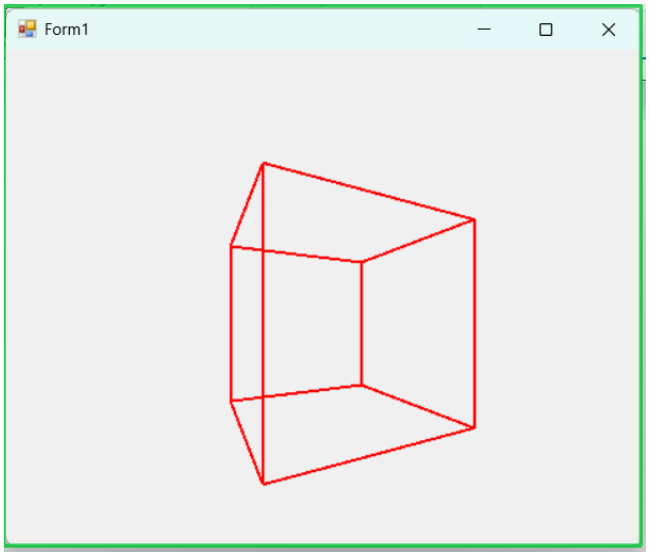
BİLGİSAYAR PROGRAMLAMA-II- 6.ders

3D GRAFİK ÇİZİM-(3 Boyutlu Küp Hareketi)




```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace DersUygulamasi_1
{
    public partial class Form1 : Form
    {
        Timer Zamanlayici = new Timer();
        float Aci = 0;

        //Küp köşelerini tutacak kendi nesnemizi dizi olarak oluşturuyoruz (DiziNokta3D).
        //Bu dizi Nokta3D sınıfından türetiliyor. her nokta bilgisi sınıfın içindeki
metoda gönderiliyor.
        //Şu ifadenin olduğu yer sınıfın adıdır. new Nokta3D[.. , Şu ifade ise içindeki
Metodun adıdır. new Nokta3D(..
        //Sınıfın içindeki alt metodda aynı isimde olmak zorunda. Bunlara "Constructor"
metod denir. Aşağıda detayı verildi.
        Nokta3D[] DiziNokta3D = new Nokta3D[]
        {
            new Nokta3D(-1, -1, -1),
            new Nokta3D( 1, -1, -1),
            new Nokta3D( 1,  1, -1),
            new Nokta3D(-1,  1, -1),
            new Nokta3D(-1, -1,  1),
            new Nokta3D( 1, -1,  1),
            new Nokta3D( 1,  1,  1),
            new Nokta3D(-1,  1,  1)
        };

        //Hangi noktalar arasında çizgi olacağını tutuyor. {0,1}: 0 inci nokta ile 1 nokta
arasın bir çizgi demek
        //Küp çizimi için, ilk 4 satır üst kareyi, sonraki 4 satır alt kareyi, son 4 satır
ise bu iki kareyi birleştiren dikmeleri çizer.
        int[,] DiziKenar = new int[,]
        {
            {0,1},{1,2},{2,3},{3,0},
            {4,5},{5,6},{6,7},{7,4},
            {0,4},{1,5},{2,6},{3,7}
```

```
        };
        //Bu kod bir Windows formunun(Form1) yapıcı metodudur(constructor). Form açılırken
otomatik çalışır.
        public Form1()
        {
            //Visual Studio'nun otomatik oluşturduğu metot.
            //Form üzerindeki butonlar, etiketler, textbox'lar gibi tüm kontrolleri
oluşturur ve yerleştirir.
            //Bu metod çağrılmazsa form boş gelir ve kontroller görünmez.
            InitializeComponent();

            //Çift arabellekleme(double buffering) aktive edilir.
            //Ne işe yarar? Ekranda hareketli veya karmaşık çizimler yaparken
titreme(flicker) sorununu azaltır.
            //Özellikle Paint olayında çizim yapıyorsanız(animasyon, grafik, oyun vb.) bu
çok önemlidir.
            this.DoubleBuffered = true;

            //Zamanlayıcının(timer) her 30 milisaniyede bir tetiklenmesini sağlar.
            //30 ms ≈ saniyede yaklaşık 33 kare(33 FPS) demektir.
            Zamanlayici.Interval = 30;

            //Timer'ın Tick olayın Anonim yada Lambda denilen bir fonksiyon şeklinde
kullanılacak.
            //Bu fonksiyon Timer her tick olduğunda metod içindeki kodlar çalıştırılır.
            //Geleneksel private void Timer_Tick(object sender, EventArgs e){..}
kullanımına göre avantaj dezevantajları vardır.
            //Lambda metodu, tanımlandığı yerdeki yerel değişkenlere doğrudan erişebilir.
            //Geleneksel metodda bu değişkenleri taşımak için sınıf seviyesinde(global)
tanımlama yapmak gerekirdi.
            //Sadece bir kez kullanılacak küçük bir işlem için sınıfın içine fazladan
private void ... metodu eklemeye gerek yoktur.
            //Timer'ın ne yapacağını başka bir yere gitmeden hemen görmek mümkün olur.
            //Bu metod Form çalıştığı sürece hafızada olacaktır. Eğer hafıza yönetimi için
form kapanmadan hafızadan kaldırmak isteniyorsa geleneksel metod kullanılsa daha iyi olur.
            //+=: Olay başlatan operatördür.
