

Ders Konu Kapsamı

4. OpenGL Kütüphanesi

4.1. Pipeline Adımları.

4.2. Çizme Fonksiyonlarının Kullanımı ve Örnek Uygulama.

Dr. Ömer ÇETİN

3

Amaç

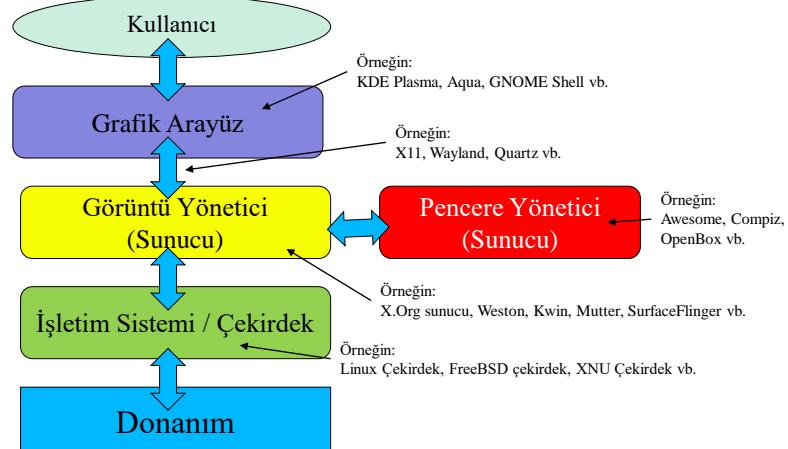
- OpenGL API'sinin geliştirilmesi
- OpenGL Mimarisi
 - Durum makinesi olarak OpenGL
 - Veri akışı makinesi olarak OpenGL
- Fonksiyonlar
 - Türleri
 - Biçimleri
- Basit bir program

Dr. Ömer ÇETİN

4

Ön Bilgi

GRAFİK SİSTEMİ

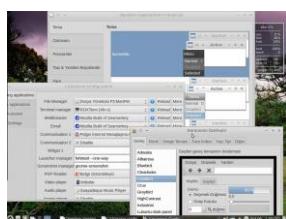


Dr. Ömer ÇETİN

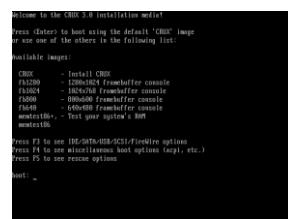
5

Ön Bilgi

- Bir yazılım ile kullanıcı arası iletişimini görsel olarak sağlamak için üretilmiş olan görsel öğeler bütününe **grafiksel kullanıcı arayüzü (Graphical User Interface - GUI)** denir.
- İşletim sistemlerine ait grafiksel arayüzde birbirine sıklıkla karışan çeşitli kavramları örneklerle açıklayalım:



Grafiksel Arayüz



Metin tabanlı kullanıcı arabirimleri

Dr. Ömer ÇETİN

6

Ön Bilgi

GRAFİK KARTI SÜRÜCÜLERİ

- En yaygınları Intel, ATI, NVIDIA gibi üreticiler olan grafik kartı donanımlarını işletim sisteminin tanıyıp kullanabilmesi için gerekli olan yazılımdır.
 - NVIDIA ve ATI üreticileri için **açık veya kapalı kaynak kodlu sürücü** seçenekleri bulunur.
 - Açık kaynak sürücüler yalnızca 2D desteği ile gelir. Ancak «**Mesa kütüphanesi**» yüklenerek 3D desteği de kazanılabilir.
 - Grafik kartı sürücüsü konusunda karşılaşılabilecek diğer bir kavram da «**vesa**»dır.
 - Vesa 2D veya 3D desteği olmayan en temel grafik kartı sürücüsüdür. Dünya çapında standart olduğu için bütün grafik kartları tarafından desteklenir.

www.mesa3d.org

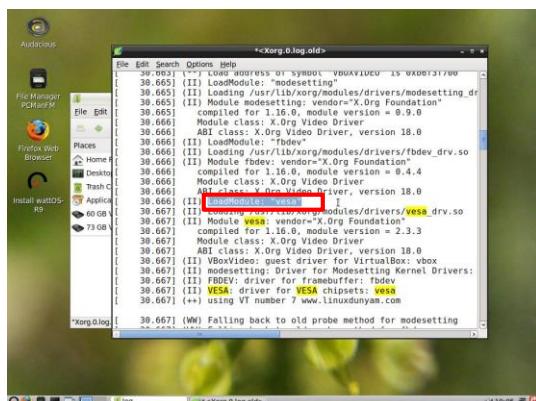
www.x.org/releases/current/doc/man/man4/vesa.4.xhtml

Dr. Ömer GETİN

7

Ön Bilgi

GRAFİK KARTI SÜRÜCÜLERİ



Dr. Ömer ÇETİN

8

Ön Bilgi

Görüntü Yönetici (Sunucu)

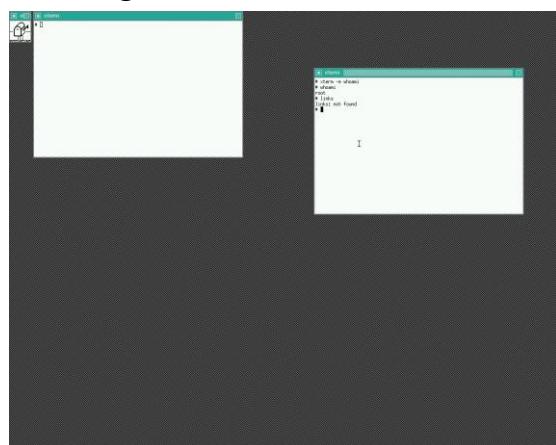
- Örneğin bir görüntü yöneticisi olan Xorg yazılımı, bilgisayarlarınızın donanımının, çalıştırığınız grafiksel arabirime sahip herhangi bir programa ait grafikleri size sunabilmesini sağlayacak olan ortamı oluşturabilmesini sağlar. Ayrıca klavye ve fare kullanımını da yönetir.
- Sunucu, istemci, kurallar ve kitaplıklar şeklinde ana bölgelere ayrılır. Sunucu size grafik arabirimini sunar. Uygulamaların istekleri Xorg'un istemcisi tarafından alınır. xlsclients komutu ile istekte bulunan uygulamalarınızı listeleyebilirsiniz. Kurallar istemci-sunucu iletişimini sağlar, kitaplıklar da gerekli olan bütün altyapıyı içerir.

Dr. Ömer ÇETİN

9

Ön Bilgi

Xorg'un sunduğu temel X ortamı:



Dr. Ömer ÇETİN

10

Ön Bilgi

GÖRÜNTÜ YÖNETİCİ (Display Manager)

- Size görsel arayüz sunacak olan grafik sunucusunun başlatılması, devam edebilmesi, kullanıcı yetkileri ve oturum açma gibi ayarları yapan küçük programlardır. Örneğin xdm, gdm, lightdm, mdm:



xdm



lightdm

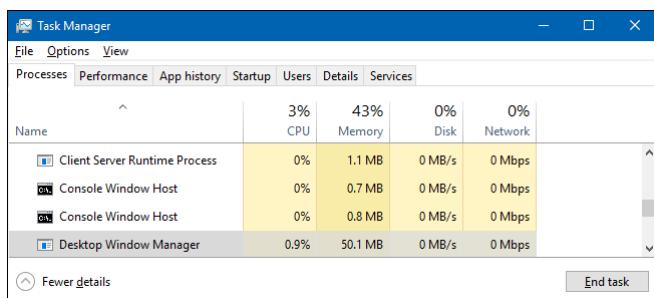
Dr. Ömer ÇETİN

11

Ön Bilgi

GÖRÜNTÜ YÖNETİCİ (Display Manager)

- MS Windows işletim sisteminde kullanılan görüntü yöneticisi uygulamasının adı Desktop Windows Manager (DWM) olarak adlandırılır.



Dr. Ömer ÇETİN

12

Ön Bilgi

MASAÜSTÜ ORTAMI (Desktop Manager)

- Kullandığınız işletim sisteminde size grafiksel arayüz olarak sunulan bütün resim, öge ve pencerelerin ortaklaşa kullandığı ortama denir. GNOME, KDE, Xfce, LXDE, Enlightenment birer masaüstü ortamıdır. KDE ve GNOME:



KDE



GNOME

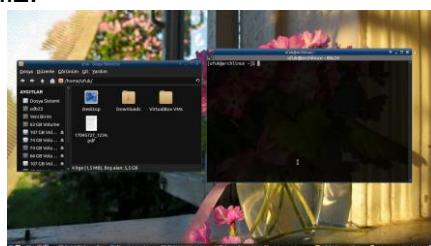
Dr. Ömer ÇETİN

13

Ön Bilgi

PENCERE YÖNETİCİSİ

- Masaüstü ortamında görüntülenen pencerelerin biçim, hareket ve yönetimini sağlayan uygulamadır. Pencere yöneticisi için masaüstü ortamı değişken olabilir. Örneğin Xfce ile ister xfwm4'ü ister marco'yu isterseniz metacity'i kullanabilirsiniz.



Dr. Ömer ÇETİN

14

API Tarihçesi

- International Federation for Information Processing (IFIPS-1973), **standart grafik API'sı** oluşturma için iki komite kurdu:
 - **Grafik Çekirdek Sistemi (GKS)**
 - 2B iş istasyonu modeli içeriyordu
 - **Çekirdek**
 - 2D ve 3D
 - **GKS, ISO** ve daha sonra **ANSI** standardı olarak kabul edildi (1980'ler).
- GKS kolayca 3D'ye uyarlanamadı (**GKS-3D**)
 - Donanım geliştirmenin çok gerisinde kaldı

PHIGS ve X

- Programmers Hierarchical GSystem (**PHIGS**)
 - CAD topluluğundan ortaya çıktı
 - Korunmuş grafikleri olan **veritabanı modeli** (yapılar)
- X Pencere Sistemi
 - DEC / MIT çabası
 - Grafikler ile **istemci-sunucu** mimarisi
- PEX ikisini birleştirdi
 - Kullanımı kolay değil (her birinin kusurları)

SGI ve GL

- Silicon Graphics (SGI), **boru hattını** donanıma uygulayarak grafik iş istasyonunda devrim yarattı (1982).
- Sisteme erişmek için uygulama programcılar **Graphic Language (GL)** adlı bir kütüphane kullandılar.
- GL ile **Üç boyutlu** etkileşimli uygulamaları programlamak nispeten kolaydı.

OpenGL

- GL'nin başarısı, **platformdan bağımsız** bir API olan **OpenGL**'ye geçiş sağladı (1992)
 - Kullanımı kolay
 - Mükemmel performans elde etmek için donanıma yeterince yakın
 - Oluşturmaya odaklanma sağlayan
 - Pencere sistemi bağımlılıklarını önleme

OpenGL Gelişimi

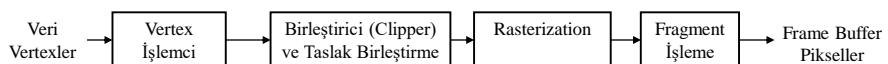
- Aslen bir Architectural Review Board (ARB) tarafından kontrol edilir.
 - Üyeleri SGI, Microsoft, Nvidia, HP, 3DLabs, IBM,
 - Şimdi adı: **Kronos Grubu**
 - Giderek kararlıdır (sürüm >2.5 ile)
 - Geriye uyumlu geliştirme
 - Yeni donanım yeteneklerini yansıtma
 - 3B doku eşleme ve doku nesneleri üretebilme
 - Vertex ve fragment programlarını gerçekleştirmeye
 - Shader programlama dili
 - Uzantılar üzerinden platforma özgü özelliklere izin verir...

Dr. Ömer ÇETİN

19

Modern OpenGL

- CPU yerine **GPU** kullanılarak performans elde edilir.
- **Shaders** adı verilen programlar aracılığıyla GPU'yu denetleyin.
 - Varsayılan shader yok... (Özelleştirilebilir boru hattı sağlar)
 - Her uygulama hem bir vertex hem de bir fragment shader sağlamalıdır
- Uygulamanın işi GPU'ya **veri** göndermek.
- GPU tüm oluşturma (**rendering**) işlemlerini yapar.



Dr. Ömer ÇETİN

20

Diger Versiyonlar

OpenGL ES

Gömülü Sistemler

Version 1.0 basitleştirilmiş OpenGL 2.1

Version 2.0 basitleştirilmiş OpenGL 3.1

Shader tabanlı

WebGL

Javascript ES 2.0 uyarlaması

Yeni nesil tarayıcıların tamamı tarafından desteklendi

OpenGL 4.X ve son sürüm 4.6

Dr. Ömer ÇETİN

21

Direct X

• Avantajları

- Kaynakların daha iyi kontrolü
- Üst düzey işlevlere erişim

• Dezavantajları

- Yeni sürümler geriye uyumlu değil
- Sadece Windows platformu desteklenir
- Shader yapısı ile ilgili son gelişmeler OpenGL ile yakınlaşmaya yol açmaktadır ...

Dr. Ömer ÇETİN

22

OpenGL Kütüphaneleri

- **OpenGL çekirdek kütüphanesi**
 - Windows'ta OpenGL32
 - Çoğu unix / linux sistemlerinde GL (libGL.a)
- **OpenGL Hizmet Programı Kütüphanesi (GLU)**
 - OpenGL çekirdeğinde işlevsellik sağlar, ancak kodu yeniden yazmak zorunda kalmaz
 - Sadece eski kodla çalışacak
- **Pencere sistemli bağlantılar**
 - X pencere sistemleri için GLX
 - Windows için WGL
 - Macintosh için AGL

Dr. Ömer ÇETİN

23

GLUT

OpenGL Yardımcı Araç Takımı (GLUT)

- Tüm pencere sistemlerinde ortak işlevsellik sağlar
- Bir pencere aç
- Harici aygıtlardan (Fare ve klavyeden vb.) giriş al
- Menüler oluştur
- Olay güdümlü çalışma sağla
- Kod taşınabilir ancak GLUT belirli bir platform için iyi bir araç setinin işlevselligidinden yoksundur...
- Örneğin kaydırma çubuğu yoktur.

Dr. Ömer ÇETİN

24

freeglut

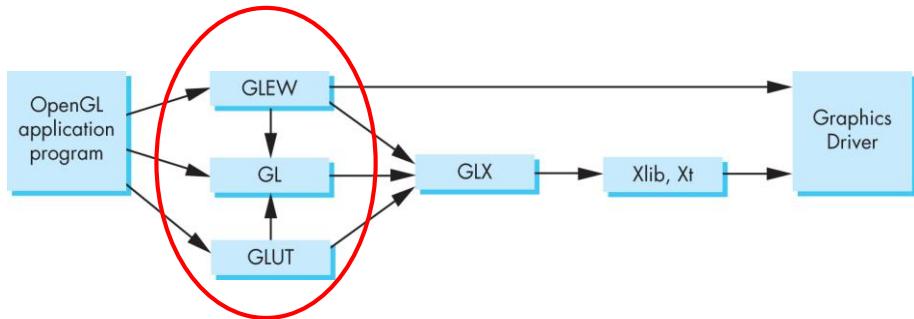
- GLUT uzun zaman önce geliştirildi ve değişmedi:
 - OpenGL 3.1 ile çalışması şaşırtıcı...
 - Kullanımdan kaldırılmış işlevler gerektirdiğinden bazı işlevleri çalışmıyor.
- **freeglut** GLUT ürününü güncelledi
 - Eklenen yetenekler geldi.
 - Bağlam kontrolü geliştirildi.

GLEW

OpenGL Eklenti Wrangler Kütüphanesi

- Belirli bir sistemde bulunan OpenGL uzantılarına erişmeyi kolaylaştırır.
- Windows kodunda belirli giriş noktalarına sahip olmaktan kaçınır.
- Uygulama sadece **glew.h**'yi içermeli ve bir **glewInit ()** çalıştırmalıdır.

Yazılım Organizasyonu



Ödev-1 OpenGL Kurulumu

Dr. Ömer ÇETİN

27

OpenGL Fonksiyonları

- İlkeller (Primitives)
 - Noktalar
 - Doğru parçaları
 - Üçgenler
- Öz nitelikler
- Dönüşümler
 - İzlenimi
 - Modelleme
- Denetim (GLUT)
- Giriş (GLUT)
- Sorgu

Dr. Ömer ÇETİN

28

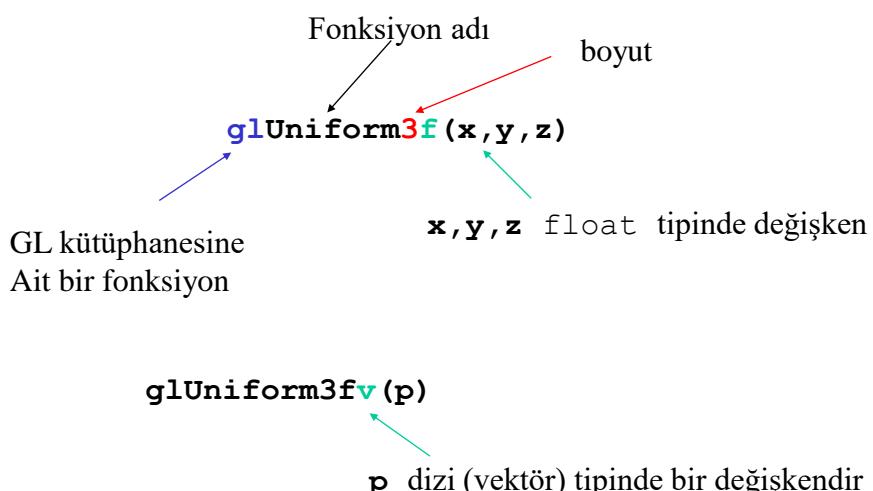
OpenGL Durum Makinesi

- OpenGL bir durum makinesidir.
- OpenGL fonksiyonları iki tiptir:
 - **İlkel üretim fonksiyonları**
 - İlkel görünür durumdaysa çıkışa neden olabilir
 - Köşeler nasıl işlenir ve ilkel görünüm durum makinası tarafından kontrol edilir
 - **Durum değiştirme fonksiyonları**
 - Dönüşüm fonksiyonları
 - Özellik fonksiyonları
 - Versiyon 3.1 altında çoğu durum değişkeni uygulama tarafından tanımlanır ve shader'lara gönderilir

Dr. Ömer ÇETİN

29

OpenGL fonksiyon formatı



Dr. Ömer ÇETİN

30

OpenGL #defines

- Çoğu sabit include dosyaları içerisinde yer almaktadır:
- gl.h, glu.h ve glut.h**
 - `#include <GL/glut.h>` diğerlerini otomatik olarak içermelidir.
 - Örneğin;
 - `glEnable(GL_DEPTH_TEST)`
 - `glClear(GL_COLOR_BUFFER_BIT)`
- include dosyaları ayrıca OpenGL veri tiplerini de içerir:
GLfloat, GLdouble,....

Dr. Ömer ÇETİN

31

GLSL

OpenGL Shading Dili (GLSL)

- C benzeri
 - Matris ve vektör çeşitleri (2, 3, 4 boyutlu)
 - Tanımlanmış (overloaded) operatörler
 - C ++ benzeri yapılar
- Nvidia'nın Cg ve Microsoft HLSL'ine benzerdir.
- Shader'lara kaynak kodu olarak gönderilen koddur.
- OpenGL fonksiyonları, shader'lar ile derleme, bağlantı kurma ve bilgi alma işlevlerini yerine getirir.

Dr. Ömer ÇETİN

32

OpenGL ve GLSL

- Shader tabanlı OpenGL, veri akışı modelinden ziyade **durum makinesi** modeline dayanır...
- Çoğu durum değişkeni, öznitelik ve versiyon 3.1 öncesi OpenGL işlevleri kullanım dışı kalmıştır...
- Eylemler shader içinde gerçekleşir...
- İş, uygulama GPU'ya veri almaktır...

Dr. Ömer ÇETİN

33

Örnek OpenGL Uygulaması

- İlk adım **programlama dilinizi** seçmek. OpenGL için **binding** işlemi, C# ve Java'dan Python ve Lua'ya kadar birçok dilde mevcuttur. Bazı dillerde, hiçbir resmi olmayan birden fazla OpenGL bağlama grubu vardır. Sonuça hepsi C veya C ++ bağlarına dayanır.
- C / C ++ kullanmıyorsanız, seçtiğiniz dil için OpenGL bağlayıcılarını içeren bir paket veya kütüphane indirmeli ve yüklemelisiniz. Bazıları önceden yüklenmiş olarak gelir, ancak bazılarında ayrı indirmeler vardır.
- C / C ++ kullanıyorsanız, önce OpenGL'e bağlanabilecek bir derleme ortamı (Visual Studio projesi, GNU makefile, CMake dosyası, vb.) oluşturmalısınız. Windows altında, **OpenGL32.lib** adlı bir kitaplığa statik olarak bağlanması gereklidir.
- Linux'ta, libGL'ye bağlanması gereklidir. Bu, "-IGL" komut satırı parametresiyle yapılabilir.

Dr. Ömer ÇETİN

34

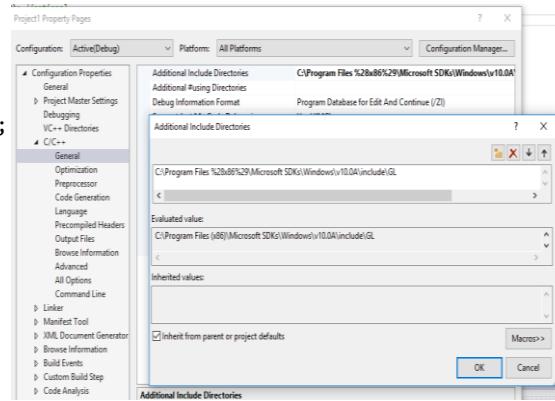
Örnek OpenGL Uygulaması

```
#include "glew.h"
#include "freeglut.h"
#include <iostream>

void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);
    glPointSize(5.0f);

    glBegin(GL_QUADS);
        glVertex2i(10, 10);
        glVertex2i(10, 210);
        glVertex2i(210, 210);
        glVertex2i(210, 10);
    glEnd();

    glFlush();
}
```



Dr. Ömer ÇETİN

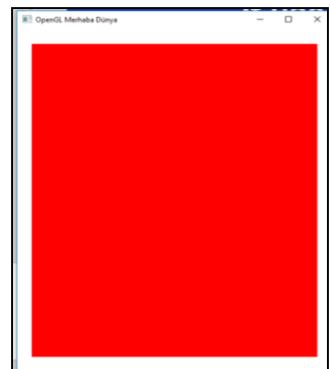
35

Örnek OpenGL Uygulaması

```
int main(int argc, char *argv[]) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutInitWindowPosition(500, 200);
    glutInitWindowSize(500, 600);
    glutCreateWindow("OpenGL Merhaba Dünya");

    glClearColor(1.0, 1.0, 1.0, 1.0);
    gluOrtho2D(0.0, 220.0, 0.0, 220.0);

    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```



Dr. Ömer ÇETİN

36

GLUT Fonksiyonları

```
glutInit(&argc, argv);
```

- Bu çağrı **GLUT'u başlatır**. GLUT kütüphanesi kullanılmaya başlanmadan önce mutlaka bu fonksiyon çağrılmalıdır. Parametreler doğrudan komut satırından sağlanabilir ve X'in eş zamansız yapısını devre dışı bırakan ve otomatik olarak GL hatalarını denetleyen ve bunları görüntüleyen (sırasıyla) '-sync' ve '-gldebug' gibi kullanışlı seçenekler içerir.

```
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
```

- Burada bazı GLUT seçeneklerini yapılandırıyoruz. **Renk paleti ve derinliği gibi ayarlamalar yapılır**. Örneğin, **GLUT_DOUBLE**, çift buffer ile çalışmayı (başka bir buffer gösterilirken arka plan bufferine çizim) ve çoğu işlemenin sona erdiği (yani ekran) renk tamponunu etkinleştirir.

<https://www.opengl.org/resources/libraries/glut/spec3/node12.html>

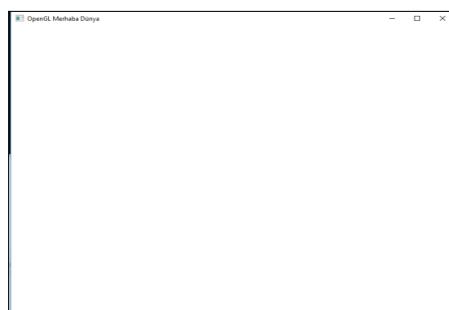
Dr. Ömer ÇETİN

37

GLUT Fonksiyonları

```
glutInitWindowPosition(500, 200);
glutInitWindowSize(500, 600);
glutCreateWindow("OpenGL Merhaba Dünya");
```

- Bu çağrılar **pencere parametrelerini** belirler ve yaratır. Ayrıca pencere başlığını belirleme seçeneğiniz de vardır.



Dr. Ömer ÇETİN

38

GLUT Fonksiyonları

```
glutDisplayFunc(fonksiyon_adı);
```

- Pencereleme sisteminde çalıştığımız için, çalışan programla etkileşimin çoğu olay geri çağrıma (callback) işlevleriyle gerçekleşir. GLUT, temeldeki pencereleme sistemi ile etkileşime girmeye özen gösterir ve bize birkaç geri çağrıma seçeneği sunar. **Burada tek bir çerçeveyin tüm işlemlerini yapmak için sadece bir "ana" geri çağrı kullanıyoruz.** Bu fonksiyon sürekli olarak GLUT iç döngüsü tarafından çağrılır.
- Oluşturulan pencere üzerinde **çizim işlemlerini yapan fonksiyonu parametre olarak alır** ve pencere içeriği yeniden çizildiğinde veya oluşturulduğunda bu fonksiyonu çalıştırır.

Örnek OpenGL Uygulaması

```
glClearColor(1.0, 1.0, 1.0, 1.0);
```

- Yukarıdaki çağrı, **çerçeve içeriğini temizlerken kullanılacak rengi** ayarlar. Renk dört kanala (RGBA) sahiptir ve 0,0 ile 1,0 arasında normalize edilmiş bir değer olarak belirtilir.

GLUT Fonksiyonları

```
glutMainLoop();
```

- Bu çağrı ile kontrol GLUT'a geçer ve **kendi iç döngüsüne başlar**. Bu döngüde, pencereleme sistemindeki olayları dinler ve yapılandırdığımız geri çağrılar yoluyla bunları iletir.

```
glClear (GL_COLOR_BUFFER_BIT);
```

```
glutSwapBuffers();
```

- Render fonksiyonumuzda yaptığımız tek şey çerçeveyi temizlemek (yukarıda belirtilen rengi kullanarak - değiştirmeyi deneyin). İkinci çağrı GLUT'a arka tamponun ve ön tamponun rollerini değiştirmesini söyler. Oluşturma geri çağrısı boyunca bir sonraki turda mevcut karelerin ön tamponuna işleyeceğiz ve mevcut geri arabellek görüntülenecektir.

Dr. Ömer ÇETİN

41

Örnek OpenGL Uygulaması

```
gluOrtho2D(0.0, 220.0, 0.0, 220.0);
```

```
void gluOrtho2D (Gldouble sol, Gdouble sag, Gdouble alt, GLdouble üst);
```

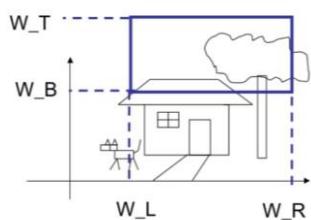
- Parametreler**

Sol ve sağ dikey kırpma düzlemleri için koordinatları belirtin.

Alt ve üst yatay kırpma düzlemleri için koordinatları belirtin.

- Açıklama**

gluOrtho2D iki boştur bir ortodrafik görüntüleme bölgesi oluşturur.



```
gluOrtho2D(W_L,W_R,W_B,W_T);
```

Dr. Ömer ÇETİN

42

Örnek OpenGL Uygulaması

```
void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);
    glPointSize(5.0f);

    glBegin(GL_QUADS);
        glVertex2i(10, 10);
        glVertex2i(10, 210);
        glVertex2i(210, 210);
        glVertex2i(210, 10);
    glEnd();

    glFlush();
}
```

Dr. Ömer ÇETİN

43

Örnek OpenGL Uygulaması

```
void glBegin(enum kip);
```

Bir çizime başlandığını belirtir. Parametresi çizilen geometrik şekli tanımlar. Bu parametre yerine geçilmesi gereken sembolik sabitler şunlardır :

- **GL_POINTS** : Nokta çizileceğini belirtir.
- **GL_LINES** : Verilen noktaların birleştirilerek bir çizgi çizileceğini belirtir.
- **GL_POLYGON** : Verilen noktaların birleştirilerek doğrular oluşturulacağını ve oluşan şeklin alansal bir şekil olacağını, içini renklendirileceğini belirtir.
- **GL_QUADS** : Verilen dört noktadan içi renkelendirilmiş bir dörtgen oluşturulacağını belirtir.
- **GL_TRIANGLES** : Verilen üç noktadan içi renklendirilmiş üçgen oluşturulacağını belirtir.
- **GL_TRIANGLES_STRIP** : Verilen noktaların üçer üçer sırasıyla birleştirerek üçgenler oluşturulacağını belirtir.
- **GL_QUAD_STRIP** : Verilen noktaların dörder dörder sırasıyla birleştirilerek dörtgenler oluşturulacağını belirtir.
- **GL_TRIANGLE_FAN** : Verilen noktaların ilk nokta ikişer ikişer alınıp her adımda ilk noktayı üçüncü nokta kabul ederek birleştirileceğini ve yelpazemsi bir şekil oluşturulacağını belirtir.

Dr. Ömer ÇETİN

44

Örnek OpenGL Uygulaması

```
void glEnd(glVoid);
```

glBegin() fonksiyonu ile başlatılmış çizim işleminin bittiğini belirtir. Çizdirilen şekil ekrana bastırılmak üzere saklanır. Saklanmış bu şeklin çizdirilmesi için başka bir fonksiyon kullanılır.

```
void glFlush(glVoid);
```

Tampon bellekteki tüm şekillerin ekrana çizdirilmesini sağlar.

Dr. Ömer ÇETİN

45

Örnek OpenGL Uygulaması

```
void glVertex2f(float x, float y);
```

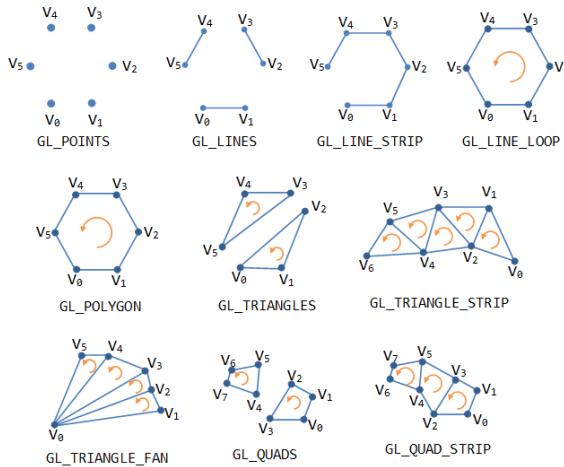
Bir geometrik şekle ait kontrol noktasının koordinat değerlerini belirtir. Geometrik şeklin ne olacağı **glBegin** fonksiyonunun parametresi ile belirlenir.

```
glBegin(GL_POLYGON);
    glColor3f(1.0f, 1.0f, 0.0f);
    glVertex2f(0.4f, 0.2f);
    glVertex2f(0.6f, 0.2f);
    glVertex2f(0.7f, 0.4f);
    glVertex2f(0.6f, 0.6f);
    glVertex2f(0.4f, 0.6f);
    glVertex2f(0.3f, 0.4f);
glEnd();
```

Dr. Ömer ÇETİN

46

Örnek OpenGL Uygulaması



```
glBegin(GL_POLYGON);
    glColor3f(1.0f, 1.0f, 0.0f);
    glVertex2f(0.4f, 0.2f);
    glVertex2f(0.6f, 0.2f);
    glVertex2f(0.7f, 0.4f);
    glVertex2f(0.6f, 0.6f);
    glVertex2f(0.4f, 0.6f);
    glVertex2f(0.3f, 0.4f);
glEnd();
```

Dr. Ömer ÇETİN

47

Örnek OpenGL Uygulaması

```
void glVertex2f(float x, float y);
```

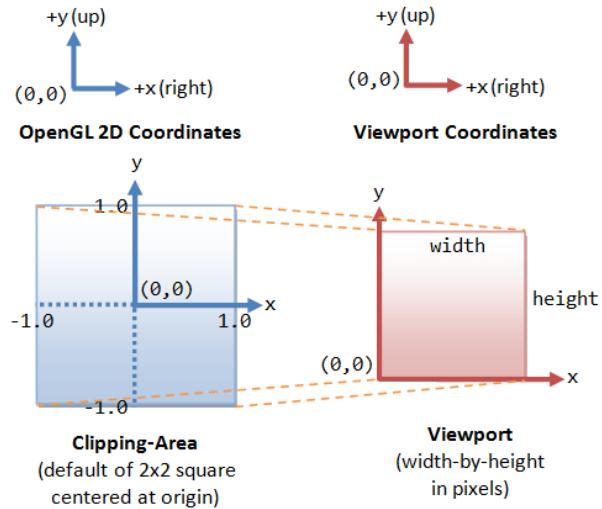
Bu fonksiyon iki boyutlu çizim yapılmacıği zaman kullanılır. Koordinatın sadece *x* ve *y* değerleri verilir. Bu fonksiyon ile aynı işi yapan glVertex2s, glVertex2i, glVertex2d fonksiyonları sırasıyla short, integer, double türden parametre değişkenleri alırlar.

```
typedef unsigned char   GLboolean;
typedef unsigned int    GLbitfield;
typedef void            GLvoid;
typedef signed char    GLbyte;      /* 1-byte signed */
typedef short           GLshort;    /* 2-byte signed */
typedef int             GLint;      /* 4-byte signed */
typedef unsigned char   GLubyte;    /* 1-byte unsigned */
typedef unsigned short  GLushort;   /* 2-byte unsigned */
typedef unsigned int    GLuint;     /* 4-byte unsigned */
typedef int             GLsizei;    /* 4-byte signed */
typedef float           GLfloat;    /* single precision float */
typedef float           GLclampf;   /* single precision float in [0,1] */
typedef double          GLdouble;   /* double precision float */
typedef double          GLclampd;  /* double precision float in [0,1] */
```

Dr. Ömer ÇETİN

48

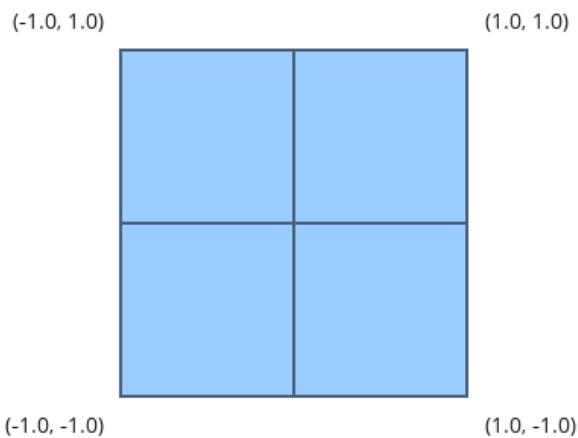
Örnek OpenGL Uygulaması



Dr. Ömer ÇETİN

49

Örnek OpenGL Uygulaması



Dr. Ömer ÇETİN

50

Örnek OpenGL Uygulaması

```
void glVertex3f(float x, float y, float z);
```

Bir geometrik şekle ait kontrol noktasının koordinat değerlerini belirtir. Geometrik şeitin ne olacağı glBegin fonksiyonunun parametresi ile belirlenir. Bu fonksiyon üç boyutlu çizim yapılacağı zaman kullanılır.

Koordinatın *x* ve *y* değerlerine ilaveten *z* değeri de verilir.

Bu fonksiyon ile aynı işi yapan glVertex3s, glVertex3i, glVertex3d fonksiyonları sırasıyla short, integer, double türden parametre değişkenleri alırlar.

Dr. Ömer ÇETİN

51

Örnek OpenGL Uygulaması

```
void glColor3f(float kirmizi, float yesil, float mavi);
```

Çizilecek şeitin rengini belirler. Ön tanımlı değerler 0'dır.

```
void glRectf(float x1, float y1, float x2, float y2);
```

Parametre olarak geçen koordinat değerlerini sol alt köşe ve sağ üst köşe olarak kabul ederek içi dolu bir dikdörtgen çizer.

Dr. Ömer ÇETİN

52

Derleme

Unix/linux

Genellikle .../include/GL dizinindeki dosyaları dahil et
–Iglut –Igl yükleyici bayraklarıyla derleyin
X kütüphaneleri için –L bayrağı eklemeniz gerekebilir
Mesa uygulaması çoğu linux dağıtımına dahildir
Mesa ve glut'un en son sürümleri için web'e bakın

Dr. Ömer ÇETİN

53

Derleme

Visual C++

Web'den glut.h, glut32.lib ve glut32.dll dosyasını edinin
Karşılık gelen OpenGL dosyalarıyla aynı yerlere kurun
Boş bir uygulama oluşturun
Proje ayarlarına glut32.lib ekleyin (bağlantı sekmesi
altında)
Freeglut ve GLEW için aynı

Dr. Ömer ÇETİN

54

OpenGL İlkelerin Çizimi

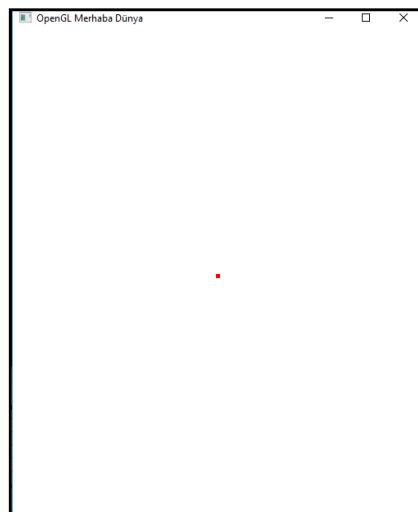
```
#include "glew.h"
#include "freeglut.h"
#include <iostream>
void display(void) {
//...
}
int main(int argc, char *argv[]) {
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
glutInitWindowPosition(500, 200);
glutInitWindowSize(500, 600);
glutCreateWindow("OpenGL Merhaba Dünya");
glClearColor(1.0, 1.0, 1.0, 1.0);
glutDisplayFunc(display);
glutMainLoop();
return 0;
}
```

Dr. Ömer ÇETİN

55

OpenGL İlkelerin Çizimi

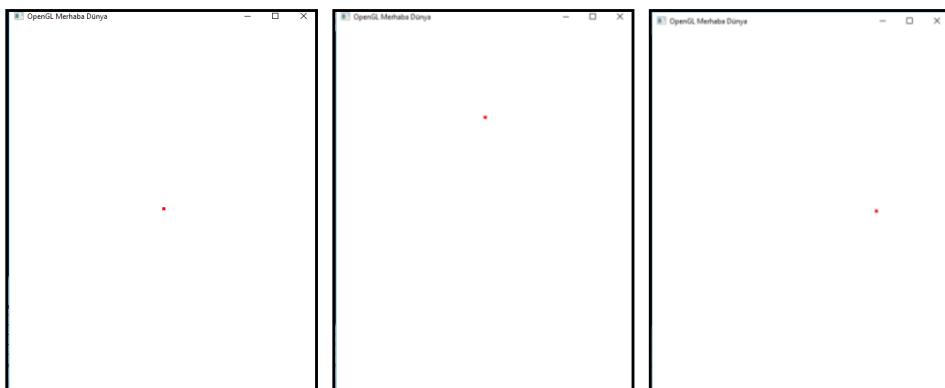
```
void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);
    glPointSize(5.0f);
    glColor3f(1, 0, 0);
    glBegin(GL_POINTS);
        glVertex2f(0, 0);
    glEnd();
    glFlush();
}
```



Dr. Ömer ÇETİN

56

OpenGL İlkelerin Çizimi



glVertex2f(0, 0);

glVertex2f(0.5, 0);

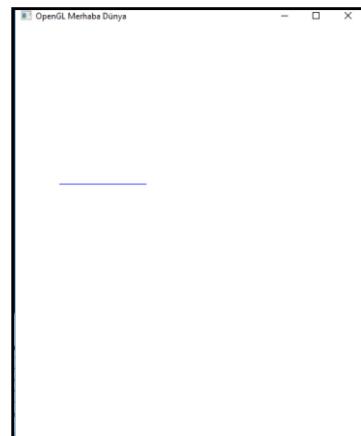
glVertex2f(0, 0.5);

Dr. Ömer ÇETİN

57

OpenGL İlkelerin Çizimi

```
void display(void) {  
    glClear(GL_COLOR_BUFFER_BIT);  
    glColor3f(1.0, 0.0, 0.0);  
    glPointSize(5.0f);  
    glColor3f(0, 0, 1);  
    glBegin(GL_LINES);  
        glVertex2f(-0.25, 0.25);  
        glVertex2f(-0.75, 0.25);  
    glEnd();  
    glFlush();  
  
}
```

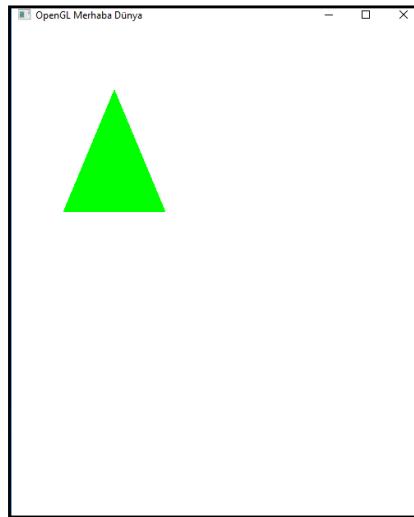


Dr. Ömer ÇETİN

58

OpenGL İlkelerin Çizimi

```
void display(void) {  
    glClear(GL_COLOR_BUFFER_BIT);  
    glColor3f(1.0, 0.0, 0.0);  
    glPointSize(5.0f);  
    glColor3f(0, 1, 0);  
    glBegin(GL_POLYGON);  
        glVertex2f(-0.25, 0.25);  
        glVertex2f(-0.75, 0.25);  
        glVertex2f(-0.5, 0.75);  
    glEnd();  
    glFlush();  
}
```

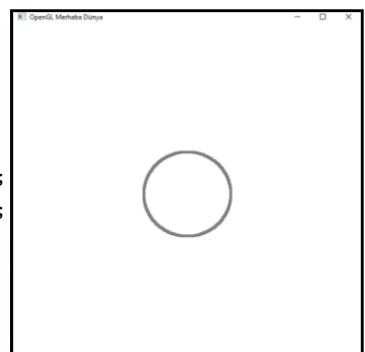


Dr. Ömer ÇETİN

59

OpenGL İlkeller Kullanarak Çizim

