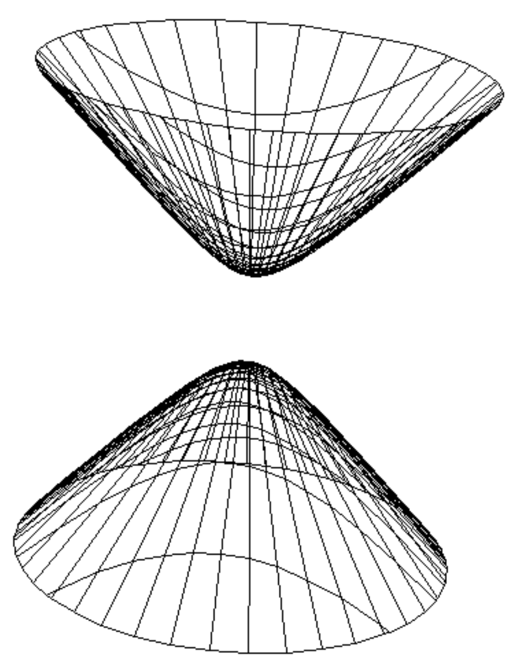
ÇOK BOYUTLU CİSİMLERİN İZDÜŞÜMLERİNİN VE ARAKESİTLERİNİN ALINMASI



Eser Aygün - Işık Barış Fidaner

2000-2001

İzmir

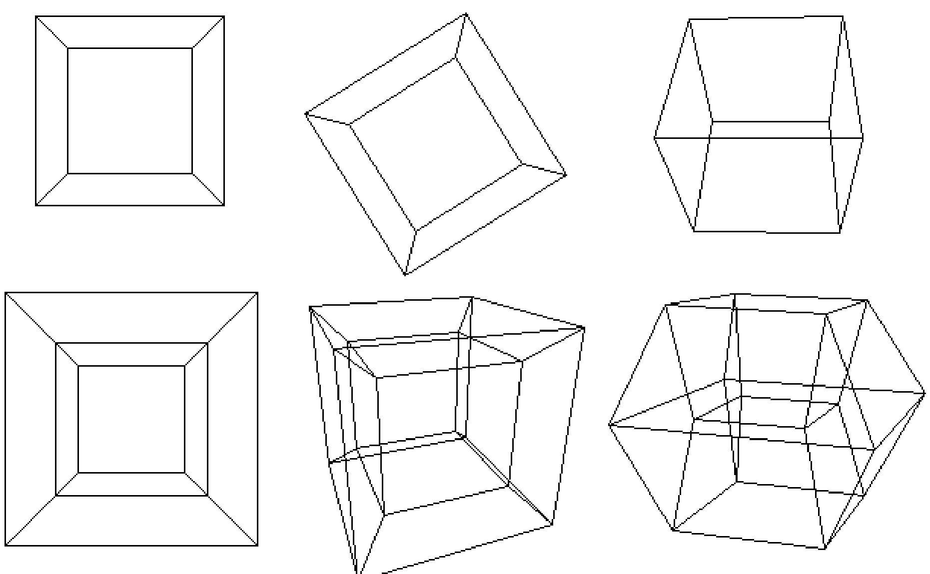
İçindekiler

Giriş 1
Amaçlar 2
Araçlar 2
Çok Boyutlu Uzay Matematiği 3
Vektör ve Matris Dönüşümleri 3
Birim Matris (Identity) 3
Öteleme Matrisi (Transition) 4
Döndürme Matrisi (Rotation) 4
İzdüşüm (Projeksiyon) 4
Arakesit 5
Veri Yapısı 6
Dosya Yapısı 7
Algoritmalar 8
Dönüşüm Algoritması 8
İzdüşüm Algoritması 8
Arakesit Algoritması 8
Cisim Yaratma Algoritmaları 9
Prizma Yaratma 9
Silindir Yaratma 10
Piramit Yaratma 11
Çizim Algoritması 11
Arayüz 12
Sonuç ve Tartışma 14
Ek 1: Programın Çıktıları 15
Ek 2: Birimler 19
Kaynaklar 21
Teşekkür 22

Giriş

Matematiğin insanlık kadar eski tarihi boyunca insanlar sayıları; önce eşyalarını saymak için tek tek, daha sonra tarlalarını ölçmek için ikişerli, ve nihayet bilimsel çalışmaları için üçerli kullandılar. Günümüzde karmaşık matematik problemlerinin çözümü ve doğa olaylarını açıklamak için çok daha fazla sayıya ihtiyacımız var. Artık eminiz ki evren en az dört boyutludur (daha fazla olabilir). Örneğin evreni anlamaya yönelik olarak ortaya çıkan süpersicim (*superstring*) teorisi gibi fiziksel açıklamalar kimi zaman onu yirmi altı boyutlu kabul etmek zorunda kalmıştır. Bu çalışmayla biz, bilgisayarların sunduğu olanakları değerlendirerek, gelişen bu yeni bakış açılarına biraz olsun destek olmak istedik.

Çok boyutlu cisimler, çok boyutlu uzaylarda yer alan nokta kümeleridir. Çok boyutlu uzaylar kavramının içine nokta, doğru, düzlem ve ikiden çok boyutlu uzaylar girer. Bunları üç boyuta kadar görürüz ve kolayca tasarlarız; fakat dört boyut ve sonrasını anlamak için, anlayabildiğimiz kısmı incelemek gerekir. Örneğin üç boyutlu uzayın iki boyutlu uzaydan (düzlemden) farkı, yukarı aşağı ve sağa sola harekete ek olarak, bu iki doğrultuya dik bir eksende ileri geri hareket edilebilmesidir. Aynı fark dört boyutlu uzay ile üç boyutlu uzay arasında vardır. Bir boyutlu doğrudaki *uzunluk* boyutuna düzlemde *yükseklik* boyutu, üç boyutlu uzayda *derinlik* boyutu, dört boyutlu uzayda yeni bir boyut daha –belki de *zaman*- eklenir. Bunlar x, y, z, t eksenlerine karşılık gelir. Boyut arttıkça uzaya yeni eksenler katılır. Bu kavramları anlamak için üç boyutlu basit bir şekil olan *küp* ile dört boyutlu karşılığı olan *hiperküp*ü karşılaştıralım. Üç boyutlu küp 6 tane kareden, hiperküp ise 8 tane küpten oluşur:



Üstteki şekiller küpü, alttaki şekiller hiperküpü gösteriyor. Soldaki iki şekil bu cisimlerin döndürülmemiş görüntülerdir. Ortadaki şekillerde, küp sadece ilk iki eksende döndüğü için hala iç içe iki kare gibi görünmektedir. Hiperküp ise ilk üç eksende döndürülmüştür ve iç içe iki küp gibi görünür. Sağdaki şekillerde, küp üçüncü eksende, hiperküp dördüncü eksende dönmüştür. Küpün bu değişiminin düzlemde gerçekleştiğini düşünseydik, içteki karenin biçimini kaybedip yavaş yavaş büyüyerek dışarı çıktığını, dıştaki büyük karenin bükülerek içeri girdiğini görecektik. Gözlerimizle üç boyutu algılayabildiğimiz için karenin biçimini kaybetmediğini, uzaklığı değişirken eğim kazandığı için öyle göründüğünü anlayabiliyoruz. Fakat hiperküpü döndürünce içteki küpün nasıl bükülerek dışarı çıktığını anlamak zorlaşıyor. Çünkü, küpün içindeki karenin uzakta olup küçük görünmesi gibi hiperküpte de içte görünen küçük küp aslında içte değil, *uzakta*dır. Küpü biri yakın biri uzak iki kare arasında kalan hacim olarak düşünebiliriz. Bu durumda hiperküp, dördüncü boyutta biri *yakın* biri *uzak* iki küp arasında kalan *hiperhacim* olur.

Amaçlar

Bu projenin amaçları;

a) Çok boyutlu geometrik cisimleri bilgisayar ekranında hareketli olarak göstererek üç boyutlu beynimiz için daha anlaşılır kılacak bir bilgisayar programı yazmak,
b) Programda cisimlerin arakesitlerini alarak, alt uzaylarındaki yaratıkların onları nasıl görebileceklerini; örneğin dört boyutlu bir küpün üç boyutlu uzayımızdan geçerken sekiz yüzlü, dört yüzlü veya sadece bir küp gibi görünebileceğini görsel olarak ortaya koymak,
c) Programda ele alınacak çok boyutlu cisimleri, küp, küre gibi daha basit cisimlerden –kareden prizma, çemberden küre, üçgenden piramit oluşturur gibi- türetmek için izlenecek yöntemler bulmak,
d) Teoride kalmış matematiksel ve fiziksel düşünceleri uygulamaya geçirmek

olarak özetlenebilir.

Çalışmayı bu amaçlar çerçevesinde ayırdığımız altı ana bölümde yürüttük:

- **Veri yapısı:** Çok boyutlu bir cisim; nokta, kenar, yüzey ve hacimler içerdiği gibi bunların sınırladığı daha üst boyutlu elemanlar da taşır (*hiperhacim* gibi). Bu yüzden, cismin bileşenlerinin ve bileşenler arası ilişkilerin bir tür veri yapısında tutulması gerekir.
- **Dönüşümler:** Üç boyutlu cisimleri döndürmek ve ötelemek için kullanılan matris dönüşümlerini çok boyuta uyarlamak gerekir.
- **İzdüşüm alma:** Üç boyut motorlarında sanal bir uzayı ekrana yansıtmak için kullanılan izdüşüm yöntemlerinin bir benzeri, çok boyutlu cisimleri görmek için gereklidir.
- **Arakesit alma:** Çok boyutlu bir cismin alt uzaylarla kesişim kümeleri olan arakesit cisimlerini oluşturmak ve incelemek, onu anlamak için önemli bir adım olacaktır.
- **Cisim yaratma:** Küp, küre, silindir gibi cisimlerin düzenliliği, bilgisayarda bu cisimleri yaratabilecek işlevler yazılabileceğini düşündürür. Aynı durum *simplex*, *hiperküp*, *hiperkoni* gibi daha fazla boyutlu düzgün şekiller için geçerlidir.
- **Görüntüleme:** Cisimlerin izdüşümlerini aldıktan sonra yapılacak iş, onları ekrana çizmektir. Karmaşık cisimleri hareket ettirirken hızdan kazanmak için bu konu üzerinde durduk.

