0, y2 = 0;

    //Resim merkezini buluyor. Resim merkezi etrafında döndürecek.
    int x0 = ResimGenisligi / 2;
    int y0 = ResimYuksekligi / 2;

    for (int x1 = 0; x1 < (ResimGenisligi); x1++)
    {
        for (int y1 = 0; y1 < (ResimYuksekligi); y1++)
        {
            OkunanRenk = GirisResmi.GetPixel(x1, y1);

            //Aliaslı Döndürme -Sağa Kaydırma
            x2 = (x1 - x0) - Math.Tan(RadyanAci / 2) * (y1 - y0) + x0;
            y2 = (y1 - y0) + y0;

            x2 = Convert.ToInt16(x2);
            y2 = Convert.ToInt16(y2);

            //Aliaslı Döndürme -Aşağı kaydırma
            x2 = (x2 - x0) + x0;
            y2 = Math.Sin(RadyanAci) * (x2 - x0) + (y2 - y0) + y0;

            x2 = Convert.ToInt16(x2);
            y2 = Convert.ToInt16(y2);

            //Aliaslı Döndürme -Sağa Kaydırma
            x2 = (x2 - x0) - Math.Tan(RadyanAci / 2) * (y2 - y0) + x0;
            y2 = (y2 - y0) + y0;

            x2 = Convert.ToInt16(x2);
            y2 = Convert.ToInt16(y2);
        }
    }
}

```