```
void display(void) {  
    glClear(GL_COLOR_BUFFER_BIT);  
    glColor3f(0.5, 0.5, 0.5);  
    glPointSize(5.0);  
    glBegin(GL_POINTS);  
        for (int i = 1; i < 360; i++) {  
            float x = 0.25 * sin((float)i) * 3.14 / 180;  
            float y = 0.25 * cos((float)i) * 3.14 / 180;  
            glVertex2f(x, y);  
        }  
    glEnd();  
    glFlush();  
}
```

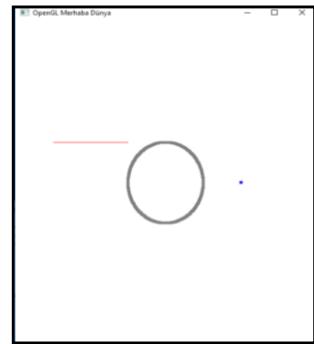


Dr. Ömer ÇETİN

60

Şekillerin Bir Arada Çizimi

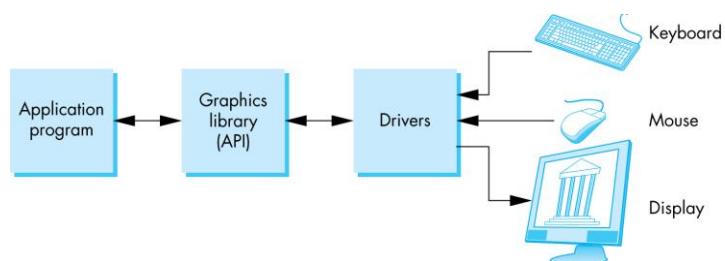
```
void display(void) {  
    glClear(GL_COLOR_BUFFER_BIT);  
    glPointSize(5.0f);  
  
    glColor3f(0, 0, 1);  
    glBegin(GL_POINTS);  
    glVertex2f(0.5, 0);  
    glEnd();  
  
    glColor3f(1, 0, 0);  
    glBegin(GL_LINES);  
    glVertex2f(-0.25, 0.25);  
    glVertex2f(-0.75, 0.25);  
    glEnd();  
}  
  
glColor3f(0.5, 0.5, 0.5);  
glBegin(GL_POINTS);  
for (int i = 1; i < 360; i++) {  
    float x = 0.25 * sin((float)i) * 3.14 / 180;  
    float y = 0.25 * cos((float)i) * 3.14 / 180;  
    glVertex2f(x, y);  
}  
glEnd();  
glFlush();  
}
```



Dr. Ömer ÇETİN

61

OpenGL Mimarisi

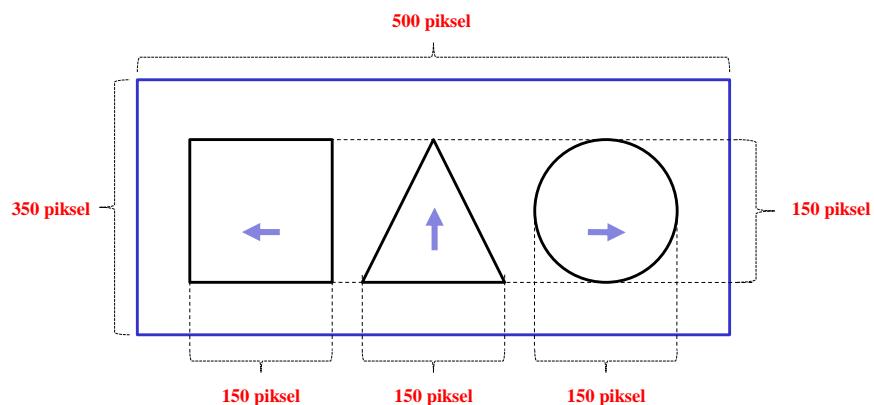


Dr. Ömer ÇETİN

62

Ödev

OpenGL - klavye ile etkileşim...



BİLGİSAYAR GRAFİKLERİ

Bilgisayar Grafiklerine Giriş



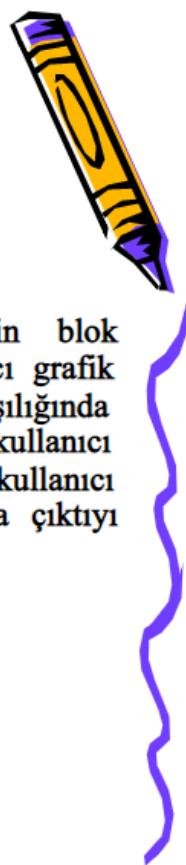
- Bilgisayar Grafikleri Nedir ?

Oluşturulan veya toplanan verilerin bilgisayar teknolojileri vasıtasıyla görüntülü şekilde sunulma yöntemlerini inceleyen bilgisayar bilimleri dalıdır. Başka bir deyişle bilgisayarda görüntü oluşturma ile ilgili herşeydir. "Bilgisayar Grafikleri"nin en önemli hedefi oluşturulan görüntülere gerçeklik kazandırmaktır(realistic looking images).

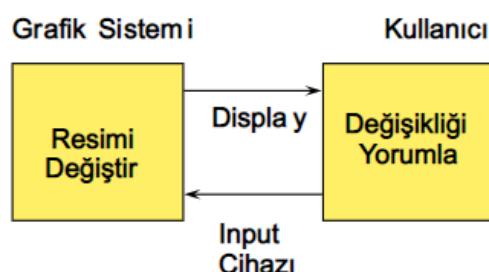
Bunun yanında birde İnteraktif bilgisayar grafikleri kavramı mevcuttur . Bunun anlamı, kullanıcının geribesleme sürecinin döngüsü içerisinde olmasıdır. Burada kullanıcı ile grafik uygulaması arasında etkileşim söz konusudur.



Bilgisayar Grafiklerine Giriş-1

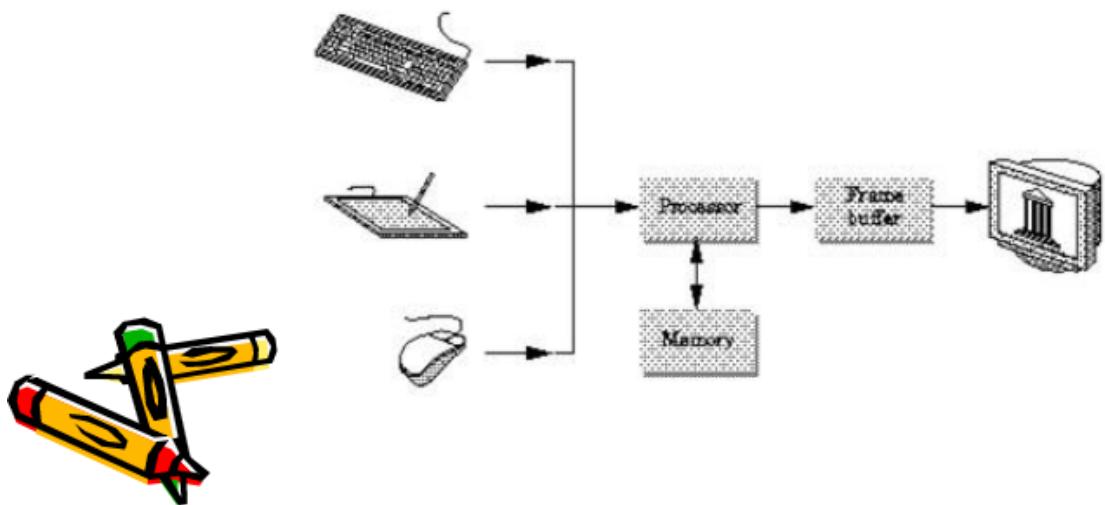


Aşağıdaki şekilde “interaktif bilgisayar grafikleri” nin blok diyagramı gösterilmiştir. Şekilden de görüldüğü gibi kullanıcı grafik sistemine bir takım girdiler ve komutlar göndermekte, karşılığında da bir görüntü almaktadır. Grafik sisteminin görevi,kullanıcı tarafından verilen girdi üzerinde birtakım işlemler yaparak kullanıcı istekleri doğrultusunda girdiyi değiştirmektir. Kullanıcı da çıktıyı analiz ederek bir takım yorumlarda bulunmaktadır.



Grafik Sistemi

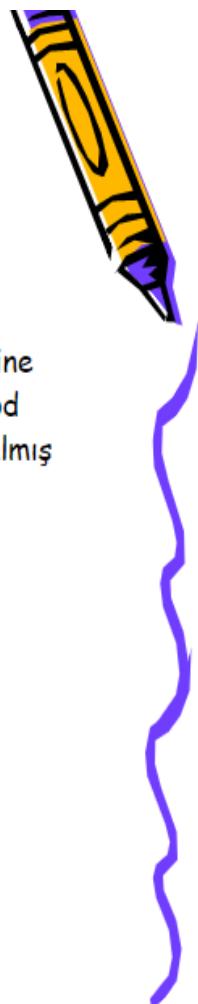
Aşağıdaki şekilde bir grafik sisteminin elemanları gösterilmiştir. Klavye, fare ve touchpad vb. input cihazlarıdır. Bu input cihazlarından alınan veriler kabaca CPU, hafıza(Ram) ve Frame Buffer dan oluşan bir işleme mekanizmasına ilettilir. Burada işlenen görüntüler ekrana yansıtılır.



Bilgisayar Grafiklerinin Tarihi

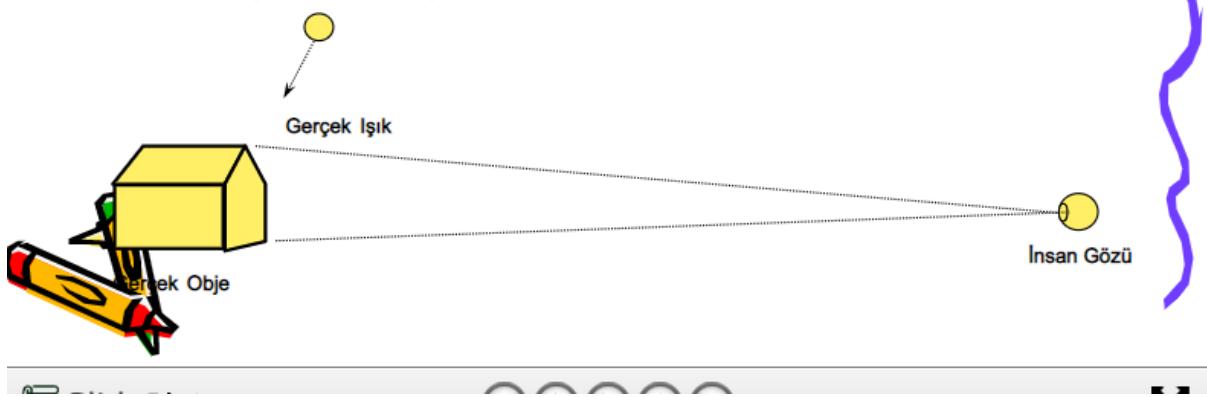
Aşağıdaki maddelerde bilgisayar grafiklerinin kullanım alanlarının kronolojik tablosu çizilmiştir. Görüldüğü gibi *bilgisayar grafikleri* yaygın kullanım alanları bulmakta ve hayatımızın değişmez parçası haline gelmiş bulunmaktadır. Çoğumuzun çok hoşlanarak izlediğimiz Hollywood filmlerinin o çarpıcı sahneleri, görüntü efektleri de bundan nasibini almış bulunmaktadır.

- Bilgisayar Destekli Tasarım (CAD) 1965 -
- Simülatörler (c.1975-)
- Bitmap grafikli kullanıcı arayüzleri (70'in sonu)
- Interaktif raster grafikler (80'li yıllar)
- Virtual reality (80'li yılların sonlarında)
- Bilgisayar animasyonu
- Visualization (Hayal canlandırma)
- Bilgisayar sanatı (Computer Arts)
- Web/Internet tabanlı grafikler

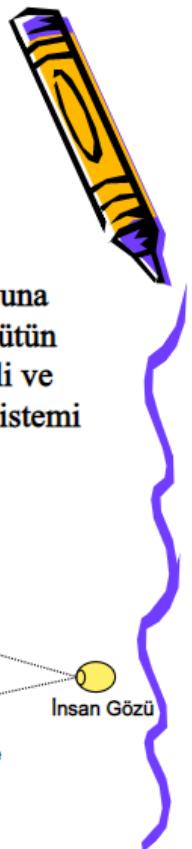


Grafik: Kavramsal(^{conceptual}) Model-Gerçek Dünya

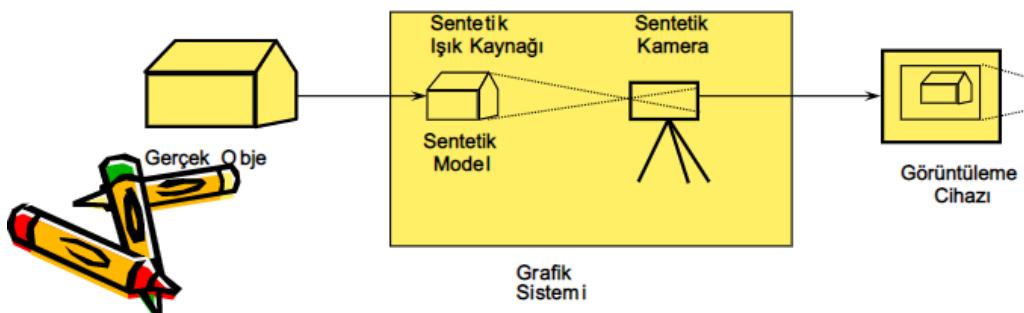
Kavramsal Model, görüntünün bilgisayarda sunulmasının soyut bir biçimidir. Bir görüntüyü oluşturan obje, ışık kaynağı ve bakan açısı gibi kavramların nasıl biraraya getirilerek bir görüntü modelinin oluşumunu inceler. Aşağıdaki resime bakıldığından bir görüntünün oluşumunda etkin olan 3 faktör ve gerçek dünyadaki modeli gösterilmiştir. Burada, gerçek obje tarafından yansıtlanan ışık insan gözü tarafından algılanır, göz bu bilgileri aynı zamanda beyine göndererek görme hafızasında görüntünün oluşmasını sağlar. Basitçe görme olayı bu şekildedir.



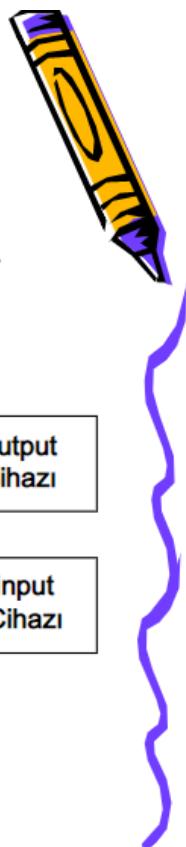
Grafik: Kavramsal(^{conceptual}) Model-Bilgisayar Dünyası



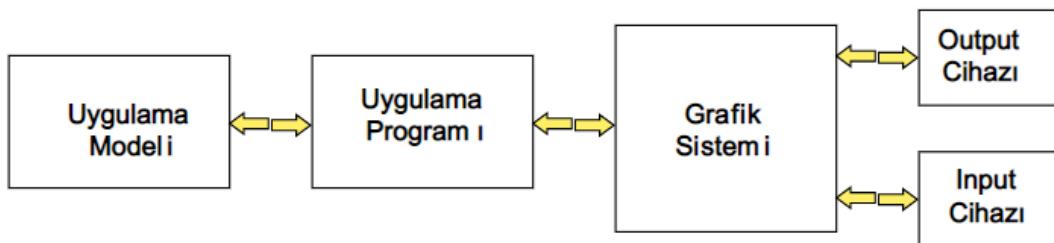
Bigisayar dünyasında ise gerçek dünyanın yapay ve sanal bir modeli mevcuttur. Gerçek obje geometrik şekiller vasıtasıyla oluşturulur ve buna sentetik model denir. Sentetik model, sentetik ışık kaynağı kullanır. Bütün bunlara da sentetik kameradan bakılır. Oluşan görüntü tamamen hayali ve mantıksaldır. Herşey bizim hayal gücümüze bağlıdır. Sonuçta grafik sistemi tarafından oluşturulan hayali görüntü görüntüleme cihazı vasıtasıyla gerçek dünyaya yansıtılır.



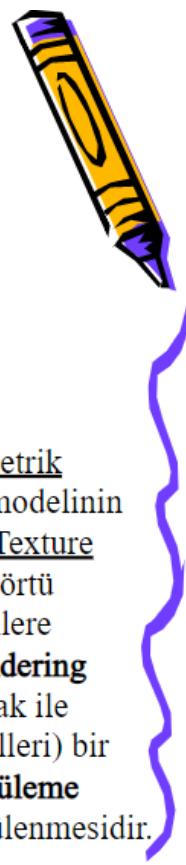
Kavramsal Model Diyagramı



Kavramsal modelin diyagramı aşağıdaki resimdeki gibidir. Uygulama modeli, uygulama programı, grafik sistemi ve I/O birimlerinden oluşmaktadır. Bu birimleri birbirleriyle etkileşim içerisinde ederler.



Grafik Oluşum Süreci

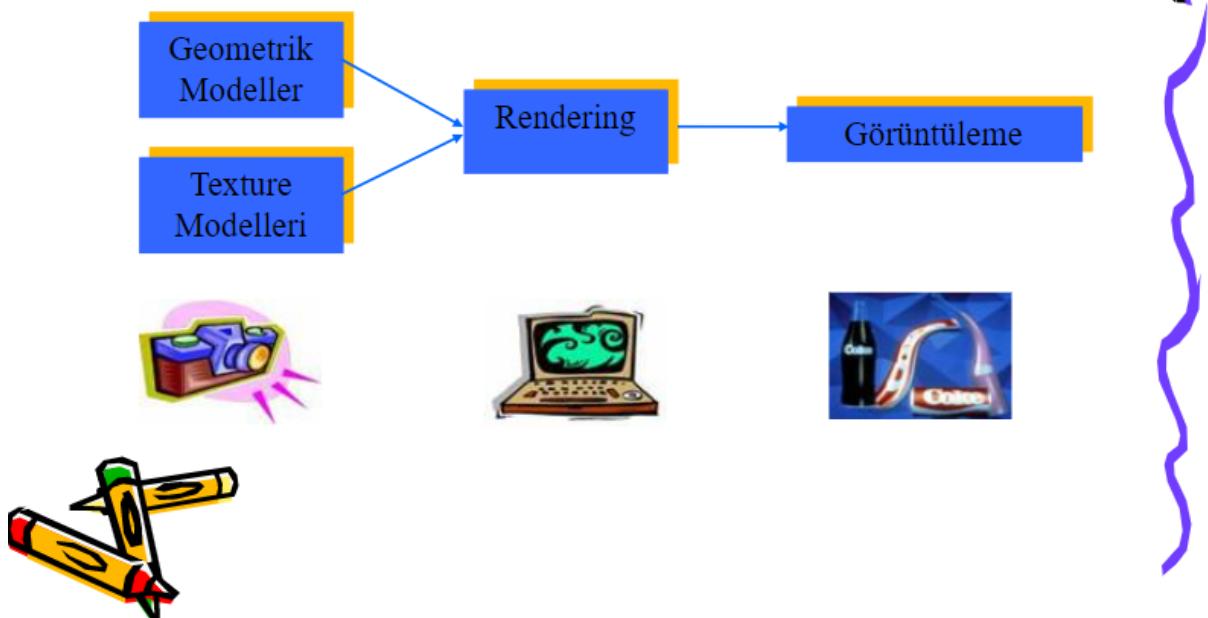


Grafik oluşum süreci 3 temel aşamadan oluşmaktadır. Bunlar

- Modelleme**
- Renderleme**
- Görüntüleme**

Modelleme, geometrik ve texture modellerinden oluşmaktadır. Geometrik modelleme objenin bir takım matematiksel ve geometrik figürler ile modelinin kurulmasıdır. Başka bir deyişle modelin iskeletinin oluşturulmasıdır. Texture modelleme de oluşturulan iskeletin giydirilmesidir. Texture, kaplama, örtü anlamlarına gelir ve 2 boyutlu texture'ları 3 boyutlu geometrik modellere uygulayarak 3 boyutlu katı cisimler oluşturulur. İkinci aşama olan **rendering** sentetik modelin görüntüleme cihazlarının anlayabileceği şekle sokmak ile ilişlidir. Renderleme, görüntü parçalarını (geometrik ve texture modelleri) birleştirerek bir bütün haline getirmek işlemidir. Son olarak **görüntüleme** aşamasında görüntünün görüntüleme cihazlarına iletilmesi ve görüntülenmesidir.

Grafik Oluşum Süreci Şeması

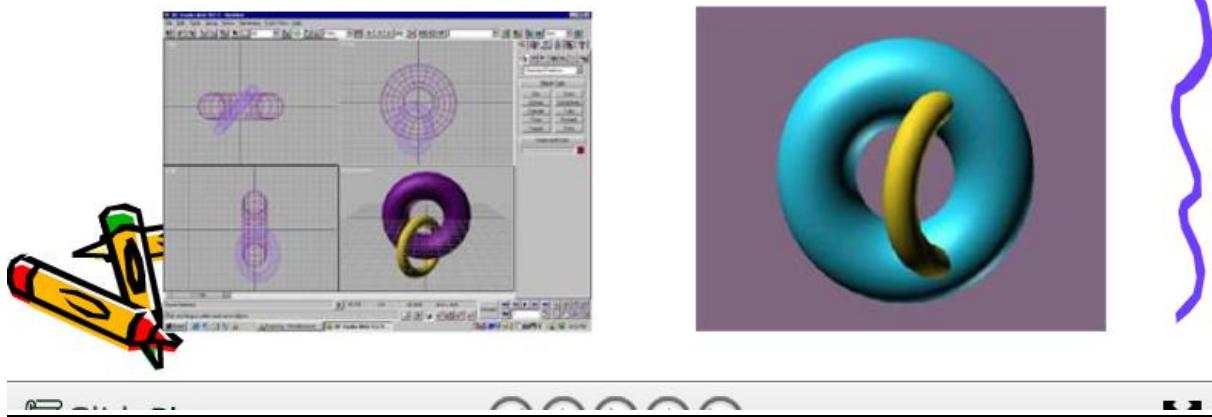
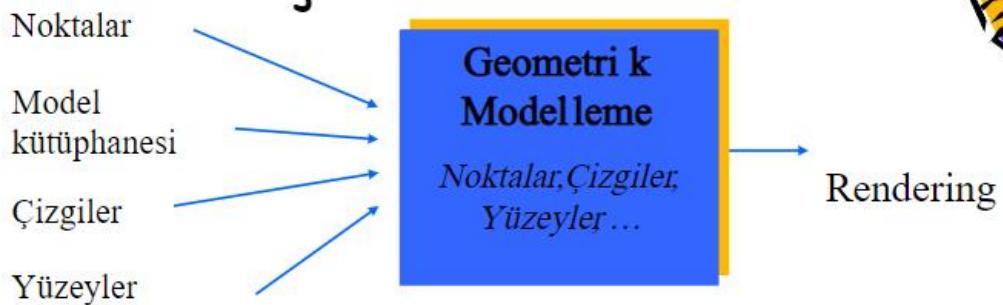


Geometrik Modelleme

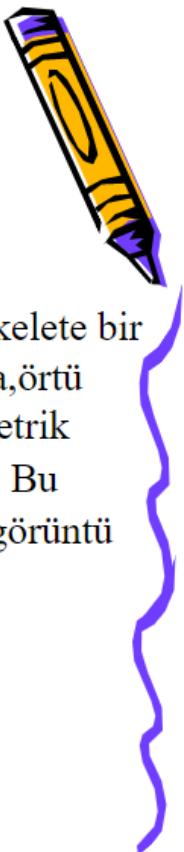
Geometrik modelleme objenin bir takım matematiksel ve geometrik ilkeller ile modelinin kurulmasıdır. Başka bir deyişle modelin iskeletinin oluşturulmasıdır. Buradaki ilkeller kelimesi İngilizce *primitives* için kullanılmıştır. Yani en basit geometrik primitifler veya şekiller de denebilir. Bunlar noktalar, çizgiler, kare, üçgenler ve çokgenler, yüzeylerdir. Geometrik modellemede gerçek objenin sentetik (yapay) modeli bu primitifler yardımıyla kurulur ve renderleme aşamasına gönderilir. Renderlenmiş model gerçek objenin bir prototipi olur.



Geometrik Modelleme-Şeması



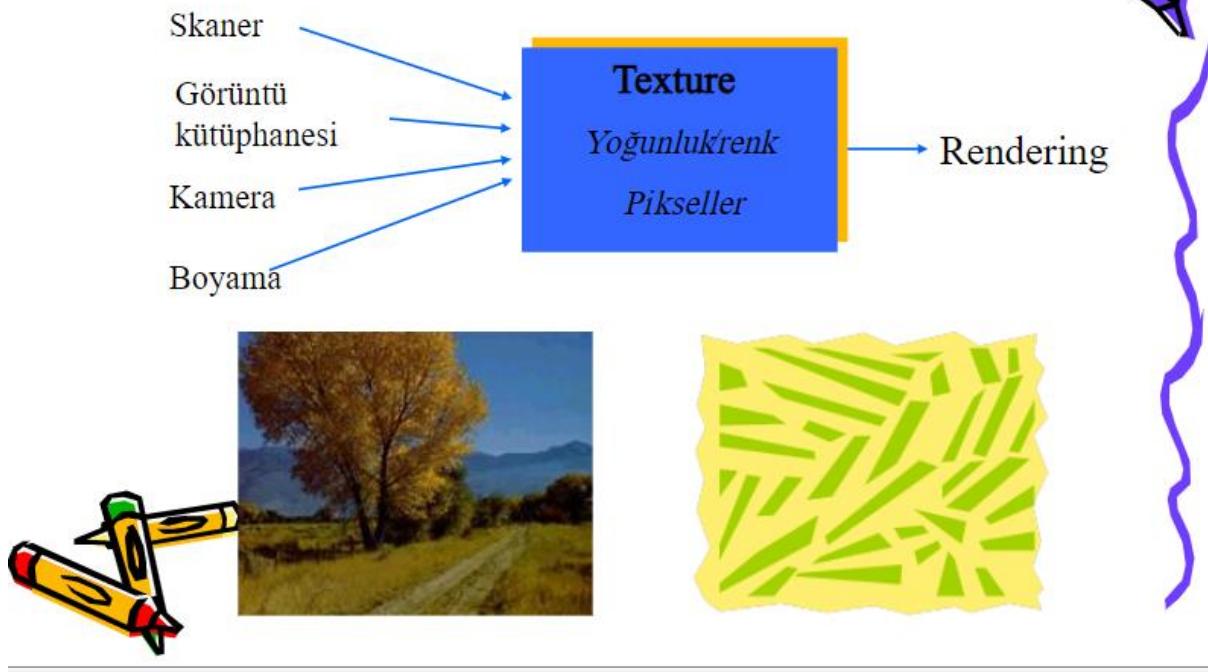
Texture Modelleme



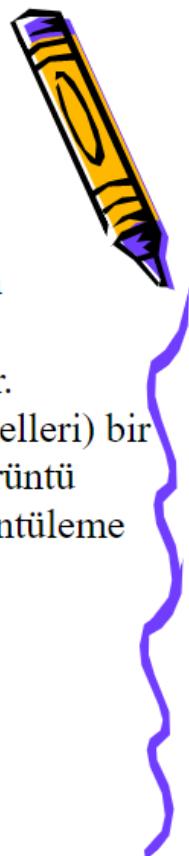
Geometirk modellemede oluşturulacak objenin bir iskeleti oluşturulur demistik. Texture modelleme de oluşturulan iskelete bir beden giydirmeye gibi bir şeydir. Texture, İngilizcede kaplama, örtü anlamlarına gelir ve 2 boyutlu texture'ları 3 boyutlu geometrik modellere uygulayarak 3 boyutlu katı cisimler oluşturulur. Bu aşamadan sonra model renderleme aşamasından geçer ve görüntü hazır hale getirilmiş olur.



Texture Modelleme - Şeması



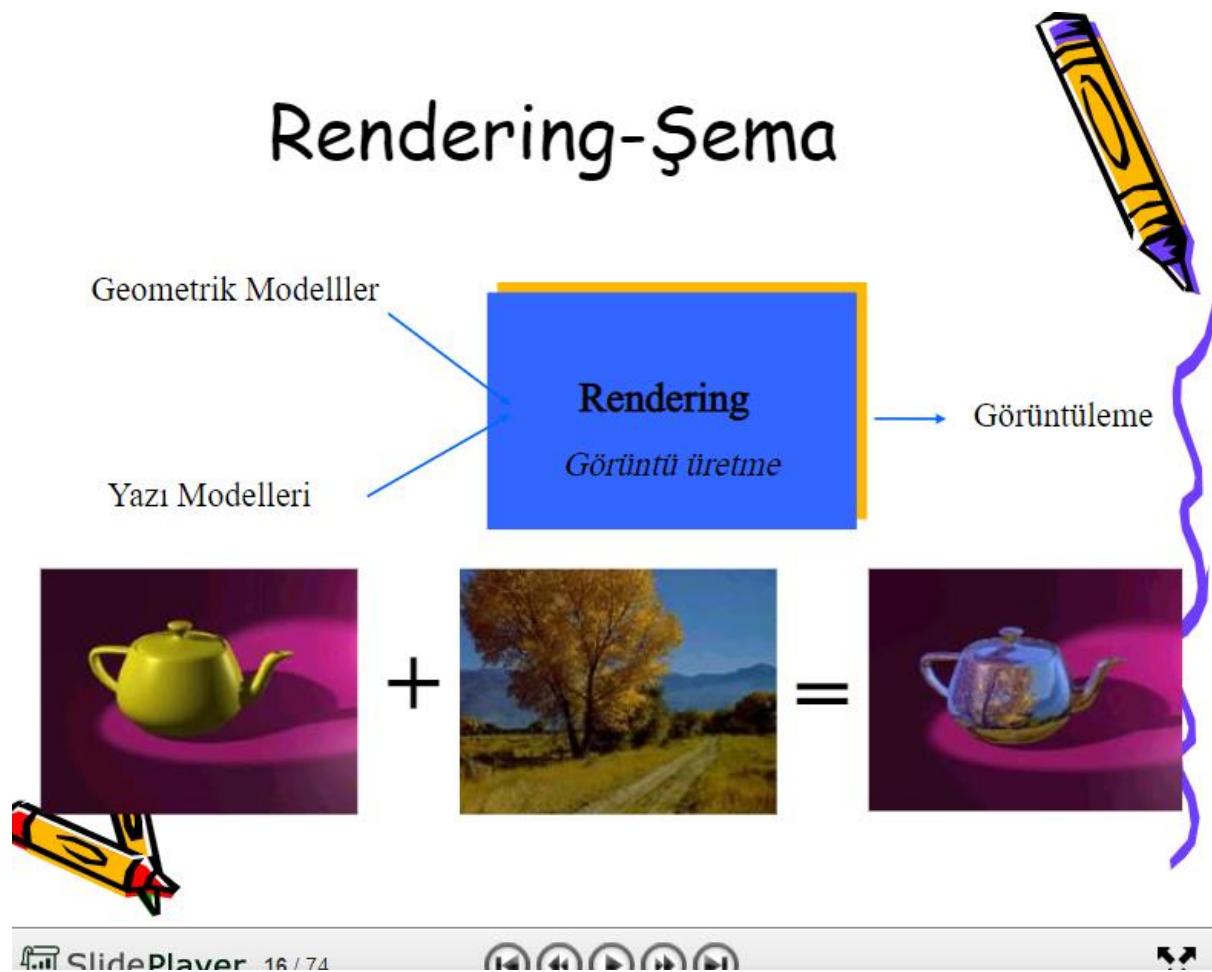
Rendering



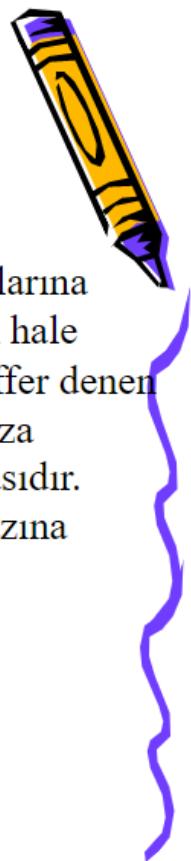
Rendering kelimesi İngilizce *render – sunma* kelimesinden gelmektedir ve sentetik modelin görüntüleme cihazlarının anlayabileceği şekilde dönüştürülerek sunulmasıyla ilgilidir. Rendering, görüntü parçalarını (geometrik ve texture modelleri) bir araya getirerek bir bütün haline getirme işlemidir. Asıl görüntü üretimi bu aşamada gerçekleştirilir. Üretilen görüntü görüntüleme cihazları vasıtasıyla kullanıcıya sunulur.



Rendering-Şema



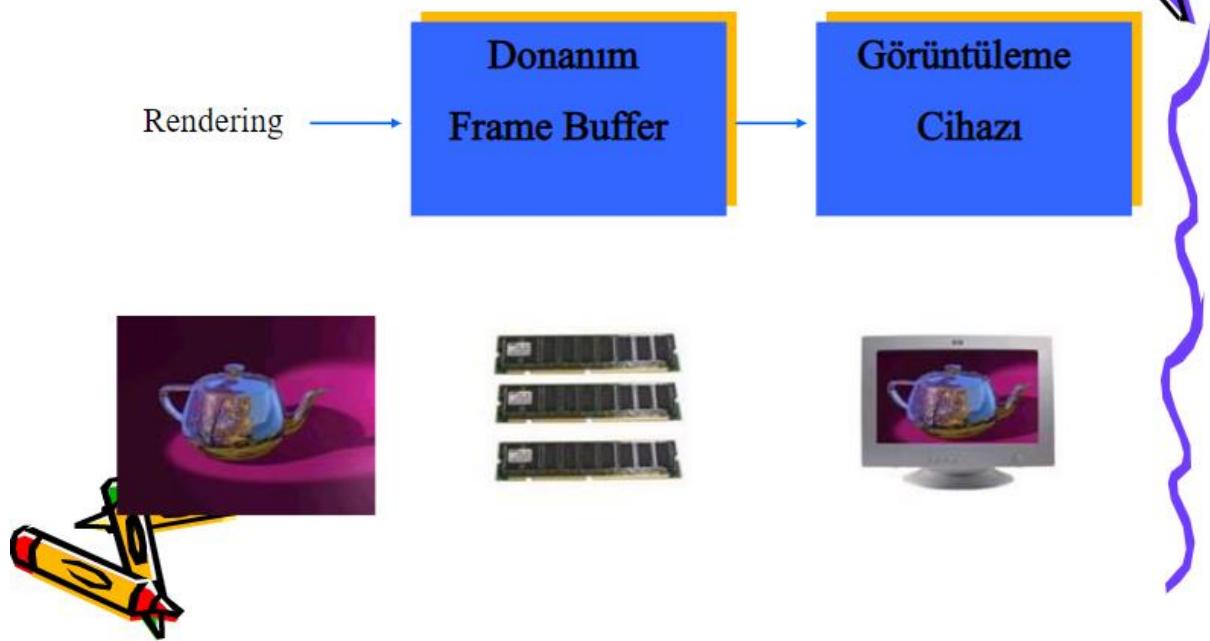
Görüntüleme



Görüntüleme aşamasında görüntünün görüntüleme cihazlarına iletilmesi ve görüntülenmesi ile ilgili konuları kapsar. Son hale getirilmiş görüntü verisi (renderlenmiş görüntü) frame buffer denen ve sadece görüntü saklanması için tahsis edilmiş olan hafıza biriminde tutulur. Frame buffer grafik sisteminin bir parçasıdır. Grafik sistemi buradaki grafik verilerini görüntüleme cihazına gönderir.



Görüntüleme-Şema

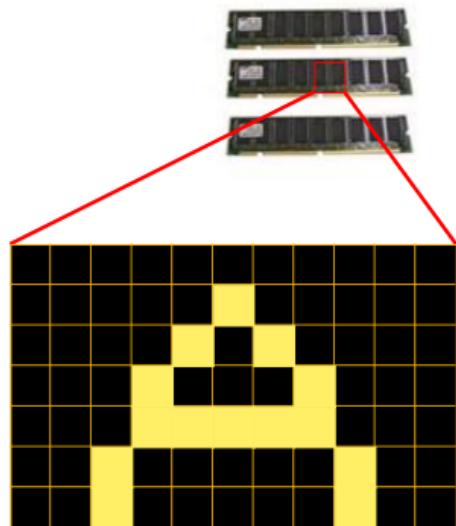




Frame Buffer

Frame Buffer

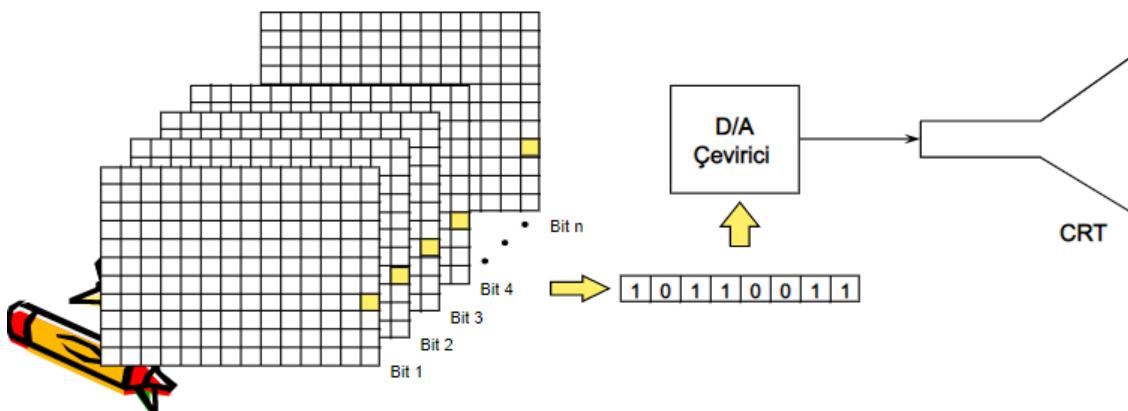
Görüntülenecek görüntüyü depolamak için kullanılan hafıza birimidir. Sadece görüntü verisini tutmakla yükümlüdür. İki tür frame buffer mevcuttur. Bunlar monochrome (siyah beyaz) ve renkli frame bufferlardır.



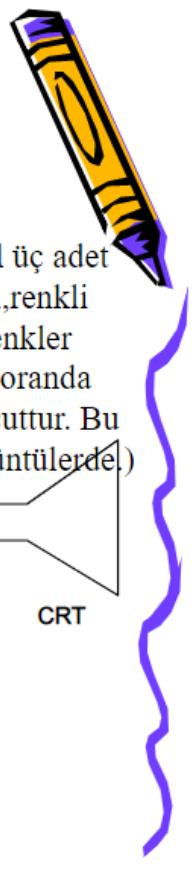
Siyah-Beyaz (monochrome) Frame Buffer



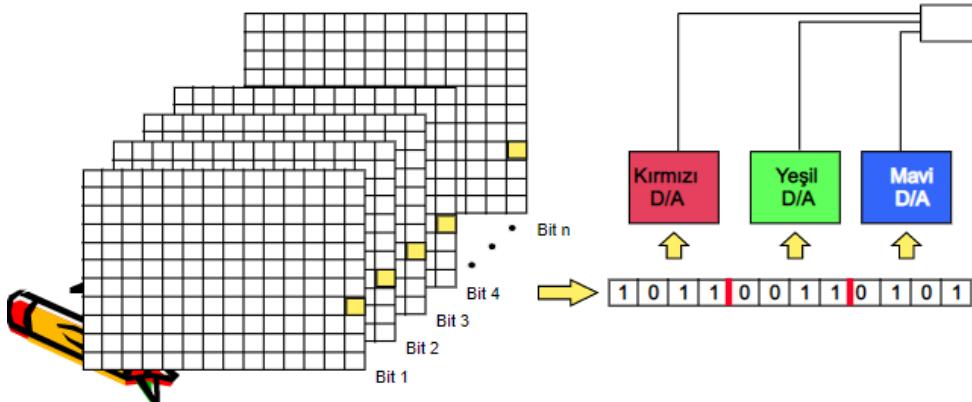
Siyah-Beyaz frame bufferda bir adet dijital/analog çevirici mevcuttur. Bu d/a çevirici frame bufferda tutulan dijital görüntü verilerini analog bir cihaz olan CRT cihazının anlayabileceği şekilde çevirmek ile yükümlüdür.



Renkli Frame Buffer



Siyah beyaz frame bufferin aksine renkli frame bufferda bir adet değil üç adet dijital/analog çevirici bulunmaktadır. Üç d/a çevirici olmasının sebebi, renkli dijital görüntülerde 3 adet temel rengin kullanılması ile ilgilidir. Bu renkler kırmızı, yeşil ve mavidir. Bütün diğer renkler bu üç rengin bellirli bir oranda karışımılarından elde edilir. Her bir renk için ayrı bir d/a çevirici mevcuttur. Bu da grafik görüntüleme sürecini hızlandıran faktörlerdendir (renkli görüntülerde.)



Bazı Genel Kavramlar



Piksel: Birim – frame buffer da tutulan bir birim görüntü verisine denir.

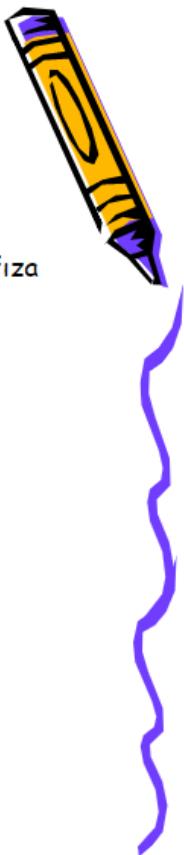
Görüntü boyutu : Görüntünün piksel cinsinden eni ve boyu anlamına gelir.

Renk Derinliği : Piksel başına düşen bit sayısı

Buffer boyutu : Buffer için ayrılan toplam hafıza miktarı



Ne Kadar Hafıza?



Bir görüntüyü Frame Bufferda saklamak için ne kadar hafıza gereklidir? Bu hafıza nasıl hesaplanır? Bunun için aşağıdaki formülü kullanmak yeterlidir :

$$\text{Gereken Buffer boyutu} = \text{genişlik} * \text{yükseklik} * \text{renk derinliği}$$

Burada genişlik ve yükseklik görüntü parametreleridir.

Örnek:

Eğer: Genişlik = 640, Yükseklik = 480, renk derinliği = 24 bit
Buffer boyutu = $640 * 480 * 24 = 921,600$ byte

Eğer Genişlik = 640, Yükseklik = 480, renk derinliği = 32 bit
Buffer boyutu = $640 * 480 * 32 = 1,228,800$ byte



Display (görüntüleme) Cihazları



Günümüzde yaygın olarak kullanılan iki tip görüntüleme teknolojisi mevcuttur.
Bunlar :

- CRT (Cathode Ray Tube) teknolojisi
- LCD (Liquid Crystal Displays) teknolojisi

CRT, daha fazla televizyon ve bilgisayar monitörlerinde kullanılırken LCD teknolojisi daha çok tajınabilir bilgisayar, avuç içi bilgisayarlar, kol saatlerinde kullanılmaktadır.



CRT



LCD

Çıktı Cihazlarının sınıflandırılması

- Sert Çıktı (Hard Copy)
 - Yazıcılar
 - Nokta matris
 - Püskürtmeli
 - Thermal transfer
 - Laser
- Yumuşak Çıktı (Soft copy)
 - CRT ekranlar
 - Düz panel
 - LCD ekranlar
 - Electrolumines cent
 - Plasma
 - Projektör





CRT (Cathode Ray Tubes) Teknolojisi

Günümüzde kullandığımız masa üstü pc ve workstation'ların yanı sıra hemen hemen bütün televizyon ekranları bu teknolojiyi kullanmaktadır. CRT cihazları analog cihazlardır ve dijital bir cihaz olan bilgisayar tarafından kullanılır. CRT'nin bileşenleri şunlardır : **elektron tabanca, focusing system, deflection system, fosfor kaplı ekran.**

Elektron tabancasından ateşlenen ışık demeti fosfor kaplı ekranın belli bir bölgesine çarparak değişik renklerde ışınlar meydana getirir. Bu ışınların ömrü kullanılan fosfor tipine değişmekte olup 10-60 mikrosaniye arasındadır. Oluşturulan görüntünün ekranda bir müddet kalabilmesi için bu ışınların belli bir aralıklarla tekrarlanması gereklidir. Bu işlemeye refreshing denir ve ekranı tazeleme anlamına gelir. Ekranın tazelenme hızına da refresh rate denir.

Avantajları:

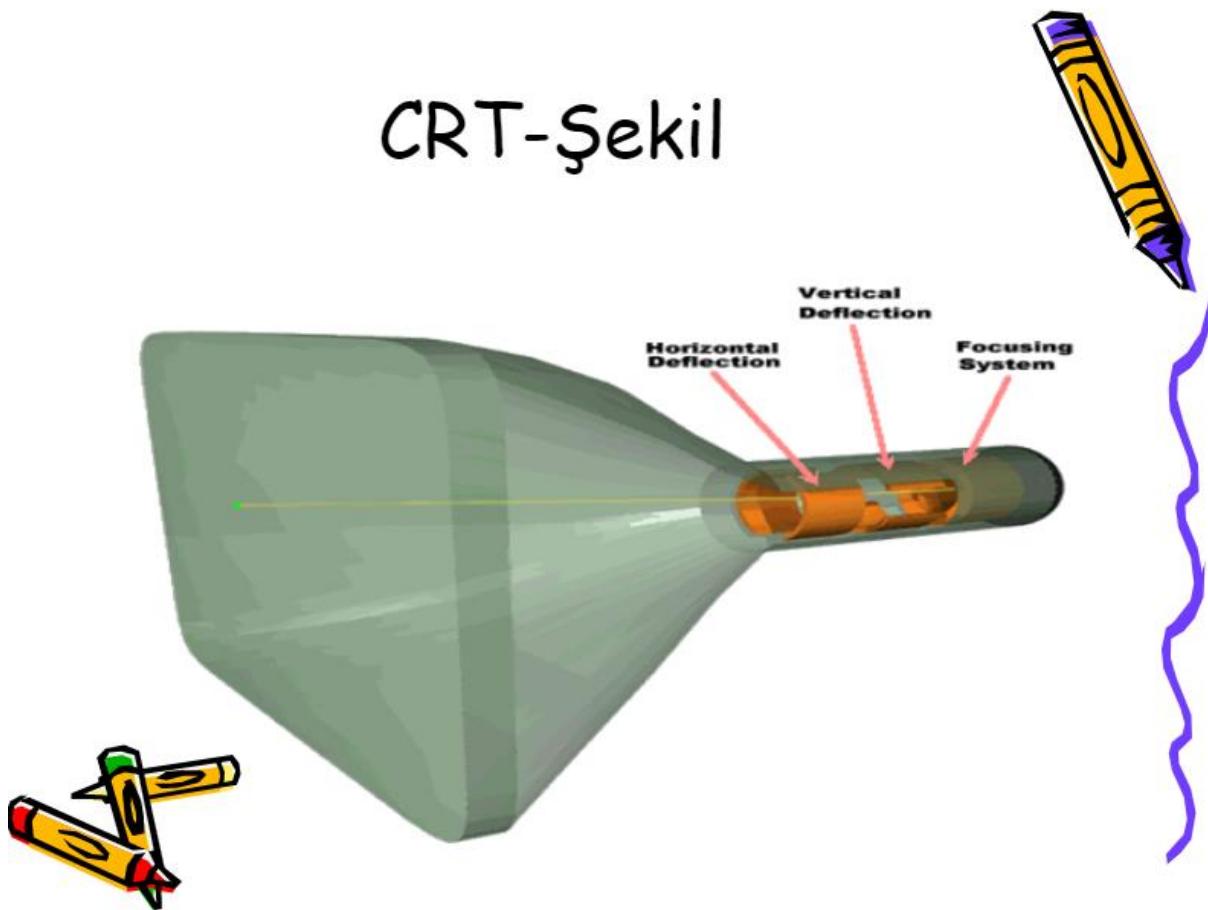
- Düşük fiyat
- İyi görüntü kalitesi
- Geniş görüntüleme açısı
- Kolay grayscale
- Kolay adresleme



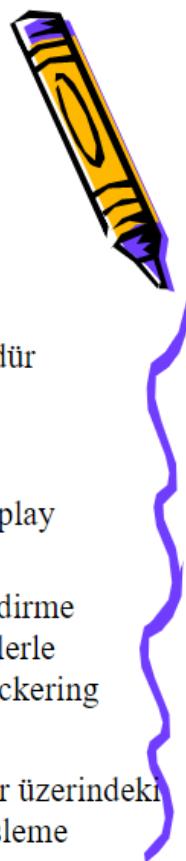
Dezavantajları:

- Büyük hacim/ağırlık
- Boyut 45 inçten küçük
- Tekrarlanabilirlik
- Yüksek voltaj

CRT-Şekil



CRT (Cathode Ray Tubes) Bileşenlerinin Görevleri



CRT bileşenlerinin görevleri aşağıdaki şekildedir:

- Elektron Tabanca** : Elektron üretme ve gönderme işlemini yapar
- Focusing System** : Ekrana çizilecek şekli oluşturmakla yükümlüdür
- Deflection System** : Elektronun ekrana çarptığı yerde ışık oluşturur
- Fosfor Kaplı Ekran** : Görüntünün uygun yerde oluşmasını sağlar

CRT teknolojisi 2 ye ayrılır: Raster scan display ve Random scan display cihazları.

Random Scan Display : Görüntü refresh edilmesi gereklidir. Gölgeleştirmeye yapılamaz. Görüntü bir noktadan başlayarak son noktaya kadar çizgilerle oluşturulur. Bu nedenle büyük görüntülerde ekran gidip gelir. Bu flickering denir.

Raster Scan Display : Görüntü çizgiler şeklinde oluşturulur. Çizgiler üzerindeki noktalar kontrol edilir ve gerekli bölgelerdeki renkleri ayarlanır. Bu işlem **rasterizasyon** denir. Random scan'dan daha gelişmiş teknolojidir.

LCD (Liquid Crystal Displays)



Avantajları

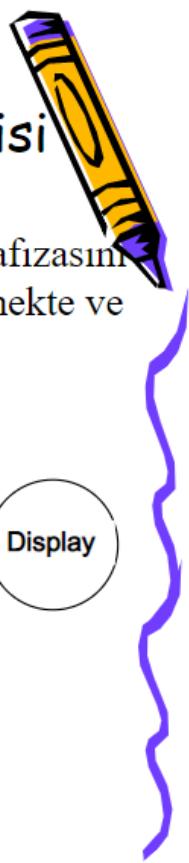
İnce/Hafif
Sağlam ve güvenilir
Düşük voltaj

Dezavantajları

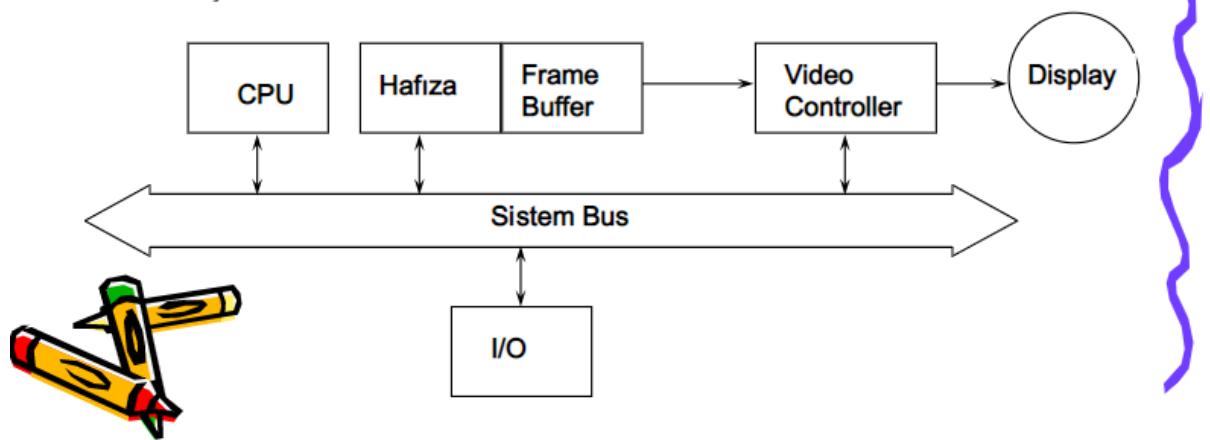
Görüntü < 13 inch (pasif)
 < 16 inch (aktif matris)
Yanıt zamanı (response time)
Fiyatı yüksek
Sınırlı görüntüleme açısı



Basit Raster Display Sistem Mimarisi



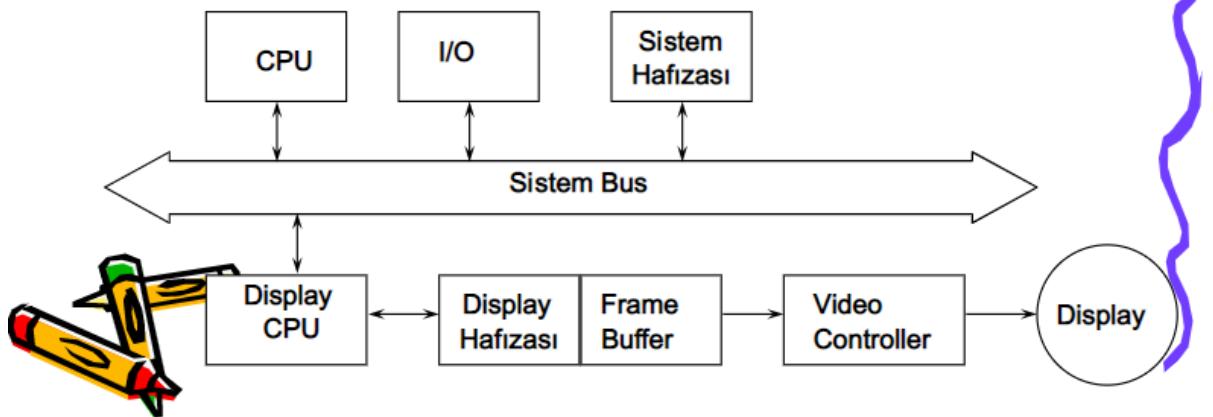
Bu sistemde Grafik sistemi sistem işlemcisini ve sistem hafızasını kullanmaktadır. Bu nedenle işlemci boş yere meşgul edilmekte ve verimi düşmektedir. Verim düşüşünü önlemek için grafik sisteminin bünyesine görüntü işlemcisi ve video hafızası eklenmiştir.



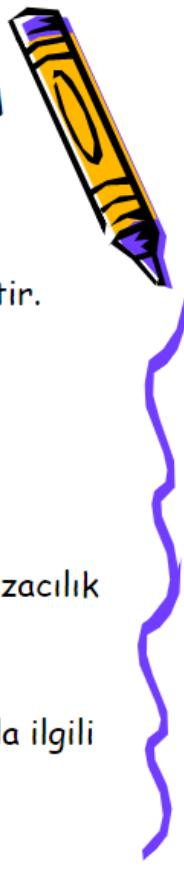


Görüntü İşlemcisine Sahip Raster Display Sistem Mimarisi

Aşağıdaki şekilde daha gelişmiş bir sistem gösterilmektedir. Görüntü sisteminin kendi görüntü işlemcisi ve görüntü hafızası mevcuttur. Dolayısıyla işlemciyi çok meşgul eden görüntüleme işlemleri sistem içindeki diğer işlemlerden özerk hale getirilmiş ve sistemin verimi artırılmıştır.



Bigisayar Grafiklerinin kullanım alanları



Günümüzde bilgisayar grafiklerinin kullanım alanları çok genişdir.
Aşağıdaki örnekler bunun en çok kullanılanlarıdır.

Dizayn : mimari, makine tasarıımı, moda alanında

Simülasyon : eğitim, uçuş ve araba sürme simülasyonları

Sanat ve Eğlence : oyunlar, filmler ve reklamlar

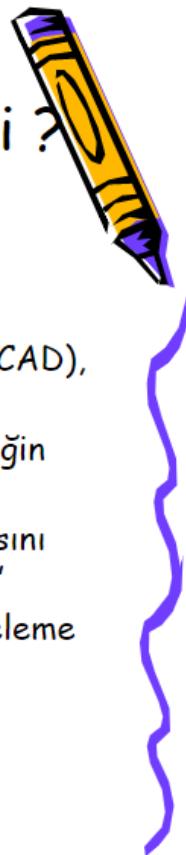
Bilgi Sunma : bilimsel vizualizasyon: hava durumu, kimya ve eczacılık

Kullanıcı Arayüzleri : pencereleme sistemi, sanal gerçeklik

Sıradaki slaytlarda yukarıda sıralanmış olan kullanım alanlarıyla ilgili uygulamaları görebilirsiniz.



Peki Neden Bilgisayar Grafikleri ?



Aşağıdaki nedenlerden dolayı bilgisayar grafikleri ürünleri tercih konusu olmuştur :

- **Fiyat** : sistemin ucuz prototiplerini oluşturma olanağı (CAD), eğitim (örneğin uçuş simülasyonu vb.) ucuzluğu.
- **Mümkün olmayanı mümkün kılama** : sanal gerçeklik. Örneğin mars gezegeninde yürüme simülasyonu.
- **Bandwith** : Band genişliği. İnsanları daha çabuk anlamasını sağlama. Yani "Bir resim bin sözden daha iyi anlatabilir."
- **Kullanıcı Dostu** : 3 boyutlu kullanıcı arayüzleri, pencereleme sistemi



Neler Yapabiliriz...



Bilgisayar Grafikleri Uygulamaları-Filimler

- Filimler – Geri's Game



"Geri's Game"
Academy Awards Ödülü Sahibi- animasyonlu en iyi filmler film dalında, 1997.



"Jurassic Park"
3 adet Academy Awards® ödülü-görsel ve ses efektleri için



Bilgisayar Grafikleri Uygulamaları-Filmiler-1

- Filimler

"Shrek"
Dreamworks - 2001.



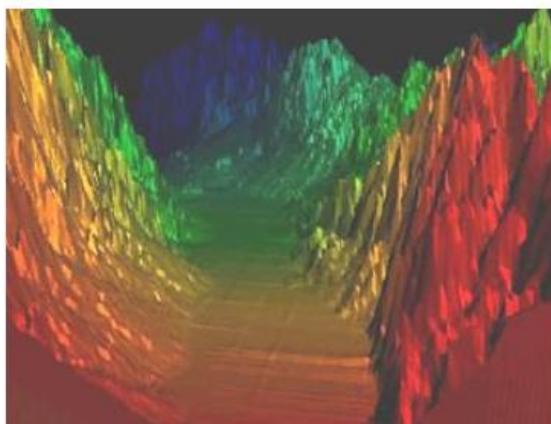
Bilgisayar Grafikleri Uygulamaları-Oyunlar



"Tekken Tag Tournament"
Playstation 2

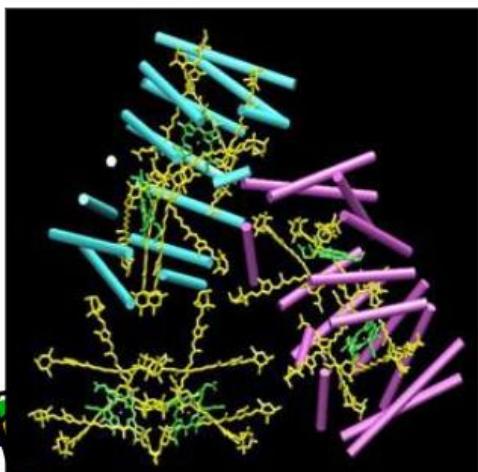
Bilgisayar Grafikleri Uygulamaları-Bilimsel Vizualizasyon

-Bilimsel Vizualizasyon



Bilgisayar Grafikleri Uygulamaları-Bilimsel Vizualizasyon-1

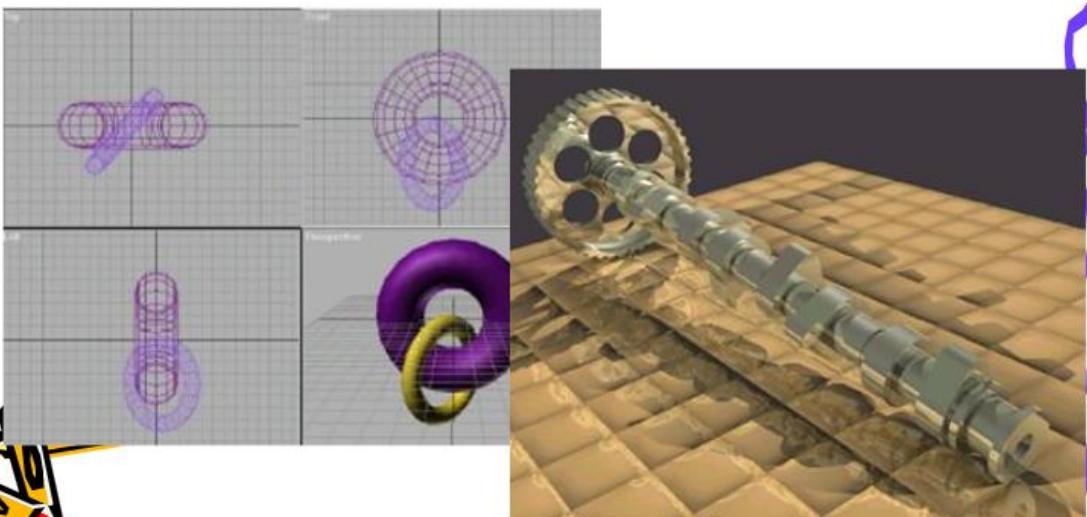
-*Bilimsel Vizualizasyon (visualization)*



Bilgisayar Grafikleri Uygulamaları-CAD

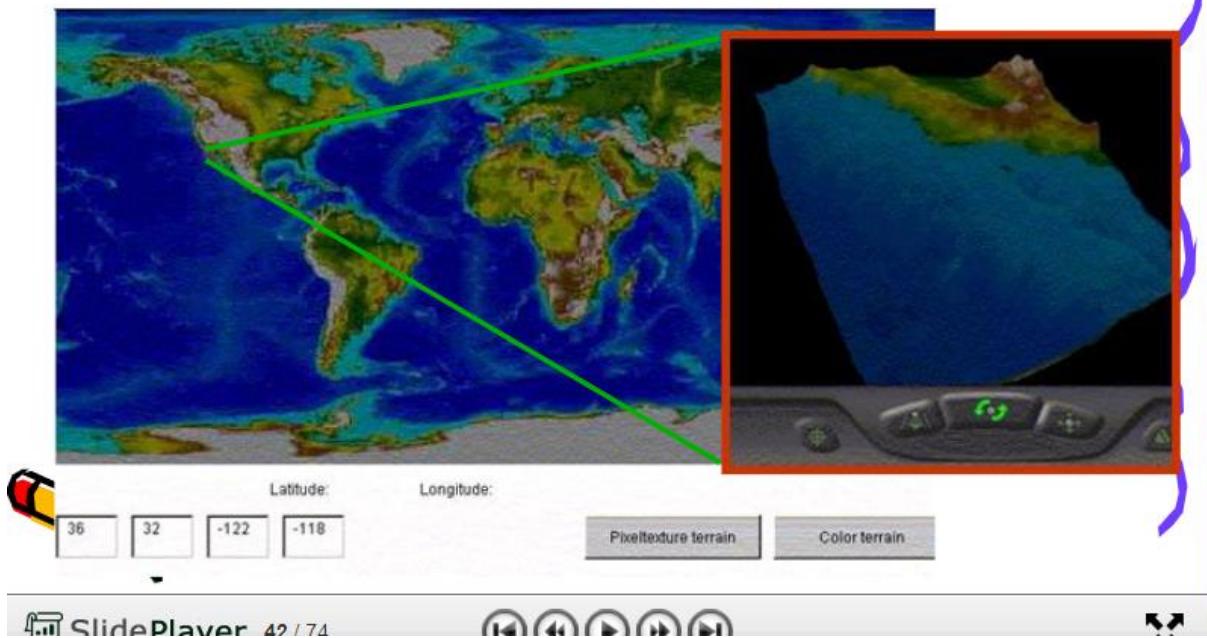


- CAD-Bilgisayar Destekli Tasarım



Bilgisayar Grafikleri Uygulamaları-Web Grafikleri

-Web Grafikleri

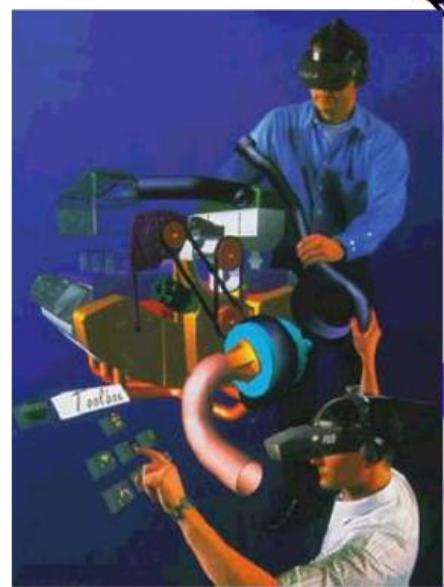


Bilgisayar Grafikleri Uygulamaları-Sanal Gerçeklik

-Sanal Gerçeklik

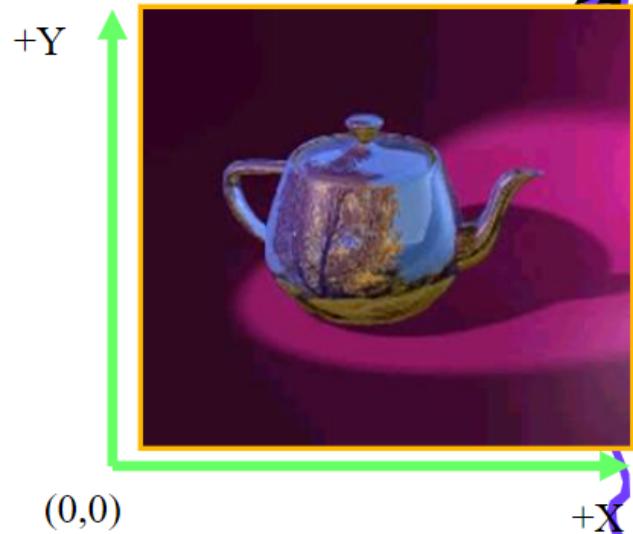
-Genişletilmiş Gerçeklik (Augmented Reality)

-İnsan Etkileşimi



İki Boyutlu Görüntüler

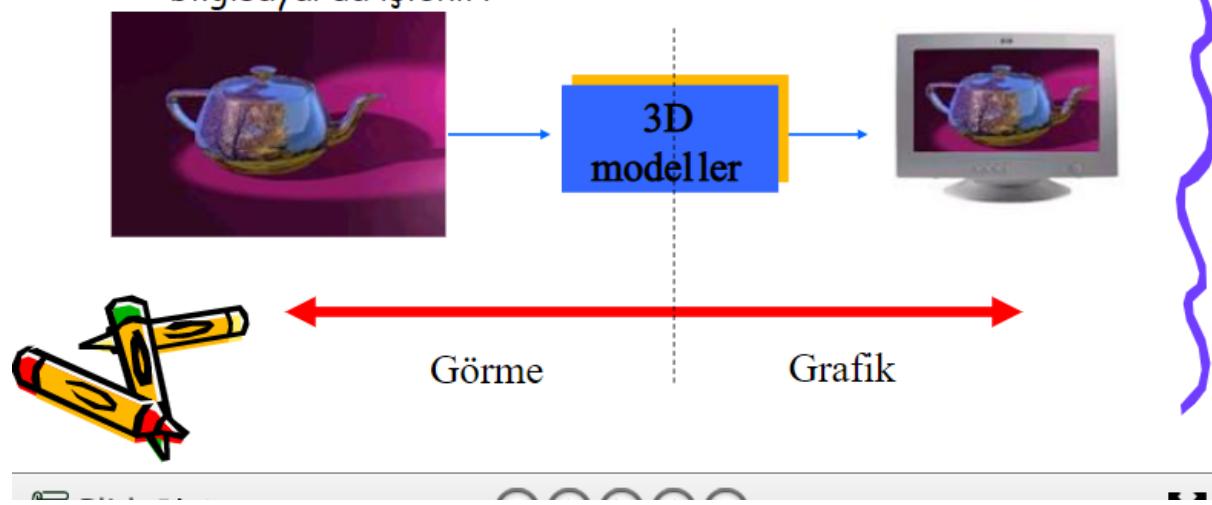
- Görüntüler iki boyutlu (2D) şekillerdir
- X ekseni (yatay), ve Y ekseni (düsey) kullanılarak tanımlanırlar
- Görüntüler piksellerden oluşmaktadır



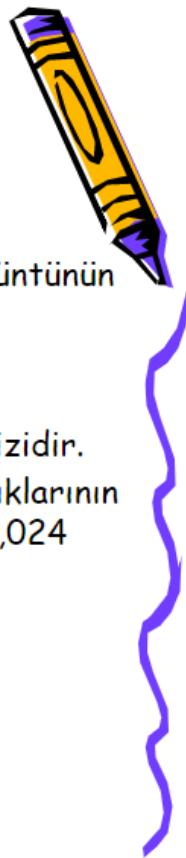
Görüntüler



- Grafik Görüntüler Sentetiktirler. Model fiziki olarak olabilirde olmayabilirde . Geleneksel görüntüler gibi oluşturulurlar. Gerçek hayatı gördüğümüz gerçek objeler 3Boyutlu modeller şeklinde grafik verisine çevirilerek bilgisayarda işlenir.



Görüntüler-1



Görüntüler piksellerden oluşur. Piksellerden oluşan görüntünün kalitesi renk yoğunluğu ve çözünürlüğe bağlıdır.

Piksəl : Dijital görüntünün en küçük bölünemeyen birimidir. Pikseller baştan başa her zaman aynı renkteler. Görüntü , piksellerden oluşan 2 boyutlu bir dizidir.

Çözünürlük : Görüntüdeki piksellerin birbirine olan uzaklıklarının ölçüsüne denir. Eğer 12 inç genişliğindeki bir ekranada 1,024 piksel görüntüleniyorsa , görüntü- inç başına 85 piksel çözünürlüğündedir.



Unutmayın...



- **Grafik Görüntüler Sentetiktir**
 - Model fiziki olarak olabilir yada olmayabilir
 - Geleneksel görüntüler gibi oluşturulurlar
- **Görüntüler piksellerden oluşur**
 - Yoğunluk/renk
 - Çözünürlük
- **Algılama**
 - Işıklandırma
 - İnsanın Görme Sistemi



- Input Cihaz Tipleri
- Interaktif Grafik Programlama





Etkileşim

Etkileşim, bilgisayar grafikleri uygulamalarının en önemli bileşenlerindendir. Buradaki etkileşim uygulama programı ile grafik kütüphaneleri arasındaki etkileşimi içine almakla birlikte kullanıcı-uygulama programı ekseni etrafında yoğunlaşmaktadır. Üst seviye etkileşim olarak adlandırdığımız kullanıcı-uygulama programı seviyesinde kullanıcı tarafından uygulama programına verilen komutlar ve programdan alınan yanıtlar söz konusudur. Uygulama programı aynı zamanda üst seviye etkileşimde elde ettiği verileri alt seviye etkileşim olarak adlandırdığımız "uygulama programı-grafik API-Grafik Sistemi" sürecine taşır.



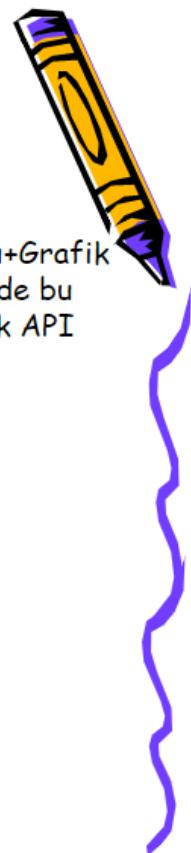


Etkileşim

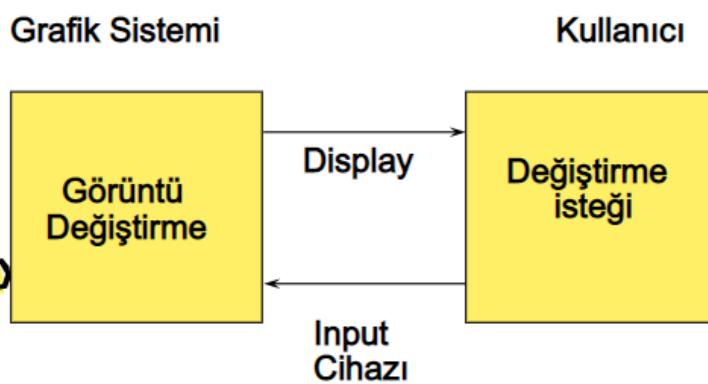
Etkileşim, bilgisayar grafikleri uygulamalarının en önemli bileşenlerindendir. Buradaki etkileşim uygulama programı ile grafik kütüphaneleri arasındaki etkileşimi içine almakla birlikte kullanıcı-uygulama programı eksenin etrafında yoğunlaşmaktadır. Üst seviye etkileşim olarak adlandırdığımız kullanıcı-uygulama programı seviyesinde kullanıcı tarafından uygulama programına verilen komutlar ve programdan alınan yanıtlar söz konusudur. Uygulama programı aynı zamanda üst seviye etkileşimde elde ettiği verileri alt seviye etkileşim olarak adlandırdığımız "uygulama programı-grafik API-Grafik Sistemi" sürecine taşır.



Grafik Sistemi İle Etkileşim

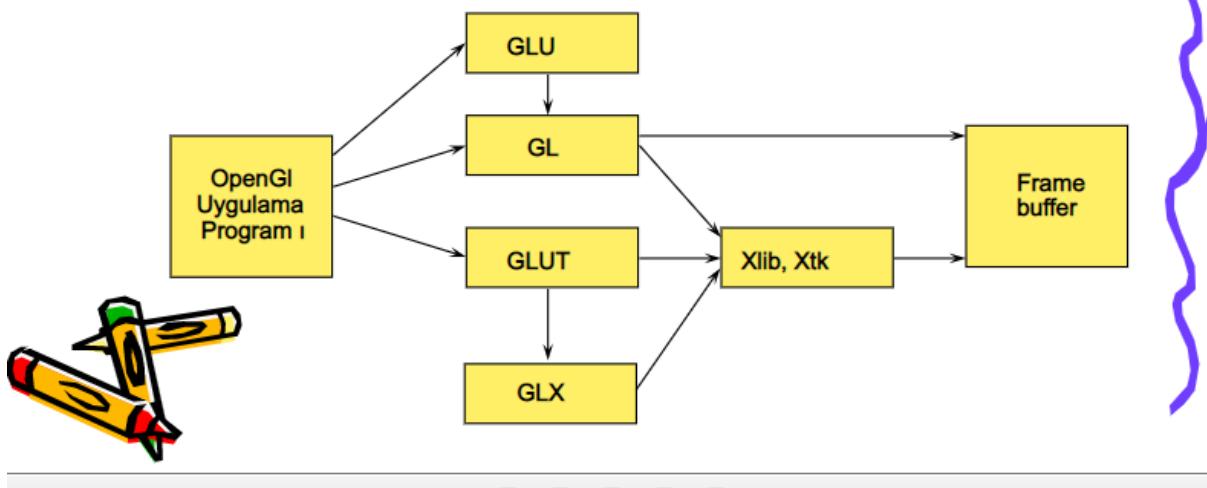


Grafik sistemi ve kullanıcı arasındaki etkileşim; uygulama programı+Grafik API toplamı tarafından gerçekleştirilmektedir. Aşağıdaki şekilde bu tablo kabaca tasvir edilmiştir. Ara katman olan uygulama-grafik API katmanı gizli katmandır.



GL Kütüphanesi Organizasyonu (X Windows altında)

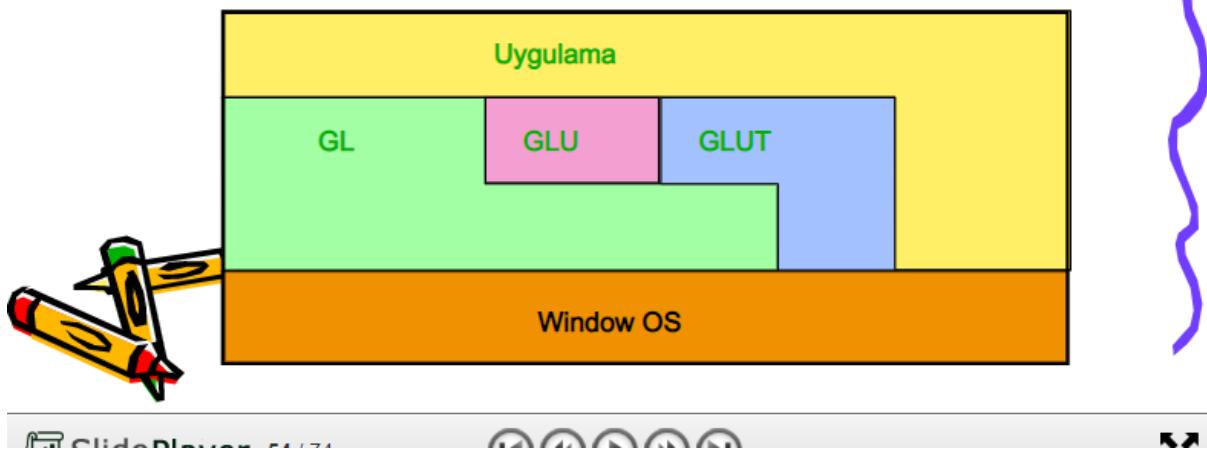
X Window grafik sisteminde uygulama programı temel ve yardımcı OpenGL kütüphaneleri yardımıyla işletim sisteminin grafik kütüphaneleri olan Xlib ve Xtk ile etkileşimde bulunur.

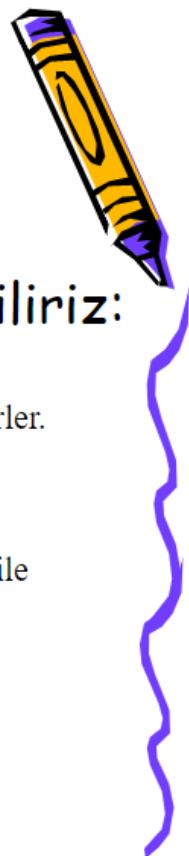


GL Kütüphanesi Organizasyonu (MS Window altında)



MS Windows ortamında durum biraz farklıdır. Win32 sisteminde uygulama programı OpenGL'in temel ve yardımcı kütüphanelerinin yanısıra direkt olarak işletim sisteminin kaynaklarından faydalananabilmektedir.





Input Cihazları

İnput Cihazlarını 2 kategoriye ayıralım:

- **Fiziki cihazlar** – gerçek-dünya fiziki özellikleriyle tanımlanabilirler.
(fare, klavye, joystick...)
- **Lojikal cihazlar** – kullanıcı programının yüksek seviye arayüzü ile karakterize edilir. Cihaz verisinin soyutlanmış şeklidir.
(fonksiyonlar, windows cihaz sürücüleri)





Fiziki İinput Cihaz Türleri

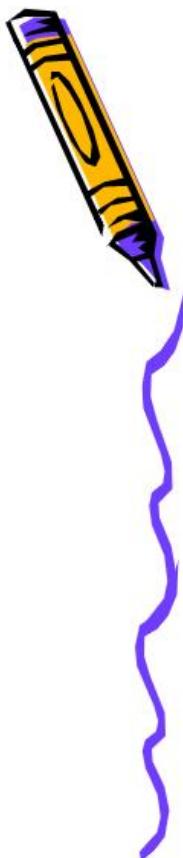
- Klavye
- Seçim cihazları
- Locator'lar

Yukarıdaki fiziki cihaz türlerinden
herbiri ileriki slaytlarda açıklanmıştır.



Klavye

- Spesifik özelliklere sahip karakterler döndürür



Seçim

- Bir kaç opsion içinden seçilenini döndürür. Genel olarak önceden tanımlanmış olan işlemler mevcuttur ve kullanıcı bu işlemlerden herhangi birini seçmektedir.



- Buton Kutusu
- Fonksiyon keypad





Locator Cihazlar

- konum ve/veya yönlendirme (orientation) bilgisi döndürürler. Bu cihazların aşağıdaki türleri mevcuttur :

- Fare
- Trackball
- Tablet
- Joystik
- Touch screen



3D Input Cihazlar

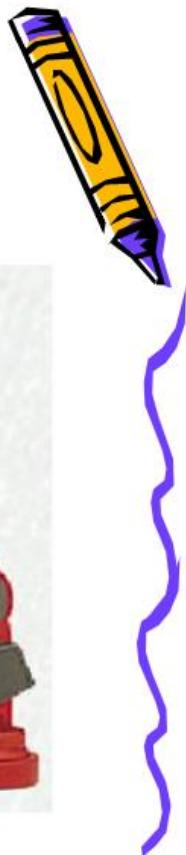
- 3 boyutlu konum ve / veya yönlendirme(orientation) bilgisi döndürürler. Aşağıdaki türleri mevcuttur :

- Digitizer
- 3D Spaceball
- Glove



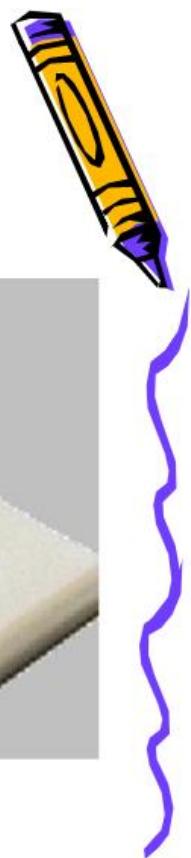
3D Input Cihazları - Digitizer

Digitizer
- 3D model



3D Input Cihazları- Spaceball

3D Spaceball



3D Input Cihazlar-Glove (eldiven)



Tracking özellikli el ve
konum / yönlendirme

Haptic Glove



3D Input Cihazları- Tracker



Hareket Tracker'ı

- Manyetik
- Akustik
- Inertial(atıl)
- Optikal





Lojikal Cihazlar

Uygulama programı açısından karakterize edilen cihazlardır. Aşağıdaki türleri mevcuttur:

- Kullanıcı programının yüksek-seviye arayüzü olarak
- Cihaz verisinin soyutlanmış şekli olarak



Lojikal Cihaz Tipleri

- **Katar (String)**
 - ASCII katarlarını döndürür
- **Locator**
 - Konum ve yönlendirme bilgisi döndürür
- **Pick**
 - Objetanımlayıcı bilgisi döndürür
- **Seçim(choise)**
 - Bir kaç opsyon içinden seçilenini döndürür
- **Çevirim (Dial)**
 - Analog input'u döndürür
- **Stroke** - Konum dizisi döndürür.



Input Cihazlarının Davranışları

İnput cihazlarının davranışları iki şekilde tanımlanır.

1-Ölçme: cihaz , kullanıcı programına ne döndürür

2-Trigger: ölçümelerin döndürülmesi



Input Cihazını Okuma Yolları



İnput cihazından üç şekilde okuma
yapılabilir. Bunlar :

- Örnek modu(sample mode)
- İstek modu(request mode)
- Olay modu(event mode)



Örnek Modu(sample mode)



Çalışma şekli aşağıdaki gibidir:

"Merhaba, şu anda cihazda *ne var?* Bana veriyi *hemmen* ver!"

Ölçme
süreci

Program



- trigger'a ihtiyaç yok
- anında döndürme
- Fonksiyon çağrısından önce veri hazırlanır.

İstek Modu(request mode)



Cihaz tetiklenene kadar ölçülen değerler döndürülmez. Tetikleme geldiği anda ölçülen değerler istek birimine gönderilir.

Trigger
Süreci

Ölçme
Süreci

Program

- Tetiklemeyi bekler
- C deki “ scanf() ” fonksiyonu gibidir

Olay Modu-1

**Cihaz tetiklenene kadar ve
kullanıcı herhangi bir şey yapana
kadar bekle**

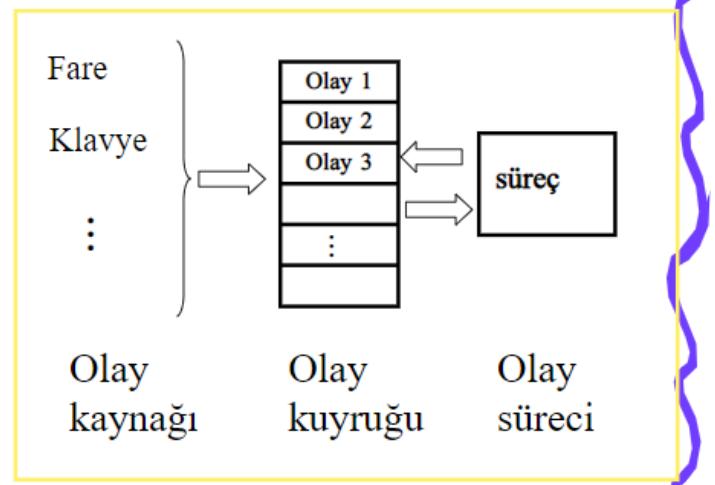


- İki asenkron süreç
 - Olay üretimi (cihaz tetiklenmesi)
 - Kullanıcı isteği (olay sorgulama)
- Olay üretimi ve süreçler birbirinden bağımsız
 - Birden fazla input cihazı olabilir

Olay Modu-2

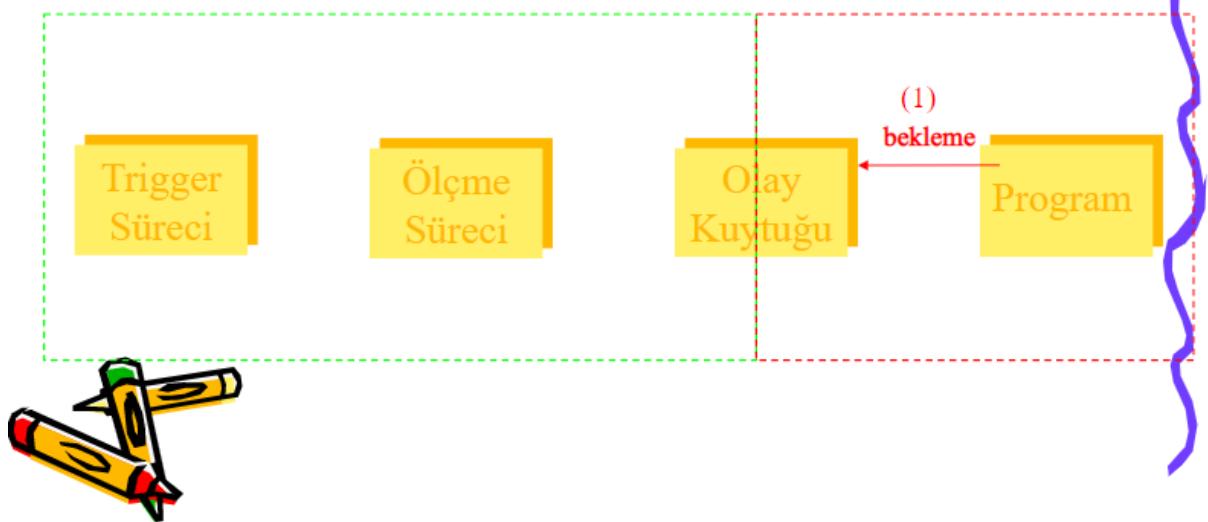
Windows'un olay mekanizmasını ele alalım

- Cihaz triggeri ile bir olay üret
- Olayı "olay kuyruğu" na koy
- olay kuyruğunu ve süreci incele



Olay Modu-3

İki asenkron süreç





- OpenGL'e giriş
- İlk OpenGL uygulamaları
- Ödev: OpenGL'i kullanmayı düşündüğünüz herhangi bir programlama dilinde yapılması gereken ayarlamalar ve indirilmesi gereken kütüphanelerin neler olduğunu araştırınız. Bir deneme program kodunu yazarak deneyiniz.



Bilgisayar Grafikleri

Kaynak Kitaplar :

Mathematical Elements for Computer Graphics

David F.Rogers, J.Alan Adams

McGraw-Hill Publishing Company

Procedural Elements for Computer Graphics

David F.Rogers

McGraw-Hill Publishing Company

Konular

- OpenGL'e giriş
- 2D ve 3D nokta ve cisim tanımı
- 2D Temel işlemler
- 2D uzay eğrileri
- 3D Temel işlemler
- Projeksiyon
- 3D uzay yüzeyleri
- Saklı yüzey
- Saklı kenar
- Kaplama
- Aydınlatma

OpenGL'e giriş

OpenGL'in Avantajları

- Tüm OpenGL uygulamaları, işletim sistemi ne olursa olsun, OpenGL API uyumlu donanımlar üzerinde mükemmel görsel sonuçlar üretebilir.
- Grafik donanımlarının yeni gelişmiş özellikleri, OpenGL tarafından, geliştirme mekanizması (extension mechanism) sayesinde kullanılabilir. Yani OpenGL, donanıma özel, gelişmiş özellikleri kullanmak için API fonksiyonları içerebilir.
- OpenGL temelli grafik uygulamaları, çok çeşitli sistemler üzerinde koşulabilir. (tüketici elektroniği – consumer electronics, PC, iş istasyonu – workstation, süper bilgisayarlar gibi)
- OpenGL grafik kütüphanesi kullanılarak, çok daha az bir kod satırıyla daha yüksek performansa sahip uygulamalar geliştirmek mümkündür.
- OpenGL grafik kütüphanesine dair teknik bilgi içeren birçok kaynak mevcuttur.(internet, kitaplar, vs.)
- Birçok programlama dili (C, C++, Fortran, Ada, Java gibi) OpenGL tabanlı uygulama geliştirmemize olanak sağlar.

OpenGL Söz dizimi

Opengl komutları, gl öneki ile başlarlar. (örnek; glClearColor()). Benzer şekilde OpenGL tarafından tanımlı sabitler de GL_ öneki ile başlarlar ve kelimeler birbirinden _ ile ayrılacak şekilde büyük harfle yazılırlar.

(örnek; GL_COLOR_BUFFER_BIT).

glColor3f komutundaki 3 sayısı da 3 parametre alacağı anlamına gelmektedir. f ise verilen parametrelerin float olacağı anlamına gelmektedir.

glVertex2i(1,3); ya da
glVertex2f(1.0,3.0); gibi

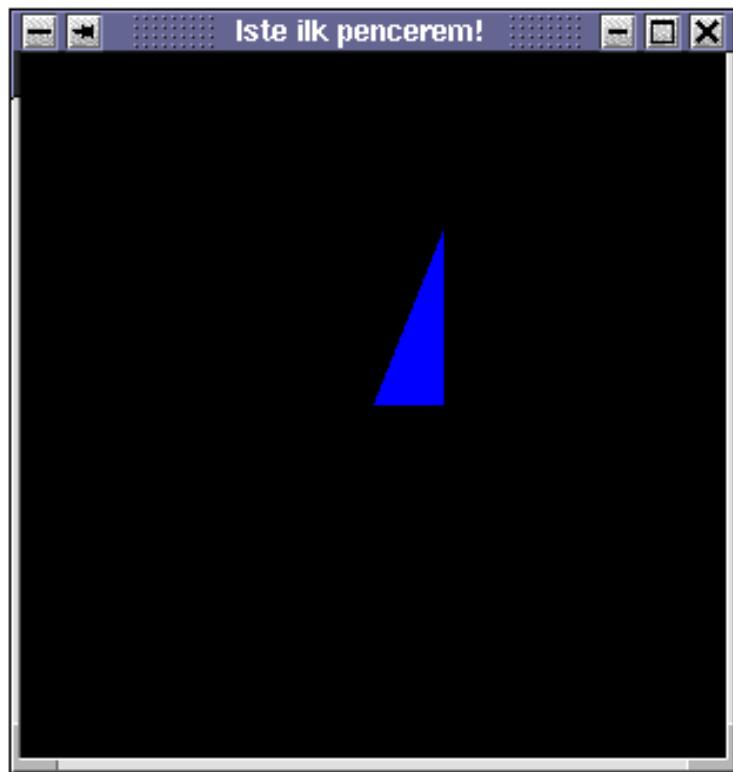
glilk.c