            //(s, e): Metodun parametreleridir. Buradaki s gönderen (sender) nesneyi
temsil eder, e ise olaya ait argümanlarını(eventArgs) temsil eder. İsim kişiye özel
olabilir ama standart budur.
            // Geleneksel metodda bu kullanım (object sender, EventArgs e) şeklinde
yazılır.
            // =>: Git operatörü. Parametreleri metoda gönderir.
            Zamanlayici.Tick += (s, e) =>
            {
                Aci += 0.05f; // Açı Aci değişkenini her çalıştığında 0.05 radyan artır.
Aci = Aci+ 0.05f; demektir. C# butün açılar Radyan olmak zorundadır.

                //Formu önceki çizimini geçersiz kılıp, yeniden çizilmeye zorlar.
                //Bunun sonucunda formun OnPaint veya Paint olayı tetiklenir.
                this.Invalidate();
            };

            //Zamanlayıcıyı (Zamanlayici) başlatır. Artık 30 ms'de bir Tick olayı
fırlatılacak.
            Zamanlayici.Start();
        }
       //OnPaint metodunu ile formun üzerinde çizim yapımı yapılıyor. (PaintEventArgs e)
ile e değişkeni kullanılarak boyama olayına ait argümanlar alınıyor.
       protected override void OnPaint(PaintEventArgs e)
       {
           //Graphics kütüphanesi çizimle alakalı tüm işlemleri barındıran sınıftır.
Bundan bir nesne türetiliyor ve buna "g" ismi veriliyor.
           //Fakat g nesnesi, forma ait olan "e" değişkenine bağlanıyor, yani form
üzerinde çizim yapılacak.
           Graphics g = e.Graphics;
```

```
            float Olcek = 200;

            //PointF sınıfı (x,y) formatı şeklindedir. 3B noktaların sayısı kadar Iz düşüm
noktalarını tutmak için dizi elemanı oluşturuluyor.
            PointF[] DiziIzDusumNokta = new PointF[DiziNokta3D.Length];

            //3 boyutlu nokta sayısınca döngüyü başlatıyor.
            for (int i = 0; i < DiziNokta3D.Length; i++)
            {
                //3Boyutlu noktaları Y ekseni etrafında uzayda döndürüp yeni Nokta
koordinatlarını hesaplıyor.
                var DonmusNoktalar3D = DondurY(DiziNokta3D[i], Aci);

                //Dönmüş noktaları Ekran (form yüzeyi) isdüzümü üzerindeki X,Y noktlarını
hesaplıyor ve dizinin elemanlarına atıyor.
                DiziIzDusumNokta[i] = PerspektifIzDusur(DonmusNoktalar3D, Olcek,
this.Width, this.Height);
            }

            //Kırmızı renkli ve 2 piksel kalınlığında Kalem nesnesini oluşturuyor.
            Pen Kalem = new Pen(Color.Red, 2);

            //DiziKenar.GetLength(0) Açıklaması:
            //C#'ta çok boyutlu dizilerde (örneğin int[,]) klasik .Length özelliği toplam
eleman sayısını verir.
            //Bu dizi için bu 24'tür. Ancak bize toplam eleman sayısı değil, satır sayısı
lazım.
            //GetLength(0): Dizinin birinci boyutunun (satır sayısı) uzunluğunu verir. Bu
dizide 12 adet {x, y} çifti olduğu için bu değer 12 dönecektir.
            //GetLength(1): Dizinin ikinci boyutunun (sütun sayısı) uzunluğunu verir. Her
satırda 2 sayı (başlangıç ve bitiş noktası) olduğu için bu değer 2 dönecektir.
            //Yani matris şeklinde düşünürsek 12 satır, 2 sutunlu bir eleman olur. Özetle
DiziKenar.GetLength(0) ifadesi, döngünün "elimizdeki 12 adet kenarın her biri için bir kez
çalış" demesini sağlar.
            for (int i = 0; i < DiziKenar.GetLength(0); i++)
            {
                int a = DiziKenar[i, 0]; //O anki kenarın İLK nokta indeksini al (Örn: 0)
                int b = DiziKenar[i, 1]; //O anki kenarın İKİNCİ nokta indeksini al (Örn:
1)

                //g.DrawLine(Kalem, BaşlangıçNoktası, BitişNoktası);