```

        if (x2 > 0 && x2 < ResimGenisligi && y2 > 0 && y2 < ResimYuksekligi)
            CikisResmi.SetPixel((int)x2, (int)y2, OkunanRenk);
        }
    }
    pictureBox2.Image = CikisResmi;
}

```

Araştırma: Yukarıda Alias düzeltme yapıldığı halde 30 derece neden hala boşlukların çıktığını araştırın.

KIRPMA

Kırpma işleminin mantığını ve algoritmasını kendiniz geliştirin. Ödev olarak istenecektir.

----- 000 -----

Haftalık Ödev: Aşağıda verilen Ö1, Ö3 ve Ö4 ödevlerinin hepsinin uygulamasını yapınız.

----- 000 -----

Ödev 1: a) **Tıklayarak Resmi Taşıma:** Resim üzerine bir noktaya tıklansın. Ardından ikinci noktaya tıkladığında resmin o noktası yeni yerine gelecek şekilde resmi taşıyın. Tıklanan noktaların gözle görülebilmesi için resim üzerinde küçük bir kare çizdirin. (PictureBox üzerinde çizgi nasıl çizdirilir öğrenmek için İnt.Tab.Prg 4. Notlara bakın)

b) **Basılı Tutarak Resmi Taşıma:** Bu işlemin aynısı mouse basılı iken de yapılabilir. Mouse basılı iken her hareket ettiği nokta alınıp resim sürekli mouse takip edecek şekilde taşınabilir.

c) **Formül Çıkarma:** Aynalama komutlarından Resmin orta noktasına göre (x0,y0) belli bir açıda (θ) aynalarken kullanılan aşağıdaki formüllerin çıkarılışını bulunuz. (Bilenler x1, y1, x0, y0 dır. Bulunacak olan x2, y2) (Örnek olarak döndürme işlemlerinin içindeki formül çıkarma yöntemini kullanabilirsiniz). Bu formüllerin doğruluğunu kontrol ediniz, nasıl çalıştığını gösteriniz.

(x0,y0) noktasından geçen herhangi bir θ açısına sahip bir eksen etrafında aynalama.


$$\Delta = (x_1 - x_0) \sin\theta - (y_1 - y_0) \cos\theta$$

olmak üzere

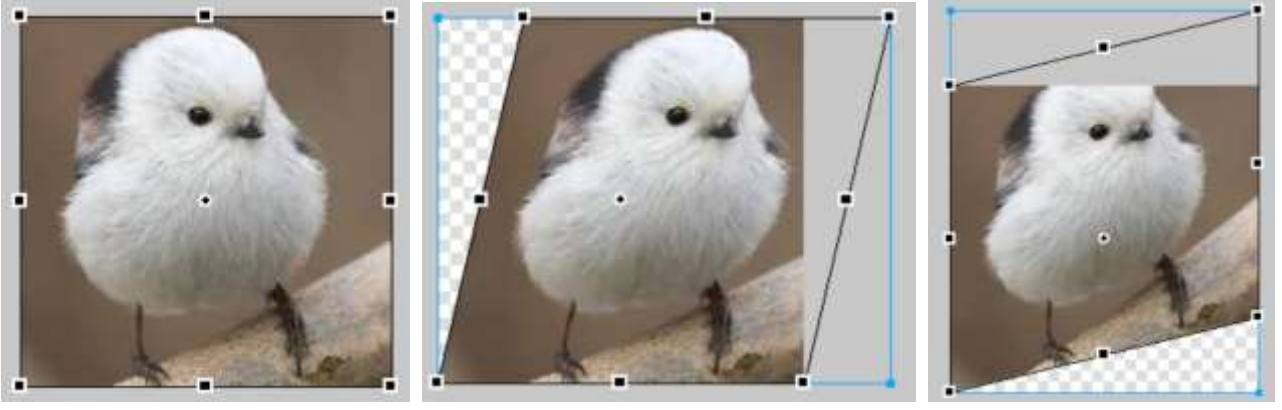
$$x_2 = x_1 + 2 \Delta (-\sin\theta) \text{ yeni x koordinatını verir.}$$

$$y_2 = y_1 + 2 \Delta (\cos\theta) \text{ yeni y koordinatını verir.}$$

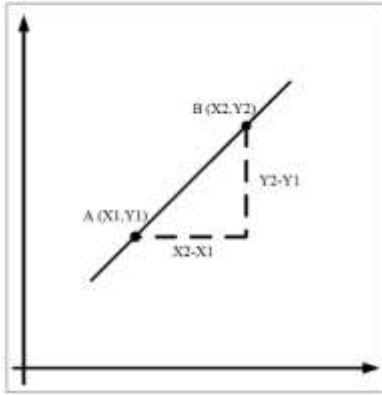
Eğer (x0,y0) noktası resmin merkezi olmazsa, yansıyan görüntü sınırların dışına çıkacaktır.

Ödev 2: Kaydırma komutuna  tıkladığında resmin köşelerinden tutulup çekebilecek şekilde kaydırma işlemi yapın. Örnek görünümü aşağıdakine benzer olsun. Burada olduğu gibi tüm köşelerden tutmak yerine üst kenara yakın bölgede mouse basılı iken yana doğru sürükleme yaparsak, resim sağa doğru kaysın. Yan kenara yakın bir bölgede mouse basılı iken aşağıda doğru sürüklersem resim aşağı doğru kaysın. Bu şekilde 4 kenar üzerinde de bunu yapabilelim. Her iki yönde kayacak şekilde ayarlayabilelim.

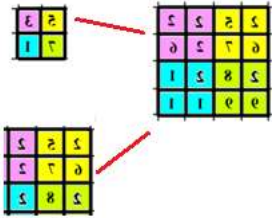
Bir resim yana kaydırıldıktan sonra o resim tekrar aynı pictureBox üzerinde gözükmeli. Yani yan tarafta değil hepsi pictureBox1 üzerinde olmalı. Böylece tekrardan eğik resim üzerinde ikinci bir kaydırma işlemi yapılabilir. Fakat bu esnada dışarı taşan piksellerin bilgisi kaybolacağından bunları kaybetmemek için tüm resim üzerindeki piksel değerlerinin hem koordinatları hemde renk bilgileri dizide tutulabilir.



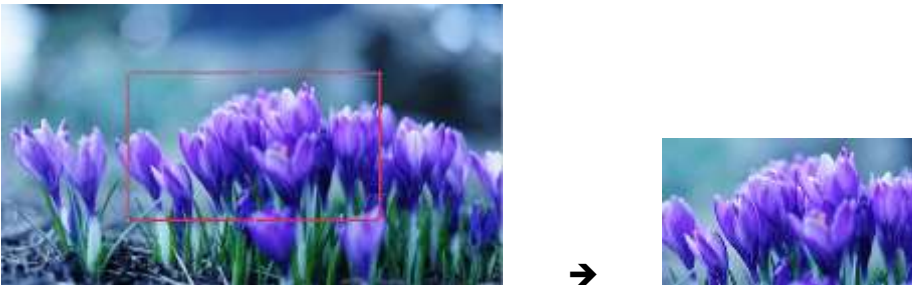
Ödev 3: Tıklanan iki noktayı eksen kabul eden çizginin etrafında aynalama yapan programı yazınız. Formüller ders notları içerisinde vardır.



Ödev 4: Küçük bir resmi belli oranlarda büyütebilen bir program yazınız. Resim büyütme esnasında aralarda oluşan piksel boşlukları (alias) için çözümde bulmaya çalışın.

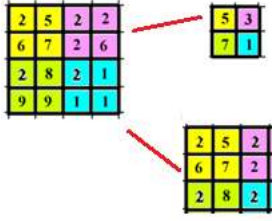


Ödev 5: Resim üzerinde mouse ile tıklanan iki noktanın arasında oluşan dikdörtgeni Kırpma aracı olarak kullanan programı yazınız. Yani çizilen dikdörtgenin dışındaki kısımlar ikinci resimde olmayacak.




----- 000 -----

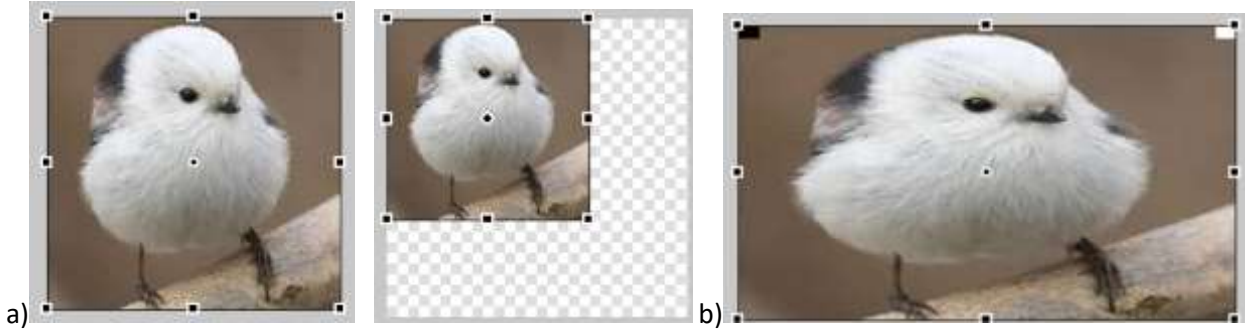
Ödev 6: Bir resmi yukarıda anlatıldığı gibi yarı yarıya küçültme değil belli oranlarda küçültme yapabilecek programı yazınız. %70, %30 gibi oranlarda da küçültme yapabilsin.



Ödev 7: Resmin ortasında tıklanan herhangi bir nokta etrafında textbox'tan girilen açı kadar çevirme yapan kodları yazın. Resmin ortasında tıklanan yerde küçük bir artı şeklinde işaret çizdirin. Dönme işleminden sonra bu işaret kalkmalı.

Ödev 8: Ölçekleme komutuna  tıklandığında resmin köşelerinde küçük kutucuklar oluşsun.

- Mouse ile bu noktadan basarak sürüklenip bırakıldığında, bırakılan nokta ile köşe arasında resim küçülmüş olsun. Bu esnada orantısı bozulmasın. Aynı işlemi büyütme içinde yapın.
- Köşe yerine kenarların ortasındaki noktalardan çekildiğinde resim yatay / dikey uzasın (orantısı bozulsun).



Ödev 9: Ölçekleme işlemlerinden büyültme işlemi hem "Piksel Değiştirme" metodu ile hem de "Piksel interpolasyon" metodu ile programlayın. Hangisinin daha iyi bir sonuç verdiğini görmek için print-screen ile alarak Paintte büyütüp inceleyin. Yanyana getirip kenarları karşılaştırın.

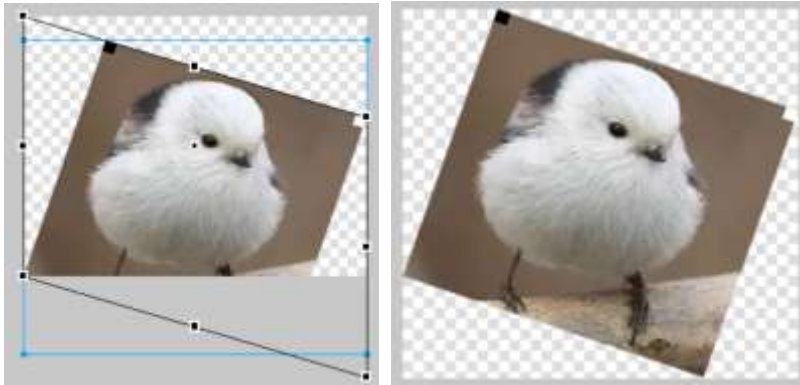
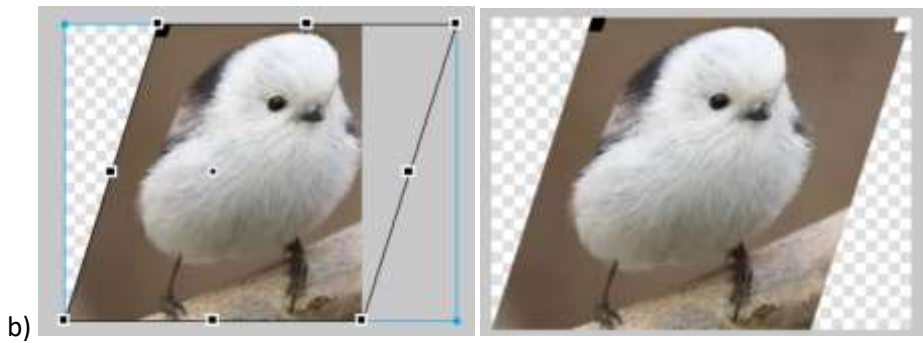
Ödev 10: Mouse ile bir resmi döndürme işlemi yaptırın. Resmin üzerinde mouse sol tuşuna basılı tutarken ve mouse ile döndürme hareketi yaparken resmin 4 kenarını çizgi şeklinde ve mouse takip edecek şekilde Orta nokta etrafında çizdirin. Mousdan elimizi çektiğimiz anda resim gösterilen en son dikdörtgenin içerisinde dönmüş olarak gözüksün. Bu esnada picturbox ın dışına taşan kısımlar gözükmeyecektir.

Ödev 11: Resim döndürme işlemi açı olmadan Kaydırma yöntemini kullanarak programlayın. (Köşelerden 50 piksel kaydırak dönder)(Hem saat yönünde, hemde tersi yönde çalışan komutlar geliştirin)

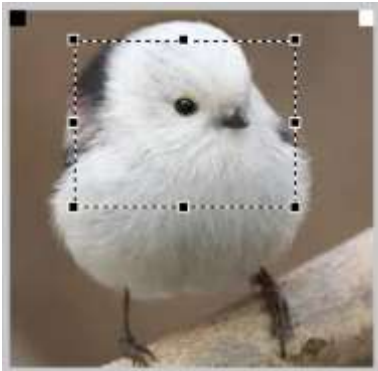
Ödev 12: a) Döndürme komutunu programlarken Geometrik dönüşümle programlayın. Oluşan Alias boşluklarını yok etmek için resim oluştuktan sonra siyah noktaların etrafındaki pikselleri okutun ve renk olarak etrafındaki piksellerin ortalamasını alacak şekilde programlayın. Bu şekilde döndürme nedeniyle oluşan hatayı yok etmeye çalışın. Eğer Aliasları yok etmek için iyi bir algoritma bulursanız büyük harflerle anlatın.

- Döndürme işlemi Yana ve aşağı kaydırarak programlayın. Notlarda bu konuda hazır formül verilmiştir fakat bunlar resmin ortasına göre kaydırma yapıyor. Siz bunu kullanmayacaksınız. Resmin köşesinden