```
#include <GL/glut.h>

void myDisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f (0,0,1);
    glBegin(GL_POLYGON);
        glVertex2f (0.0, 0.0);
        glVertex2f (0.2, 0.0);
        glVertex2f (0.2, 0.5);
    glEnd();

    glFlush();
}

int main (int argc, char ** argv)
{
    glutInit (&argc,argv);
    glutCreateWindow("Iste ilk pencerem!");
    glutDisplayFunc(myDisplay);
    glutMainLoop();
    return(0);
}
```



glKesikCizgi.c

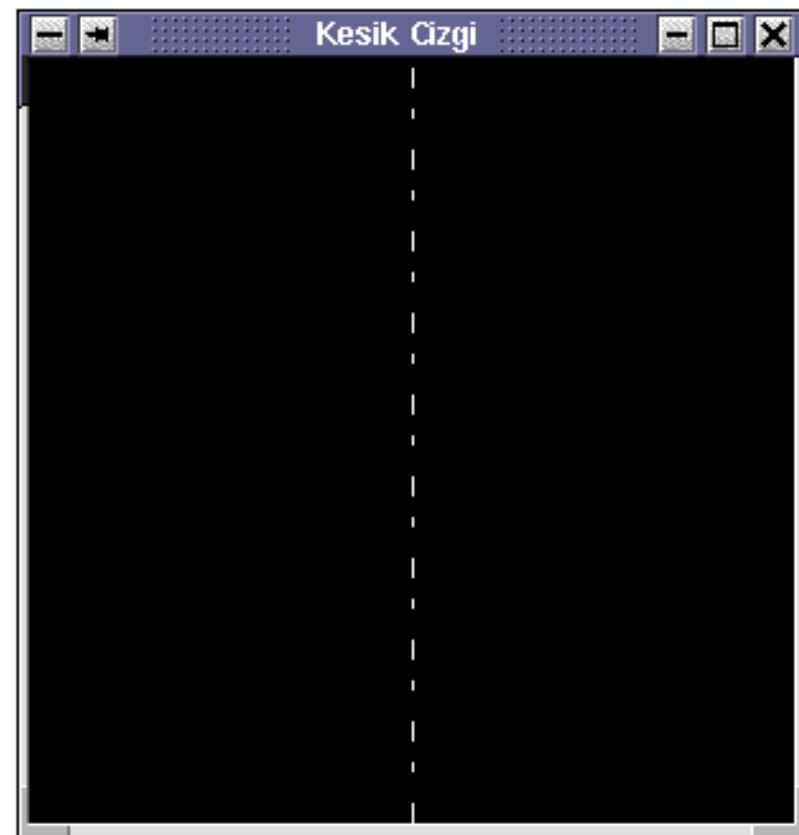
```
#include <GL/glut.h>

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(1,1,1); // white
    glEnable(GL_LINE_STIPPLE);
    glLineStipple(2, 0x0C0F);
    glBegin(GL_LINE_STRIP);
        glVertex2f(0,-1);
        glVertex2f(0,1);
    glEnd();

    glFlush();
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutCreateWindow("Kesik Cizgi");
    glutDisplayFunc(display);
    glutMainLoop();
    return(0);
}
```



glCember.c

```
#include <GL/glut.h>
#include <math.h>
#include <stdio.h>

#define RADIUS 0.75

void cember(void);

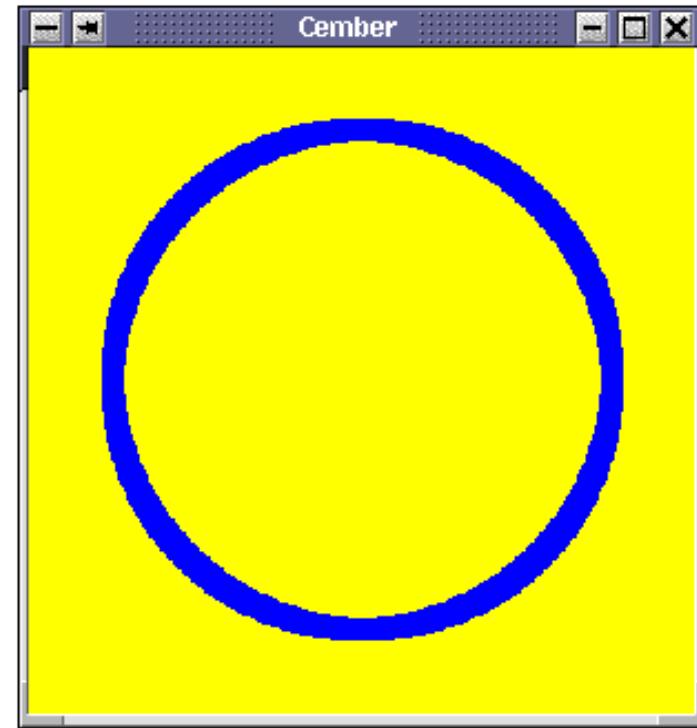
int main(int argc, char ** argv)
{
    glutInit(&argc, argv);
    glutCreateWindow("Cember");
    glutDisplayFunc(cember);
    glutMainLoop();
    return(0);
}

void cember(void)
{
    double x,y;
    int i;

    glClearColor(1.0,1.0,0.0,0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0,0,1); // blue

    glPointSize(10.0);
    glBegin(GL_POINTS);
        for(i = 1; i < 360; i++){
            x = RADIUS * sin(((double)i)*M_PI/180);
            y = RADIUS * cos(((double)i)*M_PI/180);
            #ifdef DEBUG
                fprintf(stderr, "(%f, %f)\n", x, y);
            #endif
            glVertex2f(x,y);
        }
    glEnd();

    glFlush();
}
```



Temel İşlevler

```
void glutInit (int * argc,  
              char ** argv)
```

işlev

GLUT kitaplığının ilkendirilmesi için çağrılmazı gereken işlevdir. Bu çağrılmadan GLUT kitaplığından başka bir işlev çağrılamaz. Argümanlar programın `main()` işlevinden alınır ve değiştirilmeden kullanılır. Kullanımı söyledir:

```
glutInit(&argc, argv); // main(int argc, char ** argv) olduğu varsayılmıştır.
```

```
int glutCreateWindow (char title)
```

işlev

Pencere sisteminde bir pencere oluşturur ve pencere başlığına `title` parametresindeki metni yazar. Pencere numarası ile döner.