                //a ve b: Çizilecek çizginin iki ucunun hangi noktalar olacağını seçer.
                g.DrawLine(Kalem, DiziIzDusumNokta[a], DiziIzDusumNokta[b]);
            }
        }
        // Y ekseni etrafında döndürme
        Nokta3D DondurY(Nokta3D p3, float Aci)
        {
            float cos = (float)Math.Cos(Aci);
            float sin = (float)Math.Sin(Aci);

            float x = p3.X * cos + p3.Z * sin;
            float z = -p3.X * sin + p3.Z * cos;

            return new Nokta3D(x, p3.Y, z);
        }

        //PointF sınıfı x ve y den oluşan koordinat bilgisi içerir  Perspektif projeksiyon
        PointF PerspektifIzDusur(Nokta3D p3, float Olcek, int EkranGenislik, int
EkranYukseklik)
        {
            //p3.Z: Noktanın derinliğidir (ekrandan içeriye doğru olan mesafe).
            //Uzaklik: Sanal kameranın nesneye olan mesafesidir.
```

```
            //Eğer p3.Z ve Uzaklik toplamı büyürse (yani nesne uzaklaşırsa), Faktor değeri
küçülür.
            //Faktör küçüldükçe perspektif olarak nesne küçülüyor demektir.
            //Faktor küçüldüğünde, noktalar ekranın merkezine doğru yaklaşır. Bu da
uzaklaşan nesnelerin daha küçük görünmesini sağlar.

            float Uzaklik = 3;
            float Faktor = Olcek / (p3.Z + Uzaklik);

            //Ölçekleme: 3D koordinatlar(p3.X, p3.Y), az önce hesapladığımız Faktor ile
çarpılarak derinliğe göre küçültülür / büyütülür.
            //Merkezleme(Offset): Bilgisayar ekranında(0,0) noktası sol üst köşedir. Ancak
biz 3D objemizin formun tam ortasında dönmesini isteriz. Genislik / 2 ve Yukseklik / 2
eklenerek orijin noktası formun merkezine kaydırılır.
            //Y Eksenini Ters Çevirme(-p3.Y): Matematiksel koordinat sisteminde yukarı
yön(+) iken, bilgisayar ekranında aşağı yön(+)'dır. Bu yüzden nesnenin ters görünmemesi
için Y değeri eksi ile çarpılır.
            //Eğer bu metodu kullanmayıp doğrudan X ve Y değerlerini çizseydik (Ortografik
izdüşüm), nesne ne kadar uzaklaşırsa uzaklaşsın boyutu hiç değişmezdi. Bu da derinlik
algısını tamamen yok ederdi.
            float x = p3.X * Faktor + EkranGenislik / 2;
            float y = -p3.Y * Faktor + EkranYukseklik / 2;

            return new PointF(x, y);
        }

    }


    //BİLGİ:3 boyutlu bir noktayı temsil eden sınıftır. İçinde X, Y, Z isimli 3 adet float
tipinde değişken vardır.
    //Bu sınıf 3 boyutlu noktaları tutan bir sınıf olacak. Kullanımı için özel sınıf olan
yapıcı metoda ihtiyaç vardır.
    //Amaç, Nesne ilk oluşurken alanlara ilk değer atamaktır.
    public class Nokta3D
    {
        //dışarıdan metoda gönderilecek (x,y,z) koordinatlarını sınıf içinde kullanılan
(X,Y,Z) değişkenlerine aktarılacak.
        public float X, Y, Z;

        //Bu bir Yapıcı Metod (Constructor) metoddur ve sınıfla aynı isme sahiptir.
        //Geri dönüş tipi YOKTUR(void bile yazılmaz). Çünkü yapıcı metodlar asla bir değer
döndürmez.
        //Kullanımı, Point3D nokta = new Point3D(1.5f, 2.3f, 4.0f);
        //Bu satır çalışınca yapıcı metot içindeki kod çalışır. X = 1.5f; Y = 2.3f; Z =
4.0f; atamaları yapılır.
        //Float tipi kullanırken rakamına yanına f harfi konulmalı. Yoksa Double kabul
eder.
        //Double daha büyük sayıları alır 10^308 gibi. Float ise 10^38 gibi sayıları alır.
Double 8 byte yer kaplar. Float 4 byte yer kaplar.
        public Nokta3D(float x, float y, float z)
        {
            X = x;
            Y = y;
            Z = z;
        }
    }
}
```