eğerek döndürme yapacak. Bu konu notlar içinde anlatılmıştı. Burada yana ve aşağı kaydırırken oluşan açı yı geliştireceğiniz formüller içinde kullanın. Yani açı verildiği zaman köşeden itibaren eğerek tam o açı kadar dönmeyi sağlayacak yöntemi yada formülü bulun. Resmin dışına çıkan kısımların kaybolmamasına dikkat edin. O problemi de çözün. Geliştirdiğiniz önemli uygulamaları büyük harflerle anlatın.



Ödev 13: Kırpma işlemini yapan kodları yazınız. Mouse takip eden çizgi çizdirerek yapın...



GÖRÜNTÜ İŞLEME - (4.Hafta)

PERSPEKTİF DÜZELTME

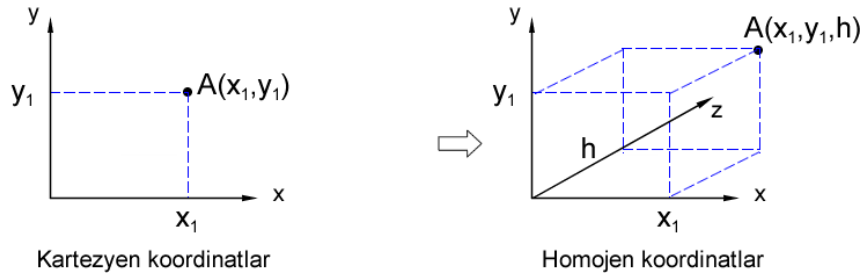
Perspektif nesnenin bulunduğu konuma bağlı olarak, gözlemcinin gözünde bıraktığı etkiyi (görüntüyü) iki boyutlu bir düzlemde canlandırmak için geliştirilmiş bir iz düşüm tekniğidir. Perspektif düzeltmede amaç kişinin veya nesnenin konum değiştirmesi sonucu oluşacak etkiyi düzeltmektir. Bu işlem sayesinde görüntü oluştuktan sonra dahi belirli kısıtlar içerisinde resme baktığımız açıyı değiştirebiliriz. Örneğin, perspektif olarak çekilmiş bir resmi, tam karşıdan çekilmiş gibi düzeltme yapabiliriz, yada çekilen kamera açısını farklı bir noktaymış gibi ayarlayabiliriz. Algoritma karakter tanıma uygulamalarında çekilmiş bir görüntüyü belirli kalıplar içerisinde oturmada, plaka tanıma, yüz tanıma gibi uygulamalarda normalizasyon sırasında sıklıkla kullanılmaktadır.

Konuya geçmeden önce dönüşüm matrislerinde işlemi kolaylaştıran "Homojen koordinatlar" konusunu bir inceleyelim.

Homojen Koordinatlar

Homojen koordinat sistemine göre, cisimler üzerinde uygulanan geometrik dönüşümlerin arka arkaya yapılabilmesini matematiksel anlamda çok esnek hale getirmektedir. Cisimlerin koordinatları matris formunda yazılarak çarpım notasyonuna tabi tutulur ve cismin yeni şekli veya konumu kolaylıkla elde edilmiş olur.

Kartezyen koordinat sisteminde bulunan bir cismin koordinatları $A(x_1, y_1)$ şeklindedir. Bu koordinatlara üçüncü bir bileşen eklenirse boyut artımına sebep olur ki, oluşan yeni sistem homojen koordinat sistemi adını alır.



Homojen koordinat sistemi yukarıda da bahsedildiği gibi, iki boyutlu kartezyen koordinat sistemine üçüncü bir boyut (z) eklenmesiyle oluşturulur. $A(x_1, y_1)$ koordinatları artık $A(x_1, y_1, h)$ üçlüsüne dönüşür. Dönüşüm için homojen koordinatlar bir h değeri ile çarpılır yada bölünür. Bu sayede iki koordinat sistemi arasında bir ilişki oluşur.

Araştırma: Homojen koordinatlara neden ihtiyaç duyuyoruz. Bu koordinat sistemi olmadan matris tersini alamaz mıyız. Araştırın?

Bu ilişkide kullanılan üçüncü koordinat bileşeni olan z, genel olarak h ile ifade edilir ve 1 alınır. Çünkü iki boyutlu eksenlerdeki bir noktanın, homojen koordinat ekseninde sonsuz gösterimi olacaktır. Örneğin kartezyen koordinat sisteminde $(x=2, y=3)$ koordinatlarına sahip bir cismin, Homojen koordinattaki gösterimi aşağıdaki gibi sonsuz sayıda olabilir.

$$\text{Kartezyen gösterim} = \text{Homojen gösterim}$$

$$A(x_1, y_1) = A(x_1, y_1, h)$$

$$A(2, 3) = A(2, 3, 1) \quad (h=1)$$

$$A(2, 3) = A(4, 6, 2) \quad (h=2)$$

$$A(2, 3) = A(6, 9, 3) \quad (h=3)$$

$$\dots = \dots$$

$$A(2, 3) = A(2n, 3n, n) \quad (h=n)$$

Bu noktaların hepsi kartezyen koordinatta aynı noktayı gösterir ve hepsi birbirine eşit demektir. Bu nedenle bir çok işlemi kolaylaştırması için homojen koordinatta $h=1$ alınır. Böylece matematiksel dönüşümler kolaylaşmış olacaktır. Dikkat edersek yukarıda tek tablo şeklinde verilen dönüşüm formüllerindeki matris gösterimleri homojen koordinatlarda verilmiştir. Örnek olarak birini yazacak olursak (döndürme matrisini seçelim) aşağıdaki şekilde olur.

Döndürme matrisinin normal gösterimi

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

Döndürme matrisinin homojen koordinatlar kullanılarak gösterimi

$$\begin{bmatrix} x_2 \\ y_2 \\ z \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Sonuç olarak homojen koordinat sistemi bize afin dönüşümlerinin tamamının çarpımsal matris formunda uygulanabilme kolaylığı sağlayacak. Geometrik hesaplamalarda esnek bir hesaplama imkanı sağlayacaktır. Bu avantajı gelecek örneklerde daha iyi anlayacağız.

-----000-----

Şimdi gelelim perspektif dönüşüm için kullanılan parametrelerin nasıl bulunabileceğine. Perspektif dönüşümünün bir matris çarpması ile yapıldığını biliyoruz. Dönüşüm için kullandığımız matrisi homojen koordinatları da dahil ederek aşağıdaki gibi oluşturabiliriz. Homojen koordinatlarda yazarken $h=1$ aldığımız için denklemler aşağıdaki gibi oluşturulur. Burada (x_1, y_1) giriş resmindeki pikselin koordinatlarını, (X_1, Y_1) ise çıkış resmindeki koordinatları temsil etmektedir (Dikkat: önceki notlarda çıkış resminin koordinatlarını x_2, y_2 şeklinde kullanıyorduk. Fakat aşağıda 4 tane noktaya ihtiyaç olacak. Dolayısıyla noktaları 1,2,3,4 indisleri ile kullanacağımızdan burada çıkış resimlerinin koordinatları büyük harflerle gösterilmiştir.) Her iki resimdeki bu noktaların koordinat değerleri bilinmektedir. Bilinmeyen $[A]$ matrisinin elemanlarıdır.

$$\begin{bmatrix} X_1 \\ Y_1 \\ z \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Burada amacımız A matrisinin (a_{xx} -elemanları ile gösterilen matris) elemanlarını bulmaktır. İlk olarak matris değerlerini açık denklemler şeklinde yazalım.

$$X_1 z = a_{11} x_1 + a_{12} y_1 + a_{13}$$

$$Y_1 z = a_{21} x_1 + a_{22} y_1 + a_{23}$$

$$z = a_{31} x_1 + a_{32} y_1 + 1$$

Buradaki z denklemini önceki iki denklemde yerine yazarsak ve X_1, Y_1 çekersek denklemler aşağıdaki gibi olur.

$$X_1 = \frac{a_{11} x_1 + a_{12} y_1 + a_{13}}{a_{31} x_1 + a_{32} y_1 + 1}$$

$$Y_1 = \frac{a_{21} x_1 + a_{22} y_1 + a_{23}}{a_{31} x_1 + a_{32} y_1 + 1}$$

Bu denklemlere içler dışlar çarpımı uygularsak;

$$a_{31} x_1 X_1 + a_{32} y_1 X_1 + X_1 = a_{11} x_1 + a_{12} y_1 + a_{13}$$

$$a_{31} x_1 Y_1 + a_{32} y_1 Y_1 + Y_1 = a_{21} x_1 + a_{22} y_1 + a_{23}$$

Yalnız kalan X_1 ve Y_1 terimlerini çekersek;

$$X_1 = a_{11} x_1 + a_{12} y_1 + a_{13} - a_{31} x_1 X_1 - a_{32} y_1 X_1$$