Bu altı bölüm birbirinden bağımsız olarak ele alınabilir; fakat uygulamada birlikte çalışmalıdırlar. Yazılacak bir programla çok boyutlu cisimleri eksiksiz bir şekilde incelemek için hepsine gerek vardır.

Araçlar

- **Windows:** Grafik yeteneklerinden dolayı *Windows*'u DOS'a tercih ettik.
- **Borland Delphi 5:** Programlama dili olarak, anlaşılırlık ve kullanışlılık açısından *Borland Delphi 5*'i seçtik.
- **DelphiX:** Çizim aşamasında DirectX'i kullanabilmek için, bu işi kolaylaştıran *DelphiX for Delphi 5* bileşenlerinden yararlandık.

Çok Boyutlu Uzay Matematiği

Çok boyutlu uzayı kavramak için öncelikle üç boyutlu uzayı inceledik. Bildiğiniz gibi üç boyutlu uzayda bir vektör üç koordinat değeri ile belirtilir (`TValue` tipi en geniş reel sayı tipini ifade etmektedir):

```
TVector3D = record
      x, y, z: TValue;
end;
```

Yine üç boyutta dönüşümler için 4x4'lük matrisler kullanılır:

```
TMatrix3D = array [0..3, 0..3] of TValue;
```

Bu kavramları çok boyutlu uzay için genelleştirdiğimizde şunları elde ettik:

```
TCoords = array [0..MAX_DIM_COUNT - 1] of TValue;
TVector = record                 * vektör tipi
      DimCount: Integer;         * vektörün boyut sayısı
      Coords: TCoords;           * vektörün koordinatları
end;

TCells = array [0..MAX_DIM_COUNT, 0..MAX_DIM_COUNT] of TValue;
TMatrix = record                 * matris tipi
      DimCount: Integer;         * matrisin boyut sayısı
      Cells: TCells;             * matrisin hücreleri
end;
```

Aslında koordinatları dinamik olarak boyutlandırılabilir bir yapıda tutabilecekken, hızdan kazanç sağlamak için sabit sayıda koordinat (`MAX_DIM_COUNT` tane) kullandık. Ayrıca her vektörün ve matrisin yapısına boyut sayısını tutan bir değişken (`DimCount` değişkeni) ekledik. Boyutları ise x, y, z gibi harflerle adlandıramayacağımızdan b0, b1, b2, b3, ... şeklinde bir gösterim kullandık (burada b0 x'e, b1 y'ye, b2 de z'ye karşılık gelir).

Vektör ve Matris Dönüşümleri:

Cisme uygulanmak istenen dönüşüm matrisleri sırayla çarpılarak genel bir matriste toplanır (matris çarpımında birleşme özelliği olmadığı için çarpım sırası önemlidir). Daha sonra cismin tüm vektörleri bu matris ile çarpılarak dönüşüm sağlanır.

Birim Matris (Identity):

Birinci köşegeni üzerindeki hücreleri 1, geri kalan hücreleri 0 olan matristir. Matris çarpma işleminde etkisiz (birim) elemandır. Örneğin aşağıdaki 4x4'lük birim matris, üç boyutlu cisimlerin dönüşümlerinde kullanılır ve herhangi bir üç boyutlu vektörle çarpıldığında sonuç, vektörün kendisi olur:

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

Öteleme Matrisi (Transition):

Öteleme matrisi bir vektöre uygulandığında vektörü son satırındaki değerlere göre öteler. tx, ty, tz öteleme değerleri olmak üzere üç boyut için 4x4'lük bir öteleme matrisi ve bir vektörle çarpımı aşağıdaki gibidir:

$$[x \quad y \quad z \quad 1] \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ tx & ty & tz & 1 \end{bmatrix} = [x+tx \quad y+ty \quad z+tz \quad 1]$$

Döndürme Matrisi (Rotation):

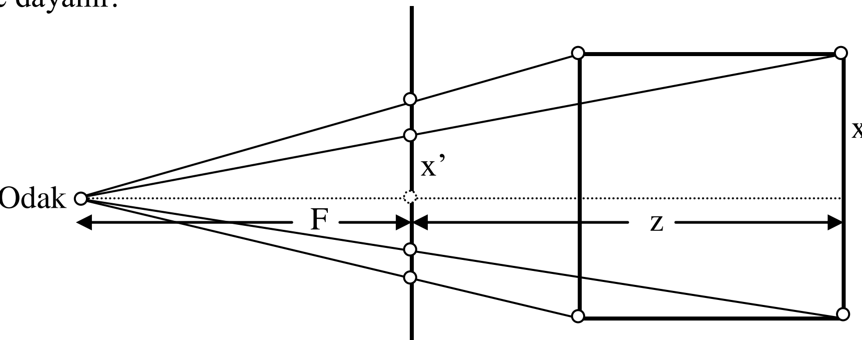
Çarpıldığı vektörü, uzayda iki eksenin belirttiği bir düzlem üzerinde θ açısı kadar döndüren matristir. Birinci eksen i, ikinci eksen j olmak üzere; birim matrisin [i, i] hücresine sinθ, [j, j] hücresine -sinθ ve [i, j] ile [j, i] hücrelerine cosθ yazılmasıyla oluşturulur. Üç boyutlu bir vektörün b0-b2 (x-z) düzleminde θ açısı kadar döndürülmesinde kullanılan bir döndürme matrisi aşağıda verilmiştir:

$$\begin{bmatrix} +\sin\theta & 0 & +\cos\theta & 0 \\ 0 & 1 & 0 & 0 \\ +\cos\theta & 0 & -\sin\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

İzdüşüm (Projeksiyon):

İzdüşüm, çok boyutlu bir cismin alt boyutlarına indirgenmesidir. Aslında bütün üç boyut motorlarının yaptığı iş de üç boyutlu cisimlerin ekran üzerine izdüşümünü almaktır. Bizim yaptığımız ise bu işlemin çok boyutlu cisimler için genelleştirilmesidir.

Birçok izdüşüm yöntemi arasından, gerçekçiliği ve uygulanmasının kolaylığı sebebiyle düzlemsel perspektif izdüşüm yöntemini seçtik. Bu yöntem, noktaları ekran düzleminin arkasındaki bir odak ile birleştiren doğruların bu düzlemi kesen noktalarının işaretlenmesine dayanır.



Üç boyut için genel formülü, basit üçgen benzerliklerinden yararlanarak aşağıdaki gibi bulduk ([x y z] üç boyutlu vektör, [x′ y′] iki boyutlu izdüşüm vektörü ve F odak uzaklığı olmak üzere):

$$x' = x\frac{F}{F+z} \quad , \quad y' = y\frac{F}{F+z}$$

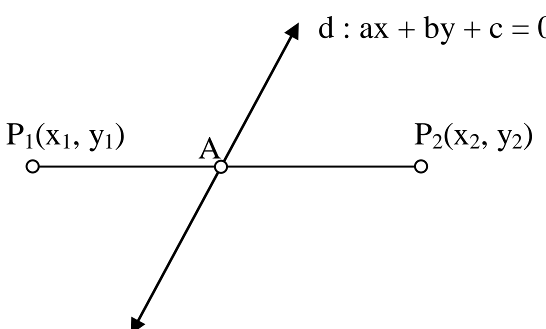
Buna göre, n boyutlu bir vektörün n-1 boyuta izdüşümü ise şu şekildedir:

$$c_0' = c_0\frac{F}{F+c_{n-1}} \quad , \quad c_1' = c_1\frac{F}{F+c_{n-1}} \quad , \ldots , \quad c_i' = c_i\frac{F}{F+c_{n-1}} \quad , \ldots , \quad c_{n-2}' = c_{n-2}\frac{F}{F+c_{n-1}}$$

Bu işlem birçok kez tekrarlanarak vektör istenen boyuta indirgenir.

Arakesit:

Çok boyutlu bir cisimle bir alt uzayın ortak elemanlarına bu cismin uzayla arakesiti denir. Dairenin doğruyla, koninin düzlemle, hiperküpün üç boyutlu uzayla arakesitleri buna örnektir. n boyutlu bir cismin n-1 boyutlu uzayla arakesitini alırken, bu cismi oluşturan doğru parçalarının uzayla kesişimlerinden yola çıktık. Matematiksel yöntemi bulmak için, analitik düzlem üzerinde bir doğru parçası ile bir doğrunun kesişiminin bulunuşunu örnek aldık:



P_1 ve P_2 noktalarının doğruya olan uzaklıkları oranı, A noktasına olan uzaklıkları oranına eşittir. Bir noktanın bir doğruya olan uzaklığı, bu doğrudaki kuvvetinin mutlak değeriyle orantılıdır. Dolayısıyla, A noktasının $[P_1P_2]$ doğru parçasını hangi oranla böldüğünü bulmak için P_1 ve P_2 noktalarını doğru denklemine koyarız:

$$k_1 = a.x_1 + b.y_1 + c$$
$$k_2 = a.x_2 + b.y_2 + c$$

`* k1 = 0 ya da k2 = 0`

Noktalardan birinin kuvveti sıfıra eşit ise; kesişim noktası o noktadır.