```
void glutDisplayFunc (int (*func) (void))
```

işlev

Pencere ekrana çizildikten sonra pencerenin içerisinde gösterilecekleri oluşturan işlevin belirtileceği işlevdir (callback).

```
void glutMainLoop ()
```

işlev

Bu işlev sayesinde program kendisine gelecek olayları (event) dinlemeye başlar ve eğer tanımlıysa gelen olaya göre tanımlanmış bir işlevi (callback) çalıştırır.

```
void glClear (GLbitfield mask)
```

işlev

Tamponların içeriğini `glClearColor`, `glClearIndex`, `glClearDepth`, `glClearStencil` ve `glClearAccum` işlevleri ile belirlenen değerlerle temizler.

Parametre olarak `GL_COLOR_BUFFER_BIT`, `GL_DEPTH_BUFFER_BIT`, `GL_ACCUM_BUFFER_BIT`, `GL_STENCIL_BUFFER_BIT` sabitlerinden birini ya da bunların VEYalanmış değerini alır.

void glBegin (enum *kip*)

Bu işlev bir çizimin başlatıldığını belirtir. Aldığı parametre ise çizilen şeyin noktaları, çizgiler veya içi dolu çizgiler şeklinde görüneceğini belirtir.

Parametre olarak aldığı sembolik sabitler:

GL_POINTS

Verilen noktaları nokta olarak çizer.

GL_LINES

Verilen noktaları doğrularla birleştirir.

GL_POLYGON

Verilen noktaları doğrularla birleştirir ve oluşan şeklin içini renklendirir.

GL_QUADS

Verilen dört noktadan içi boyanmış dörtgen oluşturur.

GL_TRIANGLES

Verilen üç noktadan içi boyanmış üçgen oluşturur.

GL_TRIANGLE_STRIP

Şu noktalar glBegin ve glEnd arasında çizdirilmiş olsun: p0, p1, p2, p3, p4, p5. p0, p1 ve p2'den bir üçgen oluşturulur ve sonraki her nokta için önceki iki nokta birleştirilerek bir üçgen daha oluşturulur. Yani p3 ile p1 ve p2 birleştirilir. Daha sonra p4 ile p3 ve p2, ... vs.

GL_QUAD_STRIP

GL_TRIANGLE_STRIP gibi çalışır, ama bu sefer verilen iki noktayı önceki iki nokta ile birleştirerek bir dörtgen oluşturur.

GL_TRIANGLE_FAN

p0, p1, p2, p3, p4, p5 verilmiş olsun. p0, p1 ve p2 üçgeni çizilir. Daha sonra p4 için p0 ve p3 birleştirilerek yeni üçgen elde edilir. p5 ile p4 ve p0 birleştirilir, ...vs. Böylece şekil yelpaze gibi olur.

void glEnd ()

glBegin() ile başlayan çizim işleminin bittiğini belirtir. Çizdirilen şekil ekrana yazılmak üzere saklanır; glFlush() ile ekrana yazılır.

void glFlush ()

Eğer çizilenler tamponlanmışsa, tampon bellekteki tüm şekillerin ekrana basılmasını sağlar.

Sadece yukarıdaki işlevleri kullanarak basit geometrik şekillerin iki boyutta çizimi mümkündür. Yukarıdaki **glilk.c** adlı program bu işlevlerden yararlanarak yazdığımız ilk OpenGL uygulamamızdır.

void glColor3f (GLclampf *kırmızı*, GLclampf *yeşil*, GLclampf *mavi*, GLclampf *donukluk*)

glClear() işleviyle temizlenen ekran renk tamponunun ne renk alacağını belirler. Her parametre 0 ya da 1 değerini alır. Örneğin **donukluk** değeri 0 ise şeffaflık, 1 ise donukluk elde edilir. Öntanımlı değerlerin tümü 0'dır.

void glColor3s (short *kırmızı*, short *yeşil*, short *mavi*)

Çizilecek şeklin rengini belirler. Öntanımlı değerlerin tümü 0'dır.

void glutInitWindowSize (int *genişlik*, int *yükseklik*)

Oluşturulan pencerenin boyutlarını belirler. Pencere boyutlarının öntanımlı değeri (300, 300)'dır.

void glutInitWindowPosition (int *x*, int *y*)

Pencerenin konumlanacağı yeri belirtir. Pencere yerinin öntanımlı değeri (-1, -1)'dir, böylece yerini pencere yöneticisi belirler.

void glLineWidth (float *genişlik*)

Çizginin kalınlığını belirtir. Bu işlevle değiştirilmedikçe kalınlığın öntanımlı değeri 1.0'dır.

```
void glLineStipple (int grpan,  
                    short pattern)
```

işlev

Çizginin nokta nokta ya da düz çizgi şeklinde görünmesini ayarlar. Eğer **pattern**deki bit 0 ise bu bite karşı gelen benek ekrana basılmaz, eğer 1 ise ekrana basılır; Böylece kesik kesik çizgi çizilebilir.

Örnek:

```
glLineStipple(3, 0xcccc); /* 0xCCCC = 1100110011001100 */
```

Bu örnekte ikilik **pattern** ile genişletilmiştir. Bu işlevle, ekranındaki çizgiler 6 beneklik gruptara ayrılmak ve bir grup ekrana basılacak, bir grup basılmayacaktır.

```
void glEnable ()
```

işlev

```
void glDisable ()
```

işlev

Performans artışı sağlamak için OpenGL'deki kesiklilik, ışıklandırma, kaplama gibi özellikler **glDisable()** ile kapatılabilir. Böylece bu özellikler şeclin ekranada oluşturulması sırasında gözardı edilecek şekilde daha hızlı ortaya çıkacaktır.

glEnable() özelliğin kullanılmasını sağlarken **glDisable()** kullanılmaz hale getirir. Örneğin:

```
glEnable(GL_LINE_STIPPLE); // kesikli çizgi çizebilmek için  
glEnable(GL_SMOOTH); // renk geçişlerini yumusatmak için
```

```
void glRecti (int x1,  
             int y1,  
             int x2,  
             int y2)
```

işlev

İşlev, bir köşesi ilk iki parametresi ile çapraz köşesi ise son iki parametresi ile belirtilen bir dörtgenin çizilmesini sağlar.

İşlevin **glRects**, **glRectf**, **glRectd** türevleri de vardır. Tek farkları parametrelerinin sırasıyla **short**, **float**, ve **double** olmasıdır.

Dönüşümler

```
void glRotate (double aç
               double x,
               double y,
               double z)
```

İşlev

Şekil, **aç** derece kadar koordinatları **x**, **y**, **z** ile belirtilen noktanın etrafında döndürülür.

```
void glTranslated (double x,
                   double y,
                   double z)
```

İşlev

Koordinatları **x**, **y**, **z** ile belirtilen noktaya koordinat sistemini öteler.

```
void gluOrtho2D (double sol,
                  double sağ,
                  double alt,
                  double üst)
```

İşlev

İki boyutlu görüş peneresinin (clipping window) büyüğünü belirler.

```
void glLoadIdentity ()
```

İşlev

Yapılmış tüm dönüşümlerin geri alınmasını sağlar.

```
void glScaled (double x,
               double y,
               double z)
```

İşlev

Bu dönüşüm sayesinde ölçekte yapılır. Eğer girilen değerler 1'den küçükse nesneler küçütlür, 1'den büyükse nesneler büyütülür. Bu işlevin float parametreler alan glScalef isimli türevi de mevcuttur.

Olay Tanımlama İşlevleri

```
void glutReshapeFunc (void (*işlev) (int genişlik,  
double yükseklik)
```

işlev

Eğer pencere yeniden boyutlandırılırsa bu işlevin parametresi olan işlev çağrılr ve parametre olarak yeni genişlik ve yükseklik değerleri atanır.

```
void glutIdleFunc (void (*işlev) (void))
```

işlev

Hiçbir olay oluşturulmadığında çalıştırılacak işlevi belirtir.

Artalanda Tamponlama

Her seferinde ekranındaki görüntünün tazelenmesi CRT (=Cathode Ray Tube) tarafından yapılır. Bu olaya programımız müdahale edemez. Bellekte OpenGL'in çizim için kullandığı ekran bölgesine nokta eklemek çizgi çizmek gibi işlemlerle değişiklik yaptıkça ekrana bu değişiklikler anında yansıtılır. Ama tüm değişikliklerin yapıldıktan sonra ekranın tazelenmesini sağlayacak bir yol da vardır.

Cizimi başka bir bellek bölgesinde yapıp sonra ekran bölgesine aktarabilirim (double buffering). Arka planda ekranın yeni görüntüsü işlevlerle hazırlanır ve bu sırada ön planda yani kullanıcının gördüğü pencerede bir değişiklik olmaz. Biz programımızda `glFlush()` yerine `glSwapBuffers()` işlevini kullanırsak arka planda hazırladığımız değişiklikler ön plana yansır.

Bu durumu "ya hep ya hiç" şeklinde işlenmesi gereken verilere benzetebiliriz. Arka planda şeitin tamamı çizildikten sonra ön plana aktarılır. Böylece ekran tazelenmesi sırasında ekranın titrememesi sağlanır.

Ama artalanda tamponlama yapabilmek için penceremiz oluşturulmadan şu şekilde ilklendirme yapılması gerekmektedir:

```
glutInitDisplayMode (GLUT_RGBA | GLUT_DOUBLE);
```

`GLUT_DOUBLE` sayesinde çift tamponlu bir penceremiz olur. Artık `glFlush()` yerine `glSwapBuffers()` kullanılarak artalanda tamponlama yapılabilir.

Klavye İşlevleri

```
void glutKeyboardFunc (void (*islev) (char tus
                                         int x,
                                         int y))
```

İşlev

Klavyeden bir tuşa basıldığında bu işlev çağrılır. *x* ve *y* farenin o andaki konumunu belirtir. *tus*se klavyede basılan tuşu belirtir.

```
void glutSpecialFunc (void (*islev) (int tus
                                         int x,
                                         int y))
```

İşlev

Klavyedeki "F tuslarını" yani işlev tuşları için bu işlev kullanılır. Örnek:

```
if(key == GLUT_KEY_F1){ printf("F1'e bastiniz.\n"); }
else if(key == GLUT_KEY_UP) { printf("Yukari ok tusuna bastiniz.\n"); }
```

```
int glutGetModifiers ()
```

İşlev

Herhangi bir tuşa basılmışken CTRL, ALT veya SHIFT tuşlarından birine basılıp basılmadığını bu işlev sayesinde öğrenebiliriz. Örneğin glutSpecialFunc işlevi tarafından çalıştırılan myFunction işlevi şöyle bir kod içerebilir:

```
void myFunction(void){
    int modifier;
    ...
    modifier = glutGetModifiers();
    if(modifier = GLUT_ACTIVE_SHIFT){ printf("SHIFT tusuna bastiniz."); }
    ...
}
```

İşlevden dönen değeri GLUT_ACTIVE_SHIFT, GLUT_ACTIVE_CTRL, GLUT_ACTIVE_ALT sabitleri ile bulabiliriz.

Fare İşlevleri

```
void glutMouseFunc (void (*işlev) (int tuş,  
                           int durum,  
                           int x,  
                           int y))
```

işlev

Bu işlev farenin herhangi bir tuşuna basıldı veya bırakıldığı zaman çalışır. *x* ve *y* farenin o andaki konumunu belirtir. *tuş* GLUT_LEFT_BUTTON, GLUT_RIGHT_BUTTON, GLUT_MIDDLE_BUTTON olarak tanımlanan sırasıyla sol, sağ ve orta fare tuşlarını belirtir. *durum* ise tuşun ne durumda olduğunu söyler. GLUT_UP, GLUT_DOWN sabitleriyle tanımlanır.

```
void glutMouseFunc (void (*işlev) (int x,  
                           int y))
```

işlev

Tuş basılı olarak fare hareket ettiğinde çalışan işlevdir. Farenin o anki konumu *x* ve *y* parametrelerine atanır.

```
void glutPassiveMotionFunc (void (*işlev) (int x,  
                           int y))
```

işlev

Herhangi bir tuşa basılmaksızın farenin hareket edişi sırasında çağrılan işlevdir. Farenin o anki konumu *x* ve *y* parametrelerine atanır.