$$Y_1 = a_{21} x_1 + a_{22} y_1 + a_{23} - a_{31} x_1 Y_1 - a_{32} y_1 Y_1$$

Bu iki denklemi matris formunda yazacak olursak $[C]=[B][A]$ şeklinde bir format ortaya çıkar. Burada $[A]$ bir vektördür (Tek boyutlu matris).

$$\begin{bmatrix} X_1 \\ Y_1 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1 X_1 & -y_1 X_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1 Y_1 & -y_1 Y_1 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{31} \\ a_{32} \end{bmatrix}$$

ifadesi elde edilir. Burada aranan [A] matrisi $[A]=[B^{-1}][C]$ işlemi ile kolaylıkla bulunabilir. Burada dikkat edilmesi gereken bir nokta elimizde sekiz bilinmeyenli bir [A] matrisi olmasına rağmen görünürde sadece iki denkleminiz olmasıdır. Sekiz bilinmeyenli bir denklemin çözümü için bağımsız sekiz tane denklem gerekmektedir. Bu amaçla tek bir nokta yerine 4 tane (x,y) noktası kullanılarak [A] matrisinin elemanları bulunacak. Bulduğumuz a_{xx} değerleri iki resim arasındaki dönüşümü yapacak olan 3x3 lük dönüşüm matrisini oluşturacaktır. Bilinen 4 nokta için matris denkleminizi tekrar yazarsak aşağıdaki şekilde olur.

$$\begin{bmatrix} X_1 \\ Y_1 \\ X_2 \\ Y_2 \\ X_3 \\ Y_3 \\ X_4 \\ Y_4 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1 X_1 & -y_1 X_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1 Y_1 & -y_1 Y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2 X_2 & -y_2 X_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2 Y_2 & -y_2 Y_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3 X_3 & -y_3 X_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3 Y_3 & -y_3 Y_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4 X_4 & -y_4 X_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4 Y_4 & -y_4 Y_4 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{31} \\ a_{32} \end{bmatrix}$$

Bu matris denkleminde, en sağdaki [A] matrisinin elemanları hariç diğer tüm elemanları biliyoruz. [A] matrisini yalnız bırakabilmek için ortadaki 8x8 lik [B] matrisinin tersini alıp, karşı taraftaki 8x1 lik [C] matrisi ile çarpmalıyız. Yani $[A]=[B^{-1}][C]$ şeklinde bir işlem yapacağız. Bu işlem için matris tersini almayı öğrenmemiz gerekiyor. Şimdi bu konuya bakalım daha sonra kaldığımız yerden devam ederiz.

Matris Tersini Alma

Matrisler sabit değerli sayıların bir düzen içerisinde tablo şeklinde yazılması ile oluşturulur. Bu konuda geliştirilen matematik görüntü işleme, istatistik gibi pek çok alandaki problemlerin bilgisayar programı ile çözümünü kolaylaştırır. Bu amaçla matrislerle ilgili Determinant hesaplama, Tersini alma gibi konuları bilmek önem arz eder.

Matris tersini alma işlemi ile ilgili olarak Algoritmik işlemler için uygun olan **Gauss Jordan** yöntemini kullanalım. Bu yöntemde amaç bir matrisin tersini almak için, tersi olan matris ile çarpımını, birim matrise dönüştürmektir. Yani bir matrisin kendisi [A] ise tersi $[A^{-1}]$ olur, buna da [B] matrisi diyelim. Bu durumda [A] matrisi ile tersi olan [B] matrisinin çarpımı birim matrisi [I] vermelidir. Birim matris köşegeni 1 olan diğer elemanları 0 olan kare matristir.

$$[A] \times [A^{-1}] = [I]$$

Bu ifade A matrisinin tersini B matrisi ile gösterirsek, yani $[A^{-1}] = [B]$ dersek;

$$[A] \times [B] = [I]$$

olur.

Matris tersini alma ile ilgili bir çok yöntem olmasına rağmen bizi programlama kolaylığı için Gauss Jordan yöntemini öğrenelim ve buna göre programını yazalım.

Gauss Jordan Yöntemi ile Matris Tersini Alma:

Gauss-Jordan yöntemi, özellikle büyük denklem takımlarını çözmede kolaylık sağlamaktadır. Bir diğer avantajı programlamaya uygun bir yöntemdir. Pratik bir şekilde en büyük matrislerin çözümünü sağlayan bir yöntemdir. Benzeri olan Gauss Eliminasyon (yok etme) metoduna benzer.

Bir örnek üzerinde bu yöntemi inceleyelim. Aşağıdaki denklem takımını çözmeye çalışalım.

$$11 = x + 2y + 2z$$

$$4 = 3x + 2y - z$$

$$3 = 2x - y + z$$

Bu denklem takımını matris formatında yazarsak aşağıdaki şekilde olur.

$$\begin{bmatrix} 11 \\ 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 2 \\ 3 & 2 & -1 \\ 2 & -1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Bu matris takımını $[C]=[A][X]$ matris formatında gösterirsek, buradaki $[X]$ matrisindeki değerleri bilmiyoruz ve bunu bulmak için $[A]$ matrisinin tersini alıp $[C]$ matrisi ile çarpmalıyız. Yani $[X]=[A^{-1}][C]$ şeklinde olmalıdır. Buna göre $[A]$ katsayılar matrisinin tersini nasıl alırız. Bunun için Gauss Jordan yöntemini kullanacağız.

Gauss Jordan yönteminde $[A]$ katsayı matrisini birim matrise dönüştürmeye çalışacağız ve bu durumda $[C]$ matrisi de ilgili sayılarla çarpılarak belli bir değer alacak. $[A]$ matrisi birim matris olunca yeni değerlere sahip olan $[C]$ matrisi x,y,z değerlerini gösteren sonuç matrisi olacaktır, yani denklem takımının çözümü olacaktır.

$$\begin{bmatrix} 11 \\ 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 2 \\ 3 & 2 & -1 \\ 2 & -1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \Rightarrow \begin{bmatrix} * \\ * \\ * \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Buradaki işlemleri matrisi formunda kısaltmalı olarak gösterirsek,

$[C] = [A] [X]$	$[A] [A^{-1}] = [I]$
$[C] = ([I] / [A^{-1}]) [X]$	$[A] = [I] / [A^{-1}]$
$[C] [A^{-1}] = [I] [X]$	

Şimdi $[C][A]$ matrislerine çeşitli satır işlemleri uygulayarak bu dönüşümü yapalım. Burada uygulanabilecek satır işlemleri aşağıdaki işlemlerden herhangi biri olabilir.

- Bir satırın bir sabit ile çarpılması
- Bir satırın diğer bir satır ile yer değiştirmesi
- Bir satırın diğer bir satırdan çıkarılması

Daha basit gösterim için aşağıdaki formatı kullanalım. Burada $[C]$ matrisi ile $[A]$ matrisi sadece yanyana getirilmiş. Herhangi bir matematiksel anlamı yok. İşlemler yapılırken her iki matrisi de uygulanacak. Böylece sağ taraftaki $[A]$ matrisi birim matrise dönüşürken sol taraftaki $[C]$ matriside başka bir değerler alacaktır. Böyle $[A]$ matrisinin birim matris olması sonucu ortaya çıkan $[C]$ matrisi aslında çözüm olarak aradığımız $[C] [A^{-1}]$ ifadesi olmuş oluyor.