`* k1 = 0 ve k2 = 0`

İkisi de sıfıra eşit ise; kesişim bir nokta değil, doğru parçasının kendisidir.

`* sgn(k1) = sgn(k2)`

k_1 ve k_2 aynı işaretli ise; iki nokta doğrunun aynı tarafındadır ve kesişim boş kümedir (k_1 ve k_2 sıfırdan farklı).

`* sgn(k1) = -sgn(k2)`

k_1 ve k_2 ters işaretli ise; iki nokta doğrunun farklı taraflarındadır ve kesişim noktası k_1 ile k_2'nin oranından bulunur (k_1 ve k_2 sıfırdan farklı).

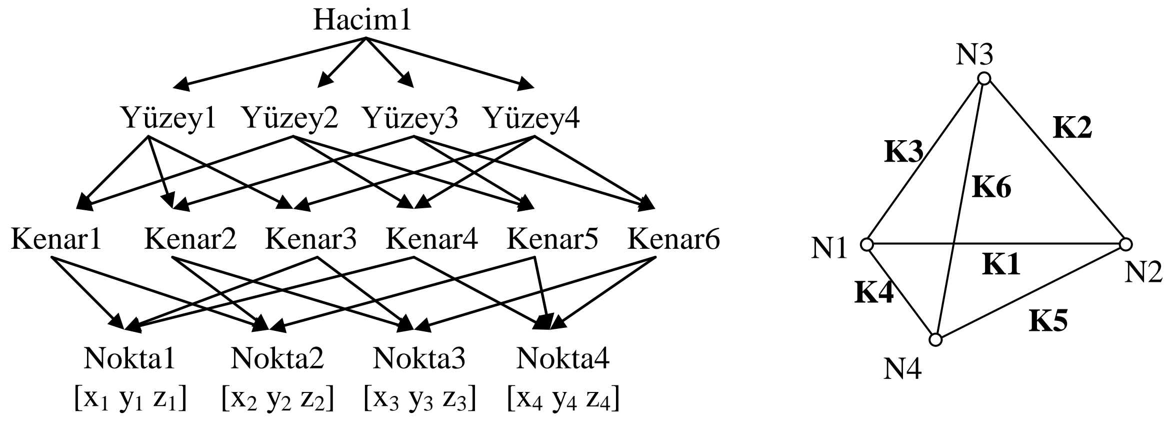
İki boyutlu düzlemde doğru denklemini n boyutlu uzaydaki n-1 boyutlu uzayın denklemi olarak genellersek (a_i'ler sabit, c_i'ler vektör koordinatı olmak üzere):

$$a_0c_0 + a_1c_1 + a_2c_2 + ... + a_ic_i + ... + a_{n-1}c_{n-1} + a_n = 0$$

denklemine ulaşırız. Düzlem için uyguladığımız yöntemi bu denklemi kullanarak çok boyuta uyarlayabiliriz.

Veri Yapısı

Üç boyutlu her cisim noktalardan, noktaların sınırladığı kenarlardan, kenarların sınırladığı yüzeylerden ve yüzeylerin sınırladığı hacimlerden oluşur. Bunlar cismin sıfır, bir, iki ve üç boyutlu elemanlarıdır. Her hacim yüzeyler, her yüzey kenarlar ve her kenar da kendini belirten iki nokta içerir. Yani nokta dışında her eleman kendini oluşturan alt boyuttaki elemanların bir listesini barındırır. Noktalar ise birer vektörden ibarettir. Buna göre düzgün dört yüzlü cisminin yapısı aşağıdaki gibidir:




```
PElement = ^TElement;
PELItem = ^TELItem;

TELItem = record                    * eleman listesi düğümü tipi
      Next: PELItem;                * sonraki düğüm
      Element: TElement;            * düğümün belirttiği eleman
end;

TElement = record                   * eleman tipi
      Dim: Integer;                 * elemanın boyutu
      Elements: PELItem;            * elemanın eleman listesinin başı (nokta değilse)
      Vector: PVector;              * elemanın vektörü (nokta ise)
end;
```

Bir elemanın boyutunu, içerdiği `Dim` değişkeni belirler. Noktalar dışındaki her eleman, kendini oluşturan elemanları tek bağlı bir liste içerisinde tutar. Listedeki her düğüm `Next` ile bir sonraki düğümü, `Element` ile listedeki bir elemanı gösterir. `Element^.Elements` ile gösterilen düğüm (*item*) bu listenin başı (*head*) olduğu için hiçbir elemanı göstermez. Böylece listenin ilk elemanı `Elements^.Next^.Element` olur. Listedeki son düğümün `Next` değişkeni boştur (*nil*).

Cisimleri tutmak için `TObj` isminde bir sınıf (*class*) yarattık:

```
TObjElementList = array [0..MAX_DIM_COUNT] of PELItem;

TObj = class
public
      Coords: TCoords;              * cismin yer vektörünün koordinatları
      Elements: TObjElementList;    * cismin eleman listeleri
      { metotlar ve özellikler... }
end;
```

Bu sınıf içerisinde elemanları boyut sayılarına göre tuttuk. Yani `TObj.Elements[0]` noktaları, `TObj.Elements[1]` kenarları, `TObj.Elements[i]` `i` boyutlu elemanları içeren listedir. Başlangıçta bunun tersine bütün elemanları tek listede topladık; fakat arama ve dolaşma işlemlerinde büyük performans kaybıyla karşılaştığımız için aynı boyutlu elemanları gruplayan bu yapıyı tercih ettik. Ayrıca cismin uzaydaki yerini `Coords` değişkeninde tuttuk. İzdüşüm ve arakesit alma işlemlerinde bu koordinatları kullandık.

Dosya Yapısı:

Çok boyutlu cisimleri metin dosyaları şeklinde tuttuk. En başta cismin boyut sayısı bulunmak üzere, en büyük boyuttan itibaren, tüm elemanları üstte o boyuttaki eleman sayısının yazdığı gruplarda topladık. Nokta dışındaki elemanları belirtirken, kapsadığı elemanların sıra numaralarını kullandık. Noktaları ise vektörlerinin koordinatlarını yan yana yazarak gösterdik. Üç boyutlu bir küp dosyası aşağıdaki gibidir:

```
3                  * cismin boyut sayısı

1                  * hacim sayısı
 1 2 3 4 5 6       * ilk hacmi oluşturan yüzeylerin listesi

6                  * yüzey sayısı
 1 2 3 4           * ilk yüzeyi oluşturan kenarların listesi
 5 6 7 8           ...
 1 5 9 10          ...
 2 6 11 12         ...
 3 7 13 14         ...
 4 8 15 16         * altıncı yüzeyi oluşuran kenarların listesi

16                 * kenar sayısı
 1 2               * birinci kenarı oluşturan noktaların listesi
 3 4               ...
 1 3               ...
 2 4               ...
 5 6               ...
 7 8               ...
 5 7               ...
 6 8               ...
 1 5               ...
 3 7               ...
 2 6               ...
 4 8               * on ikinci kenarı oluşturan noktaların listesi

8                  * nokta sayısı
 50 50 50          * birinci noktanın koordinatları
 -50 50 50         ...
 50 -50 50         ...
 -50 -50 50        ...
 50 50 -50         ...
 -50 50 -50        ...
 50 -50 -50        ...
 -50 -50 -50       * sekizinci noktanın koordinatları
```

Algoritmalar

Dönüşüm Algoritması:

`TObj.Transform` metodu ile gerçekleştirilen dönüşüm işlemi cisimdeki tüm noktaların vektörlerini bir matris ile çarpmaya dayanır. Aşağıda bu yordamın kullanılışına ait bir örnek görmektesiniz:

```
mRotation(Mx, 0, 2, Pi/3.0);   * b0-b2 düzleminde 60° döndüren matrisi Mx'e yaz
Obj.Transform(Mx);             * Obj cismini Mx ile dönüşüme uğrat
```

Döndürme işlemlerinde cismin yer vektörü dikkate alınmaz ve cisim kendi merkezi etrafında döndürülür.

İzdüşüm Algoritması:

`TObj.ProjectTo` metodu, cismin noktalarının herhangi bir alt boyuttaki izdüşümünü alır. Bu işlem sırasında gerçek vektörleri değiştirmektense ayrı vektörler tutmayı ve izdüşüm işlemini bu vektörler üzerinde yapmayı tercih ettik. Böylece `TElement` yapısı şu hali aldı:

```
TElement = record
      Dim: Integer;
      Elements: PELItem;
      Vector, Projected: PVector;  * gerçek vektör ve izdüşüm vektörü (nokta ise)
end;
```

`ProjectTo` metoduna örnek bir kullanım şöyledir:

```
Obj.ProjectTo(2);        * Obj cisminin ikinci boyuttaki izdüşümünü alır
```

Ekrana çizim için `Projected` vektörlerini kullandık. Dolayısıyla gerçek vektörlerin çizim aşamasında hiçbir görevi olmadı.