```
void glutEntryFunc (void (*işlev) (int durum))
```

işlev

Fare pencere sınırlarına girince veya sınırlarından çıkışınca çağrılan işlevdir. *durum* parametresinin aldığı değere göre GLUT_ENTERED ile farenin pencereye girdiği, GLUT_LEAVE ile farenin pencereden çıktıği anlaşılır.

2D ve 3D nokta ve cisim tanımı

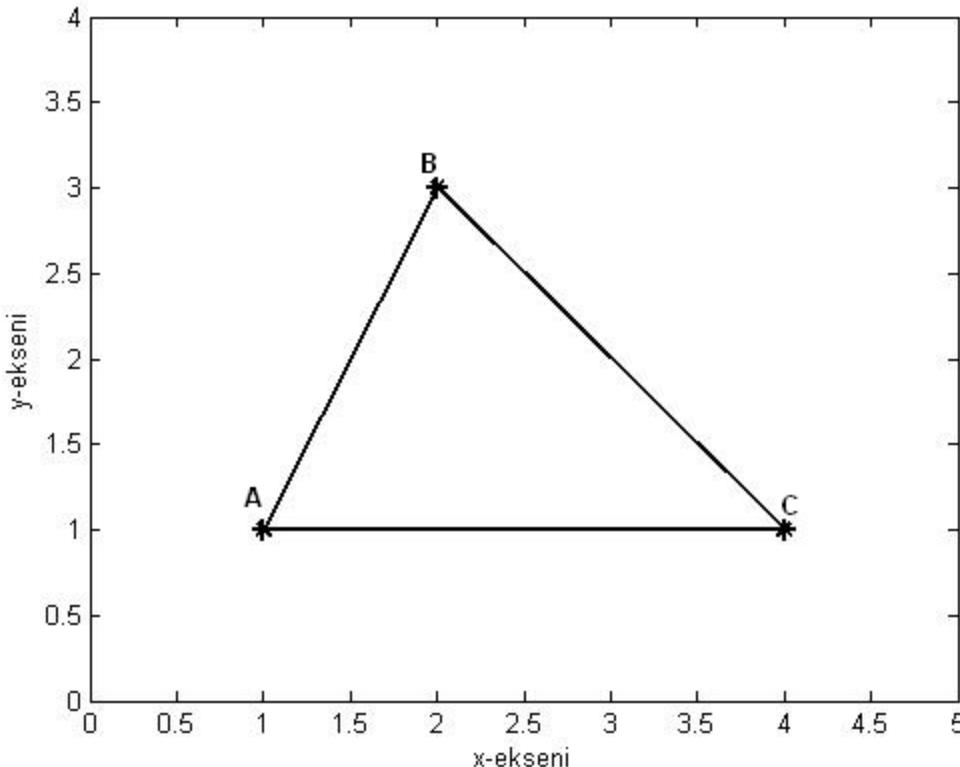
İki boyutlu uzayda bir noktayı şu şekilde tanımlarız:

$$\begin{bmatrix} x & y \end{bmatrix}$$

Bu gösterim bir $A(x,y)$ noktası için, satır ya da sütun şeklinde tanımlanabilir. Bu matris gösterimi ilerideki işlemlerde temel matris olarak alınacaktır. Birden fazla nokta olması durumunda bu noktalar, bu matrise aynı formda alt alta eklenecektir. Örneğin, $A(x_1,y_1)$ ve $B(x_2,y_2)$ olmak üzere iki nokta ile tanımlama yapıldığında sözkonusu matris şu şekilde olacaktır:

$$X_{nokta} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix}$$

İki boyutlu uzayda bir cismi tanımlamak için nokta ve kenar matrislerinin tanımlanması gereklidir. Ömek olarak bir ABC üçgenini tanımlayalım, bu üçgenin noktaları A(1,1) B(2,3) C(4,1) olarak verildiğinde üçgeni şu şekilde çizebiliriz:



Şekil 4.1.-ABC üçgeni

Burada X cisim matrisi ve kenar matrisi şu şekilde tanımlanır:

$$X_{nokta} = \begin{pmatrix} 1,1 \\ 2,3 \\ 4,1 \end{pmatrix}, X_{kenar} = \begin{pmatrix} 1,2 \\ 2,3 \\ 3,1 \end{pmatrix}$$

$$X_{nokta} = \begin{pmatrix} 1,1 \\ 2,3 \\ 4,1 \end{pmatrix}, X_{kenar} = \begin{pmatrix} 1,2 \\ 2,3 \\ 3,1 \end{pmatrix}$$

Burada ilk verilen X_{nokta} matrisi içinde, cismi oluşturan noktalar yer almaktadır. Bunları satır numarası ile birinci nokta, ikinci nokta şeklinde numaralandırılmış olarak düşününeceğiz. İkinci olarak X_{kenar} matrisinde ise sırayla cismi oluşturan kenarlar belirtilmektedir. Bu kenar matrisinde verilen sayılar, nokta matrisinde numaralandırılmış olan noktalardır. Yani ilk satırda verilen (1,2)'yi yorumlarsak; ilk kenarın birinci noktadan ikinci noktaya olduğunu göstermektedir. Diğer kenar değerleri de benzer şekilde yorumlanır.

2D Temel işlemler

- Öteleme
- Ölçekleme
- Yansıtma
- Döndürme
- Herhangi bir nokta etrafında döndürme
- Herhangi bir doğruya göre yansıtma

Homojen Koordinat sistemindeki işlemler:

$$x^* = ax + cy + m$$

$$y^* = bx + dy + n$$

$$[T] = \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ m & n & 1 \end{bmatrix} \quad \text{Dönüşüm Matrisi}$$

Öteleme :

$$\begin{bmatrix} x^* & y^* & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ m & n & 1 \end{bmatrix} = \begin{bmatrix} x+m & y+n & 1 \end{bmatrix}$$

Ölçekleme :

$$[x^* \quad y^* \quad 1] = [x \quad y \quad 1] \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [2x \quad 2y \quad 1]$$

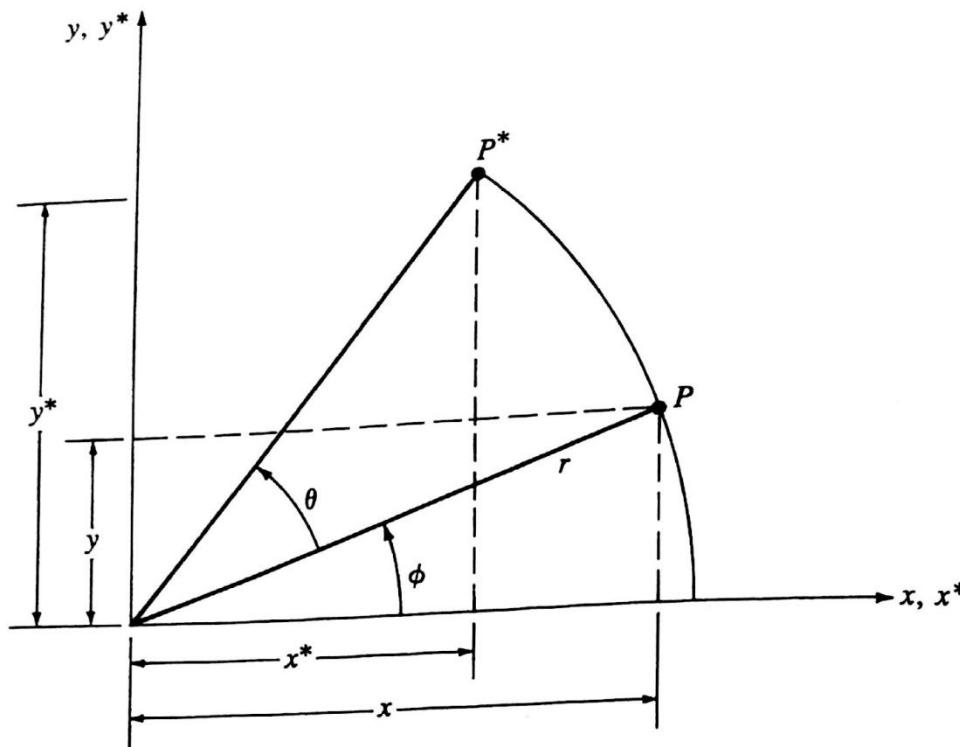
Yansıtma :

$$[x^* \quad y^* \quad 1] = [x \quad y \quad 1] \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [-x \quad y \quad 1] \quad \text{y-eks göre}$$

$$[x^* \quad y^* \quad 1] = [x \quad y \quad 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [x \quad -y \quad 1] \quad \text{x-eks göre}$$

$$[x^* \quad y^* \quad 1] = [x \quad y \quad 1] \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [-x \quad -y \quad 1] \quad y = -x \text{ göre}$$

Döndürme :



$$P = [x \ y] = [r\cos\theta \ r\sin\theta]$$

$$P^* = [x^* \ y^*] = [r\cos(\theta+\phi) \ r\sin(\theta+\phi)]$$

$$P^* = [x^* \ y^*] = [r(\cos\theta\cos\phi - \sin\theta\sin\phi) \ r(\cos\theta\sin\phi + \sin\theta\cos\phi)]$$

$$P^* = [x^* \ y^*] = [x\cos\phi - y\sin\phi \ x\sin\phi + y\cos\phi]$$

$$x^* = x\cos\phi - y\sin\phi$$

$$y^* = x\sin\phi + y\cos\phi$$

$$[X^*] = [X][R]$$

$$\begin{bmatrix} x^* & y^* & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Herhangi bir nokta etrafında döndürme :

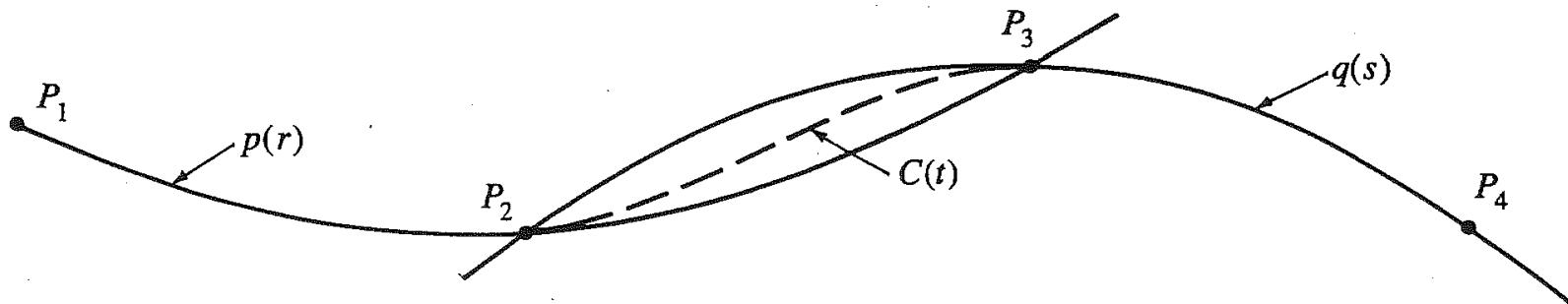
$$\begin{bmatrix} x^* & y^* & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -m & -n & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ m & n & 1 \end{bmatrix}$$

Herhangi bir doğruya göre yansıtma :

$$\begin{bmatrix} x^* & y^* & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -m & -n & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ m & n & 1 \end{bmatrix}$$

2D Uzay Eğrileri

1- Parabol uydurma:



$$C(t) = (1 - t)p(r) + tq(s)$$

$$p(r) = [\begin{matrix} r^2 & r & 1 \end{matrix}] [\begin{matrix} B \end{matrix}]$$

$$q(s) = [\begin{matrix} s^2 & s & 1 \end{matrix}] [\begin{matrix} D \end{matrix}] \quad r = k_1 t + k_2 \quad s = k_3 t + k_4$$

$$\begin{array}{lll}
 p(0) = P_1 & p(1/2) = P_2 & p(1) = P_3 \\
 q(0) = P_2 & q(1/2) = P_3 & q(1) = P_4 \\
 C(0) = P_2 & & C(1) = P_3
 \end{array}$$

$$r = k_1 t + k_2 \quad s = k_3 t + k_4$$

$$\begin{array}{llll}
 @ P_2 : & r = 1/2, t = 0 & \Rightarrow & k_2 = 1/2 \\
 @ P_3 : & r = 1, t = 1 & \Rightarrow & k_1 + k_2 = 1 \quad \Rightarrow \quad k_1 = 1/2 \\
 @ P_2 : & s = 0, t = 0 & \Rightarrow & k_4 = 0 \\
 @ P_3 : & s = 1/2, t = 1 & \Rightarrow & k_3 = 1/2
 \end{array}$$

$$r(t) = \frac{1}{2}(1+t) \quad s(t) = \frac{1}{2}t$$

$$p(0) = P_1 = [\begin{smallmatrix} 0 & 0 & 1 \end{smallmatrix}] [\begin{smallmatrix} B \end{smallmatrix}]$$

$$p(1/2) = P_2 = [\begin{smallmatrix} 1/4 & 1/2 & 1 \end{smallmatrix}] [\begin{smallmatrix} B \end{smallmatrix}]$$

$$p(1) = P_3 = [\begin{smallmatrix} 1 & 1 & 1 \end{smallmatrix}] [\begin{smallmatrix} B \end{smallmatrix}]$$

$$\begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1/4 & 1/2 & 1 \\ 1 & 1 & 1 \end{bmatrix} [B] = [M][B]$$

$$[B] = [M]^{-1} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix} = \begin{bmatrix} 2 & -4 & 2 \\ -3 & 4 & -1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

$$q(0) = P_2 = [0 \ 0 \ 1][D]$$

$$q(1/2) = P_3 = [1/4 \ 1/2 \ 1][D]$$

$$q(1) = P_4 = [1 \ 1 \ 1][D]$$

$$[D] = [M]^{-1} \begin{bmatrix} P_2 \\ P_3 \\ P_4 \end{bmatrix} = \begin{bmatrix} 2 & -4 & 2 \\ -3 & 4 & -1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_2 \\ P_3 \\ P_4 \end{bmatrix}$$

$$C(t) = (1-t)[r^2 \ r \ 1][B] + t[s^2 \ s \ 1][D]$$

$$C(t) = [-\frac{1}{4}(t^3 + t^2 - t - 1) \quad -\frac{1}{2}(t^2 - 1) \quad 1 - t][B] + \left[\begin{array}{ccc} \frac{t^3}{4} & \frac{t^2}{2} & t \end{array} \right][D]$$

$$C(t) = \begin{bmatrix} -\frac{t^3}{2} + t^2 - \frac{t}{2} & t^3 - t^2 - t + 1 & -\frac{t^3}{2} + \frac{t}{2} \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

$$+ \begin{bmatrix} \frac{t^3}{2} - \frac{3}{2}t^2 + t & -t^3 + 2t^2 & \frac{t^3}{2} - \frac{t^2}{2} \end{bmatrix} \begin{bmatrix} P_2 \\ P_3 \\ P_4 \end{bmatrix}$$

$$C(t) = \begin{bmatrix} -\frac{t^3}{2} + t^2 - \frac{t}{2} & t^3 - t^2 - t + 1 & -\frac{t^3}{2} + \frac{t}{2} & 0 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$$

$$+ \begin{bmatrix} 0 & \frac{t^3}{2} - \frac{3}{2}t^2 + t & -t^3 + 2t^2 & \frac{t^3}{2} - \frac{t^2}{2} \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$$

$$C(t) = [t^3 \quad t^2 \quad t \quad 1] [A] [G] = [T] [A] [G]$$

$$[A] = \left(\frac{1}{2} \right) \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \quad [G]^T = [P_1 \quad P_2 \quad P_3 \quad P_4]$$

$$C(t) = [t^3 \quad t^2 \quad t \quad 1] \left(\frac{1}{2} \right) \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$$

$$p(r) = [r^2 \quad r \quad 1] \begin{bmatrix} 2 & -4 & 2 \\ -3 & 4 & -1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

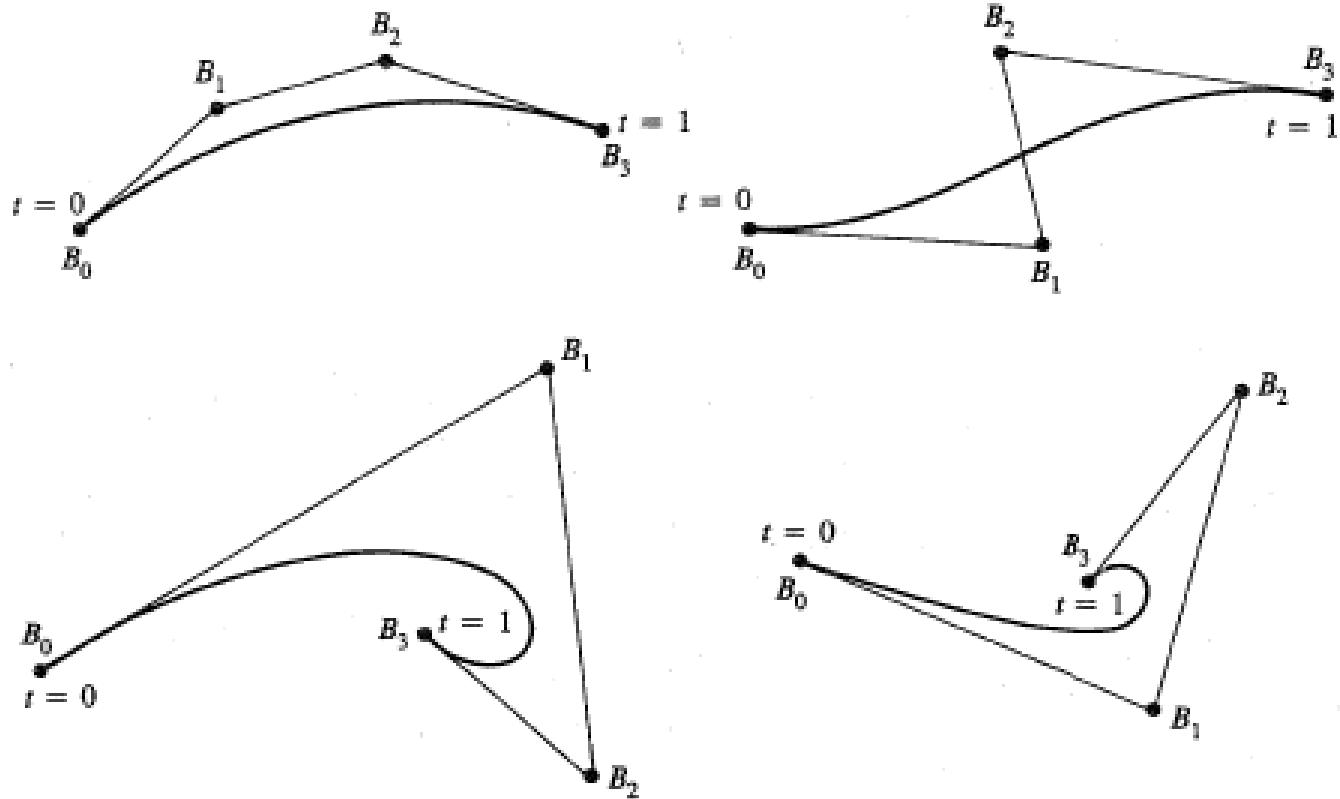
$$q(s) = [s^2 \quad s \quad 1] \begin{bmatrix} 2 & -4 & 2 \\ -3 & 4 & -1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_2 \\ P_3 \\ P_4 \end{bmatrix}$$

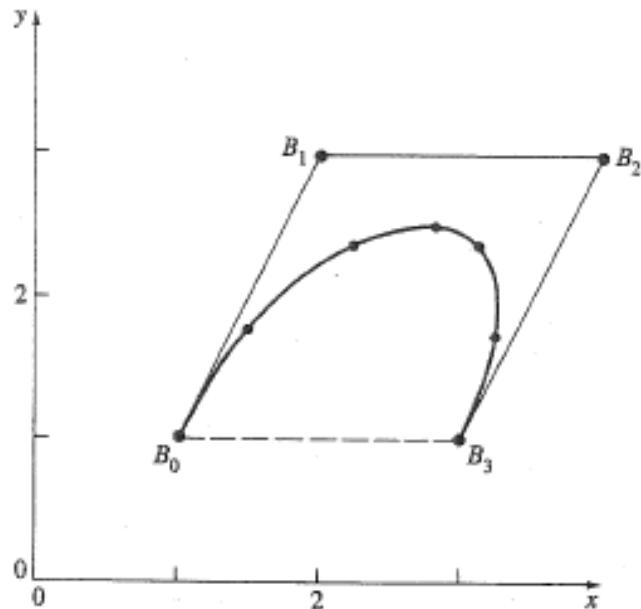
Bezier Eğrisi :

$$P(t) = \sum_{i=0}^n B_i J_{n,i}(t) \quad 0 \leq t \leq 1$$

$$J_{n,i}(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

$$\binom{n}{i} = \frac{n!}{i! (n-i)!}$$





$$J_{3,0}(t) = (1)t^0(1-t)^3 = (1-t)^3$$

$$J_{3,1}(t) = 3t(1-t)^2$$

$$J_{3,2}(t) = 3t^2(1-t)$$

$$J_{3,3}(t) = t^3$$

$$\begin{aligned}P(t) &= B_0 J_{3,0} + B_1 J_{3,1} + B_2 J_{3,2} + B_3 J_{3,3} \\&= (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t) P_2 + t^3 P_3\end{aligned}$$

$$P(0) = B_0 = [1 \quad 1]$$

$$P(0.15) = 0.614B_0 + 0.325B_1 + 0.058B_2 + 0.003B_3 = [1.5 \quad 1.765]$$

$$P(0.35) = 0.275B_0 + 0.444B_1 + 0.239B_2 + 0.042B_3 = [2.248 \quad 2.367]$$

$$P(0.5) = 0.125B_0 + 0.375B_1 + 0.375B_2 + 0.125B_3 = [2.75 \quad 2.5]$$

$$P(0.65) = 0.042B_0 + 0.239B_1 + 0.444B_2 + 0.275B_3 = [3.122 \quad 2.367]$$

$$P(0.85) = 0.003B_0 + 0.058B_1 + 0.325B_2 + 0.614B_3 = [3.248 \quad 1.765]$$

$$P(1) = B_3 = [3 \quad 1]$$

$$P(t) = \begin{bmatrix} (1-t)^3 & 3t(1-t)^2 & 3t^2(1-t) & t^3 \end{bmatrix} \begin{bmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{bmatrix}$$

$$P(t) = [T][N][G] = [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{bmatrix}$$

$$P(t) = [t^4 \quad t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} 1 & -4 & 6 & -4 & 1 \\ -4 & 12 & -12 & 4 & 0 \\ 6 & -12 & 6 & 0 & 0 \\ -4 & 4 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \\ B_4 \end{bmatrix}$$

Cubic Spline Eğrisi :

$$P(t) = \sum_{i=1}^4 B_i t^{i-1} \quad t_1 \leq t \leq t_2$$



$$P(t) = B_1 + B_2 t + B_3 t^2 + B_4 t^3 \quad t_1 \leq t \leq t_2$$

$$P'(t) = B_2 + 2B_3 t + 3B_4 t^2 \quad t_1 \leq t \leq t_2$$

$$P(0) = P_1$$

$$P(t_2) = P_2$$

$$P(0) = B_1 = P_1$$

$$P'(0) = P'_1$$

$$P'(t_2) = P'_2$$

$$P'(0) = \sum_{i=1}^4 (i-1) t^{i-2} B_i \Big|_{t=0} = B_2 = P'_1$$

$$P(t_2) = \sum_{i=1}^4 B_i t^{i-1} \Big|_{t=t_2} = B_1 + B_2 t_2 + B_3 t_2^2 + B_4 t_2^3 \quad = P2$$

$$P'(t_2) = \sum_{i=1}^4 (i-1) t^{i-2} B_i \Big|_{t=t_2} = B_2 + 2B_3 t_2 + 3B_4 t_2^2 \quad = P2'$$

$$B_3 = \frac{3(P_2 - P_1)}{t_2^2} - \frac{2P'_1}{t_2} - \frac{P'_2}{t_2}$$

$$B_4 = \frac{2(P_1 - P_2)}{t_2^3} + \frac{P'_1}{t_2^2} + \frac{P'_2}{t_2^2}$$

$$\begin{aligned} P(t) &= P_1 + P'_1 t + \left[\frac{3(P_2 - P_1)}{t_2^2} - \frac{2P'_1}{t_2} - \frac{P'_2}{t_2} \right] t^2 \\ &\quad + \left[\frac{2(P_1 - P_2)}{t_2^3} + \frac{P'_1}{t_2^2} + \frac{P'_2}{t_2^2} \right] t^3 \end{aligned}$$

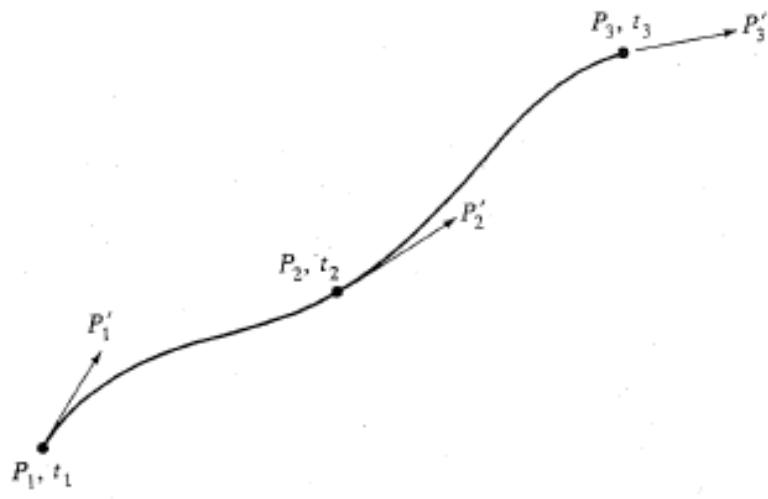
$$P''(t) = \sum_{i=1}^4 (i-1)(i-2) B_i t^{i-3} \quad t_1 \leq t \leq t_2$$

$$P'(t) = B_2 + 2B_3 t + 3B_4 t^2 \quad t_1 \leq t \leq t_2$$

$$P'' = 6B_4 t_2 + 2B_3$$

$$P'' = 2B_3$$

$$\begin{aligned} 6t_2 &\left[\frac{2(P_1 - P_2)}{t_2^3} + \frac{P'_1}{t_2^2} + \frac{P'_2}{t_2^2} \right] + 2 \left[\frac{3(P_2 - P_1)}{t_2^2} - \frac{2P'_1}{t_2} - \frac{P'_2}{t_2} \right] \\ &= 2 \left[\frac{3(P_3 - P_2)}{t_3^2} - \frac{2P'_2}{t_3} - \frac{P'_3}{t_3} \right] \end{aligned}$$



$$t_3 P'_1 + 2(t_3 + t_2)P'_2 + t_2 P'_3 = \frac{3}{t_2 t_3} [t_2^2(P_3 - P_2) + t_3^2(P_2 - P_1)]$$

$$t_4 P'_2 + 2(t_4 + t_3)P'_3 + t_3 P'_4 = \frac{3}{t_3 t_4} [t_3^2(P_4 - P_3) + t_4^2(P_3 - P_2)]$$

$$t_5 P'_3 + 2(t_5 + t_4)P'_4 + t_4 P'_5 = \frac{3}{t_4 t_5} [t_4^2(P_5 - P_4) + t_5^2(P_4 - P_3)]$$

$$\begin{bmatrix} 1 & 0 & & & & & \\ t_3 & 2(t_2 + t_3) & t_2 & 0 & & & \\ 0 & t_4 & 2(t_3 + t_4) & t_3 & 0 & & \\ 0 & 0 & t_5 & 2(t_4 + t_5) & t_4 & 0 & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ \vdots & \vdots & \vdots & 0 & t_n & 2(t_n + t_{n-1}) & t_{n-1} \\ \vdots & \vdots & \vdots & & \vdots & 0 & 1 \end{bmatrix} \times \begin{bmatrix} P'_1 \\ P'_2 \\ P'_3 \\ \vdots \\ P'_n \end{bmatrix} = \begin{bmatrix} P'_1 \\ \frac{3}{t_2 t_3} \{t_2^2(P_3 - P_2) + t_3^2(P_2 - P_1)\} \\ \frac{3}{t_3 t_4} \{t_3^2(P_4 - P_3) + t_4^2(P_3 - P_2)\} \\ \vdots \\ \vdots \\ \frac{3}{t_{n-1} t_n} \{t_{n-1}^2(P_n - P_{n-1}) + t_n^2(P_{n-1} - P_{n-2})\} \\ P'_n \end{bmatrix}$$

$$B_{1k} = P_k$$

$$B_{2k} = P'_k$$

$$B_{3k} = \frac{3(P_{k+1} - P_k)}{t_{k+1}^2} - \frac{2P'_k}{t_{k+1}} - \frac{P'_{k+1}}{t_{k+1}}$$

$$B_{4k} = \frac{2(P_k - P_{k+1})}{t_{k+1}^3} + \frac{P'_k}{t_{k+1}^2} + \frac{P'_{k+1}}{t_{k+1}^2}$$

$$\begin{aligned}[B] &= \begin{bmatrix} B_{1k} \\ B_{2k} \\ B_{3k} \\ B_{4k} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{3}{t_{k+1}^2} & -\frac{2}{t_{k+1}} & \frac{3}{t_{k+1}^2} & -\frac{1}{t_{k+1}} \\ \frac{2}{t_{k+1}^3} & \frac{1}{t_{k+1}^2} & -\frac{2}{t_{k+1}^3} & \frac{1}{t_{k+1}^2} \end{bmatrix} \begin{bmatrix} P_k \\ P'_k \\ P_{k+1} \\ P'_{k+1} \end{bmatrix} \end{aligned}$$

$$P_k(t) = [1 \quad t \quad t^2 \quad t^3] \begin{bmatrix} B_{1k} \\ B_{2k} \\ B_{3k} \\ B_{4k} \end{bmatrix} \quad 0 \leq t \leq t_{k+1}$$

$$P_k(t) = [1 \quad t \quad t^2 \quad t^3] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{3}{t_{k+1}^2} & -\frac{2}{t_{k+1}} & \frac{3}{t_{k+1}^2} & -\frac{1}{t_{k+1}} \\ \frac{2}{t_{k+1}^3} & \frac{1}{t_{k+1}^2} & -\frac{2}{t_{k+1}^3} & \frac{1}{t_{k+1}^2} \end{bmatrix} \begin{bmatrix} P_k \\ P'_k \\ P_{k+1} \\ P'_{k+1} \end{bmatrix} \quad \tau = (t/t_{k+1})$$

$$P_k(\tau) = [F_1(\tau) \quad F_2(\tau) \quad F_3(\tau) \quad F_4(\tau)] \begin{bmatrix} P_k \\ P_{k+1} \\ P'_k \\ P'_{k+1} \end{bmatrix} \quad \begin{array}{l} 0 \leq \tau \leq 1 \\ 1 \leq k \leq n-1 \end{array}$$

$$F_{1k}(\tau) = 2\tau^3 - 3\tau^2 + 1$$

$$F_{2k}(\tau) = -2\tau^3 + 3\tau^2$$

$$F_{3k}(\tau) = \tau(\tau^2 - 2\tau + 1)t_{k+1}$$

$$F_{4k}(\tau) = \tau(\tau^2 - \tau)t_{k+1}$$

Örnek: $P_1[0,0]$, $P_2[1,1]$, $P_3[2,-1]$, $P_4[3,0]$, $P'_1[1,1]$, $P'_4[1,1]$

$$t_2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} = \sqrt{(1)^2 + (1)^2} = \sqrt{2}$$

$$t_3 = \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2} = \sqrt{(1)^2 + (-2)^2} = \sqrt{5}$$

$$t_4 = \sqrt{(x_4 - x_3)^2 + (y_4 - y_3)^2} = \sqrt{(1)^2 + (1)^2} = \sqrt{2}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ t_3 & 2(t_2 + t_3) & t_2 & 0 \\ 0 & t_4 & 2(t_3 + t_4) & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P'_1 \\ P'_2 \\ P'_3 \\ P'_4 \end{bmatrix} = \begin{bmatrix} P'_1 \\ \frac{3}{t_2 t_3} \{ t_2^2 (P_3 - P_2) + t_3^2 (P_2 - P_1) \} \\ \frac{3}{t_3 t_4} \{ t_3^2 (P_4 - P_3) + t_4^2 (P_3 - P_2) \} \\ P'_4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 2.236 & 7.300 & 1.414 & 0 \\ 0 & 1.414 & 7.300 & 2.236 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P'_1 \\ P'_2 \\ P'_3 \\ P'_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 6.641 & 0.949 \\ 6.641 & 0.949 \\ 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} P'_1 \\ P'_2 \\ P'_3 \\ P'_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -0.318 & 0.142 & -0.028 & 0.062 \\ 0.062 & -0.028 & 0.142 & -0.318 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 6.641 & 0.949 \\ 6.641 & 0.949 \\ 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 \\ 0.505 & -0.148 \\ 0.505 & -0.148 \\ 1 & 1 \end{bmatrix}$$

$$F_1(1/3) = 2(1/3)^3 - 3(1/3)^2 + 1 = \frac{20}{27} = 0.741$$

$$F_2(1/3) = -2(1/3)^3 + 3(1/3)^2 = \frac{7}{27} = 0.259$$

$$F_3(1/3) = (1/3)[(1/3)^2 - 2(1/3) + 1]\sqrt{2} = \frac{4\sqrt{2}}{27} = 0.210$$

$$F_4(1/3) = (1/3)[(1/3)^2 - 1/3]\sqrt{2} = -\frac{2\sqrt{2}}{27} = -0.105$$

$$F_1(2/3) = 2(2/3)^3 - 3(2/3)^2 + 1 = \frac{7}{27} = 0.259$$

$$F_2(2/3) = -2(2/3)^3 + 3(2/3)^2 = \frac{20}{27} = 0.741$$

$$F_3(2/3) = (2/3)[(2/3)^2 - 2(2/3) + 1]\sqrt{2} = \frac{2\sqrt{2}}{27} = 0.105$$

$$F_4(2/3) = (2/3)[(2/3)^2 - 1/3]\sqrt{2} = -\frac{4\sqrt{2}}{27} = -0.210$$