$$\begin{bmatrix} 11 \\ 4 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 2 \\ 3 & 2 & -1 \\ 2 & -1 & 1 \end{bmatrix}$$

Önce köşegenin altındaki terimleri 0 yapalım. Köşegen üzerindeki değerleri de 1 yapalım. Burada hangi satırda işlem yaparsak o satırın kaçınıcı elemanı 0 yada 1 yapmaya çalışıyoruz ona bakıyoruz. Eğer 3 eleman bulunan satırın ilk elemanı değiştirmeye çalışıyorsak en yukarıdaki 1 satırda işlemleri yapmalıyız. Son 3. Elemanı değiştirmeye çalışıyorsak en son 3. Satırı kullanmalıyız. Çıkan değerleri kendi bulunduğu satır ile topluyoruz.

a) $(a_{21}=3)$ değerini 0 yapmak için birinci satırı -3 ile çarpıp (ilk terim olduğu için birinci satır ile çarpıyoruz) 2. satır ile toplayalım. Yani $[-3*11] [-3*1, -3*2, -3*2] = [-33] [-3, -6, -6]$ olur. Bu değerleri 2. satır ile toplarsak $[33+4] [-3+3, -6+2, -6-1] = [-29] [0, -4, -7]$. Bu sonucu matristeki yerlerinde gösterirsek; (Not: burada R gösterimi Row (satır manasında). İşlem yapılan satırları ifade ediyor.)

$$\begin{bmatrix} 11 \\ 4 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 2 \\ 3 & 2 & -1 \\ 2 & -1 & 1 \end{bmatrix} \xrightarrow{R_2 = -3R_1 + R_2} \begin{bmatrix} 11 \\ -29 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 2 \\ 0 & -4 & -7 \\ 2 & -1 & 1 \end{bmatrix}$$

b) $(a_{22}=-4)$ değerini 1 yapmak için ikinci satırı -4 bölmemiz yeterli olacaktır. Yani $[-29/-4][0/-4, -4/-4, -7/-4] = [29/4][0, 1, 7/4]$ olur. Bu sonucu matristeki yerlerinde gösterirsek;

$$\begin{bmatrix} 11 \\ -29 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 2 \\ 0 & -4 & -7 \\ 2 & -1 & 1 \end{bmatrix} \xrightarrow{R_2 = R_2 / -4} \begin{bmatrix} 11 \\ 29/4 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 2 \\ 0 & 1 & 7/4 \\ 2 & -1 & 1 \end{bmatrix}$$

c) $(a_{31}=2)$ değerini 0 yapmak için; o satırdaki ilk eleman olduğu için birinci satırı kullanmalıyız. Birinci satırı -2 ile çarparsak ve çıkan sonuçları da 3. satır ile toplarsak bu ilk terimi sıfır yapmış oluruz. Yani $[-2*11][-2*1, -2*2, -2*2] = [-22][-2, -4, -4]$ olur. Bu değerleri 3. satır ile toplarsak $[-22+3][-2+2, -4-1, -4+1] = [-19][0, -5, -3]$. Bu sonucu matristeki yerlerinde gösterirsek;

$$\begin{bmatrix} 11 \\ 29/4 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 2 \\ 0 & 1 & 7/4 \\ 2 & -1 & 1 \end{bmatrix} \xrightarrow{R_3 = -2R_1 + R_3} \begin{bmatrix} 11 \\ 29/4 \\ -19 \end{bmatrix} \begin{bmatrix} 1 & 2 & 2 \\ 0 & 1 & 7/4 \\ 0 & -5 & -3 \end{bmatrix}$$

d) $(a_{32}=-5)$ değerini 0 yapmak için; için ikinci satırı kullanmalıyız. İkinci satırı +5 ile çarparsak ve çıkan sonuçları da kendisi ile toplarsak bu ikinci terimi sıfır yapmış oluruz. Yani $[5*29/4][5*0, 5*1, 5*7/4] = [145/4][0, 5, 35/4]$ olur. Bu değerleri 3. satır ile toplarsak $[(145/4) -19][0+0, 5-5, (35/4)-3] = [69/4][0,0,23/4]$. Bu sonucu matristeki yerlerinde gösterirsek;

$$\begin{bmatrix} 11 \\ 29/4 \\ -19 \end{bmatrix} \begin{bmatrix} 1 & 2 & 2 \\ 0 & 1 & 7/4 \\ 0 & -5 & -3 \end{bmatrix} \xrightarrow{R_3 = +5R_2 + R_3} \begin{bmatrix} 11 \\ 29/4 \\ 69/4 \end{bmatrix} \begin{bmatrix} 1 & 2 & 2 \\ 0 & 1 & 7/4 \\ 0 & 0 & 23/4 \end{bmatrix}$$

e) $(a_{33}=23/5)$ değerini 1 yapmak için; üçüncü satırı ya $23/4$ bölmeliyiz yada $4/23$ ile çarpmalıyız.

$$\begin{bmatrix} 11 \\ 29/4 \\ 69/4 \end{bmatrix} \begin{bmatrix} 1 & 2 & 2 \\ 0 & 1 & 7/4 \\ 0 & 0 & 23/4 \end{bmatrix} \xrightarrow{R_3 = R_3 * 4/23} \begin{bmatrix} 11 \\ 29/4 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 2 \\ 0 & 1 & 7/4 \\ 0 & 0 & 1 \end{bmatrix}$$

Buraya kadar yapılan işlemler Gauss Eliminasyon yöntemi olarak geçmektedir. Köşegenin üst kısmındaki değerleri de 0 yaparsak Gauss Jordan yöntemi olmuş oluyor. Benzer şekilde kaldığımız yerden devam ederek köşegenin üst kısmını da 0 yapalım.

f) $(a_{12}=2)$ değerini 0 yapmak için; ikinci satırı -2 ile çarpıp birinci satır ile toplamalıyız. Yani $[-2*29/4][-2*0, -2*1, -2*7/4] = [-29/2][0, -2, -7/2]$ olur. Bu değerleri 1. satır ile toplarsak $[-29/2+11][0+1, -2+2, -7/2+2] = [-7/2][1,0,-3/2]$. Bu sonucu matristeki yerlerinde gösterirsek;

$$\begin{bmatrix} 11 \\ 29/4 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 2 \\ 0 & 1 & 7/4 \\ 0 & 0 & 1 \end{bmatrix} \xrightarrow{R_1 = -2*R_2 + R_1} \begin{bmatrix} -7/2 \\ 29/4 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 0 & -3/2 \\ 0 & 1 & 7/4 \\ 0 & 0 & 1 \end{bmatrix}$$

g) $(a_{13}=-3/2)$ değerini 0 yapmak için; üçüncü satırı $3/2$ ile çarpıp birinci satır ile toplamalıyız. Yani $[3/2*3][3/2*0, 3/2*0, 3/2*1] = [9/2][0, 0, 3/2]$ olur. Bu değerleri 1. satır ile toplarsak $[9/2-7/2][0+1, 0+0, 3/2-3/2] = [1][1,0,0]$. Bu sonucu matristeki yerlerinde gösterirsek;

$$\begin{bmatrix} -7/2 \\ 29/4 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 0 & -3/2 \\ 0 & 1 & 7/4 \\ 0 & 0 & 1 \end{bmatrix} \xrightarrow{R_1 = 3/2*R_3 + R_1} \begin{bmatrix} 1 \\ 29/4 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 7/4 \\ 0 & 0 & 1 \end{bmatrix}$$

g) $(a_{23}=7/4)$ değerini 0 yapmak için; üçüncü satırı $-7/4$ ile çarpıp ikinci satır ile toplamalıyız. Yani $[-7/4*3][-7/4*0, -7/4*0, -7/4*1] = [-21/4][0, 0, -7/4]$ olur. Bu değerleri 2. satır ile toplarsak $[-21/4+29/4][0+0, 0+1, -7/4+7/4] = [2][0,1,0]$. Bu sonucu matristeki yerlerinde gösterirsek;

$$\begin{bmatrix} 1 \\ 29/4 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 7/4 \\ 0 & 0 & 1 \end{bmatrix} \xrightarrow{R_1 = -7/4*R_3 + R_2} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Böylece katsayılar matrisini birim matrise dönüştürmüş olduk. Sonuç itibariyle denklem takımımız şöyle olur.

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Bu matris formatını açarsak x=1, y=2, z=3 olarak bulmuş oluruz.

Programlama (Matris Tersini alma)(Gauss Jordan yöntemi)

Matrisin tersinin alınması için geliştirilen programın kodları aşağıdadır. Bu kodları kullanımı dökümanın sonundaki diğer kodlarla birlikte bağlantısıda verilmiştir.

C# Kodları