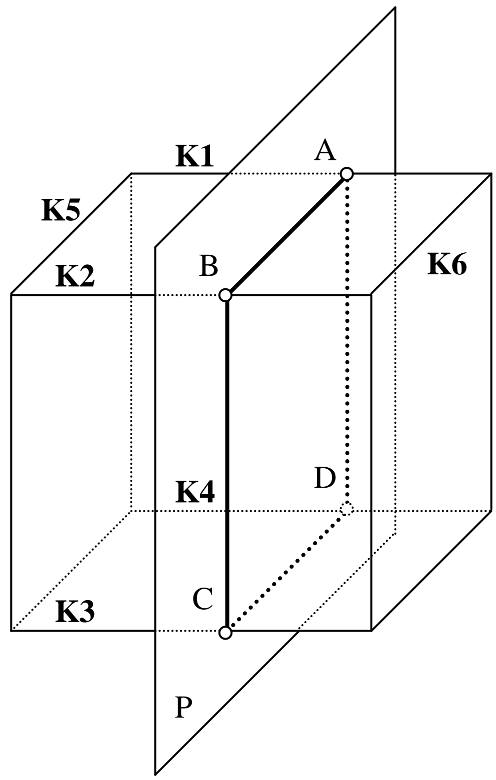
Arakesit Algoritması:

`TObj.IntersectTo` metodu kullanılarak cismin herhangi bir alt uzayı ile arakesiti alınır ve sonuç başka bir cisme yazılır. Böylece gerçek cisim korunmuş olur. Arakesit alınırken her seferinde bir alt uzayla işlem yapılır ve bu, cisim gerekli boyuta indirilene kadar sürer. Dolayısıyla bir cismin bir boyut aşağısı ile arakesitini alan yordamı yazmak yeterlidir.

Cisimdeki bir elemanın arakesiti, sahip olduğu elemanların arakesitlerini kapsayan bir elemandır. Bundan yola çıkarak ilk yaptığımız `IntersectElement`, sadece en büyük eleman için çağırılıyordu ve alt elemanlar için kendisini tekrar tekrar çağırarak çalışan özyinelemeli (*recursive*) bir işlevdi. Fakat ortak alt elemanlar için bu fonksiyon birden çok defa gereksiz olarak çağırılıyordu. Bu yüzden işleme kenarlardan başlayıp boyut boyut yükselen `IntersectObj` adında yeni bir yordam yazdık.

`IntersectObj` yordamı önce cisimdeki kenarlar için `IntersectElement` işlevini çağırarak alt uzayla kesişimlerini bulur ve bu kesişimleri arakesit cismine ekler. Daha sonra yüzeylerden başlayıp en üst boyuta kadar arakesit cisminin diğer elemanlarını yaratıp gerçek cisimdeki ilişkilerden yararlanarak listelerini düzenler. Dolayısıyla bir boyutu düzenlerken, bir alt boyuttaki her elemanın arakesitine ulaşabilmesi gerekir. Bunun için de her elemanda, kendi arakesitini işaret edecek `PElement` tipinde bir değişken olmalıdır. Bu değişkeni diğer algoritmalarda da çok amaçlı olarak kullandığımız için adını `Reserved` koyduk.

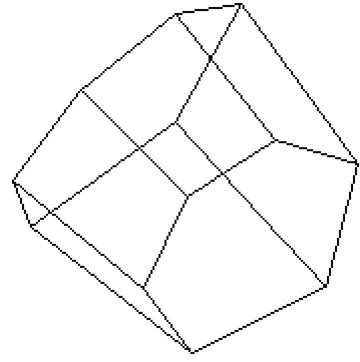
Yordam ilk aşamada her yüzey için, yüzeyin kapsadığı kenarların arakesitlerini birleştirerek yüzeyin arakesitini oluşturur. Bu işlemde her yüzey için birleştirilecek arakesitlerden sadece en yüksek boyutlu olanlar dikkate alınır. Örneğin arakesitler arasında hem kenar hem nokta varsa, nokta zaten kenarın içindedir. Böyle bir yüzey için kenarları ele almak yeterlidir. Son olarak yüzey için ele aldığımız arakesitleri bir üst boyutlu bir elemanda toplarız. Bu son eleman yüzeyin arakesiti olur. Yüzeylerin arakesitleri bu şekilde bulunduktan sonra aynı işlem hacim ve diğer üst boyutlar için yapılır.



Örneğin, üç boyutlu bir küpü P düzlemiyle kestiğimizi varsayalım (bkz. şekil). İlk önce kenarları `IntersectElement` işlevine yollarız. O da bize düzlemi kesmeyen kenarlar için boş değer (*nil*), K1, K2, K3 ve K4 kenarları için de birer nokta döndürür. Bu işlem boyunca kenarların `Reserved` değişkenlerine düzlemi kestikleri noktalar yazılır. Daha sonra yüzeyler teker teker ele alınır. Örneğimizde K1-K5-K2-K6 yüzeyi K1 için A noktasını, K2 için de B noktasını alarak [AB] kenarını oluşturur ve `Reserved` değişkenine yazar. Son olarak küp hacmi için, düzlemi kesen dört yüzeyin düzlemle arakesitleri olan dört kenarı birleştirerek arakesit cisminin ana elemanı olan ABCD karesini oluşturur.

Cisim Yaratma Algoritmaları:

Prizma Yaratma:

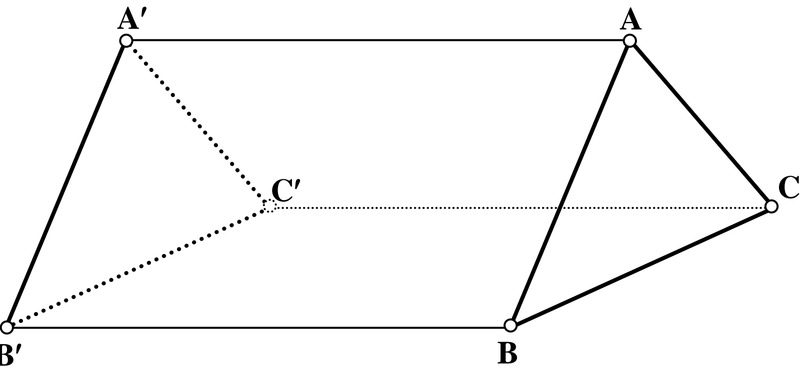


`TObj.MakePrism` metodu tarafından gerçekleştirilen prizma yaratma işlemi, cismi bir üst boyutta dik olarak kaydırmaya dayanır (bir kareyi z ekseninde dik olarak kaydırarak kare prizma oluşturmak gibi). Metot bunu, cisimden bir kopya çıkarıp kopyayı bir üst boyutta öteleyerek yapar. Öteleme işleminden sonra kopya cisim ile ilk cisim arasında gerekli bağlantıları kuran elemanları yaratıp kopyayla birlikte cisme ekler. Bağlantı yaratma işlemini `LinkObj` yordamı gerçekleştirir. Yaptığı; ilk cisimdeki

elemanları yeni elemanlar yaratarak, kopyalanmış eşlerine bağlamaktır. Bunun için her eleman eşini, içerdiği `Reserved` değişkeninde tutar.

Bir elemanı eşine bağlayan elemanı oluşturmak için alt elemanlarının bağlantılarını bilmek gerekir. Bu mantıkla yazdığımız özyinelemeli `LinkElement` işlevini sadece en büyük eleman için çağırmak yeterliydi. Alt elemanlar için kendisini çağırıp hepsini eşine bağlıyordu. Fakat `IntersectElement` işlevindeki sorun burada yine ortaya çıktı: Aynı eleman birçok kez eşine bağlanıyordu. Bunu çözmek için, bağlamaya noktalardan başlayan `LinkObj` yordamını yazdık.

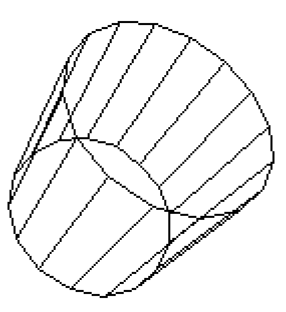
`LinkObj` yordamı, cismin noktalarını eşlerine birer kenarla bağlayarak işe başlar. Sonra kenarları ve diğer üst boyutları da birbirine bağlayarak devam eder. Bir kenarı bağlarken, bağladığı eşi de bir kenar olduğu için, bağlantı elemanı bir yüzey olmalıdır. Bu yeni yüzeyi yaratır ve listesine bu iki kenar dışında, ilk kenarın içerdiği noktaları eşlerine bağlayan kenarları da ekler. Bunun için her nokta, kendisini eşine bağlayan kenarı tutar. Daha sonra yüzeyleri ve daha üst boyutlu elemanları aynı şekilde eşlerine bağlar.

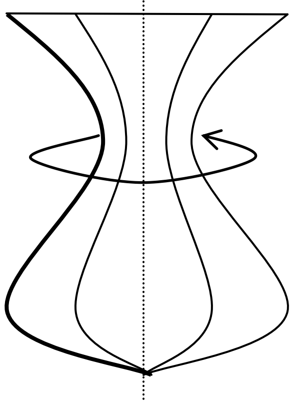


ABC üçgenini üçüncü boyutta kaydırılmış kopyası olan A′B′C′ üçgenine bağlarken `LinkObj` yordamının yapacağı ilk iş, üç noktayı eşlerine birleştiren [AA′], [BB′], [CC′] kenarlarını yaratmaktır. Daha sonra [AB] kenarını eşine bağlayacak yüzeyi yaratır. Bu yüzeye [AB] kenarını, eşi olan [A′B′] kenarını ve önceden yaratmış olduğu [AA′], [BB′] kenarlarını ekler. Aynı şekilde [BC] ve [CA] kenarları için de birer yüzey oluşturur. Son olarak ABC yüzeyini eşine bağlayacak hacmi yaratıp listesine ABC üçgenini, bu üçgenin eşini ve yine bu üçgenin kenarları için yaratmış olduğu üç yüzeyi ekler. Böylece üçgen cisminden üçgen prizma cismi yaratılmış olur.