$$P(1/3) = [0.741 \quad 0.259 \quad 0.210 \quad -0.105] \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0.505 & -0.148 \end{bmatrix}$$

$$= [0.416 \quad 0.484]$$

$$P(2/3) = [0.259 \quad 0.741 \quad 0.105 \quad -0.210] \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0.505 & -0.148 \end{bmatrix}$$

$$= [0.740 \quad 0.876]$$

| Segment | τ | $P_x(\tau)$ | $P_y(\tau)$ |
|---------|--------|-------------|-------------|
| 1 | 1/3 | 0.416 | 0.484 |
| | 2/3 | 0.740 | 0.876 |
| 2 | 1/3 | 1.343 | 0.457 |
| | 2/3 | 1.657 | -0.457 |
| 3 | 1/3 | 2.260 | -0.876 |
| | 2/3 | 2.584 | -0.484 |

Ara noktalardaki türev değerlerinin hesabı

Standart Cubic Spline:

$$\begin{bmatrix} 1 & 0 & \dots & 0 \\ t_3 & 2(t_2 + t_3) & t_2 & \vdots \\ 0 & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 1 \end{bmatrix} * \begin{bmatrix} P'_1 \\ P'_2 \\ \vdots \\ P'_n \end{bmatrix} = C \begin{bmatrix} P'_1 \\ \frac{3}{t_2 t_3} (t_2^2 (P_3 - P_2) + t_3^2 (P_2 - P_1)) \\ \vdots \\ P'_n \end{bmatrix}$$

Normalize edilmiş cubic spline:

$$\begin{bmatrix} 1 & 0 & \cdot & \cdot & \cdot & \cdot \\ 1 & 4 & 1 & 0 & \cdot & \cdot \\ 0 & 1 & 4 & 1 & 0 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 0 & 1 & 4 & 1 \\ \cdot & \cdot & \cdot & \cdot & 0 & 1 \end{bmatrix} \begin{bmatrix} P'_1 \\ P'_2 \\ \cdot \\ \cdot \\ \cdot \\ P'_n \end{bmatrix} = \begin{bmatrix} P'_1 \\ 3 \{(P_3 - P_4) + (P_2 - P_1)\} \\ 3 \{(P_4 - P_3) + (P_3 - P_2)\} \\ \cdot \\ \cdot \\ 3 \{(P_n - P_{n-1}) + (P_{n-1} - P_{n-2})\} \\ P'_n \end{bmatrix}$$

Üç noktaları selbest bırakılmış cubik spline:

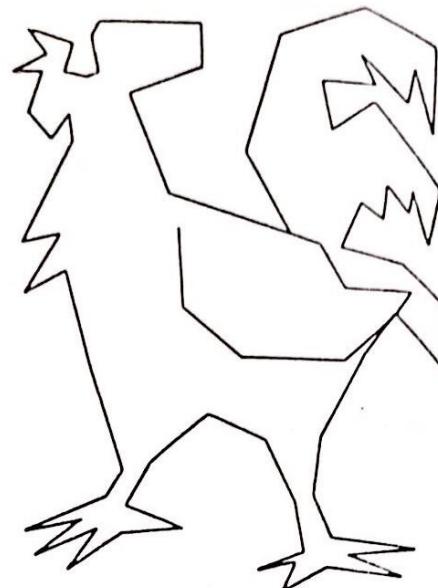
$$\begin{bmatrix} 1 & 1/2 & \dots & 0 \\ t_3 & 2(t_2 + t_3) & t_2 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 2 & 4 \end{bmatrix} * \begin{bmatrix} P'_1 \\ P'_2 \\ \vdots \\ P'_n \end{bmatrix} = \begin{bmatrix} \frac{3}{2t_2}(P_2 - P_1) \\ \frac{3}{t_2 t_3}(t_2^2(P_3 - P_2) + t_3^2(P_2 - P_1)) \\ \vdots \\ \frac{6}{t_n}(P_n - P_{n-1}) \end{bmatrix}$$

Normalize edilmiş ve üç noktaları selbest bırakılmış cubic spline :

$$\begin{bmatrix} 1 & 1/2 & 0 & 0 \\ 1 & 4 & 1 & \vdots \\ \vdots & 1 & 4 & 1 \\ 0 & 0 & 2 & 4 \end{bmatrix} * \begin{bmatrix} P'_1 \\ P'_2 \\ \vdots \\ P'_n \end{bmatrix} = \begin{bmatrix} \frac{3}{2}(P_2 - P_1) \\ 3((P_3 - P_2) + (P_2 - P_1)) \\ \vdots \\ 6(P_n - P_{n-1}) \end{bmatrix}$$



(a)



(b)



(c)



(d)

- (a)-Noktasal
- (b)-Doğrusal
- (c)-Normalize edilmiş
- (d)-Standart

B-Spline Eğrisi :

$$P(t) = \sum_{i=1}^{n+1} B_i N_{i,k}(t) \quad t_{\min} \leq t < t_{\max}, \quad 2 \leq k \leq n+1$$

$$N_{i,1}(t) = \begin{cases} 1 & \text{if } x_i \leq t < x_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,k}(t) = \frac{(t - x_i) N_{i,k-1}(t)}{x_{i+k-1} - x_i} + \frac{(x_{i+k} - t) N_{i+1,k-1}(t)}{x_{i+k} - x_{i+1}}$$

$$x_i = 0 \quad 1 \leq i \leq k$$

$$x_i = i - k \quad k + 1 \leq i \leq n + 1$$

$$x_i = n - k + 2 \quad n + 2 \leq i \leq n + k + 1$$

n=4

$$k = 2 \quad [0 \ 0 \ 1 \ 2 \ 3 \ 4 \ 4]$$

$$k = 3 \quad [0 \ 0 \ 0 \ 1 \ 2 \ 3 \ 3 \ 3]$$

$$k = 4 \quad [0 \ 0 \ 0 \ 0 \ 1 \ 2 \ 2 \ 2 \ 2]$$

$$[X] = [\begin{array}{ccccccc} 0 & 0 & 0 & 1 & 2 & 2 & 2 \end{array}] \quad k=3, n=3$$

$$0 \leq t < 1$$

$$N_{3,1}(t) = 1; \quad N_{i,1}(t) = 0, \quad i \neq 3$$

$$N_{2,2}(t) = 1 - t; \quad N_{3,2}(t) = t; \quad N_{i,2}(t) = 0, \quad i \neq 2, 3$$

$$\underline{N_{1,3}}(t) = (1-t)^2; \quad N_{2,3}(t) = t(1-t) + \frac{(2-t)}{2}t;$$

$$N_{3,3}(t) = \frac{t^2}{2}; \quad N_{i,3}(t) = 0, \quad i \neq 1, 2, 3$$

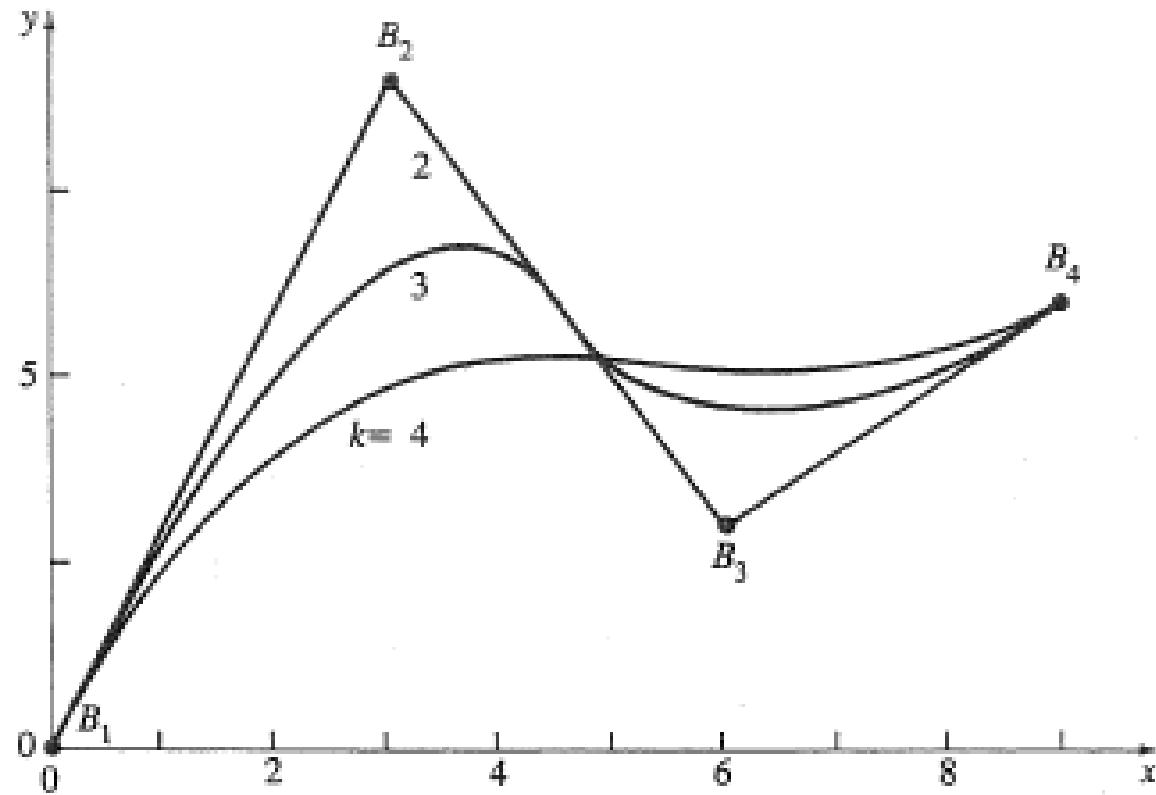
$$1 \leq t < 2$$

$$N_{4,1}(t) = 1; \quad N_{i,1}(t) = 0, \quad i \neq 4$$

$$N_{3,2}(t) = (2-t); \quad N_{4,2}(t) = (t-1); \quad N_{i,2}(t) = 0, \quad i \neq 3, 4$$

$$\underline{N_{2,3}}(t) = \frac{(2-t)^2}{2}; \quad \underline{N_{3,3}}(t) = \frac{t(2-t)}{2} + (2-t)(t-1);$$

$$\underline{N_{4,3}}(t) = (t-1)^2; \quad N_{i,3}(t) = 0, \quad i \neq 2, 3, 4$$



$$B_1 [\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}], B_2 [\begin{smallmatrix} 2 & 3 \\ 2 & 3 \end{smallmatrix}], B_3 [\begin{smallmatrix} 4 & 3 \\ 3 & 3 \end{smallmatrix}], B_4 [\begin{smallmatrix} 3 & 1 \\ 1 & 1 \end{smallmatrix}].$$

$$[\begin{smallmatrix} 0 & 0 & 1 & 2 & 3 & 3 \end{smallmatrix}] \quad k=2, n=3$$

$$0 \leq t < 1$$

$$N_{2,1}(t) = 1; \quad N_{i,1}(t) = 0, \quad i \neq 2$$

$$N_{1,2}(t) = 1 - t; \quad N_{2,2}(t) = t; \quad N_{i,2}(t) = 0, \quad i \neq 1, 2$$

$$1 \leq t < 2$$

$$N_{3,1}(t) = 1; \quad N_{i,1}(t) = 0, \quad i \neq 3$$

$$N_{2,2}(t) = 2 - t; \quad N_{3,2}(t) = (t - 1); \quad N_{i,2}(t) = 0, \quad i \neq 2, 3$$

$$2 \leq t < 3$$

$$N_{4,1}(t) = 1; \quad N_{i,1}(t) = 0, \quad i \neq 4$$

$$N_{3,2}(t) = (3 - t); \quad N_{4,2}(t) = (t - 2); \quad N_{i,2}(t) = 0, \quad i \neq 3, 4$$

$$P(t) = B_1 N_{1,2}(t) + B_2 N_{2,2}(t) + B_3 N_{3,2}(t) + B_4 N_{4,2}(t)$$

$$P(t) = (1 - t)B_1 + tB_2 = B_1 + (B_2 - B_1)t \quad 0 \leq t < 1$$

$$P(t) = (2 - t)B_2 + (t - 1)B_3 = B_2 + (B_3 - B_2)t \quad 1 \leq t < 2$$

$$P(t) = (3 - t)B_3 + (t - 2)B_4 = B_3 + (B_4 - B_3)t \quad 2 \leq t < 3$$

3D Temel işlemler

- Öteleme
- Ölçekleme
- Yansıtma
- Döndürme
- Herhangi bir doğru etrafında döndürme
- Herhangi bir düzleme göre yansıtma

Ölçekleme=

$$\begin{aligned}[X][T] &= [x \ y \ z \ 1] \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & e & 0 & 0 \\ 0 & 0 & j & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= [ax \ ey \ jz \ 1] = [x^* \ y^* \ z^* \ 1]\end{aligned}$$

Döndürme=

$$[T] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & \sin \theta & 0 \\ 0 & -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[T] = \begin{bmatrix} \cos \phi & 0 & -\sin \phi & 0 \\ 0 & 1 & 0 & 0 \\ \sin \phi & 0 & \cos \phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[T] = \begin{bmatrix} \cos \psi & \sin \psi & 0 & 0 \\ -\sin \psi & \cos \psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Yansıtma=

$$[T] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[T] = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[T] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Öteleme=

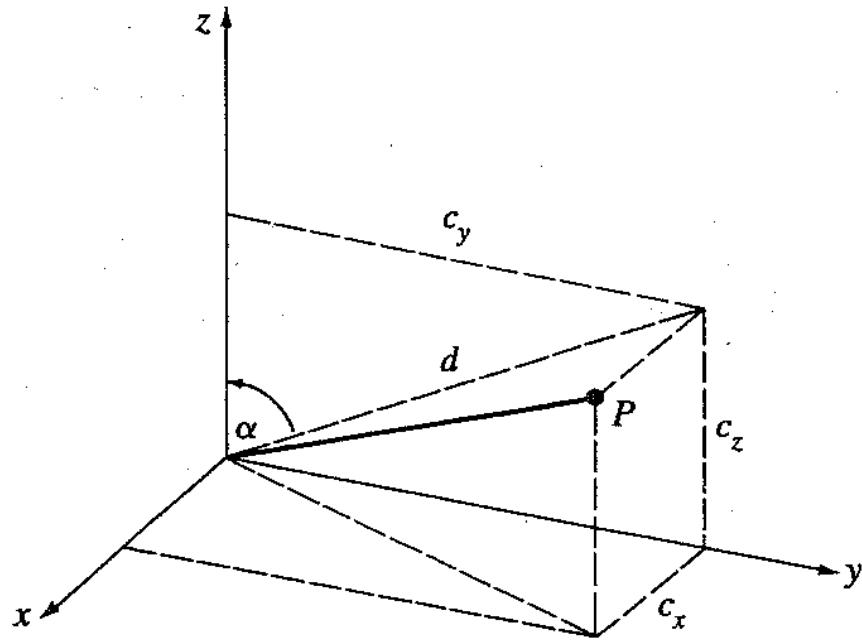
$$[T] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ l & m & n & 1 \end{bmatrix}$$

Herhangi bir doğru etrafında döndürme=

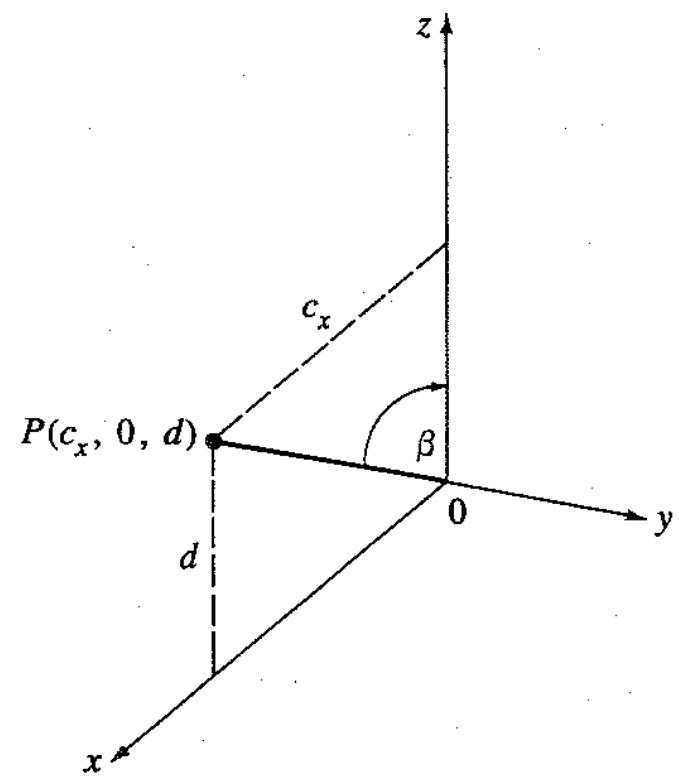
$$[M] = [T][R_x][R_y][R_\delta][R_y]^{-1}[R_x]^{-1}[T]^{-1}$$

$$[T] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_0 & -y_0 & -z_0 & 1 \end{bmatrix}$$

$$[R_x] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha & \sin \alpha & 0 \\ 0 & -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_z/d & c_y/d & 0 \\ 0 & -c_y/d & c_z/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



(a)



(b)

$$d = \sqrt{{c_y}^2 + {c_z}^2}$$

$$\cos \alpha = \frac{c_z}{d} \quad \sin \alpha = \frac{c_y}{d}$$

$$\cos \beta = d \quad \sin \beta = c_x$$

$$[R_y] = \begin{bmatrix} \cos(-\beta) & 0 & -\sin(-\beta) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(-\beta) & 0 & \cos(-\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} d & 0 & c_x & 0 \\ 0 & 1 & 0 & 0 \\ -c_x & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[R_\delta] = \begin{bmatrix} \cos \delta & \sin \delta & 0 & 0 \\ -\sin \delta & \cos \delta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[V] = [(x_1 - x_0) \quad (y_1 - y_0) \quad (z_1 - z_0)]$$

$$[c_x \quad c_y \quad c_z] = \frac{[(x_1 - x_0) \quad (y_1 - y_0) \quad (z_1 - z_0)]}{[(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2]^{\frac{1}{2}}}$$

Herhangi bir düzleme göre yansıtma=

$$[M] = [T] [R_x] [R_y] [R_{flt_z}] [R_y]^{-1} [R_x]^{-1} [T]^{-1}$$

(x_0, y_0, z_0) ve (x_1, y_1, z_1) noktalarından geçen doğrunun denklemi,

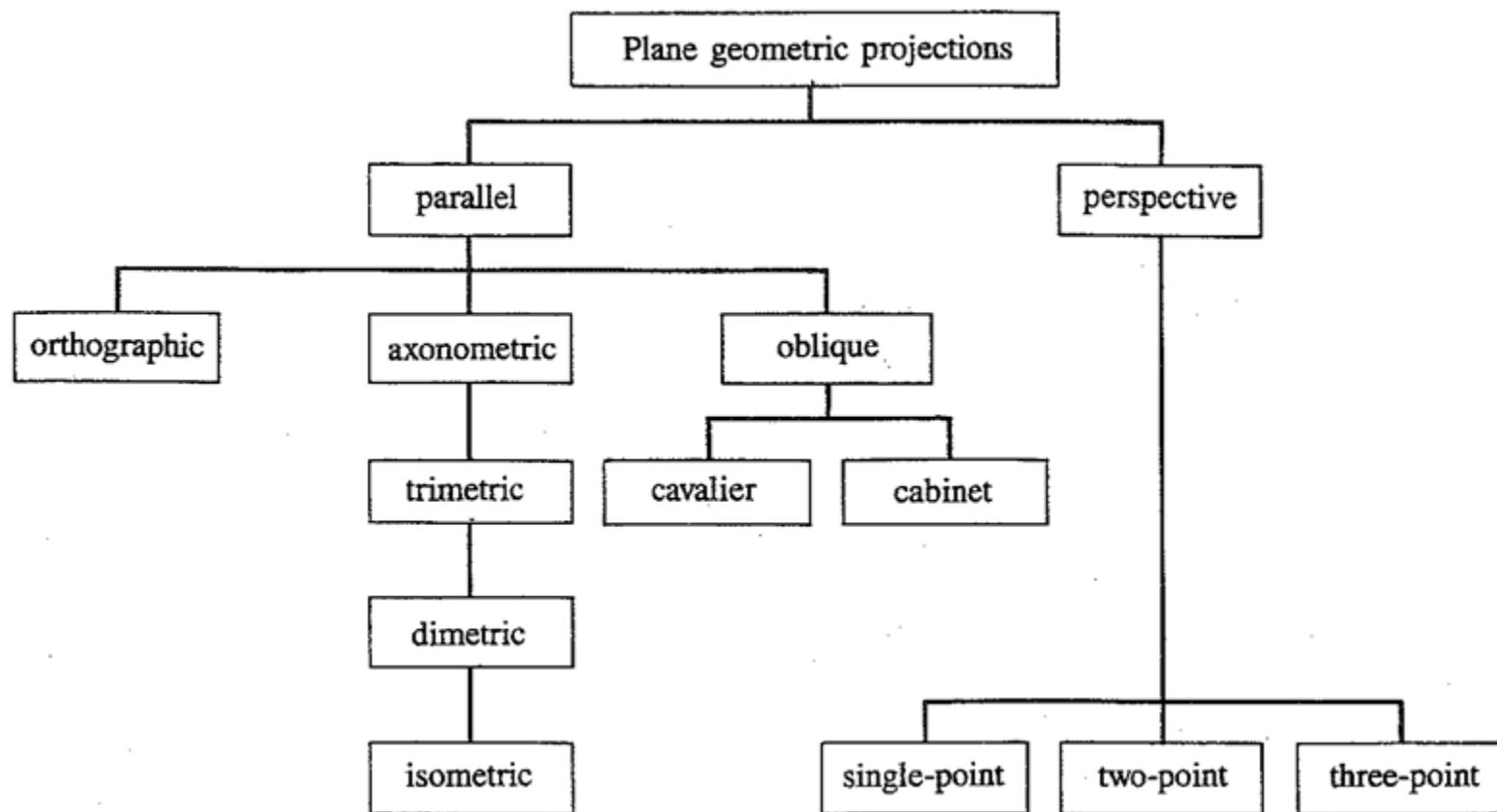
$$\frac{x - x_0}{x_1 - x_0} = \frac{y - y_0}{y_1 - y_0} = \frac{z - z_0}{z_1 - z_0}$$

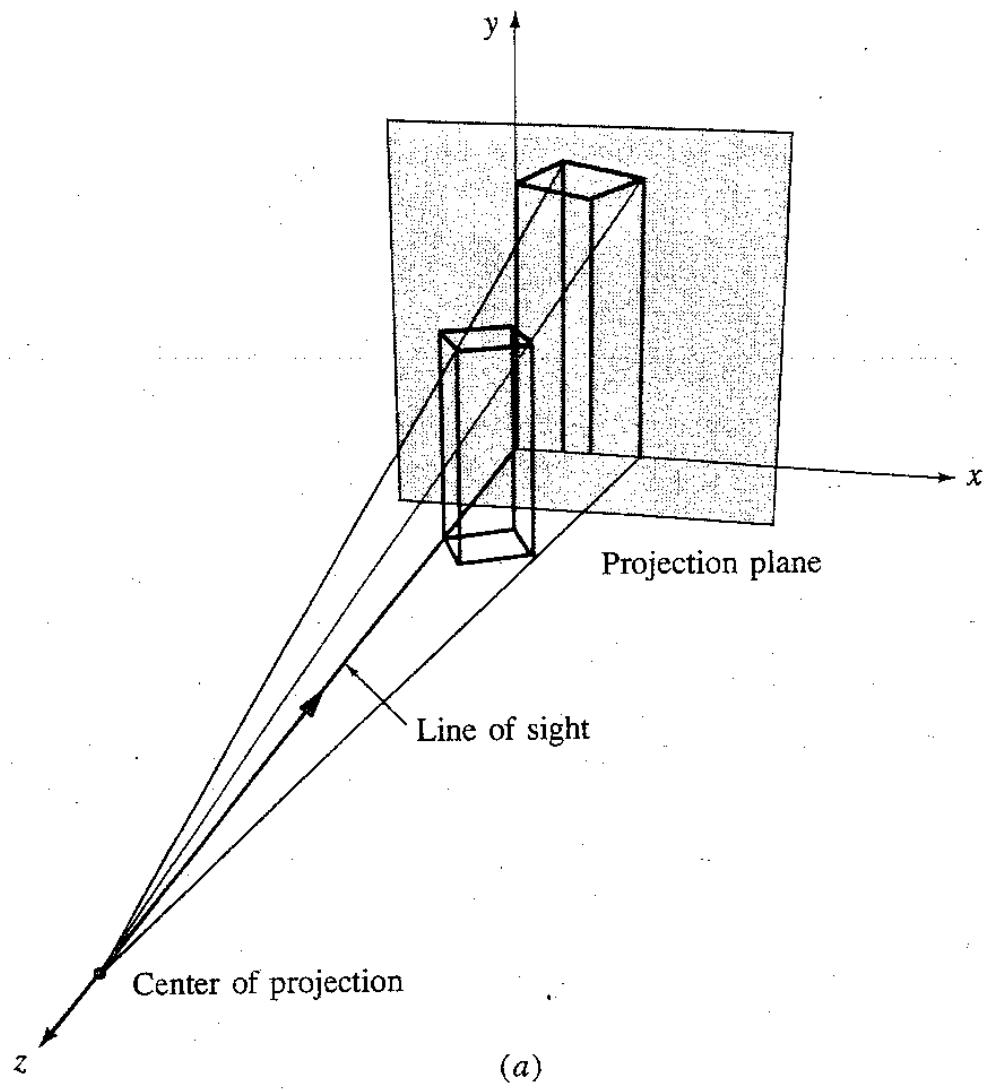
verilen iki doğrunun arasındaki açı α 'nın kosinüsü,

$$\cos \alpha = \frac{a_0.a_1 + b_0.b_1 + c_0.c_1}{\sqrt{a_0^2 + b_0^2 + c_0^2} \cdot \sqrt{a_1^2 + b_1^2 + c_1^2}}$$

Eğer bu değer, pozitif ise, doğrular arasındaki dar açı; negatif ise, doğrular arasındaki geniş açı elde edilir.

Projeksiyon(izdüşüm)





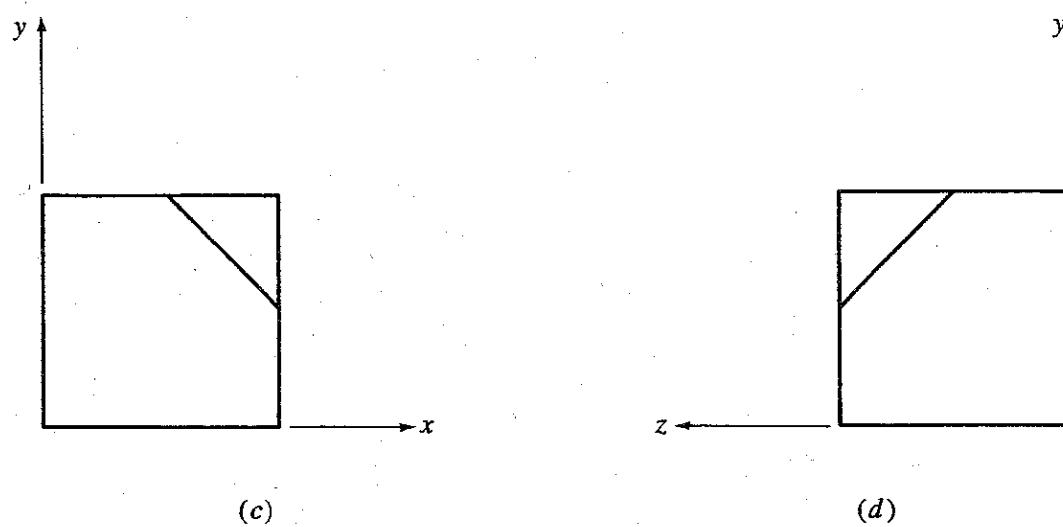
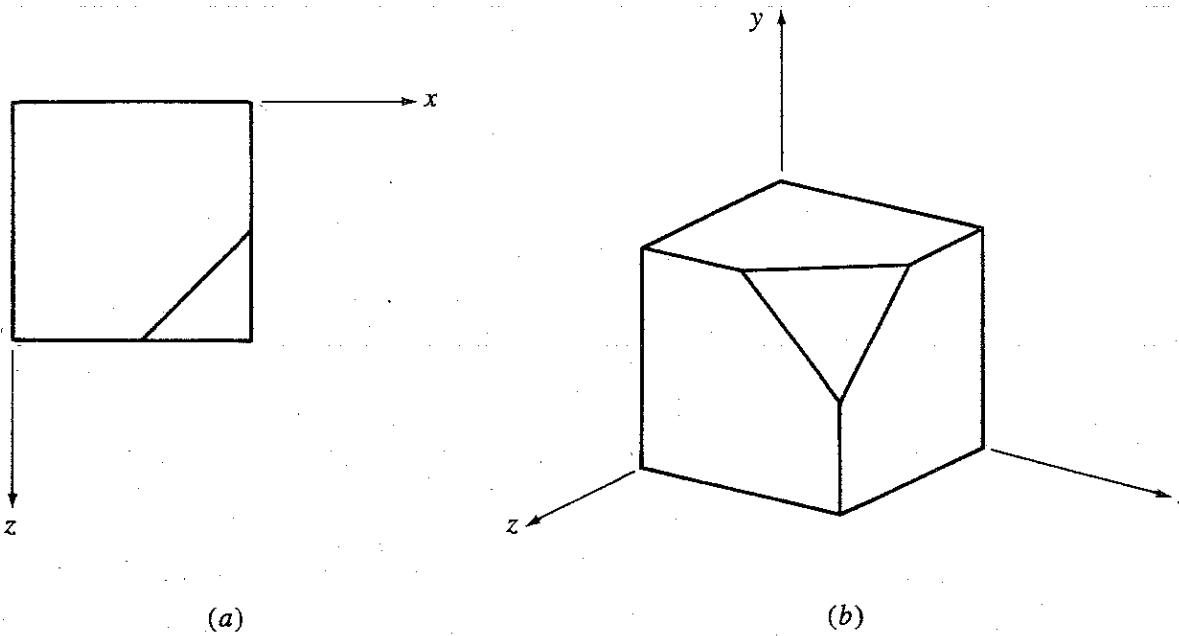
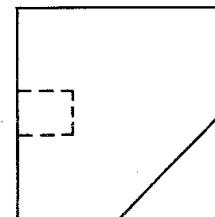
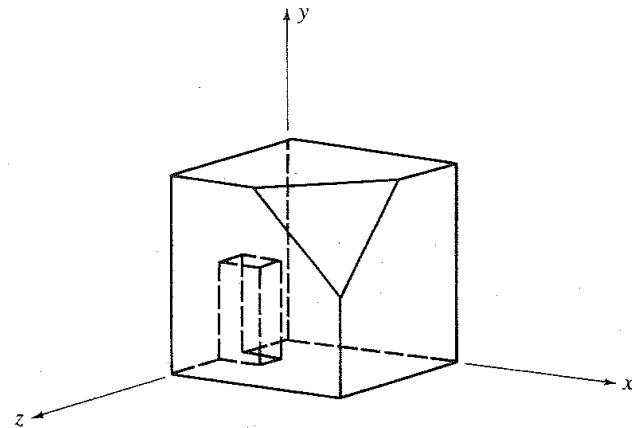
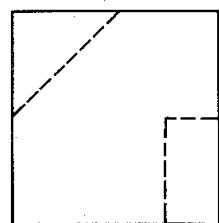


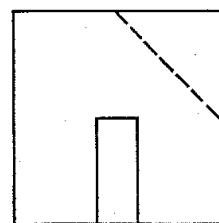
Figure 3–12 Orthographic projections onto (b) $y = 0$, (c) $z = 0$ and (d) $x = 0$ planes.



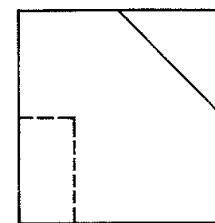
Top



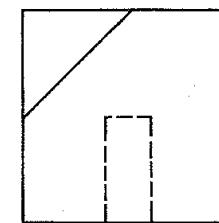
Rear



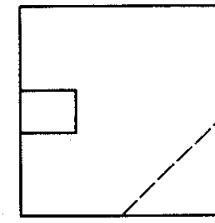
Left side



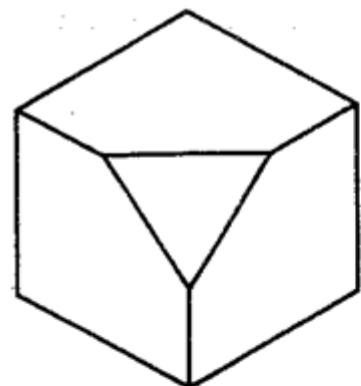
Front



Right side

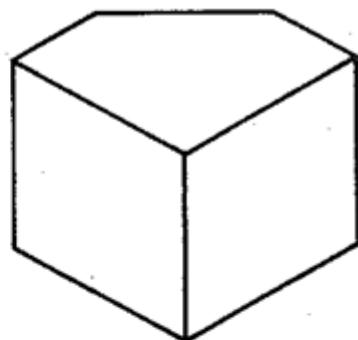


Bottom



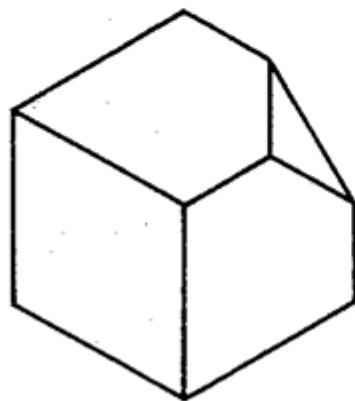
$\phi < 0, \theta > 0$

(a)



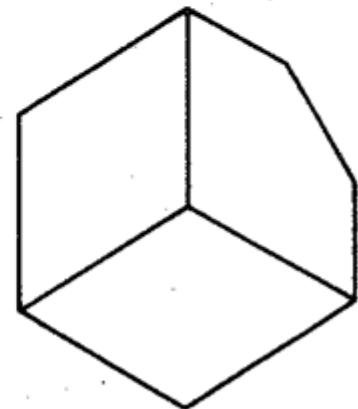
$\phi < 0, \theta < 0$

(b)



$\phi > 0, \theta > 0$

(c)



$\phi > 0, \theta < 0$

(d)

Four possible isometric projections with rotation angles $\phi = \pm 45^\circ$, $\theta = \pm 35.26^\circ$. (a) $\phi = -45^\circ, \theta = +35.26^\circ$; (b) $\phi = -45^\circ, \theta = -35.26^\circ$; (c) $\phi = +45^\circ, \theta = +35.26^\circ$; (d) $\phi = +45^\circ, \theta = -35.26^\circ$.

$$[T] = [R_y][R_x][P_z]$$

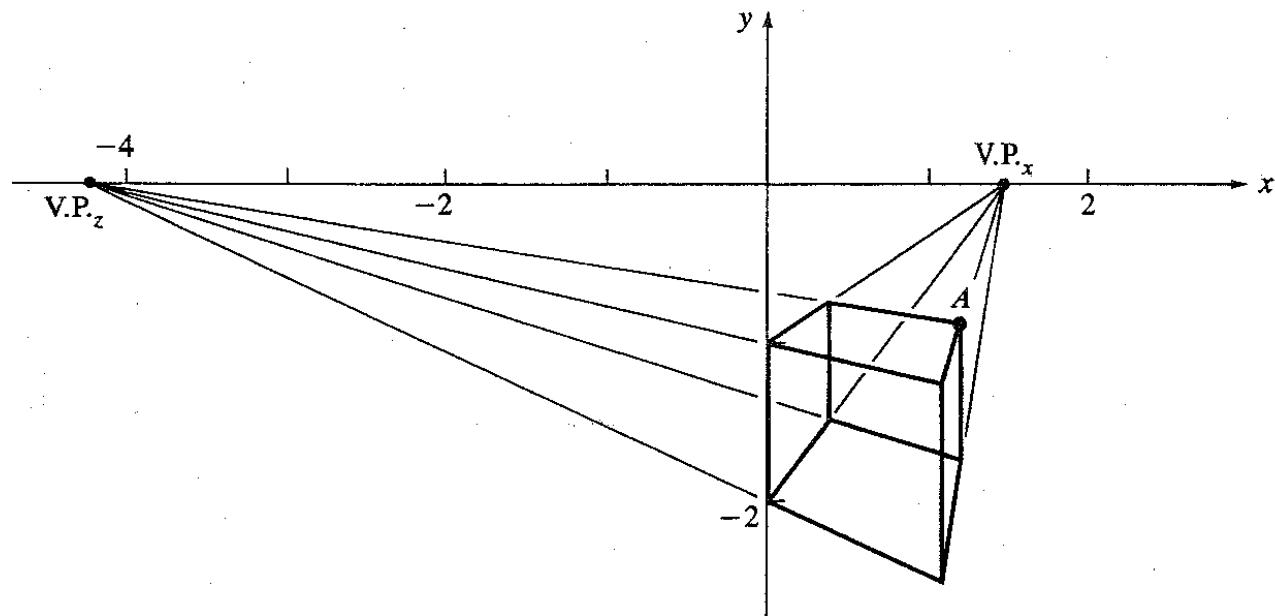
$$= \begin{bmatrix} \cos \phi & 0 & -\sin \phi & 0 \\ 0 & 1 & 0 & 0 \\ \sin \phi & 0 & \cos \phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & \sin \theta & 0 \\ 0 & -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\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

$$[T] = \begin{bmatrix} \cos \phi & \sin \phi \sin \theta & 0 & 0 \\ 0 & \cos \theta & 0 & 0 \\ \sin \phi & -\cos \phi \sin \theta & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad [X] = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0.5 & 1 & 1 \\ 0.5 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0.5 & 1 \end{bmatrix}$$

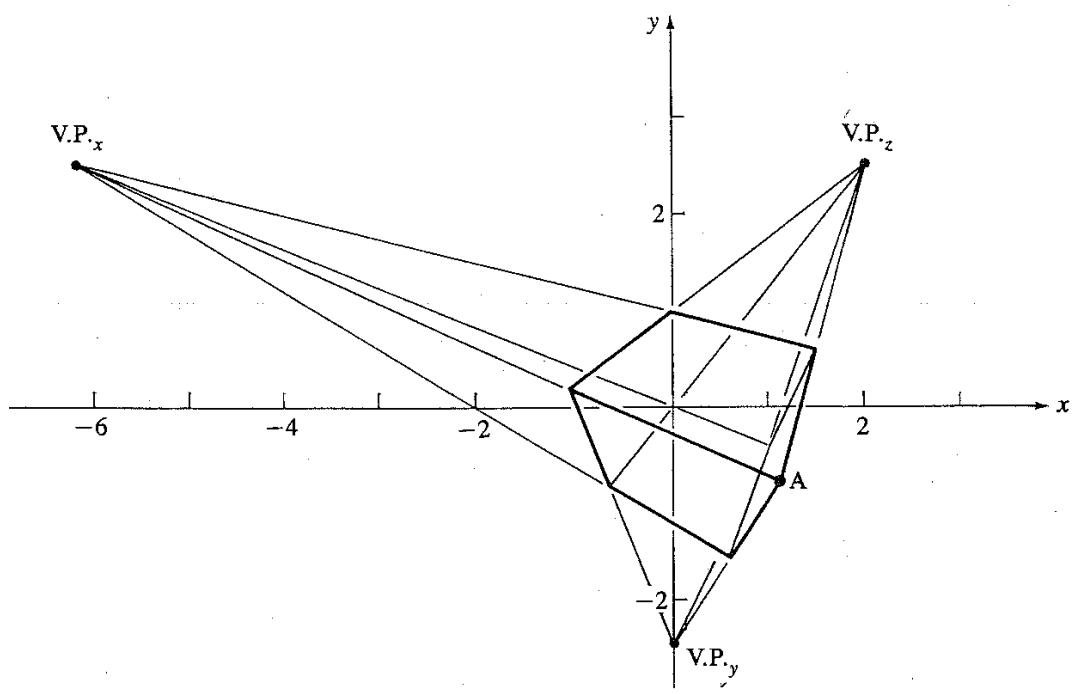
$$[T] = \begin{bmatrix} 0.707 & -0.408 & 0 & 0 \\ 0 & 0.816 & 0 & 0 \\ -0.707 & -0.408 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

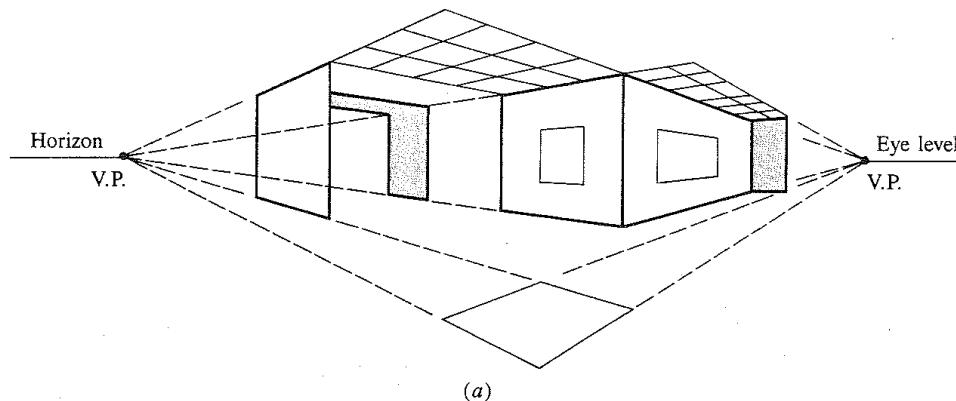
$$[X^*] = [X][T] = \begin{bmatrix} -0.707 & -0.408 & 0 & 1 \\ 0 & -0.816 & 0 & 1 \\ 0 & -0.408 & 0 & 1 \\ -0.354 & 0.204 & 0 & 1 \\ -0.707 & 0.408 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0.707 & -0.408 & 0 & 1 \\ 0.707 & 0.408 & 0 & 1 \\ 0 & 0.816 & 0 & 1 \\ 0.354 & 0.204 & 0 & 1 \end{bmatrix}$$

(a)

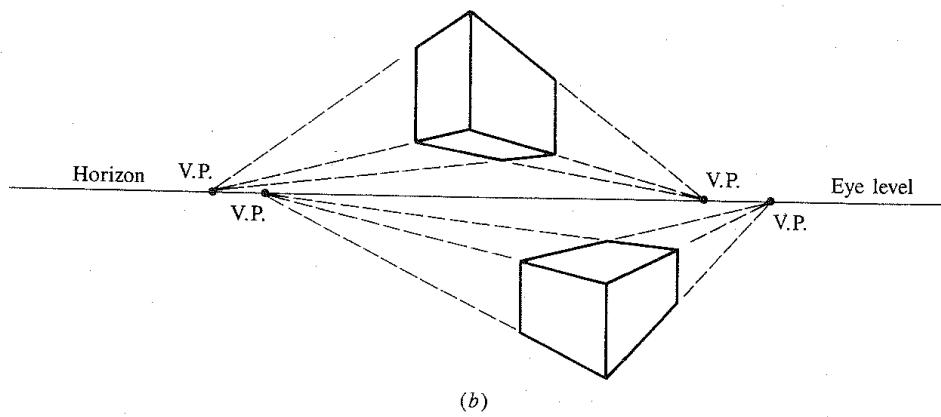


(b)

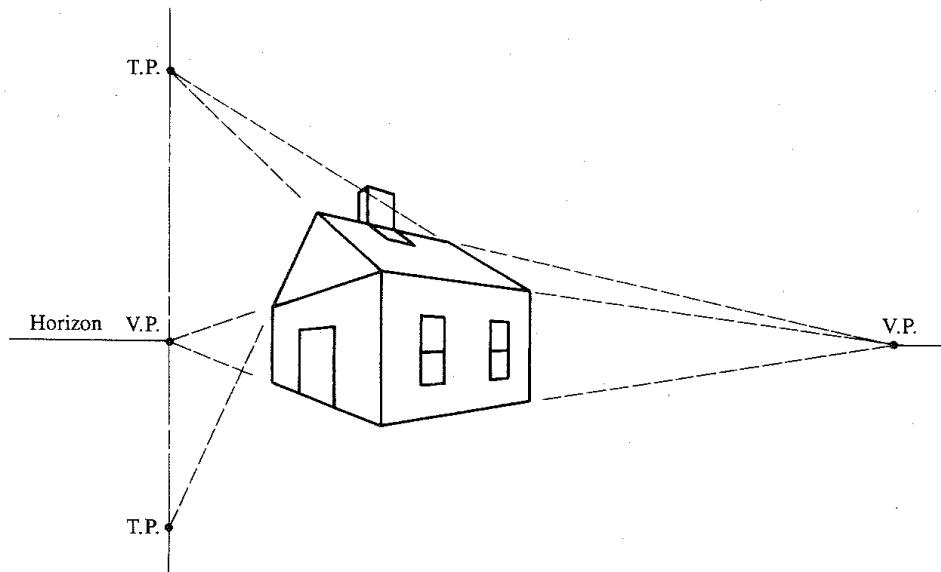


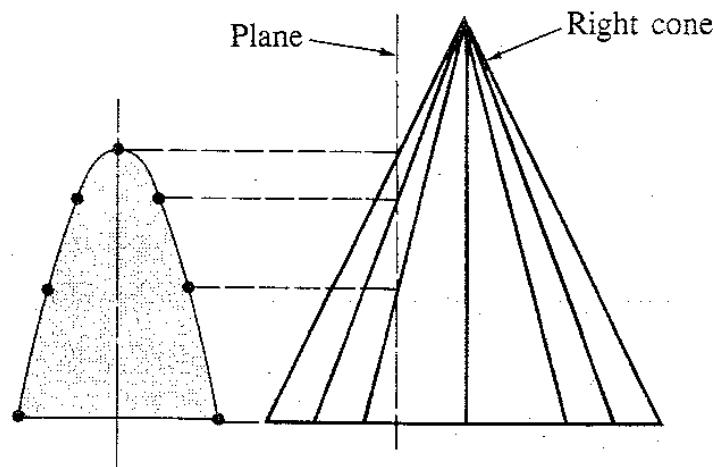


(a)



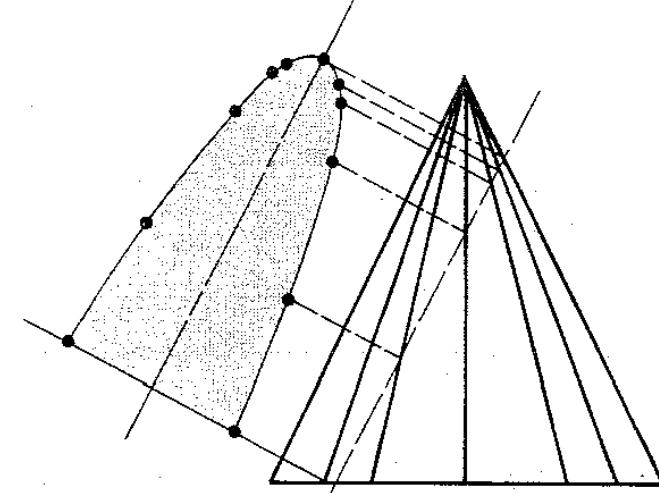
(b)





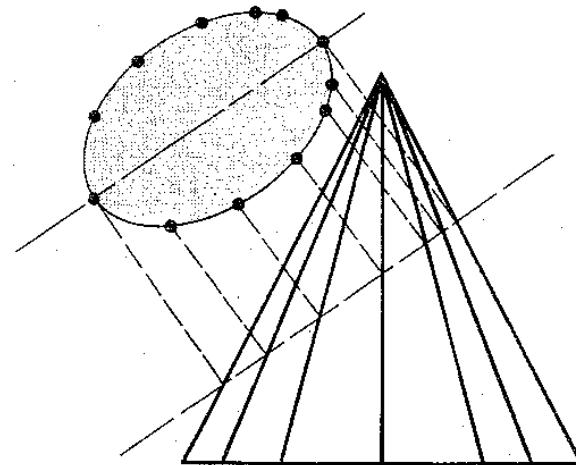
Hyperbola

(a)



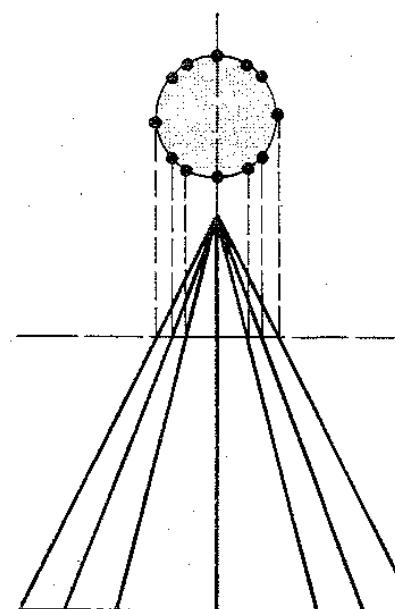
Parabola

(b)



Ellipse

(c)



Circle

(d)

$$x = r \cos \theta \quad 0 \leq \theta \leq \pi$$

$$y = r \sin \theta$$

the parametric equation of the sphere is

$$\begin{aligned} Q(\theta, \phi) &= [x(\theta) \quad y(\theta) \cos \phi \quad y(\theta) \sin \phi] \\ &= [r \cos \theta \quad r \sin \theta \cos \phi \quad r \sin \theta \sin \phi] \quad 0 \leq \theta \leq \pi \\ &\quad 0 \leq \phi \leq 2\pi \end{aligned}$$

$$x = a \cos \theta \quad 0 \leq \theta \leq \pi$$

$$y = b \sin \theta$$

gives the parametric equation for any point on the ellipsoid of revolution as

$$\begin{aligned} Q(\theta, \phi) &= [a \cos \theta \quad b \sin \theta \cos \phi \quad b \sin \theta \sin \phi] \quad 0 \leq \theta \leq \pi \\ &\quad 0 \leq \phi \leq 2\pi \end{aligned}$$

the parametric parabola $x = a\theta^2 \quad 0 \leq \theta \leq \theta_{\max}$

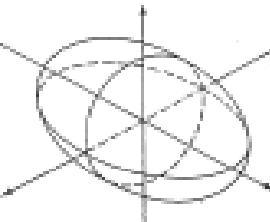
$$y = 2a\theta$$

the parametric hyperbola $x = a \sec \theta \quad 0 \leq \theta \leq \theta_{\max}$

$$y = b \tan \theta$$

Ellipsoid:

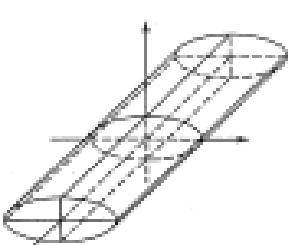
$$\begin{aligned}x &= a \cos \theta \sin \phi & 0 \leq \theta \leq 2\pi \\y &= b \sin \theta \sin \phi & 0 \leq \phi \leq 2\pi \\z &= c \cos \phi\end{aligned}$$



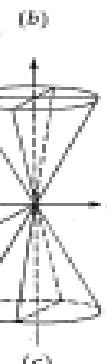
General ellipsoid
 $\alpha \neq \beta \neq \gamma$

Hyperboloid of one sheet:

$$\begin{aligned}x &= a \cos \theta \cosh \phi & 0 \leq \theta \leq 2\pi \\y &= b \sin \theta \sinh \phi & -\pi \leq \phi \leq \pi \\z &= c \sinh \phi\end{aligned}$$



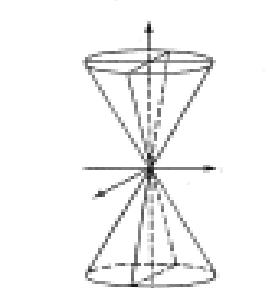
Elliptic cylinder
 $\alpha \neq \beta$



Double cone
 $\alpha \neq \beta$

Hyperboloid of two sheets:

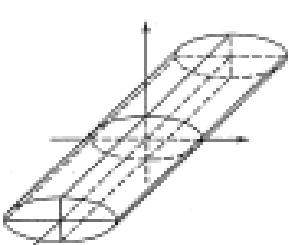
$$\begin{aligned}x &= \pm a \cosh \phi & 0 \leq \theta \leq 2\pi \\y &= b \sin \theta \sinh \phi & -\pi \leq \phi \leq \pi \\z &= c \cos \theta \sinh \phi\end{aligned}$$



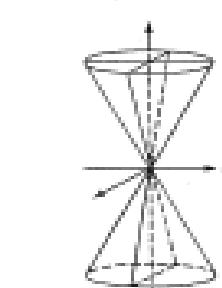
Hyperboloid of one sheet
 $\alpha \neq \beta$

Elliptic paraboloid:

$$\begin{aligned}x &= a \phi \cos \theta & 0 \leq \theta \leq 2\pi \\y &= b \phi \sin \theta & 0 \leq \phi \leq \phi_{\max} \\z &= \phi^2\end{aligned}$$



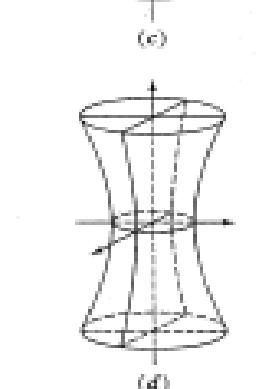
Elliptic cylinder
 $\alpha \neq \beta$



Cone asymptotic to both hyperboloids
 $\alpha = 0$

Hyperbolic paraboloid:

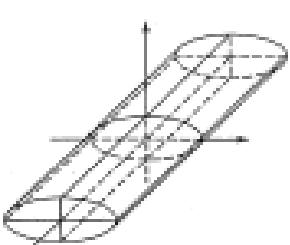
$$\begin{aligned}x &= a \phi \cosh \theta & -\pi \leq \theta \leq \pi \\y &= b \phi \sinh \theta & \phi_{\min} \leq \phi \leq \phi_{\max} \\z &= \phi^2\end{aligned}$$



Elliptic paraboloid

Elliptic cone:

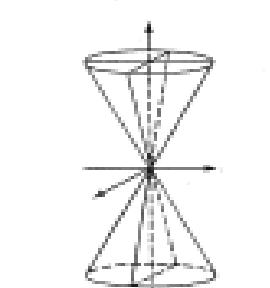
$$\begin{aligned}x &= a \phi \cos \theta & 0 \leq \theta \leq 2\pi \\y &= b \phi \sin \theta & \phi_{\min} \leq \phi \leq \phi_{\max} \\z &= c \phi\end{aligned}$$



Hyperboloid of two sheets
 $\alpha \neq \beta$

Elliptic cylinder:

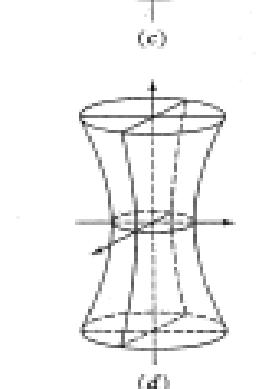
$$\begin{aligned}x &= a \cos \theta & 0 \leq \theta \leq 2\pi \\y &= b \sin \theta & \phi_{\min} \leq \phi \leq \phi_{\max} \\z &= \phi\end{aligned}$$



Hyperboloid of one sheet
 $\alpha \neq \beta$

Parabolic cylinder:

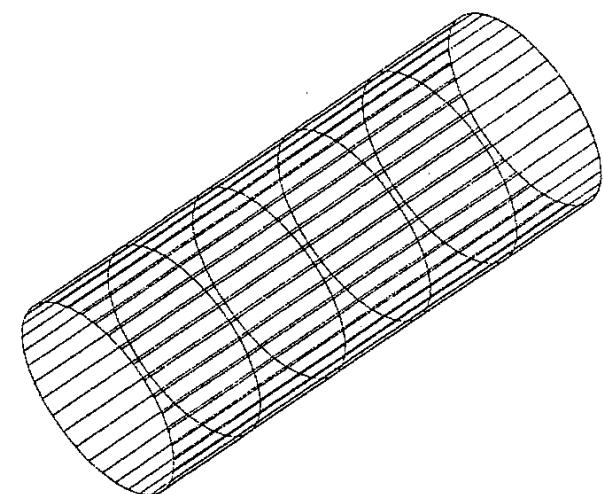
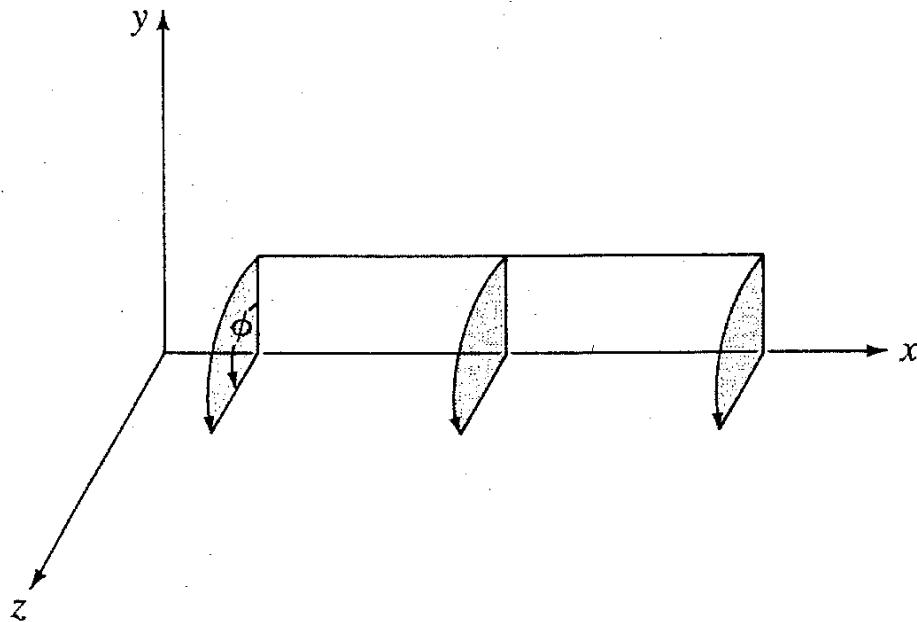
$$\begin{aligned}x &= a \theta^2 & 0 \leq \theta \leq \theta_{\max} \\y &= 2a \theta & \phi_{\min} \leq \phi \leq \phi_{\max} \\z &= \phi\end{aligned}$$



Hyperbolic paraboloid

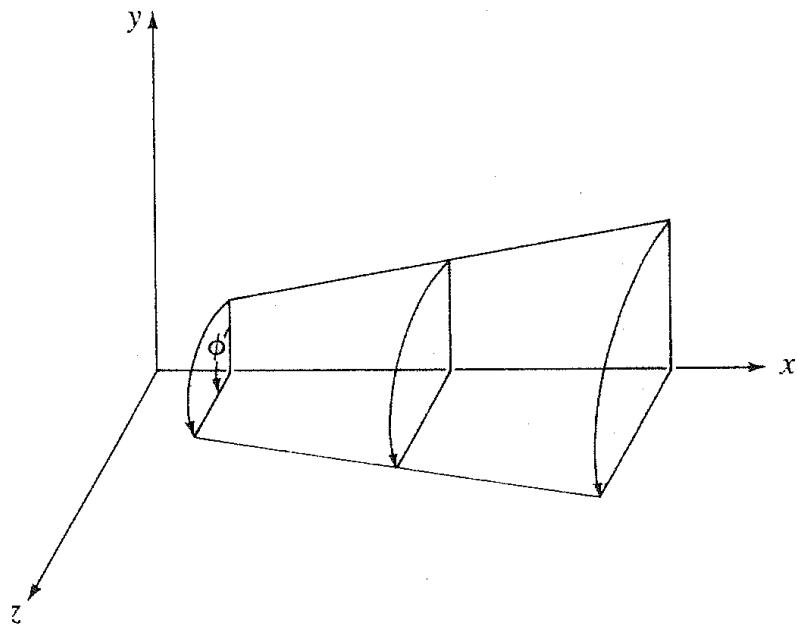
3D Uzay Yüzeyleri

Kaydırma Yüzeyi :

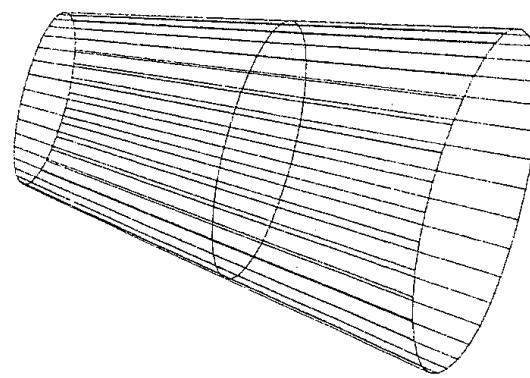


(b)

Döndürme Yüzeyi :

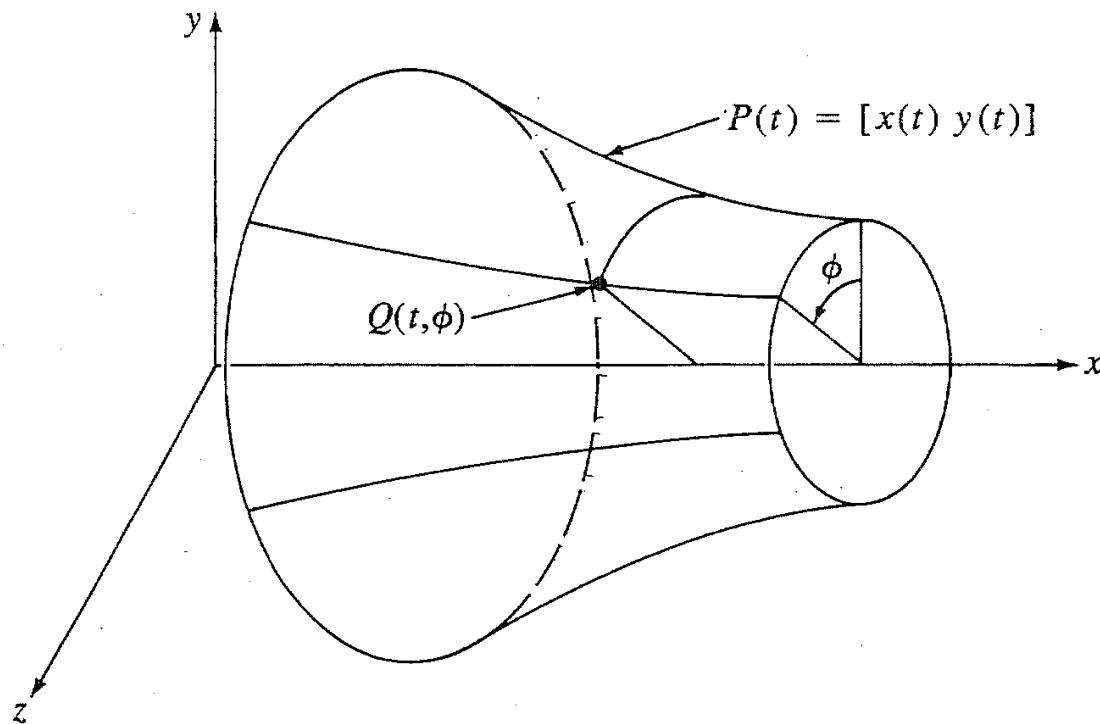


(a)

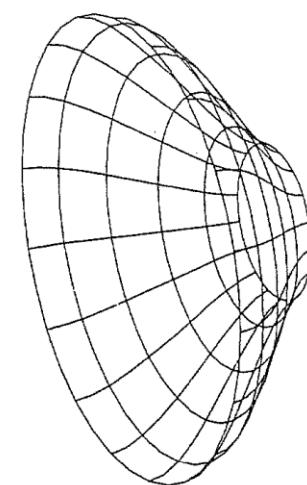
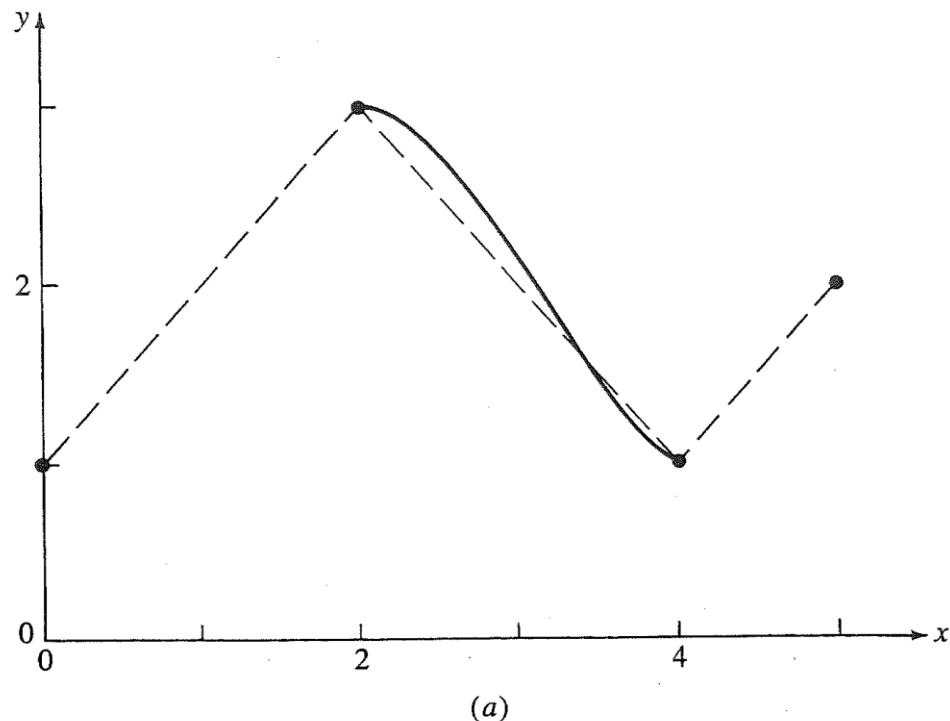


(b)

Döndürme Yüzeyi :

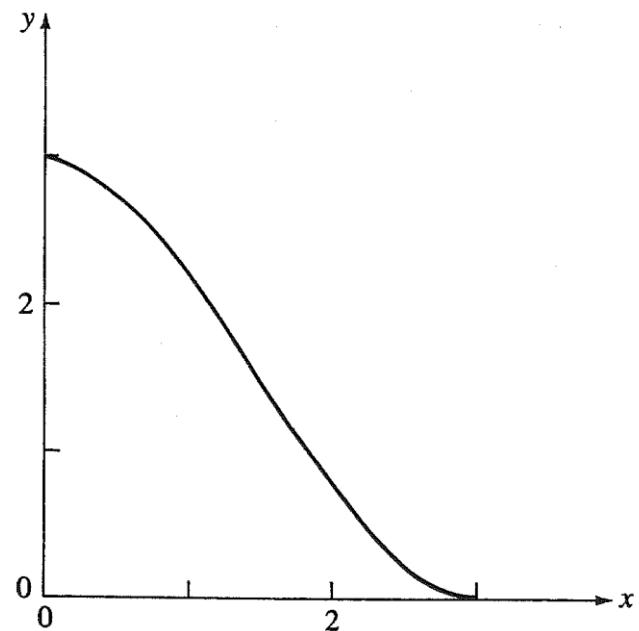


Döndürme Yüzeyi :

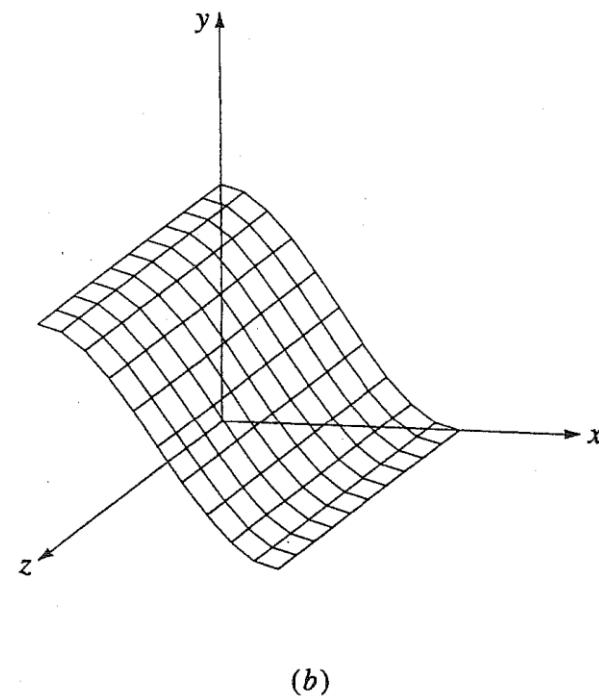


(b)

Kaydırma Yüzeyi :

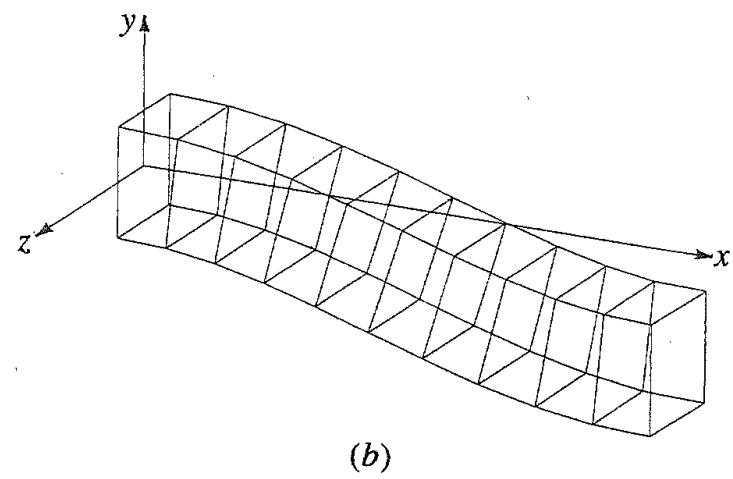
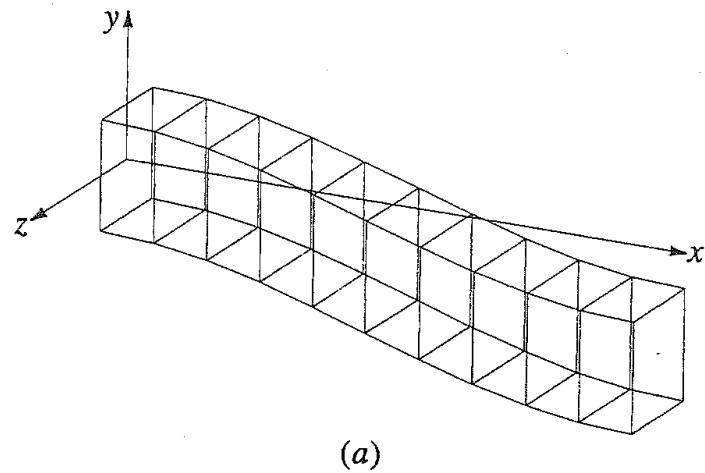


(a)

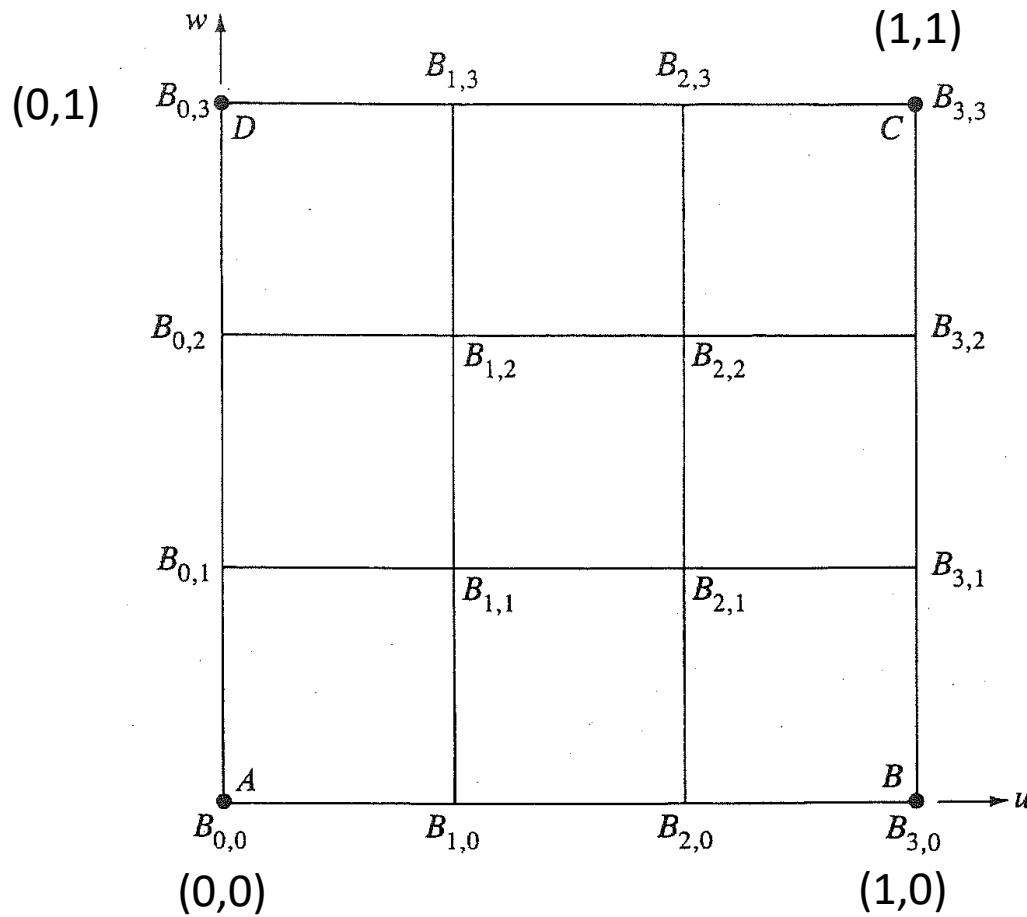


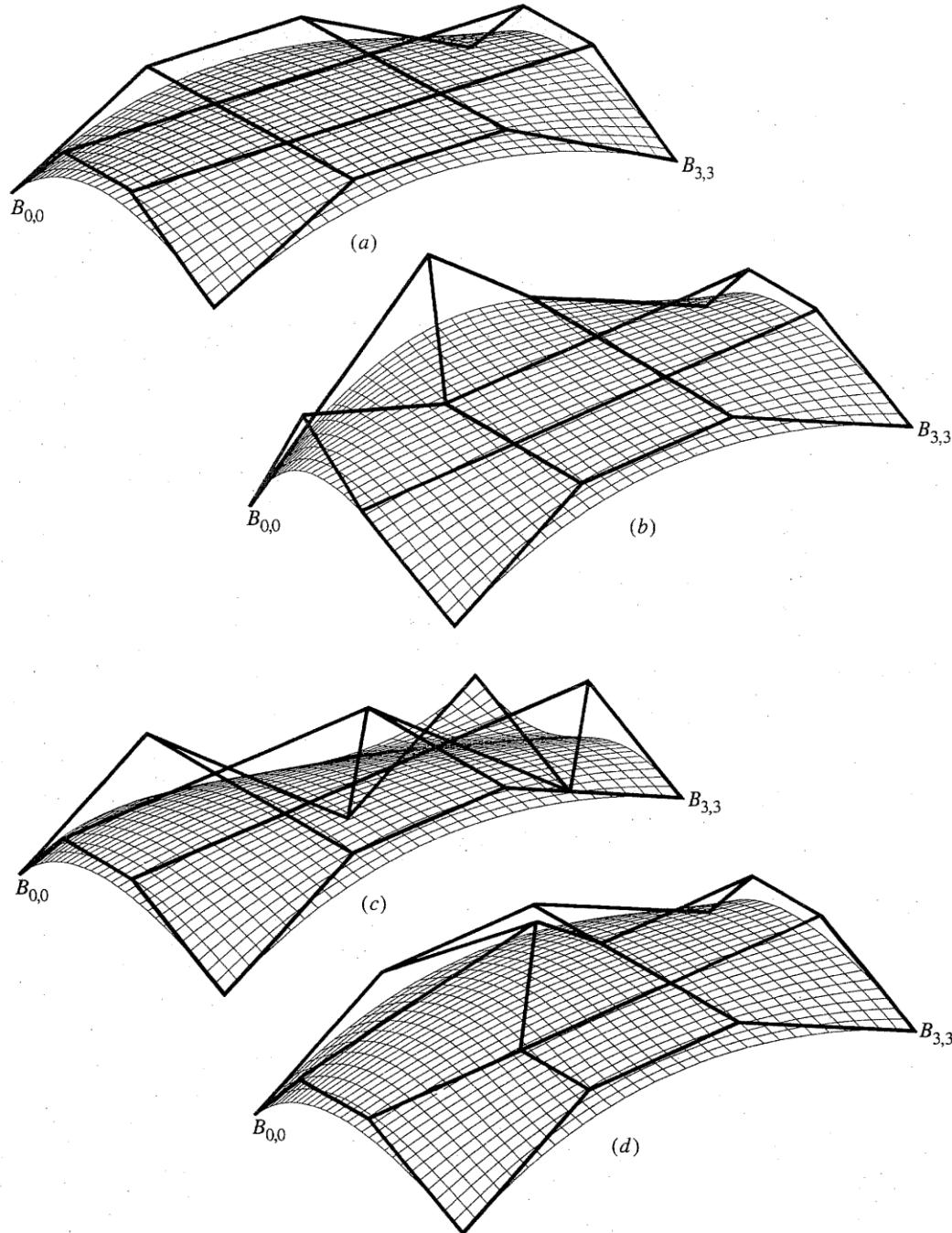
(b)

Kaydırma Yüzeyi :



Bezier Yüzeyi :





$$Q(u,w)=\sum_{i=0}^n \sum_{j=0}^m B_{i,j} J_{n,i}(u) K_{m,j}(w)$$

$$J_{n,i}(u) = \binom{n}{i} u^i (1-u)^{n-i}$$

$$K_{m,j}(w) = \binom{m}{j} w^j (1-w)^{m-j}$$