```
=====A Matrisi=====
1,2,3,4,
7,11,9,0,
9,8,7,6,
1,12,3,14,
=====I Matrisi=====
1,0,0,0,
0,1,0,0,
0,0,1,0,
0,0,0,1,
=====B Matrisi=====
-0,2,-0,0769230769230768,0,196153846153846,-0,0269230769230769,
-0,2,0,0769230769230769,-0,0461538461538462,0,0769230769230769,
0,4,0,0769230769230769,-0,0961538461538461,-0,0730769230769231,
0,1,-0,0769230769230769,0,0461538461538461,0,0230769230769231,
==Kontrol==AxB=I Matrisi=====
1,0,0,0,
0,1,0,0,
0,0,1,0,
0,0,0,1,
```

```
// MATRİS TERSİNİ ALMA-----
public double[,] MatrisTersiniAl (double[,] GirisMatrisi)
{
    int MatrisBoyutu = Convert.ToInt16(Math.Sqrt(GirisMatrisi.Length)); //matris boyutu içindeki
    eleman sayısı olduğu için kare matrisde karekökü matris boyutu olur.
    double[,] CikisMatrisi = new double[MatrisBoyutu, MatrisBoyutu]; //A nın tersi alındığında bu
    matris içinde tutulacak.

    //--I Birim matrisin içeriğini dolduruyor
    for (int i = 0; i < MatrisBoyutu; i++)
    {
        for (int j = 0; j < MatrisBoyutu; j++)
        {
            if (i == j)
                CikisMatrisi[i, j] = 1;
            else
                CikisMatrisi[i, j] = 0;
        }
    }

    //--Matris Tersini alma işlemi-----
    double d, k;

    for (int i = 0; i < MatrisBoyutu; i++)
    {
        d = GirisMatrisi[i, i];
```

```

for (int j = 0; j < MatrisBoyutu; j++)
{
    if (d == 0)
    {
        d= 0.0001; //0 bölme hata veriyordu.
    }
    GirisMatrisi[i, j] = GirisMatrisi[i, j] / d;
    CikisMatrisi[i, j] = CikisMatrisi[i, j] / d;
}
for (int x = 0; x < MatrisBoyutu; x++)
{
    if (x != i)
    {
        k = GirisMatrisi[x, i];
        for (int j = 0; j < MatrisBoyutu; j++)
        {
            GirisMatrisi[x, j] = GirisMatrisi[x, j] - GirisMatrisi[i, j] * k;
            CikisMatrisi[x, j] = CikisMatrisi[x, j] - CikisMatrisi[i, j] * k;
        }
    }
}