DÖNDÜRME FORMÜLLERİNİN AÇIKLAMASI

```
// Y ekseni etrafında döndürme
Nokta3D DonduruY(Nokta3D p3, float Aci)
```

```
{
    float cos = (float)Math.Cos(Aci);
    float sin = (float)Math.Sin(Aci);

    float x = p3.X * cos + p3.Z * sin;
    float z = -p3.X * sin + p3.Z * cos;

    return new Nokta3D(x, p3.Y, z);
}
```

Bu kodlar 3D uzaydaki bir noktayı **Y ekseni** (yukarı-aşağı doğrultusu) sabit kalacak şekilde döndürmek için kullanılan temel trigonometrik dönüşümü uyguluyor.

Formülün nasıl çıkarıldığını ve mantığını adım adım inceleyelim:

1. Sabit Kalan Eksen Y

Bir nesneyi Y ekseni etrafında döndürürsekz (tıpkı bir kapının kendi etrafında dönmesi gibi), noktaların **Y koordinatı hiç değişmez.** Sadece derinlik (**Z**) ve genişlik (**X**) değerleri değişir.

Bu yüzden metodda Y değeri olduğu gibi aktarılır. Diğer iki eksen değişimi hesaplanır.

2. Dönüş hesaplama

Y ekseni etrafında dönüşü, aslında **X-Z düzlemi üzerinde yapılan 2 boyutlu bir dönüş** gibi düşünebiliriz. Bir (Cosθ, Sinθ) birim çemberi üzerindeki temel rotasyon formülü şöyledir:

$$x' = x \cdot \cos(\theta) + z \cdot \sin(\theta)$$

$$z' = -x \cdot \sin(\theta) + z \cdot \cos(\theta)$$

3. Formülün Matematiksel Çıkarılışı

Bu formüller **Rotasyon Matrisi** dediğimiz yapıdan gelir. Y ekseni etrafında θ açısı kadar dönüşün matrisi şudur:

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

Bir P(x, y, z) noktasını bu matrisle çarptığımızda:

1. **Yeni X:** $(x \cdot \cos\theta) + (y \cdot 0) + (z \cdot \sin\theta)$ koddaki x hesabı.
2. **Yeni Y:** $(x \cdot 0) + (y \cdot 1) + (z \cdot 0)$ Y değişmeden kalır.
3. **Yeni Z:** $(x \cdot -\sin\theta) + (y \cdot 0) + (z \cdot \cos\theta)$ koddaki z hesabı.

4. Kodun Analizi

```
float x = p3.X * cos + p3.Z * sin; // X ve Z'nin bileşkesi yeni X'i verir
float z = -p3.X * sin + p3.Z * cos; // X ve Z'nin bileşkesi yeni Z'yi (derinliği) verir
```

Buradaki - işareti ve sin/cos yerleşimleri, dönüşün yönünü (saat yönü veya tersi) belirler. Koddaki yapı, standart "sağ el kuralına" uygun bir dönüş sağlar.