$$\binom{n}{i}=\frac{n!}{i!(n-i)!}$$

$$\binom{m}{j}=\frac{m!}{j!(m-j)!}$$

$$Q(u,w)=[\;U\;][\;N\;][\;B\;][\;M\;]^T[\;W\;]$$

$$Q(u, w) = [u^3 \ u^2 \ u \ 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \times$$

4x4
nokta
matrisi

$$\begin{bmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} w^3 \\ w^2 \\ w \\ 1 \end{bmatrix}$$

$$Q(u, w) = [u^4 \ u^3 \ u^2 \ u \ 1] \begin{bmatrix} 1 & -4 & 6 & -4 & 1 \\ -4 & -12 & -12 & 4 & 0 \\ 6 & -12 & 6 & 0 & 0 \\ -4 & 4 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \times$$

5x3
nokta
matrisi

$$\begin{bmatrix} B_{0,0} & B_{0,1} & B_{0,2} \\ B_{1,0} & B_{1,1} & B_{1,2} \\ B_{2,0} & B_{2,1} & B_{2,2} \\ B_{3,0} & B_{3,1} & B_{3,2} \\ B_{4,0} & B_{4,1} & B_{4,3} \end{bmatrix} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} w^2 \\ w \\ 1 \end{bmatrix}$$

B-Spline Yüzeyi :

$$Q(u, w) = \sum_{i=1}^{n+1} \sum_{j=1}^{m+1} B_{i,j} N_{i,k}(u) M_{j,l}(w)$$

$$N_{i,k}(u) = \begin{cases} 1 & \text{if } x_i \leq u < x_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,k}(u) = \frac{(u - x_i) N_{i,k-1}(u)}{x_{i+k-1} - x_i} + \frac{(x_{i+k} - u) N_{i+1,k-1}(u)}{x_{i+k} - x_{i+1}}$$

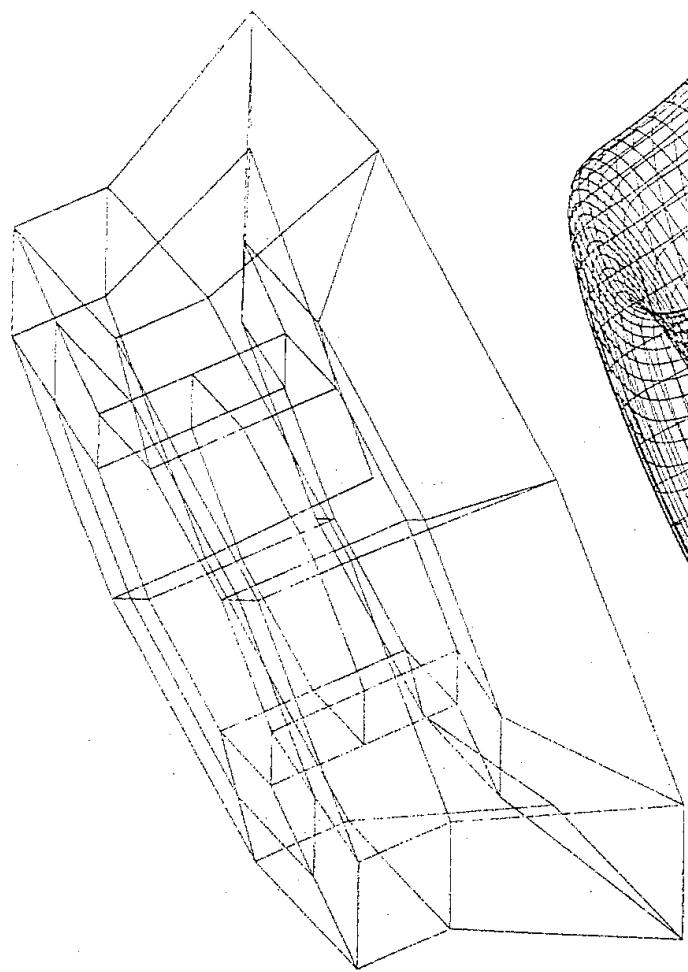
$$M_{j,l}(w) = \begin{cases} 1 & \text{if } y_j \leq w < y_{j+1} \\ 0 & \text{otherwise} \end{cases}$$

$$M_{j,l}(w) = \frac{(w - y_j) M_{j,l-1}(w)}{y_{j+l-1} - y_j} + \frac{(y_{j+l} - w) M_{j+1,l-1}(w)}{y_{j+l} - y_{j+1}}$$

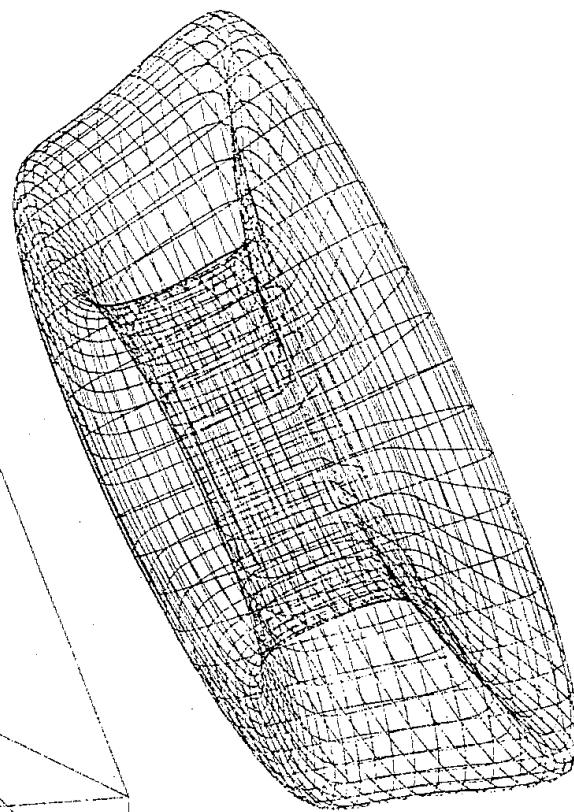
$$x_i = 0 \quad 1 \leq i \leq k$$

$$x_i = i - k \quad k + 1 \leq i \leq n + 1$$

$$x_i = n - k + 2 \quad n + 2 \leq i \leq n + k + 1$$



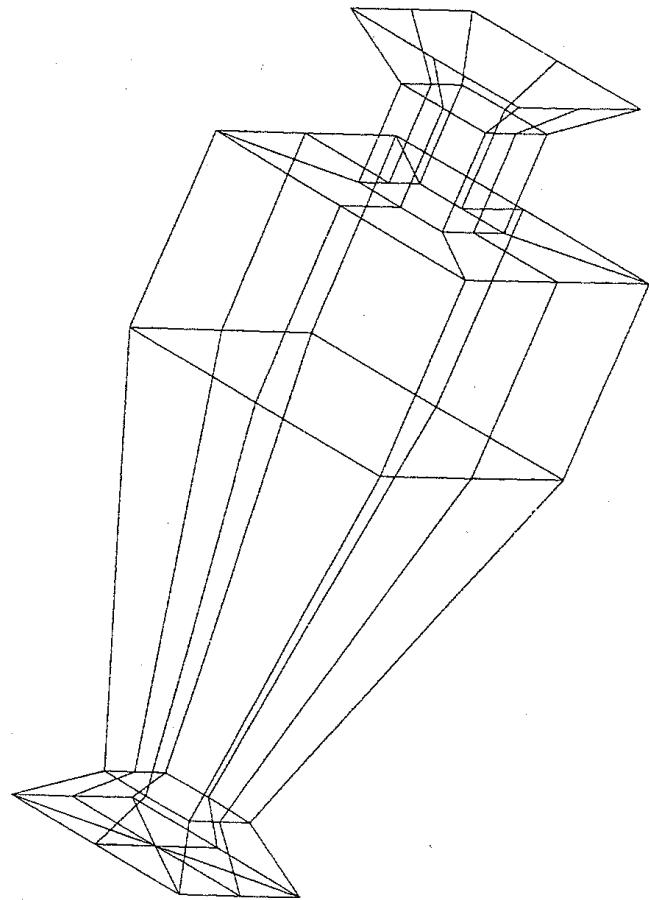
(a)



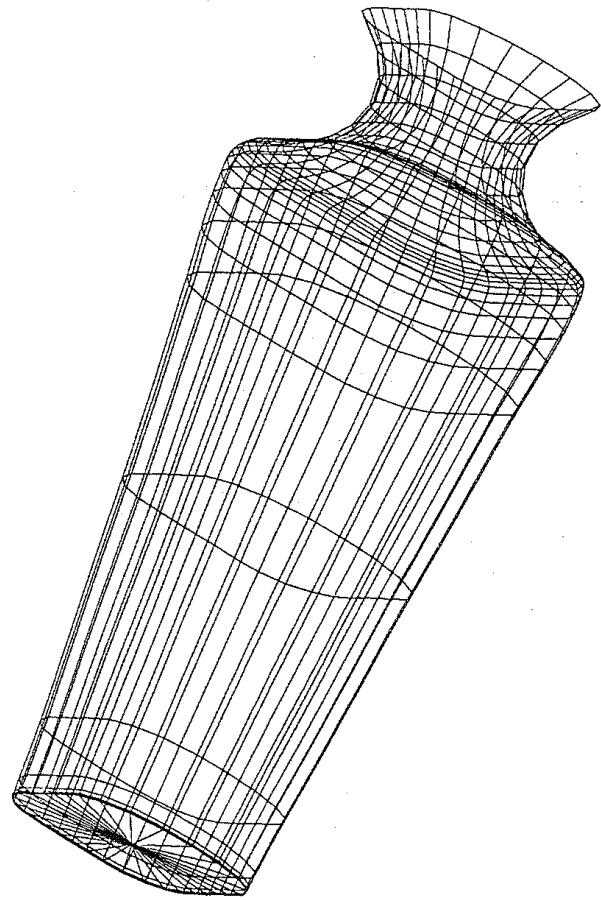
(b)



(a)



(c)



(d)

Saklı Yüzey

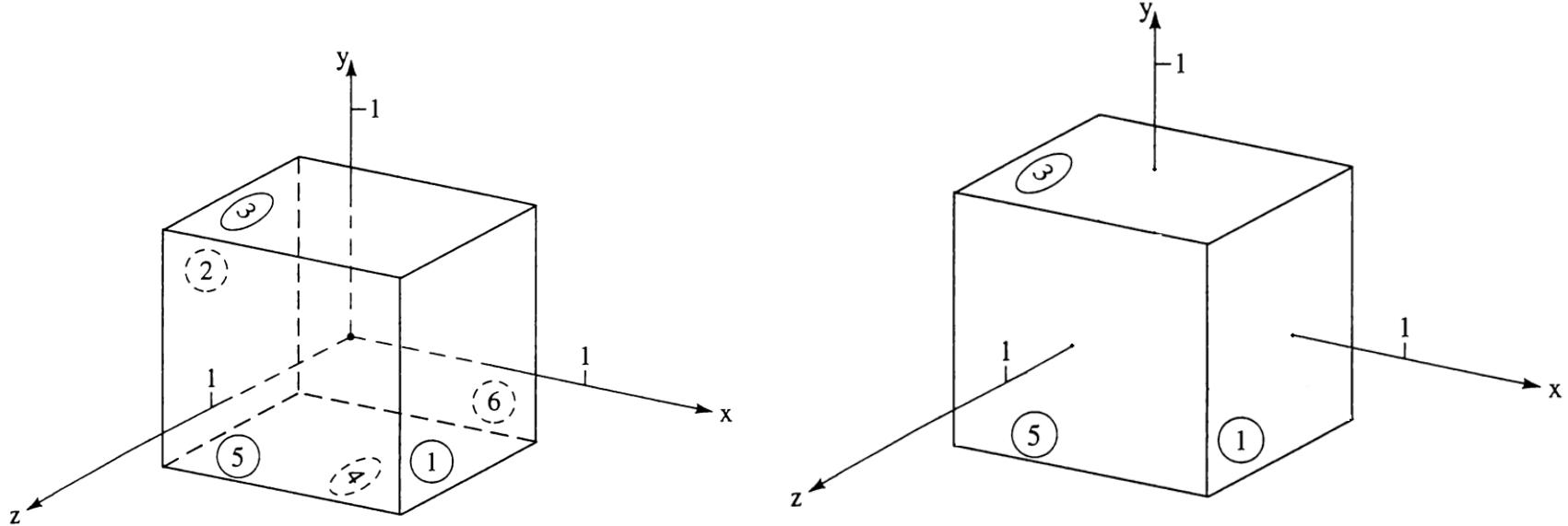
Roberts Algorithm

$$ax + by + cz + d = 0$$

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = 0$$

$$\begin{bmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \\ c_1 & c_2 & \dots & c_n \\ d_1 & d_2 & \dots & d_n \end{bmatrix}$$

$$[S] = [x \ y \ z \ 1]$$



$$[V] = \begin{bmatrix} \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{bmatrix} = \begin{bmatrix} \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} \\ 2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 2 \\ -1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix}$$

$$[S] = \left[\begin{array}{ccccc} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 1 \end{array} \right] = [1 \ 1 \ 1 \ 4]$$

$$[S] \cdot [V] = [1 \ 1 \ 1 \ 4] \begin{bmatrix} \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} \\ 2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 2 \\ -1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix}$$

$$\begin{array}{cccccc} \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} \\ = [-2 & 6 & -2 & 6 & -2 & 6] \end{array}$$

$$[V] = \begin{bmatrix} \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} \\ -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\begin{aligned} [VT] &= [T]^{-1}[V] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} \\ -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} \\ -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 7 & -5 & 1 & 1 & 1 & 1 \end{bmatrix} \end{aligned}$$

$$[S] = \begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 1 \end{bmatrix} = [1 \quad 1 \quad 1 \quad 4]$$

$$[S] \cdot [V] \geq 0.$$

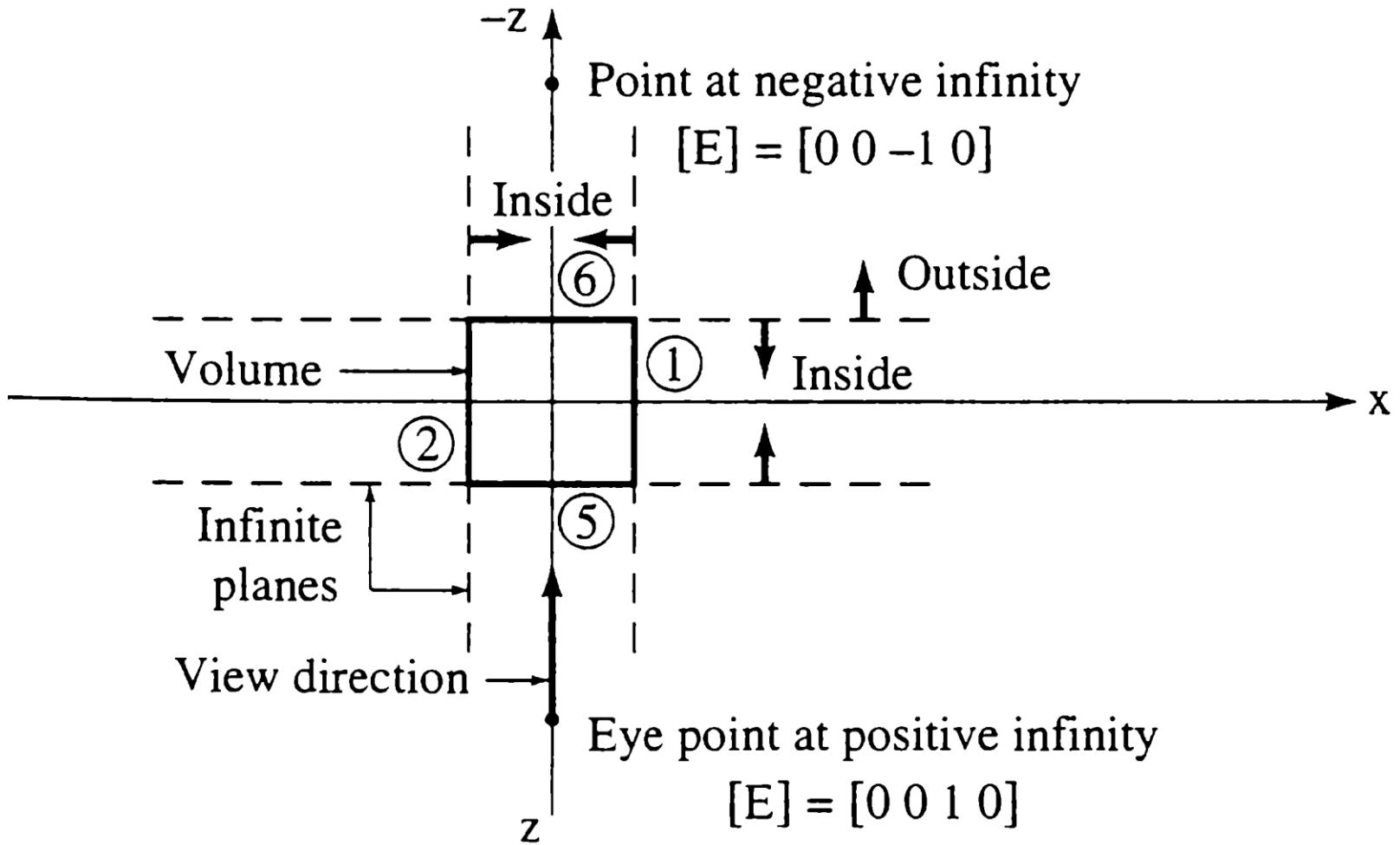
$$[ST] = [S][T] = [1 \quad 1 \quad 1 \quad 4][T] = [13 \quad 1 \quad 4] = \left[3\frac{1}{4} \quad \frac{1}{4} \quad \frac{1}{4} \quad 1 \right]$$

$$\begin{array}{ccccccc} \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} \\ [ST] \cdot [VT] = [2 & 6 & 2 & 6 & 2 & 6] \end{array}$$

$$[E] = [0 \quad 0 \quad -1 \quad 0]$$

$$[E] \cdot [V] < 0$$

$$[E] \cdot [V] = [0 \ 0 \ -1 \ 0] \begin{bmatrix} \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} \\ -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} [0 \ 0 \ 0 \ 0 \ 2 \ -2]$$



Saklı Kenar

$$P(t) = P_1 + (P_2 - P_1)t \quad 0 \leq t \leq 1$$

$$\mathbf{v} = \mathbf{s} + \mathbf{d}t$$

$$\mathbf{Q}(\alpha, t) = \mathbf{u} = \mathbf{v} + \mathbf{g}\alpha = \mathbf{s} + \mathbf{d}t + \mathbf{g}\alpha \quad 0 \leq t \leq 1, \quad \alpha \geq 0$$

$$h = \mathbf{u} \cdot [VT] = \mathbf{s} \cdot [VT] + t\mathbf{d} \cdot [VT] + \alpha\mathbf{g} \cdot [VT] > 0 \quad 0 \leq t \leq 1, \quad \alpha \geq 0$$

$$p = \mathbf{s} \cdot [VT]$$

$$q = \mathbf{d} \cdot [VT]$$

$$w = \mathbf{g} \cdot [VT]$$

$$P_1 = [-2 \ 0 \ -2 \ 1]$$

$$P_2 = [2 \ 0 \ -2 \ 1]$$

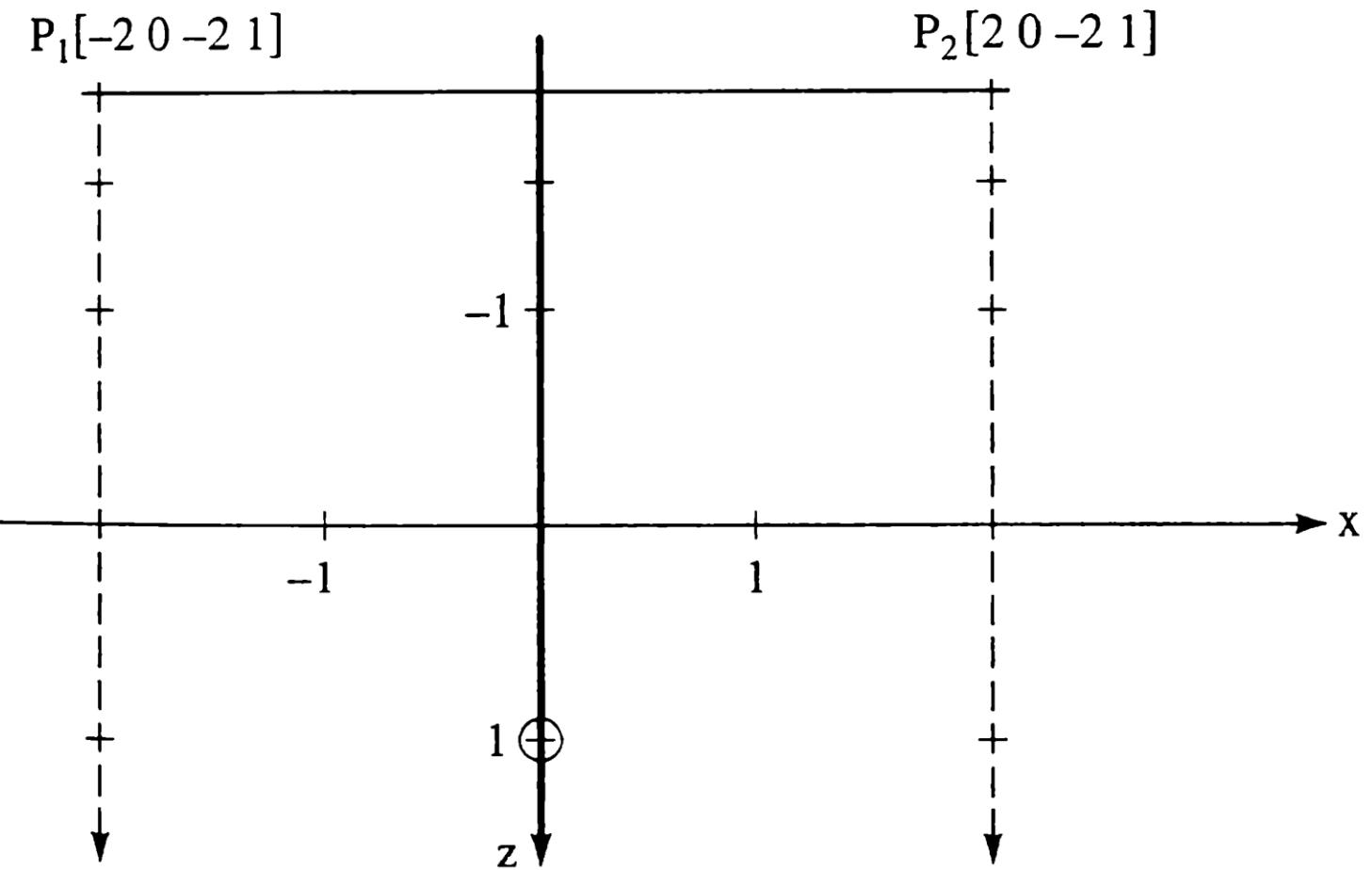
$$P(t) = \mathbf{v} = \mathbf{s} + \mathbf{d}t = [-2 \ 0 \ -2 \ 1] + [4 \ 0 \ 0 \ 0]t$$

$$\mathbf{g} = [0 \ 0 \ 1 \ 0]$$

$$\begin{aligned} Q(\alpha, t) &= \mathbf{s} + \mathbf{d}t + \mathbf{g}\alpha = [-2 \ 0 \ -2 \ 1] \\ &\quad + [4 \ 0 \ 0 \ 0]t + [0 \ 0 \ 1 \ 0]\alpha \end{aligned}$$

$$\begin{aligned} P\left(\frac{1}{2}\right) &= \mathbf{v} = [-2 \ 0 \ -2 \ 1] + [4 \ 0 \ 0 \ 0]\left(\frac{1}{2}\right) \\ &= [0 \ 0 \ -2 \ 1] \end{aligned}$$

$$\begin{aligned} Q\left(3, \frac{1}{2}\right) &= \mathbf{v} + \mathbf{g}\alpha = [0 \ 0 \ -2 \ 1] + [0 \ 0 \ 1 \ 0](3) \\ &= [0 \ 0 \ 1 \ 1] \end{aligned}$$



- Eye point $[0\ 0\ 1\ 0]$

| t | α | $\mathbf{v}(t)$ | $Q(\alpha, t)$ |
|---------------|---------------|---------------------|-------------------------------|
| 0 | 0 | $[-2 \ 0 \ -2 \ 1]$ | $[-2 \ 0 \ -2 \ 1]$ |
| | $\frac{1}{2}$ | | $[-2 \ 0 \ -\frac{3}{2} \ 1]$ |
| | 1 | | $[-2 \ 0 \ -1 \ 1]$ |
| | 2 | | $[-2 \ 0 \ 0 \ 1]$ |
| | 3 | | $[-2 \ 0 \ 1 \ 0]$ |
| $\frac{1}{2}$ | 0 | $[0 \ 0 \ -2 \ 1]$ | $[0 \ 0 \ -2 \ 1]$ |
| | $\frac{1}{2}$ | | $[0 \ 0 \ -\frac{3}{2} \ 1]$ |
| | 1 | | $[0 \ 0 \ -1 \ 1]$ |
| | 2 | | $[0 \ 0 \ 0 \ 1]$ |
| | 3 | | $[0 \ 0 \ 1 \ 0]$ |
| 1 | 0 | $[2 \ 0 \ -2 \ 1]$ | $[2 \ 0 \ -2 \ 1]$ |
| | $\frac{1}{2}$ | | $[2 \ 0 \ -\frac{3}{2} \ 1]$ |
| | 1 | | $[2 \ 0 \ -1 \ 1]$ |
| | 2 | | $[2 \ 0 \ 0 \ 1]$ |
| | 3 | | $[2 \ 0 \ 1 \ 0]$ |

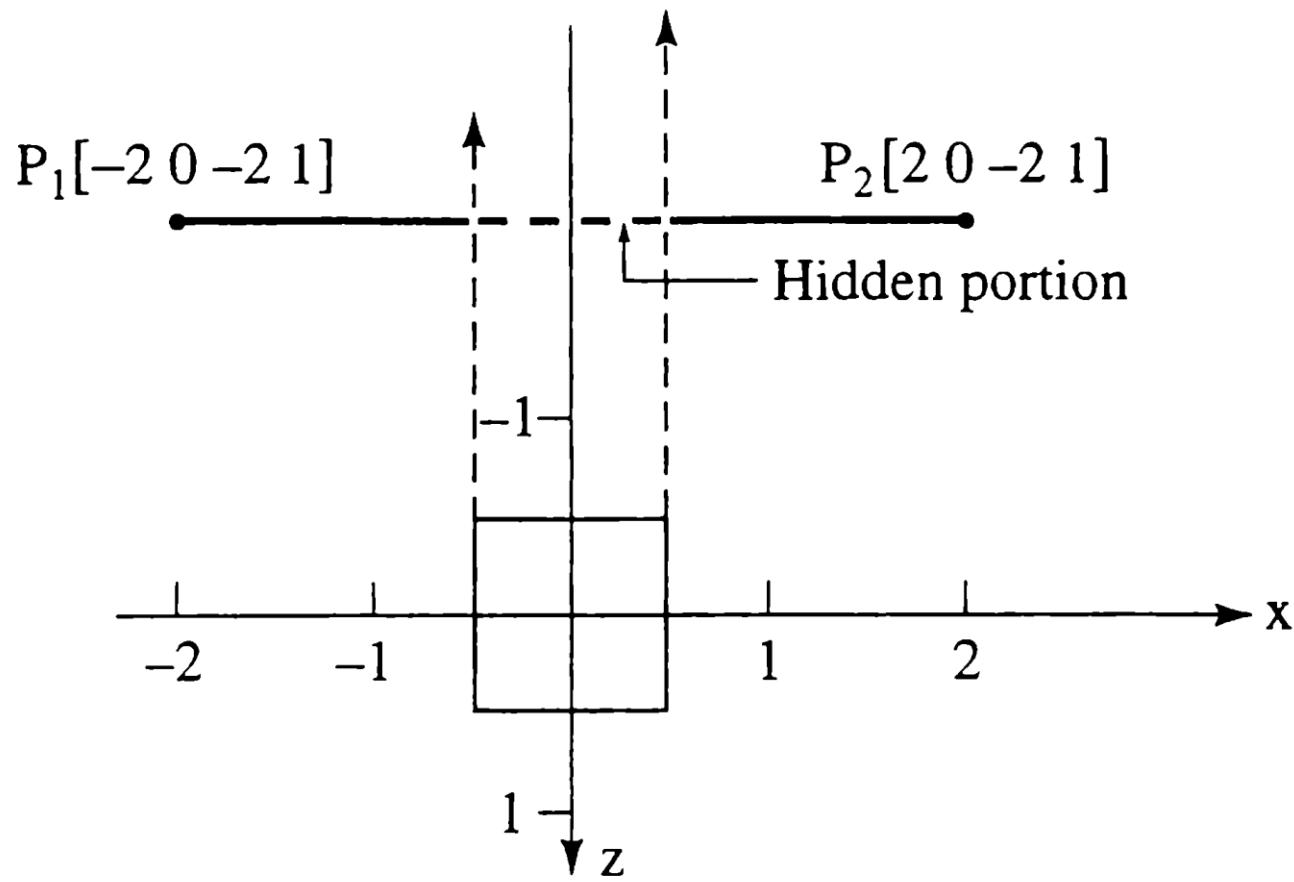
$$h = \mathbf{u} \cdot [VT] = \mathbf{s} \cdot [VT] + t\mathbf{d} \cdot [VT] + \alpha \mathbf{g} \cdot [VT] > 0 \quad \quad 0 \leq t \leq 1, \; \alpha \geq 0$$

$$p = \mathbf{s} \cdot [VT]$$

$$q = \mathbf{d} \cdot [VT]$$

$$w = \mathbf{g} \cdot [VT]$$

$$h_j = p_j + t q_j + \alpha w_j > 0 \quad \quad 0 \leq t \leq 1, \; \alpha \geq 0$$



$$P(t) = \mathbf{v} = [-2 \ 0 \ -2 \ 1] + [4 \ 0 \ 0 \ 0]t$$

$$\mathbf{s} = [-2 \ 0 \ -2 \ 1]$$

$$\mathbf{d} = [4 \ 0 \ 0 \ 0]$$

$$\mathbf{g} = [0 \ 0 \ 1 \ 0]$$

$$[VT] = [V] = \begin{bmatrix} -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$p = \mathbf{s} \cdot [VT] = [5 \ -3 \ 1 \ 1 \ 5 \ -3]$$

$$q = \mathbf{d} \cdot [VT] = [-8 \ 8 \ 0 \ 0 \ 0 \ 0]$$

$$w = \mathbf{g} \cdot [VT] = [0 \ 0 \ 0 \ 0 \ -2 \ 2]$$

$$h_j = p_j + tq_j + \alpha w_j > 0$$

$$\textcircled{1} \quad 5 - 8t > 0$$

$$\textcircled{2} \quad -3 + 8t > 0$$

$$\textcircled{3} \quad 1 > 0$$

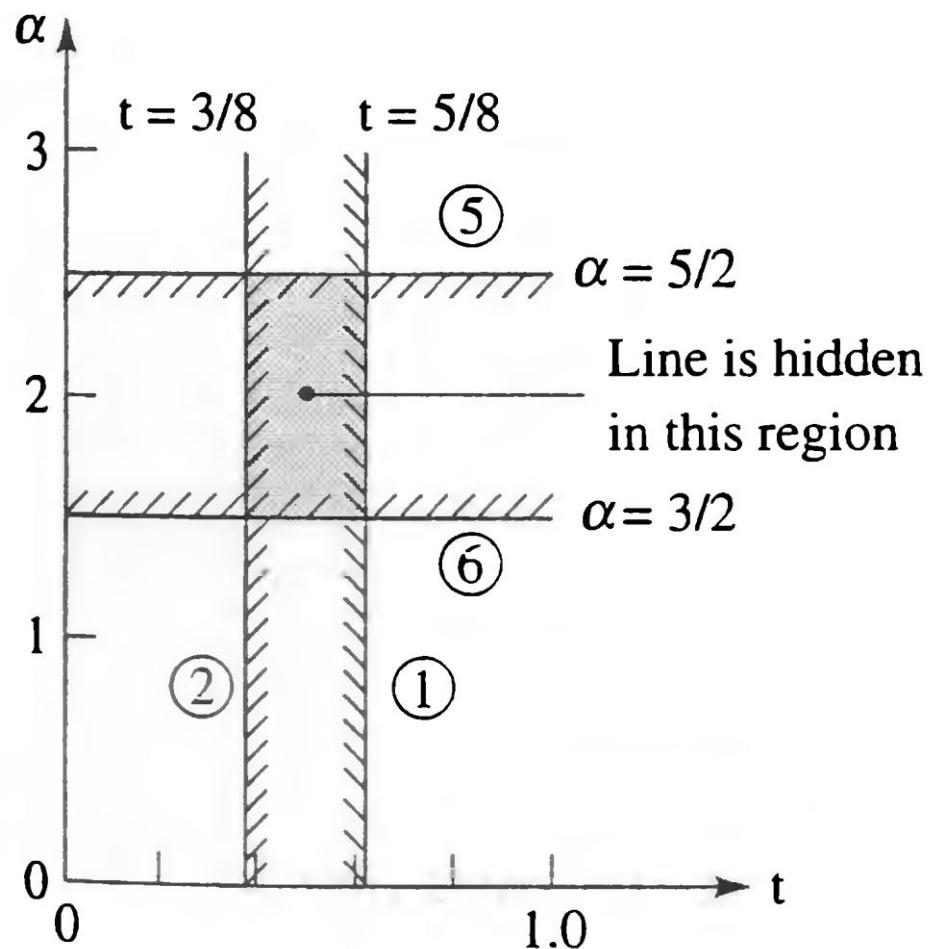
$$\textcircled{4} \quad 1 > 0$$

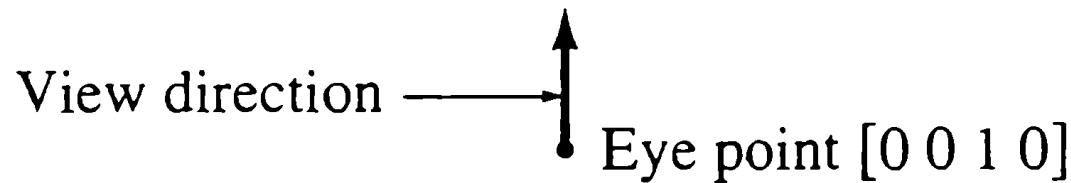
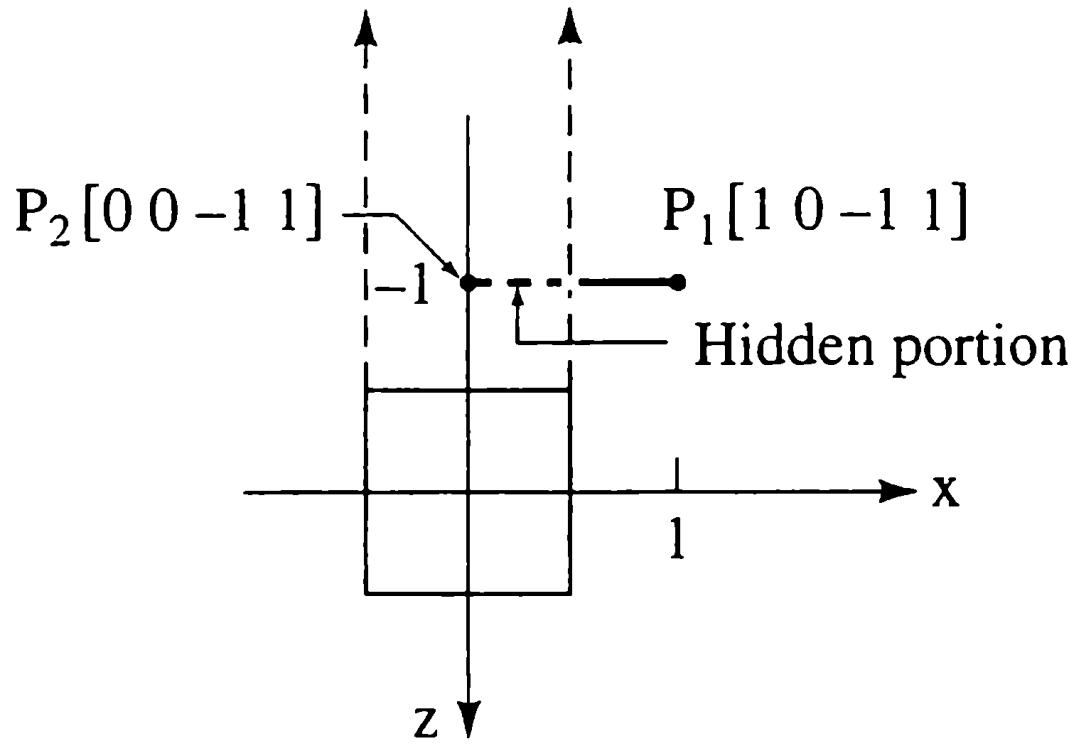
$$\textcircled{5} \quad 5 - 2\alpha > 0$$

$$\textcircled{6} \quad -3 + 2\alpha > 0$$

$$P\left(\frac{3}{8}\right) = [-2 \ 0 \ -2 \ 1] + [4 \ 0 \ 0 \ 1] \left(\frac{3}{8}\right) = \left[-\frac{1}{2} \ 0 \ -2 \ 1\right]$$

$$P\left(\frac{5}{8}\right) = [-2 \ 0 \ -2 \ 1] + [4 \ 0 \ 0 \ 1] \left(\frac{5}{8}\right) = \left[\frac{1}{2} \ 0 \ -2 \ 1\right]$$





$$\mathbf{s} = [\begin{array}{cccc} 1 & 0 & -1 & 1 \end{array}]$$

$$\mathbf{d} = [\begin{array}{cccc} -1 & 0 & 0 & 0 \end{array}]$$

$$\mathbf{g} = [\begin{array}{cccc} 0 & 0 & 1 & 0 \end{array}]$$

$$p = \mathbf{s} \cdot [VT] = [\begin{array}{cccccc} -1 & 3 & 1 & 1 & 3 & -1 \end{array}]$$

$$q = \mathbf{d} \cdot [VT] = [\begin{array}{cccccc} 2 & -2 & 0 & 0 & 0 & 0 \end{array}]$$

$$w = \mathbf{g} \cdot [VT] = [\begin{array}{cccccc} 0 & 0 & 0 & 0 & -2 & 2 \end{array}]$$

$$\textcircled{1} \quad -1 + 2t \quad > 0$$

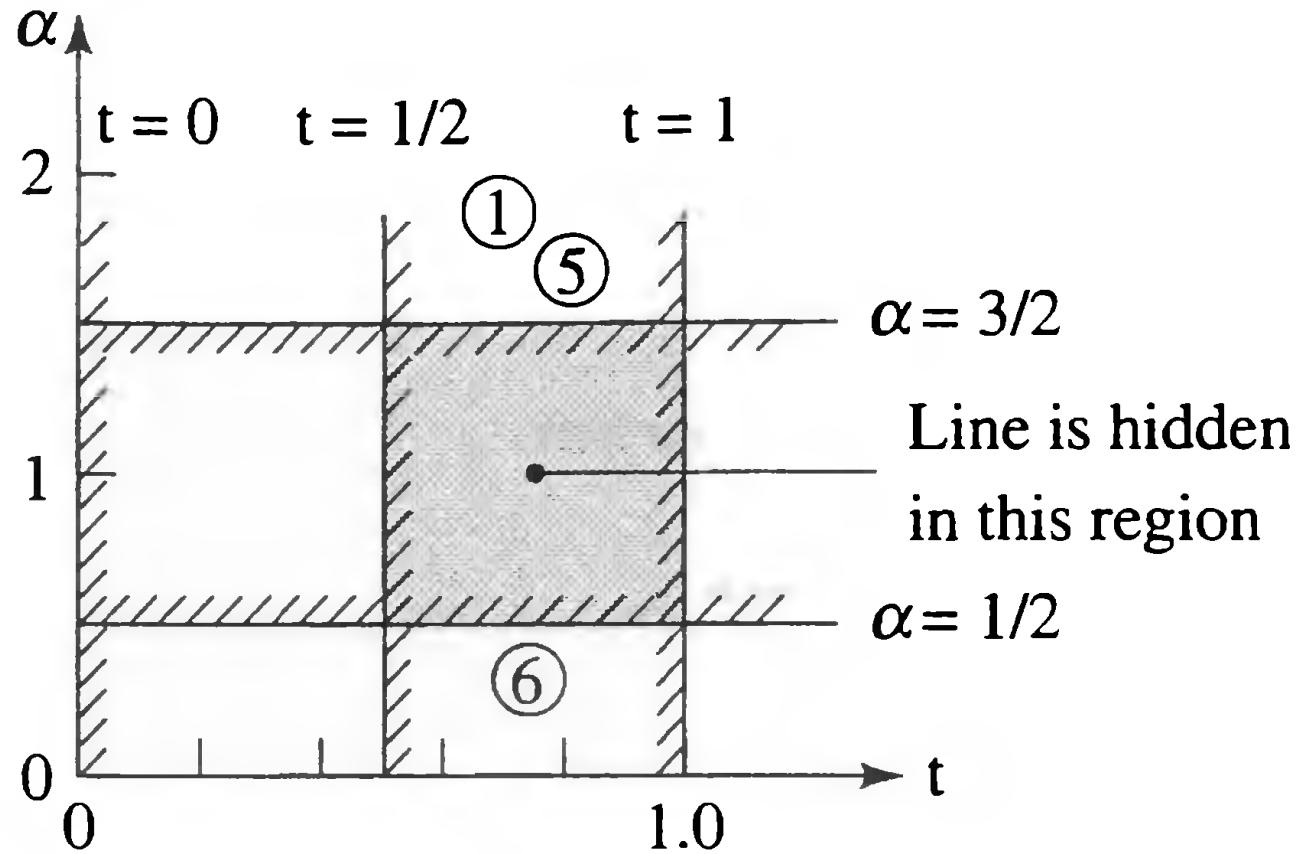
$$\textcircled{2} \quad 3 - 2t \quad > 0$$

$$\textcircled{3} \quad 1 \quad > 0$$

$$\textcircled{4} \quad 1 \quad > 0$$

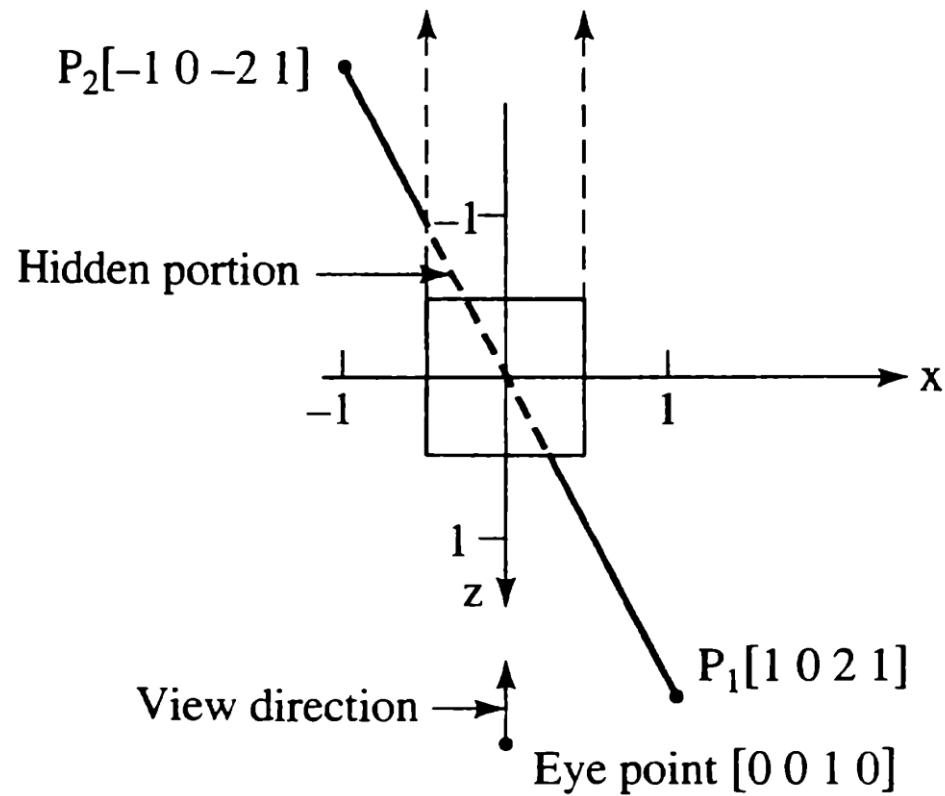
$$\textcircled{5} \quad 3 \quad - 2\alpha \quad > 0$$

$$\textcircled{6} \quad -1 \quad + 2\alpha \quad > 0$$



$$P(0) = [1 \ 0 \ -1 \ 1] + [-1 \ 0 \ 0 \ 0](0) = [1 \ 0 \ -1 \ 1]$$

$$P\left(\frac{1}{2}\right) = [1 \ 0 \ -1 \ 1] + [-1 \ 0 \ 0 \ 0]\left(\frac{1}{2}\right) = \left[\frac{1}{2} \ 0 \ -1 \ 1\right]$$



$$P(t) = \mathbf{v} = [1 \ 0 \ 2 \ 1] + [-2 \ 0 \ -4 \ 0]t$$

$$\mathbf{s} = [1 \ 0 \ 2 \ 1]$$

$$\mathbf{d} = [-2 \ 0 \ -4 \ 0]$$

$$p = \mathbf{s} \cdot [VT] = [-1 \ 3 \ 1 \ 1 \ -3 \ 5]$$

$$q = \mathbf{d} \cdot [VT] = [4 \ -4 \ 0 \ 0 \ 8 \ -8]$$

$$w = \mathbf{g} \cdot [VT] = [0 \ 0 \ 0 \ 0 \ -2 \ 2]$$

$$\textcircled{1} \ -1 + 4t > 0$$

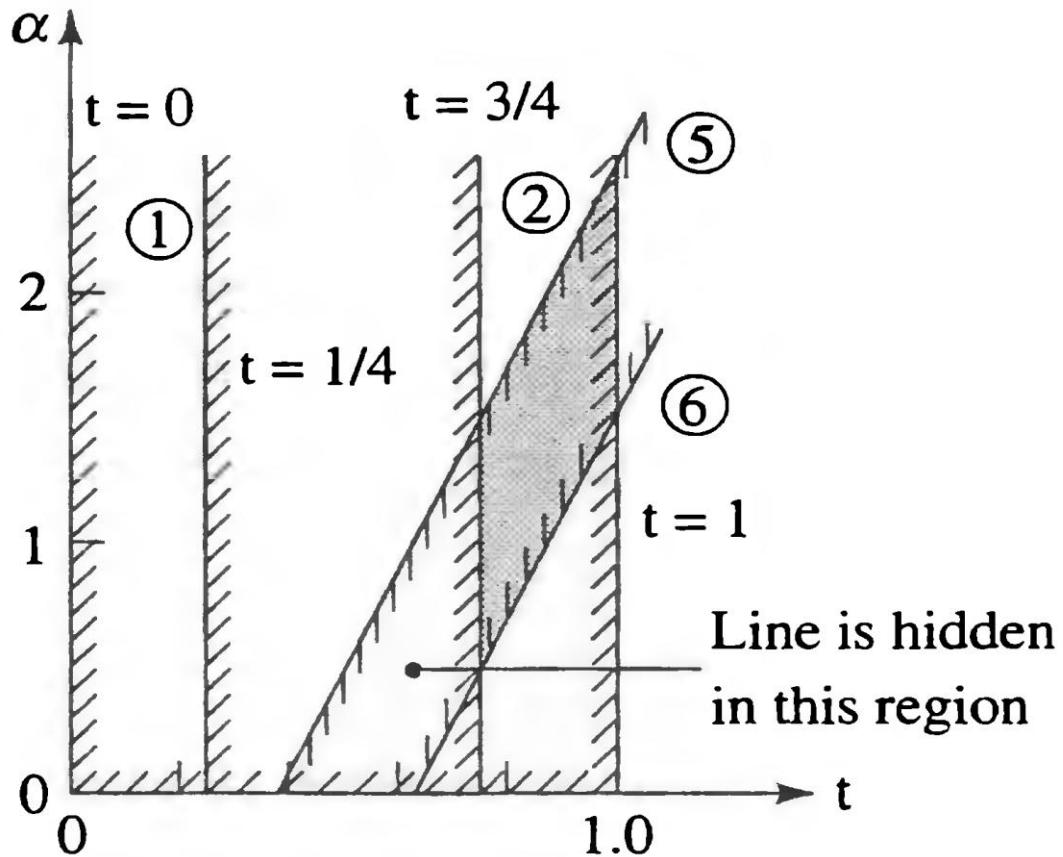
$$\textcircled{2} \quad 3 - 4t > 0$$

$$\textcircled{3} \quad 1 > 0$$

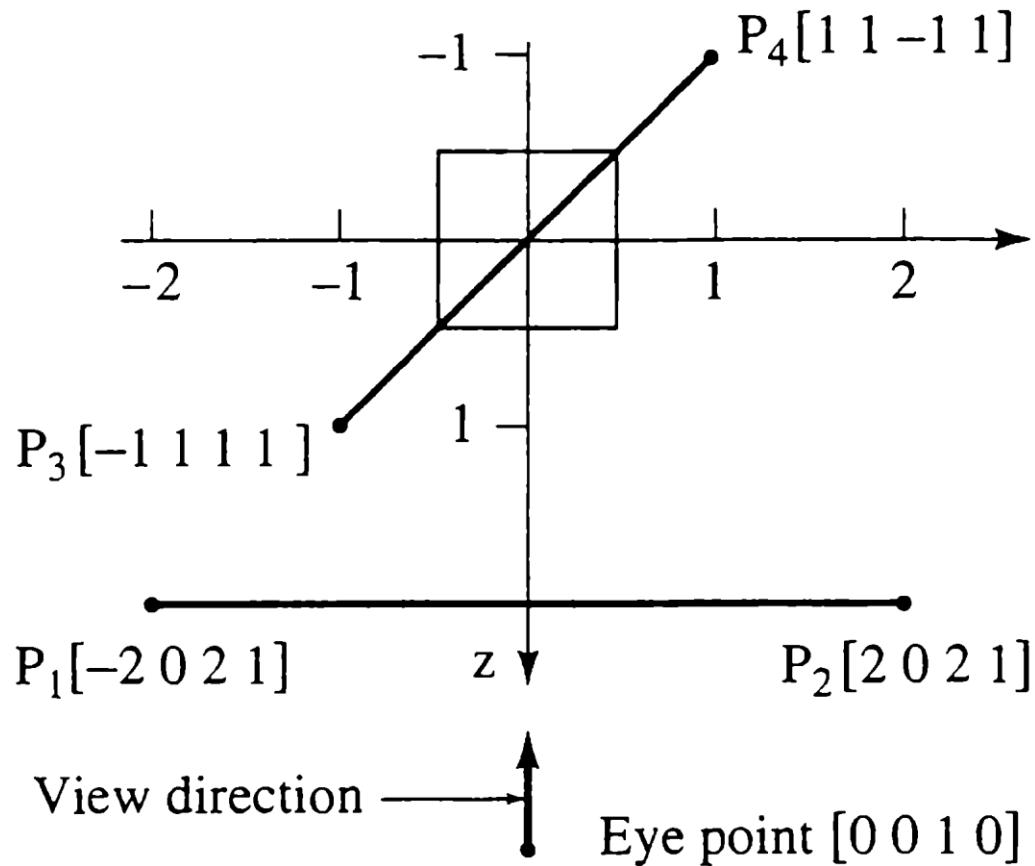
$$\textcircled{4} \quad 1 > 0$$

$$\textcircled{5} \ -3 + 8t - 2\alpha > 0$$

$$\textcircled{6} \quad 5 - 8t + 2\alpha > 0$$



$$0 \leq t \leq \frac{3}{8} \quad \text{and} \quad \frac{3}{4} \leq t \leq 1$$
$$P(0) = [1 \ 0 \ 2 \ 1] \quad \text{to} \quad P\left(\frac{3}{8}\right) = \left[\frac{1}{4} \ 0 \ \frac{1}{2} \ 1\right]$$
$$P\left(\frac{5}{8}\right) = \left[-\frac{1}{4} \ 0 \ -\frac{1}{2} \ 1\right] \quad \text{to} \quad P(1) = [-1 \ 0 \ -2 \ 1]$$



$$\mathbf{s} = [-1 \ 1 \ 1 \ 1]$$

$$\mathbf{d} = [2 \ 0 \ -2 \ 0]$$

$$\mathbf{g} = [0 \ 0 \ 1 \ 0]$$

① ② ③ ④ ⑤ ⑥

$$p = \mathbf{s} \cdot [VT] = [3 \ -1 \ -1 \ 3 \ -1 \ 3]$$

$$q = \mathbf{d} \cdot [VT] = [-4 \ 4 \ 0 \ 0 \ 4 \ -4]$$

$$w = \mathbf{g} \cdot [VT] = [0 \ 0 \ 0 \ 0 \ -2 \ 2]$$

$$w_5 < 0 \quad \text{and} \quad p_5 < 0 \quad \text{but} \quad p_5 + q_5 > 0$$

$$w_3 = 0 \quad \text{and} \quad p_3 < 0 \quad \text{and} \quad p_3 + q_3 < 0$$

Tamamen görünür.

$$\mathbf{s} = [-2 \ 0 \ 2 \ 1]$$

$$\mathbf{d} = [4 \ 0 \ 0 \ 0]$$

$$\mathbf{g} = [0 \ 0 \ 1 \ 0]$$

① ② ③ ④ ⑤ ⑥

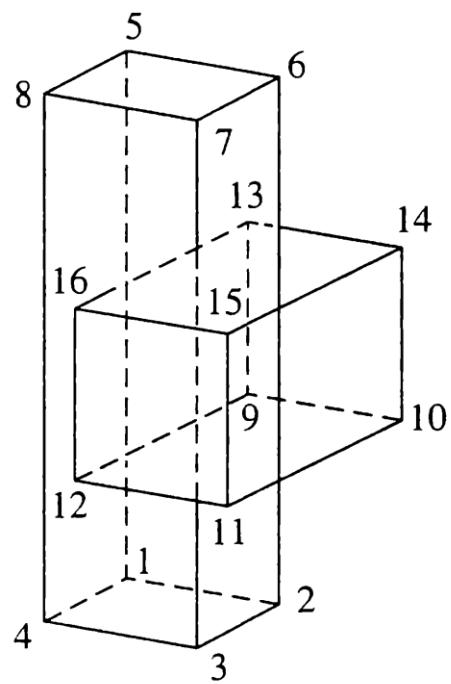
$$p = \mathbf{s} \cdot [VT] = [5 \ -3 \ 1 \ 1 \ -3 \ 5]$$

$$q = \mathbf{d} \cdot [VT] = [-8 \ 8 \ 0 \ 0 \ 0 \ 0]$$

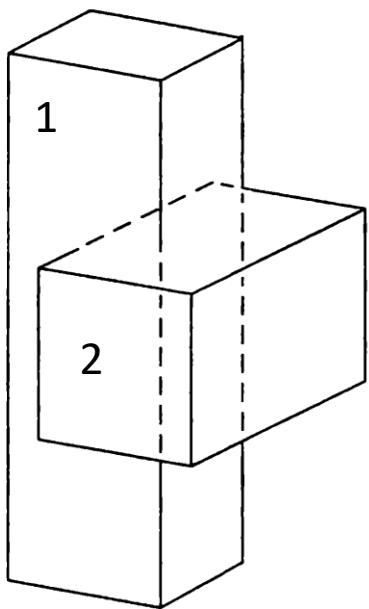
$$w = \mathbf{g} \cdot [VT] = [0 \ 0 \ 0 \ 0 \ -2 \ 2]$$

$$w_5 < 0 \quad \text{and} \quad p_5 < 0 \quad \text{and} \quad p_5 + q_5 < 0$$

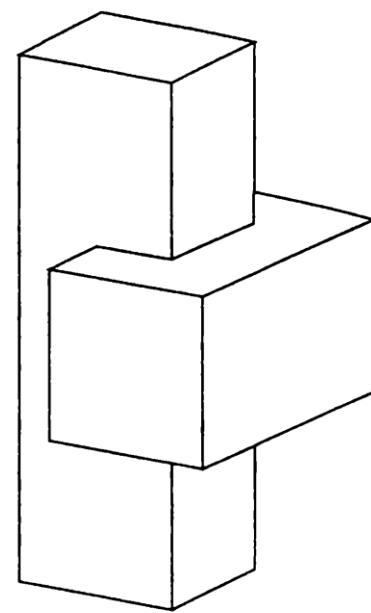
Tamamen görünür.



(a)



(b)



(c)

Nokta matrisleri :

| Block 1 | | | | Block 2 | | | |
|------------------|---|---|---|------------------|---|---|---|
| Vertex number | x | y | z | Vertex number | x | y | z |
| 1 | 0 | 0 | 1 | 9 | 1 | 2 | 0 |
| 2 | 2 | 0 | 1 | 10 | 3 | 2 | 0 |
| 3 | 2 | 0 | 3 | 11 | 3 | 2 | 4 |
| 4 | 0 | 0 | 3 | 12 | 1 | 2 | 4 |
| 5 | 0 | 6 | 1 | 13 | 1 | 4 | 0 |
| 6 | 2 | 6 | 1 | 14 | 3 | 4 | 0 |
| 7 | 2 | 6 | 3 | 15 | 3 | 4 | 4 |
| 8 | 0 | 6 | 3 | 16 | 1 | 4 | 4 |

Kenar Matrisleri :

| Block 1 | | Block 2 | |
|---------|-------------------|---------|-------------------|
| Edge | Joins vertices | Edge | Joins vertices |
| 11 | 1–2 | 13 | 19–10 |
| 12 | 2–3 | 14 | 10–11 |
| 13 | 3–4 | 15 | 11–12 |
| 14 | 4–1 | 16 | 12–19 |
| 15 | 5–6 | 17 | 13–14 |
| 16 | 6–7 | 18 | 14–15 |
| 17 | 7–8 | 19 | 15–16 |
| 18 | 8–5 | 20 | 16–13 |
| 19 | 1–5 | 21 | 19–13 |
| 10 | 2–6 | 22 | 10–14 |
| 11 | 3–7 | 23 | 11–15 |
| 12 | 4–8 | 24 | 12–16 |

Yüzey Matrisleri :

| Block 1 | | Block 2 | |
|-------------------|--------------|-------------------|----------------|
| Polygon number | Edges | Polygon number | Edges |
| 1 | 2, 11, 6, 10 | 7 | 14, 23, 18, 22 |
| 2 | 4, 12, 8, 9 | 8 | 21, 20, 24, 16 |
| 3 | 5, 6, 7, 8 | 9 | 17, 18, 19, 20 |
| 4 | 1, 2, 3, 4 | 10 | 13, 14, 15, 16 |
| 5 | 3, 12, 7, 11 | 11 | 15, 24, 19, 23 |
| 6 | 1, 10, 5, 9 | 12 | 13, 22, 17, 21 |

2.1 OpenGL nedir ?

OpenGL dediğimiz şey, grafiksel donanıma arayüz olan bir yazılımdır. Bu arayüz, şimdiki araştırmalarına göre yaklaşık olarak 250 tane birbirinden farklı olan OpenGL komutları içerir (bu komutların yaklaşık 200 tanesi OpenGL'e has olan ve kalan 50 tanesi ise OpenGL Utility Library dediğimiz kütüphaneye aittir.) Bu komutları kullanarak objeleri yaratabilir ve interaktif bir şekilde üç boyutlu uygulamaları sağlayan işlemleri gerçekleştirebilirsiniz.

OpenGL, faklı donanım özelliklerine sahip platformlarda çalışabilen bir arayüz olarak tasarlanan, yani donanımdan bağımsız bir araçtır. OpenGL'in bu niteliğini gerçekleştirebilmek için OpenGL'de kullanıcının dışarıdan bilgi girmesini sağlayan araçlar ve işletim sistemlerinde pencere yönetimini sağlayan komutlar yoktur. Bu yüzden programcı bulunduğu platformu iyi tanımalı ve kullandığı işletim sistemi komutlarını iyi bilmelidir. Aynı şey üç boyutlu nesneler için de geçerlidir. Yani söylemek istedigim şey, OpenGL'in üç boyutlu nesneler için sağlayacağı yüksek seviyeli komutları yoktur. Sizin yapmanız gereken şey OpenGL'in sağladığı ilkel (primitive) geometrik nesneleri kullanarak istediğiniz modelde iki veya üç boyutlu nesneleri yaratmanızdır. İlkel nesneler olarak bahsettiğim şeyler nokta, çizgi, poligon gibi objelerdir. Bu özellikleri sağlayan gelişmiş bir kütüphane OpenGL'in üstüne inşa edilmiştir. OpenGL Utility Library (GLU) quadric yüzeyleri, NURBS eğrileri ve yüzeyleri gibi bir çok özgüllüğü sağlamaktadır. GLU, her OpenGL implementasyonunun bir gerçekleştirim standartıdır.

2.2 Küçük bir OpenGL kodu

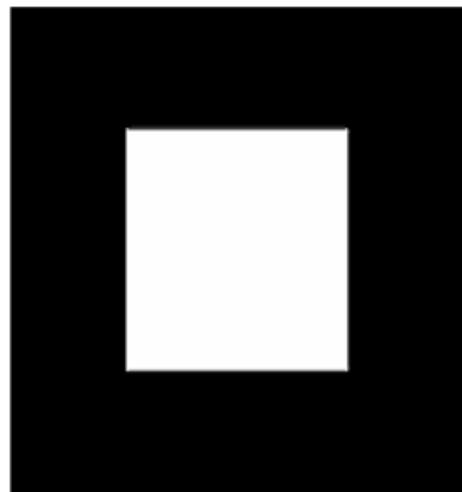
OpenGL grafik sistemi ile bir çok şey yapabildiğinizden dolayı, bir OpenGL programı karmaşık olabilir. Fakat, programın temel yapısı basit olabilir. OpenGL'in nasıl

render işleminde kontrol görevini sağlayan belirli durumları ilklemek ve render edilecek olan nesneleri ise belirlemek OpenGL'in görevleri arasında yer alır.

OpenGL koduna bakmadan önce, birkaç terimi incelemekte fayda var. Mesela biraz önce kullandığım *render* kelimesinin anlamı bilgisayarın modellerden yarattığı resimleri işleme anlamına gelir. Bu modeller ve nesneler geometrik ilkel elemanlardan yaratılırlar (poligonlar, çizgiler ve noktalar).

Sonuç olarak üretilen resim yani render edilmiş olan resim ekranınızda çizilmiş olan pikselleri içerir. Pikselin anlamı, donanım tarafından ekrana koyulan en küçük görünür elementtir. Piksellerle ilgili bilgiler, memorideki bitplaneler içinde düzenlenmiştir. Bitplane bir memori bölgesidir. Bu bölge ekran üzerindeki her bir piksel ile ilgili tek bitlik bilgi tutmaktadır. Örnek olarak, bit herhangi bir pikselin nasıl kırmızı renkte olup olamayacağı ile ilgili bilgiyi tutar. Bitplanelerin kendileri ise bir framebuffer içerisinde organize olmuştur. Bu framebuffer, ekran üzerindeki piksellerin renk kontollerini ve yoğunlukları ile ilgili ihtiyaç duyduğu bilgileri tutmaktadır.

Bir OpenGL kodunu genel yapısını incelersek, Örnek 2-1, arka planı siyah olan beyaz renkli bir karenin render edişini Resim 2-1' de göstermektedir.



Resim 2-1

Örnek 2-1