return CikisMatrisi;
}

```

-----000-----

Kaldığımız yerden devam edersek, matris tersini almayı öğrendik. Şimdi aşağıdaki 4 nokta üzerinde perspektif düzeltme yapacak matris denklemindeki a_{xx} katsayılarını bulalım. Bunun için bir örnek resim kullanalım ve üzerindeki anahtar noktaların koordinatlarını paint programında okuyup dönüşüm matrisini bulalım.



Bu resimdeki 1,2,3,4 nolu noktaları sırayla a,b,c,d noktalarının olduğu köşelere doğru gerdirilecektir. Böylece plakanın yazım alanı resmin tamamını kaplayacaktır. Fakat plakanın en boyu orantısı bozulacağı için a,b,c,d noktalarının olduğu koordinatlar resmin içerisine sığdırmak yerine, istediğimiz herhangi bir dikdörtgenin köşeleri olarak da atayabiliriz. Önce birinci uygulamayı bir yapalım. Bu 8 noktanın koordinatlarını paint gibi basit bir programa atıp mouse ile üzerinde gezdirirsek koordinatları okuyabiliriz. Tabi burada resmin üzerine sonradan eklenen çizgiler işlemlerde olmayacaktır. Yukarıdaki resmin üzerindeki noktaların koordinatları aşağıda gibi okunmuştur. $x_2=806$ değeri çerçevenin dışında bir nokta olarak belirlenmiştir.

1.nokta, $x_1= 47, y_1=155$

2.nokta, $x_2=806, y_2=101$

a noktası, $X_1 = 0, Y_1=0$

b noktası, $X_2 = 800, Y_2=0$

3.nokta, $x_3 = 46$, $y_3=291$
4.nokta, $x_4 = 726$, $y_4=417$

c noktası, $X_3 = 0$, $Y_3=600$
d noktası, $X_4 = 800$, $Y_4=600$

Bu değerleri aşağıdaki dönüşüm matrisimizde yerine yazalım.

$$\begin{bmatrix} X_1 \\ Y_1 \\ X_2 \\ Y_2 \\ X_3 \\ Y_3 \\ X_4 \\ Y_4 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1X_1 & -y_1Y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1Y_1 & -y_1X_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2X_2 & -y_2Y_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2Y_2 & -y_2X_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3X_3 & -y_3Y_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3Y_3 & -y_3X_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4X_4 & -y_4Y_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4Y_4 & -y_4X_4 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{31} \\ a_{32} \end{bmatrix}$$

8 noktanın koordinatı yerlerine yazıldığında denklem aşağıdaki gibi olur.

$$\begin{bmatrix} 0 \\ 0 \\ 800 \\ 0 \\ 0 \\ 600 \\ 800 \\ 600 \end{bmatrix} = \begin{bmatrix} 47 & 155 & 1 & 0 & 0 & 0 & -47 * 0 & -155 * 0 \\ 0 & 0 & 0 & 47 & 155 & 1 & -47 * 0 & -155 * 0 \\ 806 & 101 & 1 & 0 & 0 & 0 & -806 * 800 & -101 * 800 \\ 0 & 0 & 0 & 806 & 101 & 1 & -806 * 0 & -101 * 0 \\ 46 & 291 & 1 & 0 & 0 & 0 & -46 * 0 & -291 * 0 \\ 0 & 0 & 0 & 46 & 291 & 1 & -46 * 600 & -291 * 600 \\ 726 & 417 & 1 & 0 & 0 & 0 & -726 * 800 & -417 * 800 \\ 0 & 0 & 0 & 726 & 417 & 1 & -726 * 600 & -417 * 600 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{31} \\ a_{32} \end{bmatrix}$$

Bu matris denklemini , $[C] = [B] [A]$ şeklinde gösterirsek B matrisinin tersini $[B^{-1}]$ yukarıdaki program ile bulabiliriz. Bu durumda aşağıdaki sonucu buluruz.

$$[B^{-1}] = \begin{bmatrix} -0,005547 & -0,004954 & 0,002844 & 0,002039 & 0,004403 & 0,005187 & -0,001699 & -0,002272 \\ -0,007394 & -3,6E-05 & 2,1E-05 & 1,5E-05 & 0,007385 & 3,8E-05 & -1,2E-05 & -1,7E-05 \\ 2,406739 & 0,23849 & -0,136889 & -0,098176 & -1,351657 & -0,24969 & 0,081807 & 0,109375 \\ -0,000809 & -0,001922 & 0,000333 & 0,001351 & 0,000847 & 0,000607 & -0,000371 & -3,7E-05 \\ -0,011368 & -0,008493 & 0,00468 & 0,000473 & 0,011902 & 0,008536 & -0,005214 & -0,000517 \\ 1,800114 & 2,406739 & -0,741026 & -0,136889 & -1,884646 & -1,351657 & 0,825558 & 0,081807 \\ -3E-06 & -6E-06 & 1E-06 & 2E-06 & 3E-06 & 6E-06 & -2E-06 & -3E-06 \\ -8E-06 & 0 & 3E-06 & 0 & 9E-06 & 0 & -4E-06 & 0 \end{bmatrix}$$

$$[A] = [B^{-1}] [C]$$

$$\begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{31} \\ a_{32} \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} & b_{16} & b_{17} & b_{18} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} & b_{26} & b_{27} & b_{28} \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} & b_{36} & b_{37} & b_{38} \\ b_{41} & b_{42} & b_{43} & b_{44} & b_{45} & b_{46} & b_{47} & b_{48} \\ b_{51} & b_{52} & b_{53} & b_{54} & b_{55} & b_{56} & b_{57} & b_{58} \\ b_{61} & b_{62} & b_{63} & b_{64} & b_{65} & b_{66} & b_{67} & b_{68} \\ b_{71} & b_{72} & b_{73} & b_{74} & b_{75} & b_{76} & b_{77} & b_{78} \\ b_{81} & b_{82} & b_{83} & b_{84} & b_{85} & b_{86} & b_{87} & b_{88} \end{bmatrix} \begin{bmatrix} X_1 \\ Y_1 \\ X_2 \\ Y_2 \\ X_3 \\ Y_3 \\ X_4 \\ Y_4 \end{bmatrix}$$

Matris formatını denklem şekline dönüştürsek.

$$a_{11} = b_{11}X_1 + b_{12}Y_1 + b_{13}X_2 + b_{14}Y_2 + b_{15}X_3 + b_{16}Y_3 + b_{17}X_4 + b_{18}Y_4$$

$$a_{12} = b_{21}X_1 + b_{22}Y_1 + b_{23}X_2 + b_{24}Y_2 + b_{25}X_3 + b_{26}Y_3 + b_{27}X_4 + b_{28}Y_4$$

(...şeklinde devam eder...)

$$a_{21} = b_{41}X_1 + b_{42}Y_1 + b_{43}X_2 + b_{44}Y_2 + b_{45}X_3 + b_{46}Y_3 + b_{47}X_4 + b_{48}Y_4$$

$$a_{22} = b_{51}X_1 + b_{52}Y_1 + b_{53}X_2 + b_{54}Y_2 + b_{55}X_3 + b_{56}Y_3 + b_{57}X_4 + b_{58}Y_4$$

(...şeklinde devam eder...)

----- 0000-----

b_{xx} matrisin tersinin katsayılarını bulduğumuza göre ve X_x, Y_x koordinatlarını da biliyoruz buradan a_{xx} değerlerini buluruz. Bu değerler aynı zamanda dönüşüm denklemindeki $[A]$ matrisinin elemanlarıdır. Yani aşağıdaki denklemdeki katsayılar matrisidir.

$$\begin{bmatrix} X_1Z \\ Y_1Z \\ Z \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Bu denklemi resim üzerindeki tüm noktalara uygulamak için genelleştirsek ve aynı zamanda programlamada kullanacağımız indislere dönüştürürsek aşağıdaki şekilde olur. Programlarken diziler (0,0) değerinden başlar.

$$\begin{bmatrix} XZ \\ YZ \\ Z \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$z = a_{20}x + a_{21}y + 1$$

$$X = \frac{a_{00}x + a_{01}y + a_{02}}{z}$$

$$Y = \frac{a_{10}x + a_{11}y + a_{12}}{z}$$

Dikkat: Matrisleri programlarken indisler bir sayı aşağıdan başlar. Çünkü dizilerin başlangıç değerleri (0,0) dır. Buna göre programda a_{11} matris elemanı a_{00} olarak gösterilir.

Artık bu formülleri kullanarak tüm belirlediğimiz alanı istediğimiz çerçeve içine sığdırabiliriz.

Programlama (Perspektif Düzeltme)

Aşağıdaki sınırlarda plakanın köşeleri (x_x, y_x -koordinatları), resmin sınırlarına (X_x, Y_x) genişletilmiştir. Bu nedenle plaka doğal görünümünün dışına çıkmıştır.



Plakayı daha gerçekçi boyutlar görmek için dönüşeceği sınırları daha küçük tutmalıyız. ($X_1, Y_1=0,178$; $X_2, Y_2=800,178$; $X_3, Y_3=0,378$; $X_4, Y_4=800,378$). Çerçeve 800x200 olarak belirlenmiştir.



Resim ne kadar büyük bir çerçeveye yerleştirilirse aradaki boşluklar (piksel kayıpları-alias) o kadar fazla olmaktadır. Elde edilen resimler tekrar filtreden geçilip kayıp pikseller düzeltilmelidir. Eğer yerleştirilecek çerçeve daha küçük seçilirse kayıp piksel azalmaktadır. Aşağıdaki örnek (400x100) boyutlarındadır. Burada gösterilen resimler küçültülmüş boyutlardadır. Bu resimlerin her biri gerçekte 800x600 boyutlarındadır.



Yukarıdaki ilk örnekte plaka alanı (800x600) alana yerleştirilince kayıp piksel artmıştır (a resmi). Plaka 400x100 boyutlarına küçültüldüğünde kayıp pikseller daha az olmuştur (b resmi).



```
private void btnPERSPEKTIF_Click(object sender, EventArgs e)
{
    double x1 = Convert.ToDouble(txt_x1.Text);
    double y1 = Convert.ToDouble(txt_y1.Text);
    double x2 = Convert.ToDouble(txt_x2.Text);
    double y2 = Convert.ToDouble(txt_y2.Text);
    double x3 = Convert.ToDouble(txt_x3.Text);
    double y3 = Convert.ToDouble(txt_y3.Text);
    double x4 = Convert.ToDouble(txt_x4.Text);
    double y4 = Convert.ToDouble(txt_y4.Text);

    double X1 = Convert.ToDouble(txtX1.Text);
    double Y1 = Convert.ToDouble(txtY1.Text);
    double X2 = Convert.ToDouble(txtX2.Text);
    double Y2 = Convert.ToDouble(txtY2.Text);
    double X3 = Convert.ToDouble(txtX3.Text);
    double Y3 = Convert.ToDouble(txtY3.Text);
    double X4 = Convert.ToDouble(txtX4.Text);
    double Y4 = Convert.ToDouble(txtY4.Text);
}
```



```

double[,] GirisMatrisi = new double[8, 8];

// { x1, y1, 1, 0, 0, 0, -x1 * X1, -y1 * X1 }
GirisMatrisi[0, 0] = x1;
GirisMatrisi[0, 1] = y1;
GirisMatrisi[0, 2] = 1;
GirisMatrisi[0, 3] = 0;
GirisMatrisi[0, 4] = 0;
GirisMatrisi[0, 5] = 0;
GirisMatrisi[0, 6] = -x1 * X1;
GirisMatrisi[0, 7] = -y1 * X1;

//{ 0, 0, 0, x1, y1, 1, -x1 * Y1, -y1 * Y1 }
GirisMatrisi[1, 0] = 0;
GirisMatrisi[1, 1] = 0;
GirisMatrisi[1, 2] = 0;
GirisMatrisi[1, 3] = x1;
GirisMatrisi[1, 4] = y1;
GirisMatrisi[1, 5] = 1;
GirisMatrisi[1, 6] = -x1 * Y1;
GirisMatrisi[1, 7] = -y1 * Y1;

//{ x2, y2, 1, 0, 0, 0, -x2 * X2, -y2 * X2 }
GirisMatrisi[2, 0] = x2;
GirisMatrisi[2, 1] = y2;
GirisMatrisi[2, 2] = 1;
GirisMatrisi[2, 3] = 0;
GirisMatrisi[2, 4] = 0;
GirisMatrisi[2, 5] = 0;
GirisMatrisi[2, 6] = -x2 * X2;
GirisMatrisi[2, 7] = -y2 * X2;

//{ 0, 0, 0, x2, y2, 1, -x2 * Y2, -y2 * Y2 }
GirisMatrisi[3, 0] = 0;
GirisMatrisi[3, 1] = 0;
GirisMatrisi[3, 2] = 0;
GirisMatrisi[3, 3] = x2;
GirisMatrisi[3, 4] = y2;
GirisMatrisi[3, 5] = 1;
GirisMatrisi[3, 6] = -x2 * Y2;
GirisMatrisi[3, 7] = -y2 * Y2;