5. Çalışmanın Önemi:

Bu yöntem, ekran kartlarının (GPU) en temel çalışma prensibidir. Karmaşık 3D oyun motorları bile arka planda milyonlarca nokta için bu matris çarpımlarını saniyeler içinde binlerce kez yapar. Bu çalışma ile küçükte olsa bir kendi Render motorumuzu yazmış olduk.

PERSPEKTİF İZ DÜŞÜM

```
//PointF sınıfı x ve y den oluşan koordinat bilgisi içerir  Perspektif projeksiyon
PointF PerspektifIzDusur(Nokta3D p3, float Olcek, int Genislik, int Yukseklik)
{
    float Uzaklik = 3;
    float Faktor = Olcek / (p3.Z + Uzaklik);

    float x = p3.X * Faktor + Genislik / 2;
    float y = -p3.Y * Faktor + Yukseklik / 2;

    return new PointF(x, y);
}
```

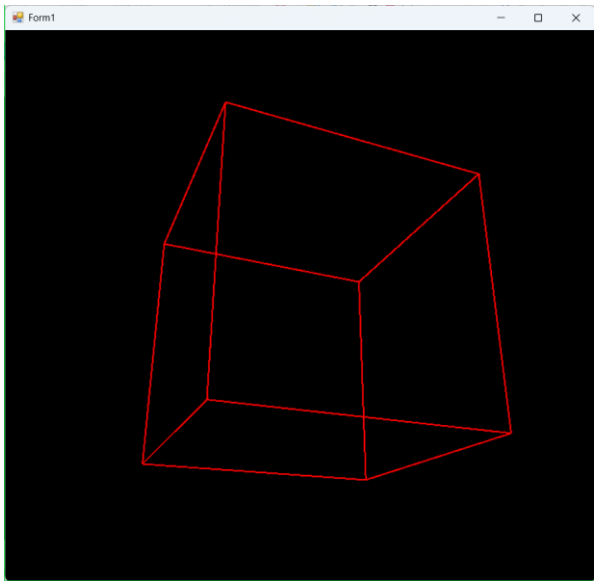
Bu metod, 3D dünyadaki bir nesnenin 2D ekran üzerinde "gerçekçi" görünmesini sağlayan sihirli dokunuştur. "Perspektif İzdüşüm" (Perspective Projection) dediğimiz bu işlem, nesnelerin bizden uzaklaştıkça küçülmesi ilkesine dayanır.

İşte kodun içindeki her bir satırın mantığı:

1. Uzaklik ve Faktor Hesabı (Derinlik Algısı)

```
float Uzaklik = 3;
float Faktor = Olcek / (p3.Z + Uzaklik);
```

SADELEŞTİRİLMİŞ VE İKİ EKSENDE DÖNÜŞ EKLENMİŞ KODLAR




```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace DersUygulamasi_1
```

```
{
    public partial class Form1 : Form
    {
        Timer Zamanlayici = new Timer();
        float Aci = 0;

        float CisminBoyu = 1000; //Cismin en uzak iki noktası arasının uzaklığı, yani
ekrana sığacak uzaklık

        Nokta3D[] DiziNokta3D = new Nokta3D[]
        {
            ////Orijin Küpün Köşesinde
            //new Nokta3D( 0, 0, 0),
            //new Nokta3D( 1000, 0, 0),
            //new Nokta3D( 1000, 1000, 0),
            //new Nokta3D( 0, 1000, 0),
            //new Nokta3D( 0, 0, 1000),
            //new Nokta3D( 1000, 0, 1000),
            //new Nokta3D( 1000, 1000, 1000),
            //new Nokta3D( 0, 1000, 1000)

            //Orijin Küpün Tam Ortasında
            new Nokta3D(-500, -500, -500),
            new Nokta3D( 500, -500, -500),
            new Nokta3D( 500,  500, -500),
            new Nokta3D(-500,  500, -500),
            new Nokta3D(-500, -500,  500),
            new Nokta3D( 500, -500,  500),
            new Nokta3D( 500,  500,  500),
            new Nokta3D(-500,  500,  500)
        };

        int[,] DiziKenar = new int[,]
        {
            {0,1},{1,2},{2,3},{3,0},
            {4,5},{5,6},{6,7},{7,4},
            {0,4},{1,5},{2,6},{3,7}
        };

        public Form1()
        {
            InitializeComponent();
            this.DoubleBuffered = true;
            Zamanlayici.Interval = 30;

            Zamanlayici.Tick += (s, e) =>
            {
                Aci += 0.05f;

                this.Invalidate(); //Formun OnPaint olayı tetiklenir
            };

            Zamanlayici.Start();
        }

        //A-- ON PAINT OLAYI - ÇİZİMİN GERÇEKLEŞTİĞİ OLAY
        protected override void OnPaint(PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            PointF[] DiziIzDusumNokta2D = new PointF[DiziNokta3D.Length];

            //****** 3 Boyutlu noktaları döndürüp, İz düşümünü alacak ***** esas iş burda
            for (int i = 0; i < DiziNokta3D.Length; i++)
            {
```

```
                // Y ekseninde döndür--------------
                var DonmusNokta3D = DondurY(DiziNokta3D[i], Aci);

                // X ekseninde döndür--------------
                DonmusNokta3D = DondurX(DonmusNokta3D, Aci);

                DiziIzDusumNokta2D[i] = PerspektifIzDusur(DonmusNokta3D);
            }

            Pen Kalem = new Pen(Color.Red, 2);

            for (int i = 0; i < DiziKenar.GetLength(0); i++)
            {
                int a = DiziKenar[i, 0];
                int b = DiziKenar[i, 1];
                g.DrawLine(Kalem, DiziIzDusumNokta2D[a], DiziIzDusumNokta2D[b]);
            }
        }

        //*********** Y ETRAFINDA DÖNDÜR ***********
        Nokta3D DondurY(Nokta3D p3, float Aci)
        {
            float cos = (float)Math.Cos(Aci);
            float sin = (float)Math.Sin(Aci);

            float x = p3.X * cos + p3.Z * sin;
            float z = -p3.X * sin + p3.Z * cos;

            return new Nokta3D(x, p3.Y, z);
        }

        // X ETRAFINDA DÖNDÜR***************
        Nokta3D DondurX(Nokta3D p3, float Aci)
        {
            float cos = (float)Math.Cos(Aci);
            float sin = (float)Math.Sin(Aci);
            float y = p3.Y * cos - p3.Z * sin;
            float z = p3.Y * sin + p3.Z * cos;
            return new Nokta3D(p3.X, y, z);
        }

        PointF PerspektifIzDusur(Nokta3D p3)
        {
            // Eğer küp 0 noktasındaysa ve biz de 0 noktasındaysak düzgün göremeyiz.
            // Küpü kameradan biraz uzağa itmemiz lazım (Z ekseninde öteleme)
            float KameraUzakligi = 2500f;
            float z_derinlik = p3.Z + KameraUzakligi;

            // Odak uzaklığı (Ekranın derinlik hissi)
            float Odak = 800f;

            // Perspektif formülü: Koordinat / Derinlik
            float KucultmeFaktoru = Odak / z_derinlik;

            float x = p3.X * KucultmeFaktoru;
            float y = p3.Y * KucultmeFaktoru;

            // Ekran ortasına hizalama
            x = x + this.Width / 2;
            y = y + this.Height / 2;

            return new PointF(x, y);
        }
    }
```

```
    public class Nokta3D
    {
        public float X, Y, Z;

        public Nokta3D(float x, float y, float z)
        {
            X = x;
            Y = y;
            Z = z;
        }
    }
}
```

6. Konunun Ana Hatları:

1 Noktalar → küp oluşur

2 Kenarlar → çizgi ile bağlanır

3 Döndürme → matematik

4 Projeksiyon → ekrana düşürme

7. İlave Konular:

* RotateY yerine RotateX de döndür

* Küp farklı herhangi bir eksende döndür

8. İlerisi İçin

- Mouse ile kontrol edilen küp

- Yüzey doldurma (solid cube)

- Işıklandırma (lighting)