Silindir Yaratma:

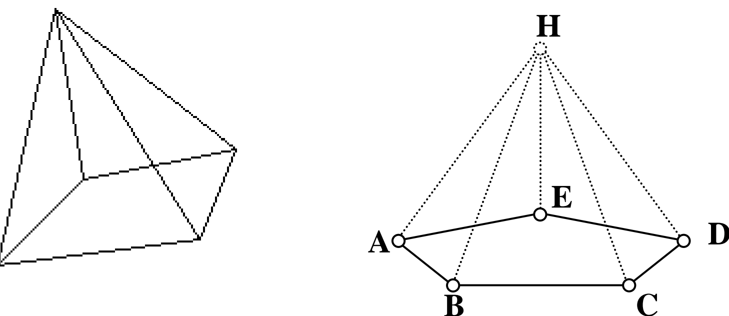
`TObj.MakeCylinder` metodu, cismi bir üst boyutta döndürerek yeni cisme ulaşır. Örneğin bir kareyi bir kenarı boyunca 360° döndürerek silindir, bir yarım daireyi çapı boyunca 360° döndürerek küre elde edilebilir. Bunun için metot cismin küçük açılarla döndürülmüş kopyalarını yaratıp `LinkObj` yordamıyla birbirlerine ardarda bağlar. Ne kadar küçük açılarla döndüreceğini `Detail` parametresiyle alır.





Piramit Yaratma:

`TObj.MakePyramid` metodu ile yapılır. Cismin ağırlık merkezinde bir nokta yaratır, bu noktayı bir üst boyuttaki eksende yükseltip bütün cismi bu noktaya bağlar. `TObj.CenterPoint` metoduyla yaratılan orta noktanın koordinatları cisimdeki noktaların aritmetik ortasından elde edilir. Bu noktanın yeni eksende yükseltildikten sonra cisme bağlanmasında `TObj.LinkPoint` metodu görev alır. Önce cismin noktalarını birer kenarla yeni noktaya bağlar. Daha sonra kenarları birer yüzeyle bağlar ve bu şekilde devam eder. İki boyutlu düzgün beşgeni piramitleştirecek olursak:



Metot, ilk olarak cismin orta noktasını yaratıp üçüncü koordinatına yüksekliği yazarak H noktasını elde eder. Sonra beşgendeki noktaları H noktasına bağlayan beş kenarı yaratır. [AB] kenarını H noktasına bağlamak için bir yüzey yaratır ve listesine [AB] ile A, B noktaları için önceden yarattığı [AH], [BH] kenarlarını ekler. Bu şekilde beş üçgen yüzeyi oluşturur. Son olarak beşgen yüzeyi H noktasına bağlayacağı hacmi yaratıp listesine beşgenin kendisi ile listesindeki kenarları H noktasına bağlayan üçgenleri ekler.

Çizim Algoritması:

`DrawObj` yordamı ile, cismin her kenarının iki noktasının `Projected` değişkeninde tutulan izdüşüm vektörleri `DrawLine` aracılığıyla birleştirilerek kafes biçimindeki görüntü oluşturulur.

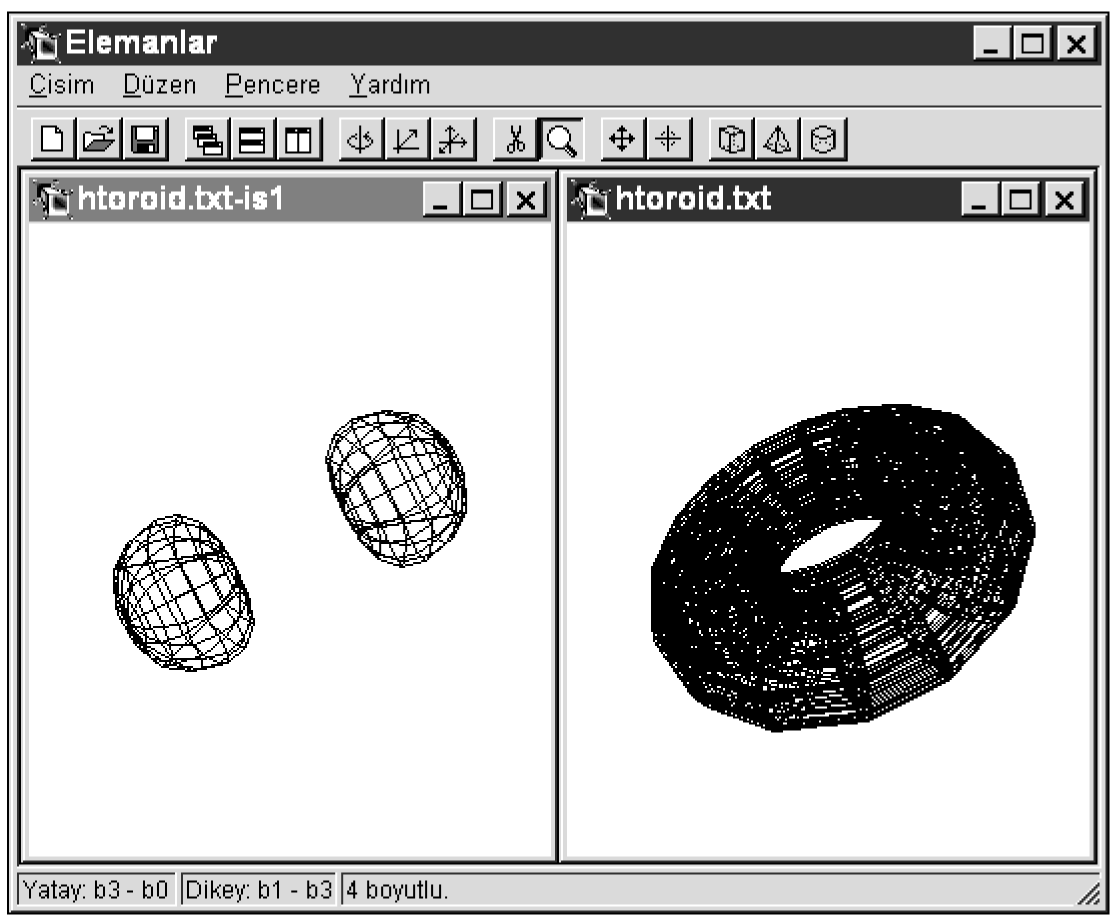
Çizimde gerekli performansı sağlayabilmek için, daha çok kelime işlem programlarına yönelik olan Windows'un GDI (*graphic device interface*) komutları yerine, doğrudan ekran belleğine erişime izin veren *DirectX*'i kullandık. Bu yüzden çizgi çizmek gibi temel komutları yeniden yaratmamız gerekti. Bilinen en hızlı çizgi çizme algoritması olan ve tam sayılarla işlem yapan *Bresenham* algoritmasını kullandık. Bu algoritmaya göre (x1, y1) noktasından (x2, y2) noktasına çizgi çizmek için şu işlemler yapılır (`dy < dx` iken):

```
dx := Abs(x2 - x1); dy := Abs(y2 - y1);
d := 2 * dy - dx;
dincr1 := 2 * dy; dincr2 := 2 * (dy - dx);
SetPosition(x1, y1);                    * (x1, y1)'e konumlan
for i := 0 to dx do
begin
      SetPixel(r, g, b);                * bulunduğun konumu boya
      if d < 0 then
            d := d + dincr1             * kırılma yoksa
      else
      begin                             * kırılma varsa
            d := d + dincr2;
            Down;                       * aşağı hareket et
      end;
      Right;                            * sağa hareket et
end;
```

Bu yöntemde çizgi soldan sağa doğru çizilir. `d` sayısı çizginin kırıldığı noktaların bulunmasında kullanılır. Her işlemde `dincr1` kadar arttılırılan `d` sıfırı geçtiğinde; bir aşağı inilir, `d` `dincr2` ile toplanır ve işlem bu şekilde devam eder.

Arayüz

"*Elemanlar*" programının arayüzü MDI'ye (*multi dialog interface*) dayanır:



Bu görüntüde *htoroid.txt* cisim dosyası açılmış ve bu dosyadaki hipertoroidin üç boyutlu arakesiti alınmıştır. Cisimler ve arakesitleri farklı alt pencerelerde görünür. Bu sayede arakesitlerin de arakesitleri alınabilir.

Araç çubuğunda yeralan düğmelerin yaptığı işler şöyledir:

1) **Yeni Cisim :** Bu düğme soldaki gibi bir diyalog kutusu açar. Program bu kutuda listelenmiş düzgün şekilleri cisim yaratma algoritmalarını kullanarak her boyutta yaratabilir. Bunları tek tek incelersek:

- **Küp:** Sıfır boyutlu küp noktadır. Diğer her küp, bir alt boyuttaki küpün kenar uzunluğu kadar prizmalaştırılmasıyla oluşturulur.
- **Küre:** İki boyutlu küre bir çemberdir. Çok boyutlu diğer her küre, bir alt boyuttaki boş yarım kürenin 360° silindirleştirilmesiyle oluşur.

- **Silindir:** Çok boyutlu silindir bir alt boyutundaki kürenin prizmalaştırılmasıyla oluşur.
- **Koni:** Çok boyutlu koni, bir alt boyuttaki iki kürenin birbirinden uzaklaştırılıp orta noktaya bağlanmasıyla oluşur.
- **Simplex:** Sıfır boyutlu simplex noktadır. Diğer her simplex, bir alt boyuttaki simplexin her noktasına eşit ve sabit uzaklıkta bir tepe noktası seçerek piramitleştirilmesi ile oluşur. Bu işlemin sonucu, bir boyutta doğru parçası, iki boyutta eşkenar üçgen, üç boyutta düzgün dörtyüzlüdür. Simplex aynı zamanda kendi boyutunun en basit düzgün şeklidir.
- **Nokta:** Her boyutta tek bir noktadır.
- **Toroid:** Bir alt boyutlu boş kürenin merkezden uzaklaştırılıp 360° silindirleştirilmesiyle oluşur. Üç boyutlu toroid simit biçimindedir.