//{ x3, y3, 1, 0, 0, 0, -x3 * X3, -y3 * X3 }
GirisMatrisi[4, 0] = x3;
GirisMatrisi[4, 1] = y3;
GirisMatrisi[4, 2] = 1;
GirisMatrisi[4, 3] = 0;
GirisMatrisi[4, 4] = 0;
GirisMatrisi[4, 5] = 0;
GirisMatrisi[4, 6] = -x3 * X3;
GirisMatrisi[4, 7] = -y3 * X3;

//{ 0, 0, 0, x3, y3, 1, -x3 * Y3, -y3 * Y3 }
GirisMatrisi[5, 0] = 0;
GirisMatrisi[5, 1] = 0;
GirisMatrisi[5, 2] = 0;

```

```

GirisMatrisi[5, 3] = x3;
GirisMatrisi[5, 4] = y3;
GirisMatrisi[5, 5] = 1;
GirisMatrisi[5, 6] = -x3 * Y3;
GirisMatrisi[5, 7] = -y3 * Y3;

//{ x4, y4, 1, 0, 0, 0, -x4 * X4, -y4 * X4 }
GirisMatrisi[6, 0] = x4;
GirisMatrisi[6, 1] = y4;
GirisMatrisi[6, 2] = 1;
GirisMatrisi[6, 3] = 0;
GirisMatrisi[6, 4] = 0;
GirisMatrisi[6, 5] = 0;
GirisMatrisi[6, 6] = -x4 * X4;
GirisMatrisi[6, 7] = -y4 * X4;

//{ 0, 0, 0, x4, y4, 1, -x4 * Y4, -y4 * Y4 }
GirisMatrisi[7, 0] = 0;
GirisMatrisi[7, 1] = 0;
GirisMatrisi[7, 2] = 0;
GirisMatrisi[7, 3] = x4;
GirisMatrisi[7, 4] = y4;
GirisMatrisi[7, 5] = 1;
GirisMatrisi[7, 6] = -x4 * Y4;
GirisMatrisi[7, 7] = -y4 * Y4;

//-----
double[,] matrisBTersi = MatrisTersiniAl(GirisMatrisi);

//----- A Dönüşüm Matrisi (3x3) -----
-
double a00 = 0, a01 = 0, a02 = 0, a10 = 0, a11 = 0, a12 = 0, a20 = 0, a21 = 0,
a22 = 0;

a00 = matrisBTersi[0, 0] * X1 + matrisBTersi[0, 1] * Y1 + matrisBTersi[0, 2] *
X2 + matrisBTersi[0, 3] * Y2 + matrisBTersi[0, 4] * X3 + matrisBTersi[0, 5] * Y3 +
matrisBTersi[0, 6] * X4 + matrisBTersi[0, 7] * Y4;
a01 = matrisBTersi[1, 0] * X1 + matrisBTersi[1, 1] * Y1 + matrisBTersi[1, 2] *
X2 + matrisBTersi[1, 3] * Y2 + matrisBTersi[1, 4] * X3 + matrisBTersi[1, 5] * Y3 +
matrisBTersi[1, 6] * X4 + matrisBTersi[1, 7] * Y4;
a02 = matrisBTersi[2, 0] * X1 + matrisBTersi[2, 1] * Y1 + matrisBTersi[2, 2] *
X2 + matrisBTersi[2, 3] * Y2 + matrisBTersi[2, 4] * X3 + matrisBTersi[2, 5] * Y3 +
matrisBTersi[2, 6] * X4 + matrisBTersi[2, 7] * Y4;

a10 = matrisBTersi[3, 0] * X1 + matrisBTersi[3, 1] * Y1 + matrisBTersi[3, 2] *
X2 + matrisBTersi[3, 3] * Y2 + matrisBTersi[3, 4] * X3 + matrisBTersi[3, 5] * Y3 +
matrisBTersi[3, 6] * X4 + matrisBTersi[3, 7] * Y4;
a11 = matrisBTersi[4, 0] * X1 + matrisBTersi[4, 1] * Y1 + matrisBTersi[4, 2] *
X2 + matrisBTersi[4, 3] * Y2 + matrisBTersi[4, 4] * X3 + matrisBTersi[4, 5] * Y3 +
matrisBTersi[4, 6] * X4 + matrisBTersi[4, 7] * Y4;
a12 = matrisBTersi[5, 0] * X1 + matrisBTersi[5, 1] * Y1 + matrisBTersi[5, 2] *
X2 + matrisBTersi[5, 3] * Y2 + matrisBTersi[5, 4] * X3 + matrisBTersi[5, 5] * Y3 +
matrisBTersi[5, 6] * X4 + matrisBTersi[5, 7] * Y4;

a20 = matrisBTersi[6, 0] * X1 + matrisBTersi[6, 1] * Y1 + matrisBTersi[6, 2] *
X2 + matrisBTersi[6, 3] * Y2 + matrisBTersi[6, 4] * X3 + matrisBTersi[6, 5] * Y3 +
matrisBTersi[6, 6] * X4 + matrisBTersi[6, 7] * Y4;

```

```

    a21 = matrisBTersi[7, 0] * X1 + matrisBTersi[7, 1] * Y1 + matrisBTersi[7, 2] *
X2 + matrisBTersi[7, 3] * Y2 + matrisBTersi[7, 4] * X3 + matrisBTersi[7, 5] * Y3 +
matrisBTersi[7, 6] * X4 + matrisBTersi[7, 7] * Y4;
    a22 = 1;
    //----- Perspektif düzeltme işlemi -----
    -----
    PerspektifDuzelt(a00, a01, a02, a10, a11, a12, a20, a21, a22);
}

```

```

//===== Perspektif düzeltme işlemi =====
public void PerspektifDuzelt(double a00, double a01, double a02, double a10, double
a11, double a12, double a20, double a21, double a22)
{
    Bitmap GirisResmi, CikisResmi;
    Color OkunanRenk;

    GirisResmi = new Bitmap(pictureBox1.Image);

    int ResimGenisligi = GirisResmi.Width;
    int ResimYuksekligi = GirisResmi.Height;

    CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi);

    double X, Y, z;

    for (int x = 0; x < (ResimGenisligi); x++)
    {
        for (int y = 0; y < (ResimYuksekligi); y++)
        {
            OkunanRenk = GirisResmi.GetPixel(x, y);

            z = a20 * x + a21 * y + 1;
            X = (a00 * x + a01 * y + a02) / z;
            Y = (a10 * x + a11 * y + a12) / z;

            if (X > 0 && X < ResimGenisligi && Y > 0 && Y < ResimYuksekligi)
            //PictureBox ın dışına çıkan kısımlar oluşturulmayacak.
                CikisResmi.SetPixel((int)X, (int)Y, OkunanRenk);
        }
    }

    pictureBox2.Image = CikisResmi;
}

```

Ödev 1. Perspektif alanı mouse ile seçme

Yukarıdaki örnek uygulamada 1,2,3,4 ve a,b,c,d noktalarını belirlemeyi mouse ile resim üzerine tıklayarak yapın. Yani bu 8 tane nokta belirlenirken mouse kullanın. Buna göre perspektif düzeltme işlemini gerçekleştirin. Örnek resim olarak bina cephelerindeki perspektif görünümlü reklam tabelalarını kullanın. Perspektif düzeltmesi yapıldıktan sonra Alias oluşan noktalarda (hesaplama yapmadığı ve resim üzerinde boşlukları temsil eden siyah

noktaları) daha sonra düzeltmeye çalışın. Bu kısımlara geldiğinde etrafındaki 9 tane pikselin değerine bakın ve ortadaki pikseli bu piksellerin ortalaması olarak aldırın. Buna benzer resmi daha da iyileştirecek bir yöntem bulmaya çalışın.

Araştırma: Gaus Jordan yönteminin ne olduğunu araştırılın. Yukarıda geçen 8x8 matrislerin tersini almak için programlamada bu yöntem mi kullanılmaktadır, değilse hangi yöntemi kullanıyor.