2) **Aç :** Açılan diyalog kutusundan seçilen cisim dosyasını açar.

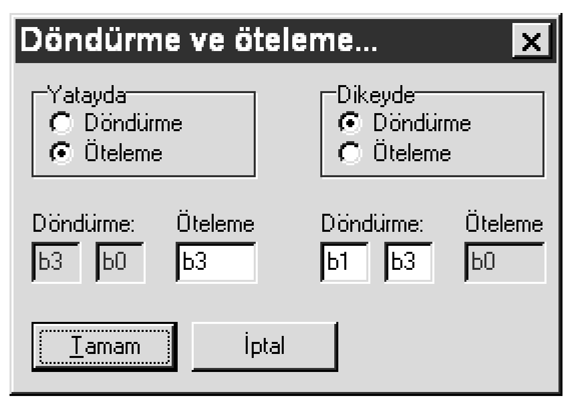
3) **Kaydet :** Etkin alt penceredeki cismi bir metin dosyası olarak kaydeder.

4) **Basamakla,** **Yatay Döşe,** **Dikey Döşe :** Pencereleri düzenlemeye yarar.

5) **Açılı Döndür :** Etkin penceredeki cisme tek bir döndürme işlemi uygulamak için yandaki diyalog kutusu açılır. Açı, derece cinsinden girilir ve döndürülecek düzlem iki eksen ile belirtilir.

6) **Yer Vektörü :** Bir diyalog kutusu aracılığıyla sayısal olarak etkin cismin yer vektörünü değiştirmeye yarar.

7) **Döndürme ve Öteleme :** Farenin sol düğmesi basılı tutularak cisim kolayca hareket ettirilebilir. Farenin hareketinin dikey ve yatay bileşenlerinin cismi nasıl etkileyeceği, bu düğmeye basılınca çıkan diyalog kutusuyla belirlenir. Fare hareketinin bir bileşenine döndürme görevi verilirse, uzayda iki eksenin belirttiği döndürme düzlemi; öteleme görevi verilirse, ötelemenin gerçekleşeceği eksen girilmelidir.



8) **Yeni Arakesit :** Cismin boyutundan küçük olması gereken arakesit boyutunu bir girdi kutusuyla kullanıcıdan aldıktan sonra, cismin verilen boyutlu uzayla arakesitini yeni bir pencerede yaratır.

9) **Arakesitleri Güncelle :** İki durumlu bir düğmedir. Basılıysa cisimler hareket ettikçe arakesitleri sürekli olarak güncellenir. Eğer basılı değilse arakesitler son alındıkları durumda kalırlar. Düğme normal olarak basılıdır.

10) **Boyutlandır :** Kullanıcının girdiği yüzdeye göre cismi boyutlandırır.

11) **Merkezle :** Cismin merkezini bütün noktalarının orta noktasına taşır.

12) **Prizmalaştır :** Prizma yapma algoritmasını kullanarak cismi prizmalaştırır.

13) **Piramitleştir :** Piramit yapma algoritmasını kullanarak cismi piramitleştirir.

14) **Silindirleştir :** Silindir yapma algoritmasını kullanarak cismi silindirleştirir.

Sonuç ve Tartışma

Çalışmalarımızın sonucunda 564 KB büyüklüğünde *elemanlar.exe* isimli Windows tabanlı bir program oluşturduk. Çok boyutlu cisimler yaratma, onları dosyadan okuma ve dosyaya yazma işlemlerini gerçekleştirebilen bu program aynı zamanda onların izdüşümlerini ve arakesitlerini alabilmekte, döndürme ve öteleme işlemlerini fare aracılığıyla gerçekleştirebilmektedir.

Yaptığımız testlerde 32 MB RAM'e sahip, Intel Pentium II işlemcili bir bilgisayar kullandık. Grafik işlemlerinin veriminde kıstas olarak, dosya büyüklüklerini ve -izdüşümleri çizgiler kullanarak çizdiğimiz için- cisimlerin bulundurdukları kenarların sayısını aldık. Ele aldığımız cisimleri kendi etraflarında döndürürken saniye başına düşen çerçeve sayısını (*framerate*) ölçtük. Bu işlemi her cisim için bir kere de arakesitiyle birlikte çizilirken yaparak sonuçları aşağıdaki çizelgede topladık:

Cismin adı	**Boyut sayısı**	**Kenar sayısı**	**Dosya boyutu**	**İzdüşüm (çerçeve / saniye)**	**İzdüşüm + arakesit (çerçeve / saniye)**
Küp	3	12	768 bayt	60	30
Hiperküp	4	32	2,23 KB	60	30
Koni	3	64	3,4 KB	60	30
Küre	3	272	15,1 KB	60	30
Hiperküp(7)	7	448	62,3 KB	60	7
Toroid	3	512	28,2 KB	60	20
Hipersilindir	4	688	47,2 KB	60	12
Hiperkoni	4	832	54,1 KB	40	10
Hiperküre	4	3600	261 KB	30	1
Hiperküp(10)	10	5120	2,23 MB	15	?
Hipertoroid	4	6656	496 KB	15	0,15

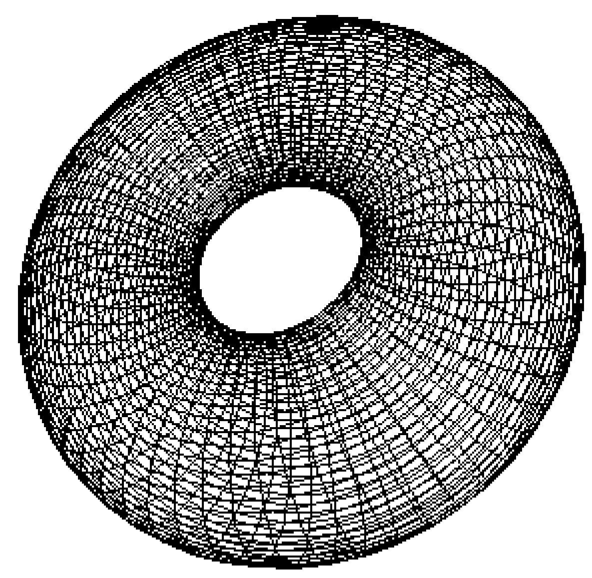
Arakesit alınırken aynı anda iki cisim birden (gerçek cisim ve arakesit cismi) çizildiği için hızda en az yarıya kadar düşme gördük. Arakesitli ölçümlerdeki daha büyük yavaşlamaları ise bazı cisimlerde (örneğin on boyutlu hiperküp) kenar sayısının, daha çok boyutlu elemanların sayılarına oranla çok küçük kalmasına bağlayabiliriz. Çünkü çizim işleminin tersine, arakesit alırken her boyuttaki elemanın etkisi vardır.

Gözümüz saniyede yaklaşık 25 çerçeve yakaladığı için üç ve daha az boyutlu tüm cisimlerde ve çoğu dört boyutlu cisimde istediğimiz performansı sağladık diyebiliriz.

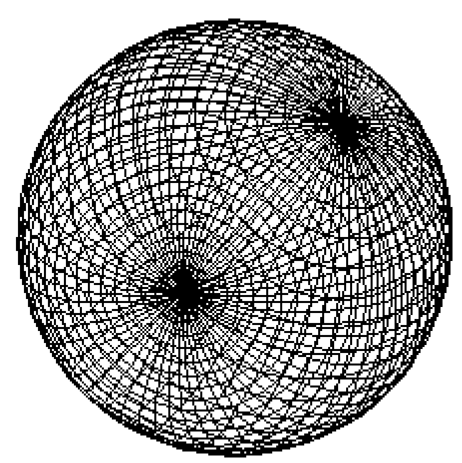
Son olarak, kenarları çizmek yerine yüzeyleri doldurmayı denediysek de üç boyutun üzerinde yeterli hıza erişemediğimiz için bu yönteme geçişi erteledik. Yine gelecekte programa eklemeyi düşündüğümüz bir özellik cisim yaratma bölümüne, girilen çok boyutlu fonksiyonu oluşturan bir seçenek eklemek. Bunu da zaman kıtlığı sebebiyle sonraya bıraktık.

Ek 1: Programın Çıktıları

Çeşitli Üç Boyutlu Cisimler

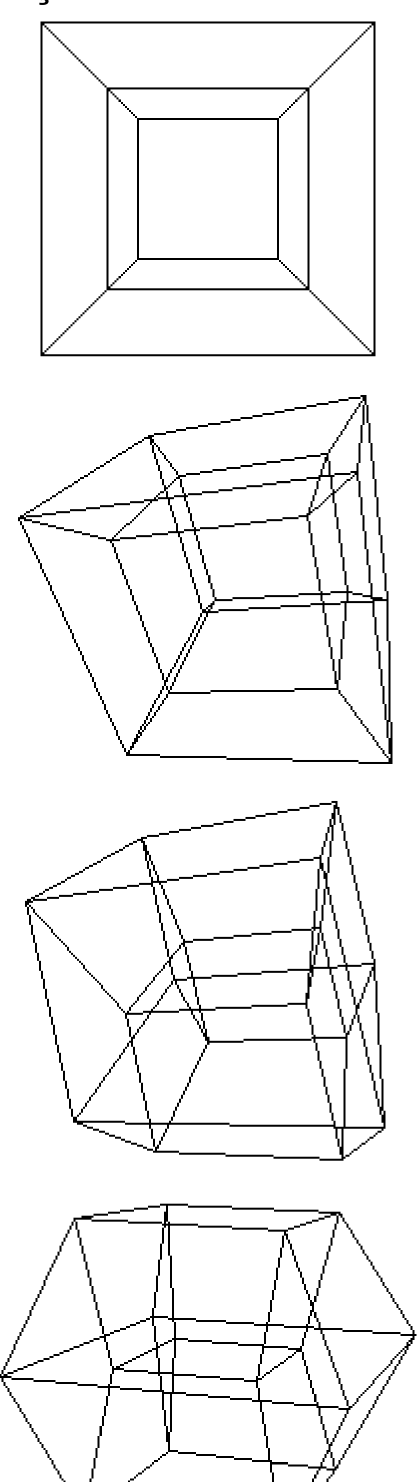


Üç boyutlu toroid:
Hacim sayısı: 1
Yüzey sayısı: 4096
Kenar sayısı: 8192
Köşe sayısı: 4096



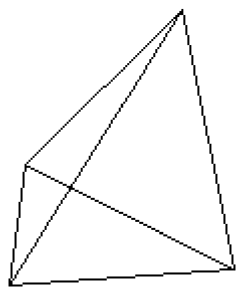
Üç boyutlu küre:
Hacim sayısı: 1
Yüzey sayısı: 2048
Kenar sayısı: 4032
Köşe sayısı: 1986

Dört Boyutlu Küpün (Hiperküp) Çeşitli Açılardan Görünümü

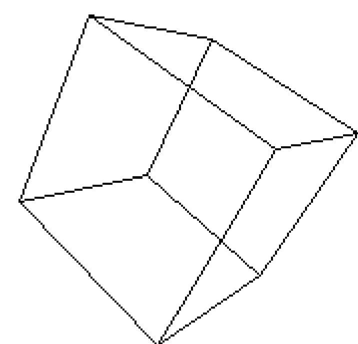


Dört boyutlu küp:
Hacim sayısı: 8
Yüzey sayısı: 24
Kenar sayısı: 32
Köşe sayısı: 16

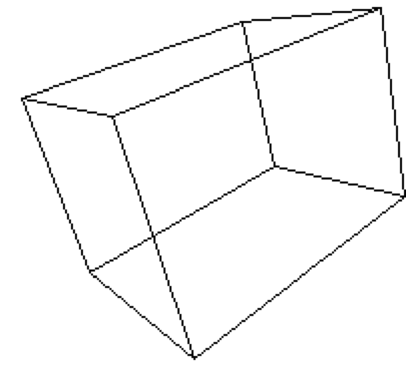
Dört Boyutlu Küpün (Hiperküp) Üç Boyutlu Uzayla Arakesitleri



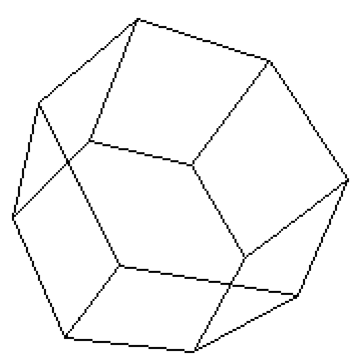
Düzgün dört yüzlü:
Hacim sayısı: 1
Yüzey sayısı: 4
Kenar sayısı: 6
Köşe sayısı: 4



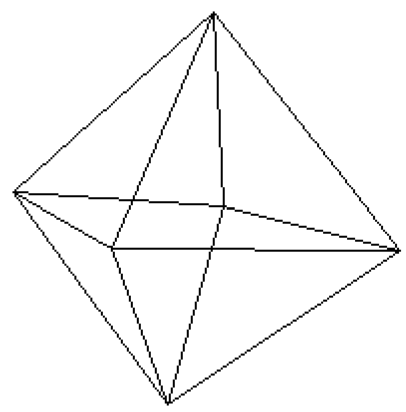
Düzgün altı yüzlü (küp):
Hacim sayısı: 1
Yüzey sayısı: 6
Kenar sayısı: 12
Köşe sayısı: 8



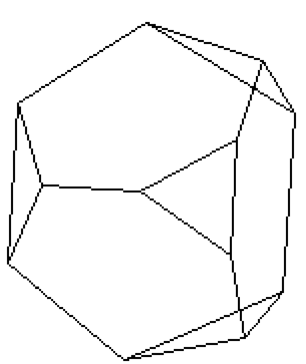
Kare Prizma:
Hacim sayısı: 1
Yüzey sayısı: 6
Kenar sayısı: 12
Köşe sayısı: 8



Altıgen Prizma:
Hacim sayısı: 1
Yüzey sayısı: 8
Kenar sayısı: 18
Köşe sayısı: 12

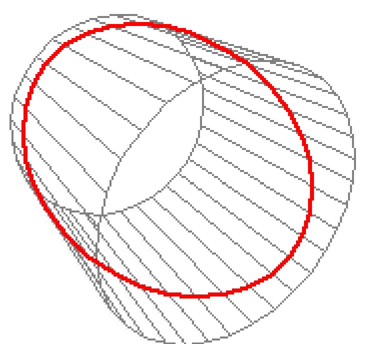


Düzgün sekiz yüzlü:
Hacim sayısı: 1
Yüzey sayısı: 8
Kenar sayısı: 12
Köşe sayısı: 6

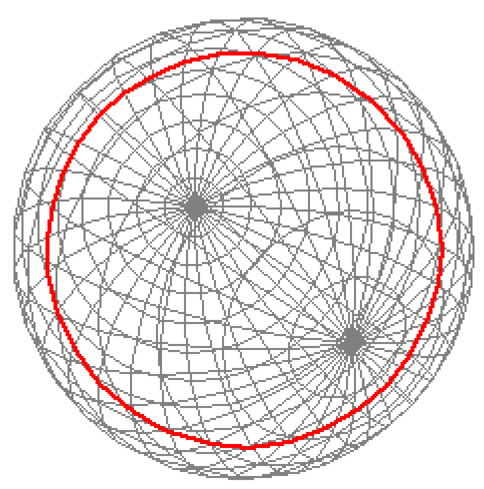


Kesik dört yüzlü:
Hacim sayısı: 1
Yüzey sayısı: 8
Kenar sayısı: 18
Köşe sayısı: 12

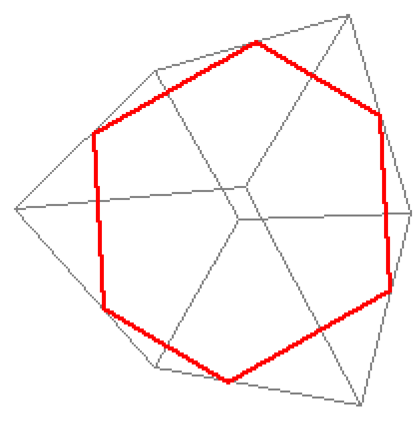
Çeşitli Üç Boyutlu Cisimler Ve İki Boyutlu Arakesitleri



Silindir ve arakesiti elips

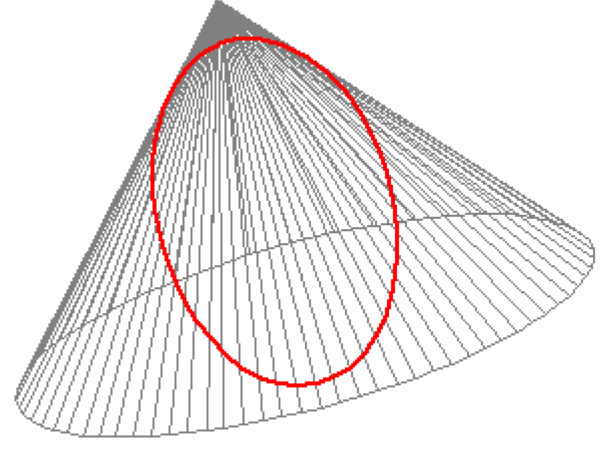


Küre ve arakesiti çember

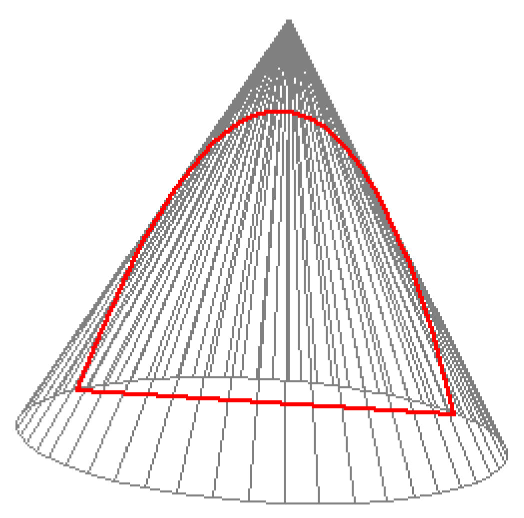


Küp ve arakesiti altıgen

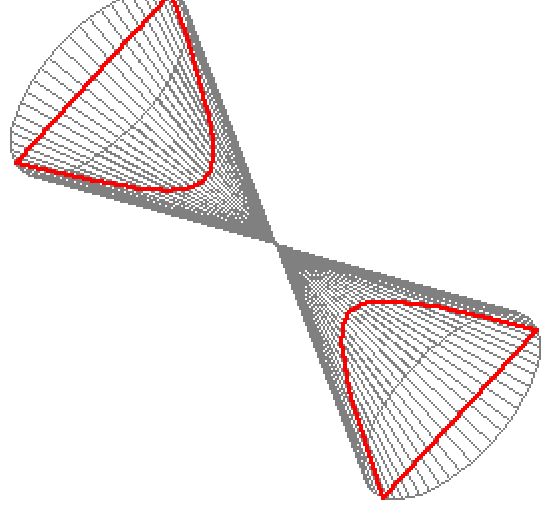
Üç Boyutlu Koni Ve İki Boyutlu Arakesitleri (Konikler)



Elips

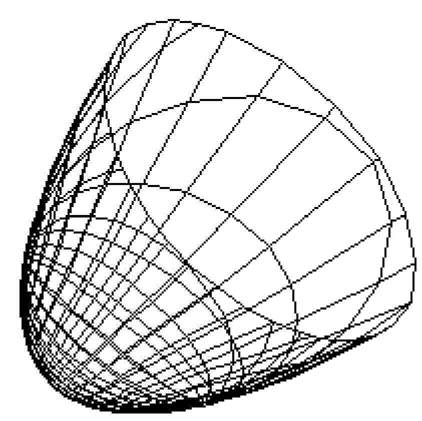


Parabol

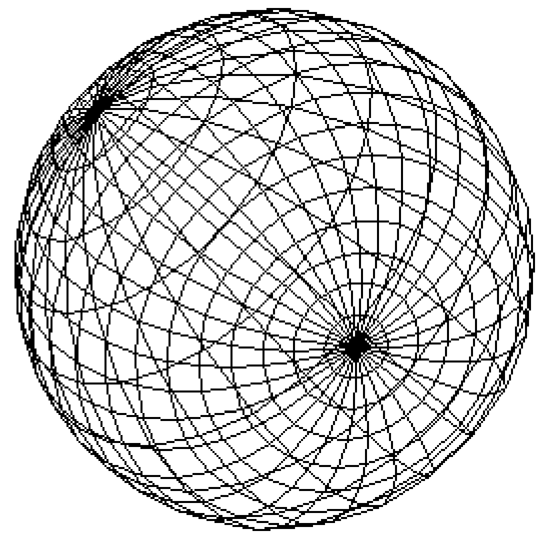


Hiperbol

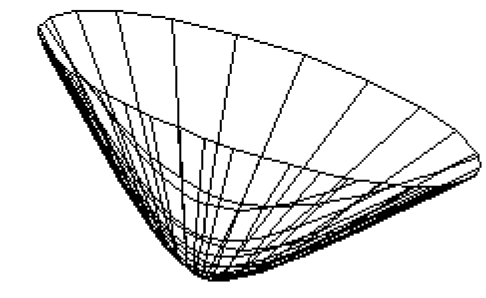
Dört Boyutlu Koninin Üç Boyutlu Uzayla Arakesitleri (Hiperkonikler)

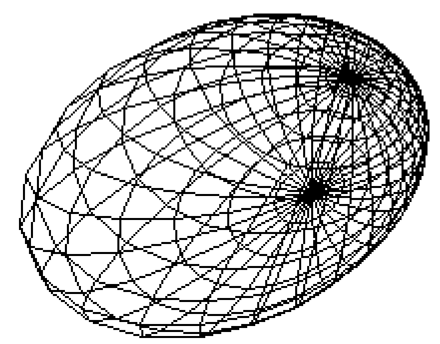


Üç boyutlu parabol (paraboloid):
Hacim sayısı: 1
Yüzey sayısı: 451
Kenar sayısı: 889
Köşe sayısı: 440

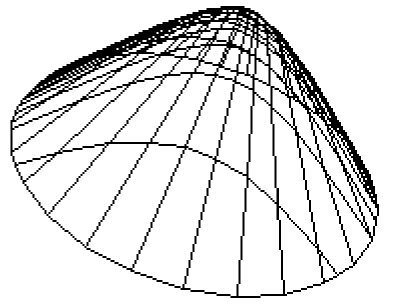


Üç boyutlu küre:
Hacim sayısı: 1
Yüzey sayısı: 512
Kenar sayısı: 992
Köşe sayısı: 482





Üç boyutlu elips (elipsoid):
Hacim sayısı: 1
Yüzey sayısı: 512
Kenar sayısı: 992
Köşe sayısı: 482



Üç boyutlu hiperbol (hiperboloid):
Hacim sayısı: 2
Yüzey sayısı: 374
Kenar sayısı: 793
Köşe sayısı: 423

Ek 2: Birimler

Structures, *Operations* ve *Graphs* birimleri programın temel işlev birimleridir. *main*, *about*, *childwin*, *creator*, *rotate*, *rotplanes, vector* birimleri ise programın arayüzünü oluşturur. Yalnız *creator* biriminde *küp*, *toroid* gibi bazı nesneleri yaratma yordamları bulunur. Bunlar yine `TObj` sınıfındaki üç cisim yaratma metodunu kullanır.

Operations Birimi:

Vektör ve matris işlemlerinin çok boyut için genelleştirilmiş hallerini bu birimde topladık

```
procedure vTransform(var V: TVector; var Mx: TMatrix);
```
`V` vektörüyle `Mx` matrisini çarpar ve sonucu `V` vektörüne yazar. Böylece `V` vektörüne `Mx` dönüşümü uygulanmış olur

```
procedure vProjectTo(var V: TVector; DimCount: Integer);
```
`V` vektörünün `DimCount` boyutuna izdüşümünü alır

```
procedure mMultiply(var Mx1, Mx2: TMatrix);
```
`Mx1` matrisiyle, `Mx2` matrisini çarpar ve sonucu `Mx1`'e yazar

```
procedure mIdentity(var Mx: TMatrix);
```
`Mx` matrisini birim matrise çevirir

```
procedure mTransition(var Mx: TMatrix; Coord: Integer; x: TValue);
```
`Coord` koordinatında `x` kadar öteleyen öteleme matrisini `Mx`'e yazar

```
procedure mRotation(var Mx: TMatrix; Axis1, Axis2: Integer; Angle: TValue);
```
`Axis1-Axis2` düzleminde `Angle` açısı kadar döndürme yapan matrisi `Mx`'e yazar

Graphs Birimi:

Çizim ile ilgili işlevler bu birimdedir:

```
procedure grDrawLine(x1, y1, x2, y2: Integer; r, g, b: Byte);
```
(`x1,y1`) ve (`x2,y2`) noktaları arasına çizgi çizer

```
procedure grDrawElement1(Element: PElement);
```
`Element` çizgisini çizer

```
procedure grDrawObj(Obj: TObj);
```
`Obj` cismini çizer

Structures Birimi:

Bu birimde yer alan veri yapısı ve arakesit alma ile ilgili işlevlerin başlıcaları:

```
function IntersectElement(Obj: TObj; Src: PElement): PElement;
```
`Obj` cismindeki `Src` çizgisinin $c_{n-1} = 0$ denklemli (son koordinatı sıfır olan) alt uzayla kesişimini döner

```
procedure IntersectObj(var Obj: TObj);
```
`Obj` cisminin $c_{n-1} = 0$ alt uzayı ile arakesitini bulur

`function LinkElement(Element: PElement): PElement;`
`Element` noktasını `Reserved` değişkeninde tutulan eşine bir çizgiyle bağlayıp bu çizgiyi döner

`procedure LinkObj(Obj: TObj);`
`Obj` cisminin bütün elemanlarını eşlerine bağlar

TObj sınıfının başlıca metotları:

`procedure TObj.SaveToFile(FileName: string);`
Cismi `FileName` isimli dosyaya yazar

`procedure TObj.LoadFromFile(FileName: string);`
Cismi `FileName` isimli dosyadan okur

`procedure TObj.MakePrism(d: TValue);`
Prizma yapma algoritmasını kullanarak cismi prizmalaştırır

`procedure TObj.MakeCylinder(Axis: Integer; Detail, Ratio: TValue);`
Silindir yapma algoritmasını kullanarak cismi silindirleştirir

`procedure TObj.MakePyramid(h: TValue);`
Piramit yapma algoritmasını kullanarak cismi piramitleştirir

`procedure TObj.IntersectTo(var Obj: TObj; DimCount: Integer);`
Cismin `DimCount` boyutundaki arakesitini `Obj` cismine yazar

`procedure TObj.ProjectTo(DimCount: Integer);`
Cisimdeki noktaların `DimCount` boyutuna izdüşümünü alır

`procedure TObj.Transform(var Mx: TMatrix);`
Cisme `Mx` matris dönüşümünü uygular

`procedure TObj.LinkPoint(Point: PElement);`
Cismin bütün elemanlarını `Point` noktasına bağlar

Kaynaklar

1) Bilim ve Teknik Dergisi, Sayı 318, Dördüncü Boyut ve Ötesi

2) Bresenham's Line Drawing Algorithm
http://ugweb.cs.ualberta.ca/~c411/notes/bresenhams/bres.html

3) 3dica
http://www.fyslab.hut.fi/~jyp/tut/3dica.htm

4) Geometry Forum Articles
http://www.geom.umn.edu/docs/forum/polytope/

5) Student of Hyperspace
http://www.uccs.edu/~eswab/hyprspac.htm

6) Hyperspace Structures
http://www.lut.ac.uk/departments/ma/gallery/hyper/

7) Hyperspheres, Hyperspace and The Fourth Spatial Dimension
http://www.cyburban.com/~mrf/index.html

8) Hyperspace
http://www.thepla.net/~denzi/tor.html

Teşekkür

Bu projenin yapılmasında yardımlarını bizden esirgemeyen;

İzmir Fen Lisesi idaresine, öğretmenlerine, personeline

Bilim Kurulu Başkanı **Kemal Ocaktürk**'e

Bilgisayar Öğretmeni **Hasan Korkmaz**'a

tüm arkadaşlarımıza

küçüklükten beri bizlere bilimsel merakı aşılayan ailelerimize

çok teşekkür ederiz.

Eser Aygün – Işık Barış Fidaner

2000–2001

İzmir