```
#include <istenile başlık dosyaları>

int main(int argc, char** argv) {

    Pencereİlklemişlemleri();

    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
    glBegin(GL_POLYGON);
        glVertex3f(0.25, 0.25, 0.0);
        glVertex3f(0.75, 0.25, 0.0);
        glVertex3f(0.75, 0.75, 0.0);
        glVertex3f(0.25, 0.75, 0.0);
    glEnd();
    glFlush();

    PencereyiGüncelleVeKontrolleriYap();

    return 0;
}
```

main() fonksiyonun ilk satırı, ekranda gözükecek olan pencere ile ilgili hazırlık işlevleri yapmaktadır. **Pencereİlklemişlemleri()** fonksiyonu, OpenGL'e ait olamayan işletim sistemine has pencere sistemi ile ilgili rutinleri içeren bir yer tutucudur. Sonraki iki satır ise pencerenin arka planın siyah olarak temizleyen OpenGL komutlarıdır: **glClearColor()**, pencerenin hangi renk ile temizleneceğini belirler ve **glClear()** komutu ise pencereyi temizler. **glColor3f()** komutu çizilecek olan nesnenin rengini belirtir. Bu

örnek içerisinde çizilmiş olan karenin rengi beyaz olarak belirlenmiştir. Bu noktadan itibaren tüm nesneler bu komut tarafından yeni bir renk ayarlaması olana kadar bu rengi kullanırlar. Sonraki OpenGL komutu **glOrtho()**'dur. Bu komut pencerede nesne çizilmeden önce, koordinat sistemini belirtir. **glBegin()** ve **glEnd()** komutları tarafından sarılmış olan komutlar, çizilecek nesneyi tanımlamaktadır. Örnek içerisinde göreceğiniz gibi dört köşeli bir poligon çizilmiştir. Poligonu köşeleri, **glVertex3f()** komutları tarafından tanımlanmaktadır. **glVertex3f()** komutu koordinat sistemi ile alakalı üç tane parametre almaktadır. Bu parametreler x, y ve z dir. Bu örnek içerisinde z'nin değeri 0 olarak belirlenmiştir. Son olarak **glFlush()** komutu çizim komutlarını çalıştırır. **PencereyiGüncelleVeKontrolleriYap()** rutini ise pencere ile ilgili güncellemeri yapmaktadır ve diğer processler arasındaki mesaj alış-verişini dinlemektedir.

Aslında örnek 2-1, basit olarak OpenGL komut yapısını vermektedir. Ayrıntılara raporumun sonraki sayfalarında değineceğim.

2.3 OpenGL Komut Sintaksi

Örnek 2-1' de ki basit OpenGL programını incelediyseniz, her OpenGL komutunun önüne **gl** ön eki getirilmektedir. Aynı şekilde, tanımlanmış OpenGL sabitleri de GL ön eki kullanılmıştır (örneğin **GL_COLOR_BUFFER_BIT** gibi...). Dikkat edeceğiniz gibi bazı ekler de bu komutlara birer anlam katmak için kullanılmışlardır. (örneğin **glColor3f()** komutunun son eki olan **3f**). Yaratılmış olan nesnenin o anki rengini belirleyen komutun yani **glColor3f()** komutunun **Color** parçasına bakarak anlamak yetlidir sanırım. Fakat, komutların farklı sayıda argüman almalarını sağlamak için, farklı komutlar tanımlanmıştır. Mesela **glColor3f()** komutunda aklınıza takılan 3 sayısı, bu komutun üç tane argüman aldığıını bildirmektedir. Komutun **f** karakteri, alınacak olan parametrelerin **float** tipinde olmaları gerektiğini anlatmaktadır. Aslında kısacası, OpenGL komutlarını yazarken, fonksiyonların aşırı yüklenmesi söz konusu değildir. Bunu dikkate almakta fayda var.

Bazı OpenGL komutları, sekiz farklı tipte argüman almaktadır. Bir sonraki sayfada tablo, OpenGL' in ISO C gerçekleştirmesi için belirlenmiş veri tiplerini önek olarak tanımlamaları ile beraber kullanımını göstermektedir. Bu veri tiplerini programlama dilinde kullanmakta fayda var, çünkü kullanılacak olan komutların argüman tipleri, programla dilinde kullanılan veri tipleri ile uyuşmayabilir. Bu yüzden, OpenGL komutlarının taşınabilirliği söz konusu ise bu komutları kullanmakta fayda var.

Tablo 2-1 Komut Önekleri ve Argüman Veri Tipleri

| Öne k Veri Tipi | C programlama dilinde | OpenGL Tip Tanımı |
|--------------------------------|----------------------------------|---|
| Kullanılan Tipler | | |
| B | 8-bit integer | signed char |
| S | 16-bit integer | Short |
| İ | 32-bit integer | int veya long |
| F | 32-bit floating-point | Float |
| D | 64-bit floating point | Double |
| Ub | 8-bit unsigned integer | unsigned char |
| Us | 16-bit unsigned integer | unsigned short |
| Ui | 32-bit unsigned integer | unsigned int veya unsigned long |
| | | GLbyte GLshort GLint, GLsizei GLfloat, GLclampf GLdouble, GLclampf GLubyte, GLboolean GLushort GLuint, GLenum, GLbitfield |

Aşağıdaki iki komutu incelersek

```
glVertex2i(1, 3);
glVertex2f(1.0, 3.0);
```

her iki komut ta aynı işlev sahiptir. Fakat ilki, vertex'in koordinatları argüman olarak 32-bit integerlar ve diğe komut ise argüman olarak float tipindedir.

Bazı OpenGL komutları, son ek olarak **v** karakterini alır. Bu karakter, programcıya parametre olarak alınacak şeyin bir vectörü veya bir diziyi işaret eden bir işaretçi olduğunu bildirir. Birçok komutun parametre olarak vector alan veya almayan versiyonları bulunmaktadır. Aşağıdaki satırlar **glColor***() komutunun vektörlü ve vektörsüz olarak parametre alımını göstermektedir.

```
glColor3f(1.0, 0.0, 0.0);  
  
GLfloat color_array[] = {1.0, 0.0, 0.0};  
glColor3fv(color_array);
```

Son olarak, OpenGL veri tipi olarak GLvoid'i tanımlar. Bu veri tipi dizileri işaret eden pointerları, argüman olarak alan OpenGL komutları için kullanılmaktadır.

2.4 GLUT, OpenGL Hizmet Aracı (OpenGL Utility ToolKit)

Bildiğiniz gibi, render komutlarını OpenGL içermektedir, fakat bir pencere sisteminde ve bir işletim sisteminden bağımsız olarak tasarlanmıştır. Sonuç olarak ne açılan pencerelelere ait, ne de klavye ve mause olaylarına ait komutlar içermemektedir. Penceresiz bir grafik programı yazmak imkansızdır ve çok ilgi çeken programlar dışarıdan kullanıcı bilgisi alırlar veya işletim sisteminin ve pencere sisteminin hizmetlerinden yaralanırlar. GLUT kütüphanesini kullanarak, pencere yaratma dışarıdan bilgi alma gibi fonksiyonları kolayca kullanabiliriz. Raporumun ileriki sayfalarında Windows işletim sistemini pencere sistemi gerçekleştirimine degeneceğim.

OpenGL' in ne olduğunu açıklarken, OpenGL'in basit ilkel nesneleri çizebilecek komutlarla sınırlı olduğunu söylemiştim, fakat GLUT sayesinde karmaşık üç boyutlu nesnelerimizi kolaylıkla yaratabiliriz.

2.5 Pencere Yönetimi

Pencere yaratabilmek için gerekli olan beş rutin yerine getirilmelidir.

- **glutInit(int *argc, char** argv)** fonksiyonu GLUT'ı ilkler ve komut satır argümanlarını işlemektedir. Diğer GLUT rutinlerinden önce bu komutun yazılması zorunludur.
- **glutInitDisplayMode(unsigned int mode)** renk modunu belirlemektedir. Bu renk modu RGBA veya color-index mod olabilir. Ayrıca pencerenin single veya double buffered olacağını belirlersiniz. Eyer bir pencerenin RGBA ve double buffered modda olmasını istiyorsanız, bu fonksiyonu **glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE)** olarak çağrırmalısınız.
- **glutInitWindowPosition(int x, int y)** fonksiyonu grafik penceresinin sol-üst köşesini baza alarak, ekrandaki yerini belirler.
- **glutInitWindowSize(int width, int height)**, pencerenizin büyüklüğünü ayarlar. (piksel birimi olarak)
- int **glutCreateWindow(char* string)**, OpenGL kontekstli bir pencere yaratır. Yeni pencereye ait tek yani eşsiz bir satı dönderir. Not: glutMainLoop() çağrırlana kadar pencere yaratılamaz.

2.6 Görüntüyü Geri Çağırma (The Display Callback)

glutDisplayFunc(void (*func)(void)) fonksiyonu göreceğiniz ilk ve en önemli fonksiyon çağrıma fonksiyonudur. Nezaman GLUT, pencerenizin içeriğinin yeniden

gösterileceğine ihtiyaç duyduğunu belirlerse, **glutDisplayFunc()** tarafından çalıştırılacak fonksiyon çağrılrılır. Anlaşılacağı gibi, glutDispalyFunc()' a parametre olarak aktarılan bir pointer to function tipinde bir işaretçidir. Bu nedenden ötürü, pencerenizi yeniden çizmeniz için gerekli gördüğünüz rutinleri çağıracağınız fonksiyon içerisine yani void (*func)(void) poiner to function içerisine koyn.

2.7 Renkli bir Üçken

Yapacağınız en son şey ise **glutMainLoop(void)** funksyonun çağırılmak olacaktır. Bunu sayesinde yarattığınız pencere ekranada gösterilir. Bu fonksiyon GLUT olay işlem döngüsüne (the GLUT even processing loop) girer. Program içerisinde sadce bir kez çalıştırılmalıdır. Fonksiyon bir kere çağrıldığı zaman, fonksiyondan geri dönülmez. Kayıt edilen fonksiyonları çalıştırırmak için bekleyecektir.

Aşağıdaki örnek 2-2 GLUT tarafından yaratılan bir pencerede renkli bir üçken göstermektedir. Programın verimliliğini artırmak için, sadece bir kez çağrılmaması gereken fonksiyonlar, **init()** fonksiyonu içerisinde bulunmaktadır. Diğer render işlemleri ise **display()** fonksiyonu içerisinde konulmuştur (üçkeni çizen fonksiyon) ve bu fonksiyonun ismi parametre olarak **glutDispalyFunc()** fonksiyonuna aktarılacaktır.

Örnek 2-2 GLUT kullanılarak yapılan basit bir OpenGL programı

```
#include <GL/glut.h>

void display(void) {
    /* tum pikselleri temizle */
    glClear(GL_COLOR_BUFFER_BIT);
    /* koordinatlari (0.25, 0.25),
     * (0.75, 0.25), (0.50, 0.75) olan ucken ciz
     */
    glBegin(GL_POLYGON);
```

```

        glColor3f(1.0, 0.0, 0.0);
        glVertex2f(0.25, 0.25);
        glColor3f(0.0, 1.0, 0.0);
        glVertex2f(0.75, 0.25);
        glColor3f(0.0, 0.0, 1.0);
        glVertex2f(0.50, 0.75);

    glEnd();
    glFlush();
    return;
} // end of void display(void)

void init(void) {

/* arka plan rengini belirle */
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_SMOOTH);

/* bakis degerlerini ilkle */
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    return;
} // end of void init(void)

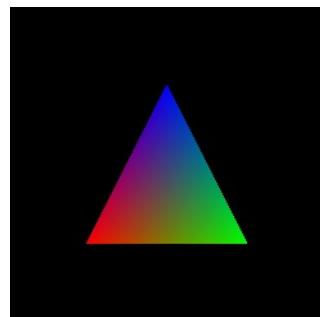
int main(int argc, char* argv[]) {

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(250, 250);
    glutInitWindowPosition(100, 100);
    glutCreateWindow(argv[0]);
    init();
    glutDisplayFunc(display);
    glutMainLoop();
}

```

```
    return 0;  
} // end int main(int, char**)
```

Programın ekrandaki çıktısı aşağıdaki gibi olacaktır.



Resim 2-2

2.8 Dışarıdan Input Girişini Tutmak

Belli bir olay olduğunda, bu olayları tutabilmek için GLUT, bize aşağıdaki fonksiyonları sağlamıştır.

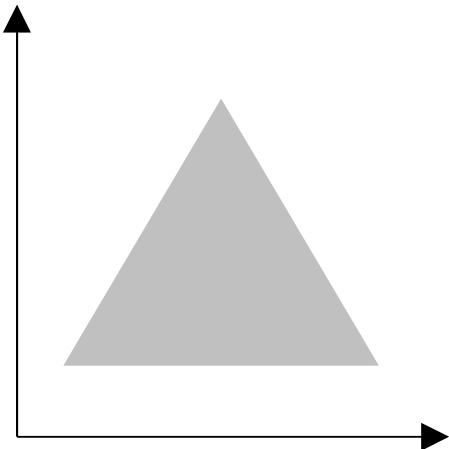
- **glutReshapeFunc**(void (**func*)(int *w*, int *h*)) fonksiyonu pencerenin büyütüleceği zaman çalışır.
- **glutKeyboardFunc**(void (**func*)(unsigned char *key*, int *x*, int *y*)) ve **glutMouseFunc**(void (**func*)(int *button*, int *state*, int *x*, int *y*)) fonksiyonları, sizin mouse veya klavye ile ilgili işlemlerinizi tutar. Klavyenin veya farenin herhangi bir tuşuna bastığınızda veya basık tuttuğunuzda bu olaylar meydana gelir.
- **glutMotionFunc**(void (**func*)(int *x*, int *y*)) fonksiyonu, fare haraket etmesi veya farenin düğmelerine basılması gibi olayları tutar.

- **glutIdleFunc(void (*func)(void))**, eyer arka planda bir olay olmuyorsa yani dışarıdan bir bilgi girişi yok ise, kendisine parametre olarak aktarılan fonksiyon ismini çalıştırır. Eyer **glutIdleFunc()** fonksiyonun inaktif yapmak istiyorsanız fonksiyona NULL değerini aktarın.

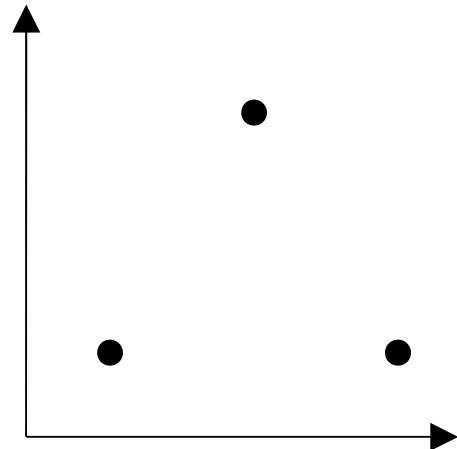
2.9 OpenGL İlkel Geometrik Nesneleri (OpenGL Primitives)

İlkel geometrik nesnelerden anlatmak istediğimiz şey, OpenGL’ın çizebildiği basit olan ilkel noktalar, çizgiler, poligonlar ve üçkenler gibi nesnelerdir. Bu nesnelerin birer koordinat bilgileri vardır. Bu koordinat bilgilerine vertex demekte fayda görüyorum. OpenGL bu nesnelerin vertex bilgileri sayesinde bu ilkel olan geometrik şekilleri çizebilmektedir. Fakat çizilecek olan nesnenin nokta, çizgi veya poligon olup olmadığını OpenGL’e bildirmek gereklidir. Bu bildirim **glBegin(Glenum mode)** fonksiyonu tarafından gerçekleştirilir. Öncelikle nesnenin koordinat bilgileri OpenGL’ e aktarılmadan öncece **glBegin()** fonksiyonu çağrılp çizim modu belirlenir. Ardından vertex bilgileri aktarılıp nesnenin çizimini ve çizme modunun bittiğini göstermek için **glEnd()** fonksiyonu kullanılır. Aşağıdaki **glBegin()** ve **glEnd()**’in kullanımını göstermektedir.

```
glBegin(GL_POLYGON);
    glVertex2f(0.25, 0.25);
    glVertex2f(0.75, 0.25);
    glVertex2f(0.50, 0.75);
glEnd();
```



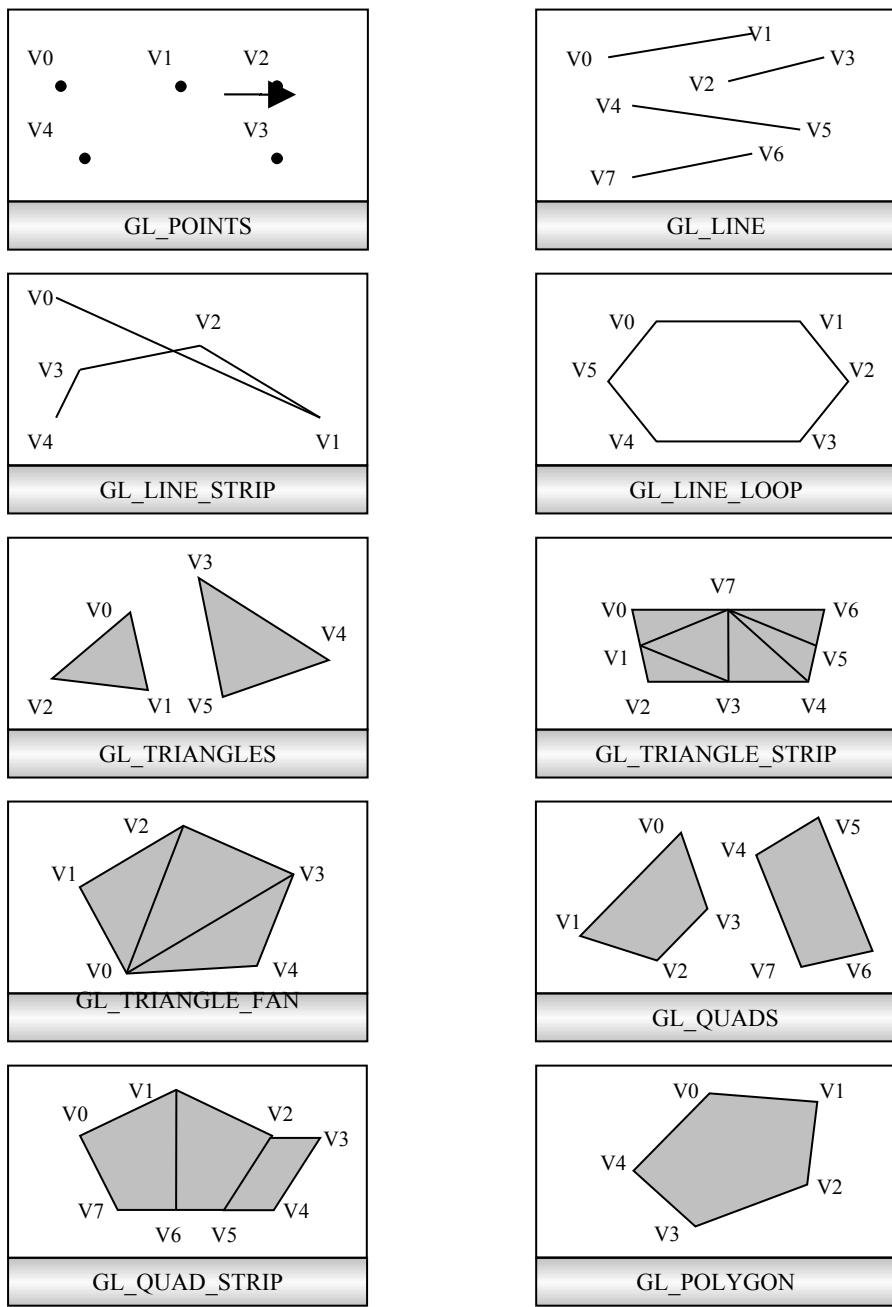
GL_POLYGON



GL_POINTS

Resim 2-3

Önceki sayfada gördüğünüz gibi, OpenGL'in çizeceği mode **glBegin** tarafından poligon olarak belirtilmiştir. Bunun için **glBegin()** fonksiyonuna parametre olarak aktarılan parametre GL_POLYGON dur. sayfada gördüğünüz resim 2-3' ün sol tarafında poligon modu geçerli olarak üçken çizilmiştir. Eğer **glBegin()**'e aktarılacak olan mod GL_POINTS olsaydı, aynı resmin sağında bulunan üç tane nokta gözükecekti. Resim 2-4, çizim modlarını göstermektedir.



Resim 2-4

Örnekte, gördüğünüz gibi, **glEnd()** fonksiyonunun parametresi yoktur. OpenGL'e belirttiğimiz modda, çizdiğimiz ilkel geometrik nesnenin çizimini birtirmesi gerektiğini haber vermekten başka, bir özel işlevi yoktur.

Sunu belirtmek gerekirse, **glBegin()** / **glEnd()** bloğu içerisinde, başka bir **glBegin()** / **glEnd()** bloğu bulunamaz. Ayrıca, tüm OpenGL fonksiyonları, bu bloğun içerisinde kullanılamayabilirler. Gerçekte, sadece **glVertex()**, **glColor()**, **glIndex()**, **glNormal()**, **glTexCoord()**, **glEvalCoord()**, **glEvalPoint()**, **glMaterial()**, **glEdgeFlag()**, **glCallList()** ve **glCallLists()** fonksiyonlarının çeşitli varyasyonları **glBegin()** / **glEnd()** bloğu içerisinde kullanılabilirler.

2.10 Vertexleri Belirleme

OpenGL ile her geometrik nesne düzgün bir sıraya sahip olan koordinat bilgileri ile ifade edilebilirler. Nesnenin bir koordinat bilgisini yani vertex'ini belirlemek için **glVertex*()** komutunu kullanmamız gereklidir.

```
void glVertex[2, 3, 4][s, i, f, d][v]( Type coords );
```

Bu fonksiyon, nesnenin özel bir koordinat bilgisine göre en fazla dört parametre (parametre bilgisi olarak (x, y, z ,w) olabilir) ve en az iki parametre parametre (parametre bilgisi olarak (x, y)) alabilir. Eyer z veya w gibi parametreleri kullanmadığınız bir **glVertex*()** fonksiyonu kullanıyorsanız, z'nin değeri 0 ve w'nun değeri 1 olarak varsayılacaktır. **glVertex*()** fonksiyon çağrıları sadece **glBegin()** / **glEnd()** bloğu içerisinde verimlidir.

Aşağıdaki örnek **glVertex*()** kullanımını basitçe göstermektedir.

```
glVertex2s(2, 3);
glVertex3d(0.0, 0.0, 3.1415926535898);
glVertex4f(2.3, 1.0, -2.2, 2.0);

Gldouble dVect[] = {5.0, 9.0, 1992.0};
glVertex3dv(dVect);
```

İlk satırdaki komut, üç boyutlu koordinat sistemine göre koordinat bilgisi (2, 3, 0) olan bir vertex'i göstermektedir. Bildiğiniz gibi bu komut, sadece iki parametre almaktadır. Yani koordinat bilgileri olarak $x = 2$, $y = 3$ tür. Kullanmadığımız z 'nin değeri 0 ve w 'nun değeri 1 dir. İkinci komut koordinat bilgileri (0.0, 0.0, 3.1415926535898) değerlerine sahip olan bir vertex'i gösterir. Üçüncü satırdaki komut ise parametre olarak (2.3, 1.0, -2.2, 2.0) alır. Dikkat ederseniz w 'nun değeri 2.0 dir ve bu yüzden demin belirttiğim parametre bilgilerini yeniden incelersek w değeri x , y ve z değerlerini bölecektir. Yani parametre aktarım (x/w , y/w , z/w) olarak aktarılır. Sonuç olarak OpenGL'in gördüğü koordinat bilgiler (2.3, 1.0, -2.2) değilde (1.15, 0.5, -1.1) olacaktır.

Son satırda ise, fonksiyona bir dizi parametre olarak atanmıştır.

Bazı donanımlarda **glVertex*()** fonksiyonunun vektör formlarını kullanmak daha etkilidir çünkü sadece tek bir parametre, grafik alt sistemine geçirilmeye ihtiyaç duyar. Özel bir donanım, tek bir seri halinde, tüm koordinat bilgilerini gönderme yeteneğine sahip olabilir. Bu da size iyi bir performans sağlar.

2.11 Nokta Büyüklüğünü Değiştirme

Nokta büyülüüğünü değiştirmek için aşağıdaki fonksiyonu kullanmak gerekir.

```
void glPointSize( GLfloat size );
```

Varsayılı olan nokta büyülüüğünün değeri yani size 1.0'a eşittir. Farklı nokta büyülükleri için ekrana çizilecek olan piksel toplulukları antialiasing durumunun aktif veya pasif olmasına bağlıdır.(Antialiasing çizilecek olan nokta veya çizgileri pürüssüz bir şekilde çizmek için kullanılacak olan bir tekniktir.) Eğer antialiasing aktif değil ise ve verilen büyülüük değerleri virgülü bir sayı ise bu sayı, tam sayıya dönüştürülür ve ekranda çizilecek olan noktalar kare şeklinde çizilirler. Mesela büyülüüğü 1.0 olan bir nokta, 1'e 1 genişliğinde bir piksel olarak çizilir. Eğer büyülüüğü 2.0 olan bir nokta enine ve boyuna 2 piksel olarak kare şeklinde çizilecektir

Antialiasing'in aktif olduğu durumlarda, çizilecek plan noktaların pikselleri dairesel bir şekilde çizilir buda nesnenin keskin köşelerinde yumuşak geçişler sağlayacaktır yani çizilecek olan nesne pürüssüz olacaktır. Bu modda büyülüğu tamsayı olmayan bir sayı yuvarlanmayaçaktır. Programın çalışması esnasında seçilmiş olan nokta büyülüğünü **glGet()** fonksiyonuyla eldedersiniz. Bu fonsiyon **GL_PONIT_SIZE** parametresini ak-

tardığınızda noktanın büyülüğü bulunur.

2.12 Çizgi Kalınlığını Değiştirme

Aşağıdaki fonksiyonu kullanarak bir çizginin kalınlığını belirleyebilirsınız.

```
void glLineWidth( GLfloat width );
```

width yani genişlik 0.0 dan büyük olmalıdır ve varsayılı olan değeri 1.0 dır. Noktaların çiziminde antialiasing etkili olduğu gibi aynı şey çizgiler için de geçerlidir. Antialiasing olmadan, çizilecek olan çizgilerin kalınlıkları, sırasıyla 1, 2 ve 3 ise, piksel genişlikleri, yine aynı sırada, 1, 2 ve üç piksel genişliğinde olacaktır. Önemle belirtmem gerekirse, ekranınız düşük çözünürlükte çalışıyorsa piksel genişliği 1.0 olan çizgiler geniş olarak gözükecektir eğer çözünürlük yüksek ise çizilecek olan çizgi gözükmeme yakını olabilir.

2.13 Pencere Temizleme

Bir resmi bilgisayar ekranına çizmek ile kağıda çizmek arasında çok fark vardır. Bilgisayarın belleğinde son çizdiğiniz resim tutulur. Yeni bir resim çizmeden önce arka planı temizlemeye ihtiyaç duyarsınız, tabi bunu yapmak için öncelikle arka planı temizleyecek renge sahip olamanız gereklidir. Kullandığınız renk gerçekleştireceğiniz uygulamaya bağlıdır.

Bilmemiz gereken şey, bitplane olarak biline grafiksel donanımda her pikselin renjinin nasıl tutulduğudur. Renklerin tutulmasına dair iki yöntem mevcut. Birinci yöntem, bir pikselin kırmızı, yeşil, mavi ve alfa değerlerinin direkt olarak bitplane içerisinde konunması ve diğer bir yöntem ise renk index modudur.

Aşağıdaki komut satırları RGBA modunda pencereyi siyaha temizlemektedir.

```
glClearColor(0.0, 0.0, 0.0, 0.0);  
glClear(GL_COLOR_BUFFER_BIT);
```

İlk satırdaki komut temizleme rengini siyah yapmaktadır ve ardından gelen komut ise seçilmiş olan temizleme rengini dikkate alarak pencereyi temizler. **glClear()** komutuna aktarılmış olan tek parametre hangi bufferin temizleneceğini OpenGL'e bildirmektedir. Bu durumda resmin saklı tutulduğu renk bufferi program tarafından temizlenecektir.

Aşağıda **glClear()** ile **glClearColor()** fonksiyonlarının prototipleri ve komutların işlevleri açıklanmaktadır.

- void **glClear(Glbitfield mask)** fonksiyonu mask olarak belirlmiş parametreye göre bufferları temizler.
- void **glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha)** fonksiyonu RGBA modda temizlemek için kullanılacak olan rengi belirler. Varsayılan temizleme değerleri (0.0, 0.0, 0.0, 0.0) olduğundan dolayı siyahdır.

2.14 Nesnenin Rengini Belirleme

OpenGL ile, çizilecek olan bir nesnenin şeklini ifade etmek, nesnenin rengini ifade etmekten bağımsızdır. Nezaman bir nesne çizileceğinde, o anki renk şeması kullanılarak nesne çizilir. Genel olarak OpenGL programcısının yapacağı ilk iş rengi veya renk şemasını ayarlamak ve sonra ise nesneyi çizmektir. O anki renk veya renk şeması değişmeyinceye kadar tüm nesneler bu rengi veya renk şemasını kullanarak çizilirler.

Aşağıdaki örneği incelerseniz, çizilecek olan nesnelerin yalancı kodunu göreceksiniz.

```
SetColor(RED);  
DrawObject(A);  
DrawObject(B);  
SetColor(GREEN);  
SetColor(BLUE);  
DrawObject(C);
```

Yukarıdaki örnekte A ve B nesneleri kırmızı olarak ve C nesnesi ise mavi olarak çizilecektir. Dördüncü satırda SetCurrentColor(GREEN) komutu gereksiz yere kullanılmıştır.

Bir rengi belirlemek için, **glColor3f()** komutunu kullanın. Bu fonksiyon 0.0 ile 1.0 arasında üç tane argüman almaktadır. Parametreleri sırasıyla red, green ve blue dur. Bu parametrelerin değerlerinin karışımından yeni renkler yaratılabilirsiniz. 0.0 değeri bileşeni yani rengi kullanmayacağımız anlamına gelmektedir. 1.0 ise tam tersidir. Mesela **glColor3f(1.0, 0.0, 0.0);** fonksiyonunu kullandığımızda sistemin bir nesneyi parlak kırmızı bir renkte çizeceği anlamına gelmektedir yeşil ve mavi renkler kullanılmaz. Tüm sıfırlar rengi siyah, tüm birler ise rengi beyaz yapar. Tüm renk değerle-

rini 0.5 olarak ayarlırsak rengimiz griye kaçacaktır. Bir sonraki sayfada sekiz tane renk ayarlaması gösterilmektedir.

```
glColor3f(0.0, 0.0, 0.0);      siyah
glColor3f(1.0, 0.0, 0.0);      kırmızı
glColor3f(0.0, 1.0, 0.0);      yeşil
glColor3f(1.0, 1.0, 0.0);      sarı
glColor3f(0.0, 0.0, 1.0);      mavi
glColor3f(1.0, 0.0, 1.0);      magenta
glColor3f(0.0, 1.0, 1.0);      çiyan
glColor3f(1.0, 1.0, 1.0);      beyaz
```

2.15 Koordinat Çevrimleri

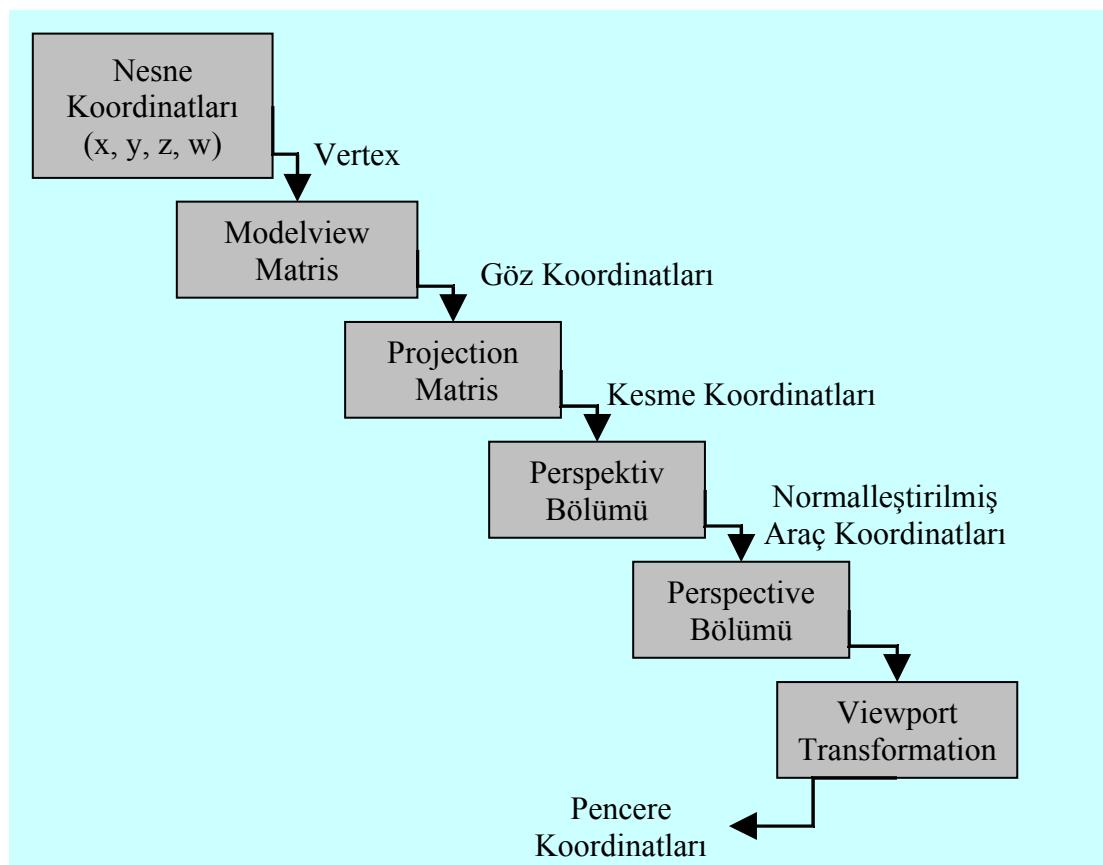
Koordinat çevrimleri üç boyutlu bir ortamda varlıklarını gerçekleştirmeye, döndürme ve hareket ettirmeye gibi olanaklar sağlar. Bu çevrimler, nesneleri ve nesnelerin koordinat sistemlerini direkt olarak modifiye eder.(degistirir) Mesela, modelinizin koordinat sisteminde bir döndürme işlemi yaptığınızda, modelinizi çizeceğiniz zaman, o da değiştirmiş olduğunuz koordinat eksenine göre dönecektir.

Nesneler çizilmeden önce, üç tip transformasyondan yani dönüşümden geçer.

- **Viewing transformation:** Kameranın yerinin belirlendiği transformasyondur.
- **Modeling transformation:** Nesnenin ekranada hareketini sağlayan transformasyondur.
- **Projection transformation:** Kameranın bakış hacmini belirleyen transformasyondur.

Viewing transformation, yazdığınız kod içerisinde modelling transformation'dan önce gelmelidir fakat çizim yapmadan önce projection ve viewport transformationları yani dönüşümlerini belirleyebilirsiniz.

Aşağıdaki şekil resim 2-5 bilgisayarınızda hangi işlemlerin sırayla olduğunu göstermektedir.



Resim 2-5 Vertex Dönüşümünün Safhaları

Viewing, modeling ve projection transformationlarını belirlemek için, 4X4 lük bir **M** matris tanımlaması yapmanız gereklidir. Bu matris, istenilen transformasyonları gerçekleştirebilmek için ekrandaki her bir vertex'in koordinatlarıyla çarpılır.

$$v' = \mathbf{M}v$$

(Not: vertexlerin dört tane koordinatı olduğunu önemle vurgulamam gereklidir. Fakat bir çok durumda w , 1'e eşit ve iki boyutlu koordinat sistemi için z değeri 0'a eşittir.)Viewing ve modeling transformasyonları otomatik olarak yüzey normal vektörlerine ve ek olarak vertexlere uygulanmaktadır.(Normal vektörleri sadece göz koordinatlarında kullanılır.Bu, normal vektörünün vertex verisi ile olan ilişkisini uygun bir şekilde sağladığın temin etmektedir.

Belirttiğiniz viewing ve modelling transformasyonları, modelview matrisini oluşturmak için bir araya getirilirler. Göz koordinatlarını vermek için, gelen nesnenin koordinat bilgilerine bu matris uygulanmaktadır.Eğer nesnenin ekranda gözükmeyen kısımlarını silmek gerekiyorsa, kesme yüzeyleri uygulanır. Bundan sonra, OpenGL, projection matrisini kesme koordinatlarını verebilmek için uygulayacaktır. Bu sayede kamerasının nesneyi gösteremediği kısımlar kesilecektir, yani gözükmeyecektir. Bu noktadan sonra perspektif bölümünden, normalleşmiş araç koordinatlarını üretmek için koordinat değerlerini w ile bölgerek yerine getirilir. Son olarak ise, viewport transformasyonu uygulanarak, dönüştürülmüş koordinatlar pencere koordinatlarına çevrilir. Böylece viewport boyutları üzerinde, resim ile oynamak için, çeşitli manipülasyonlar uygulayabilirsiniz.

2.16 Basit bir Örnek: Bir Küp Çizme

Örnek 2-3, yarattığınız pencerede, wireframe olarak bir küp çizmektedir.Wireframe olarak söylenilen şey küpün sadece gözüken kenarlarını göstermek anlamına gelmektedir, küpün yüzeyleri çizilmez. Viewing transformasyonunda **gluLookAt()** ile kamerasının nereye yerleştirileceği ve nasıl bir şekilde modeli izleyeceği belirtilir.

Ek olarak projection ve viewport transformasyonları (bezen transformation kelimesini transformasyon veya dönüşüm olarak yazabilirim) belirlenecektir. Resim 2-6 ise ekranda olacak olan küpü göstermektedir.

Örnek 2-3 Dönüştürülmüş küp

```

#include <windows.h>
#include <GL/glut.h>

void init(void) {

    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_FLAT);

} // end of void init(void)

void display(void) {

    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glLoadIdentity();
    /* viewing transformation */
    gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
    //glTranslatef(0.0, 0.0, -5.0);
    glScalef(1.0, 2.0, 1.0);
    glutWireCube(1.0);
    glFlush();

} // end of void display(void)

void reshape(int w, int h) {

    glViewport(0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);
//    gluPerspective(60.0, 1.0, 1.5, 20.0);
    glMatrixMode(GL_MODELVIEW);

} // end of void reshape(int, int)

int main(int argc, char** argv) {

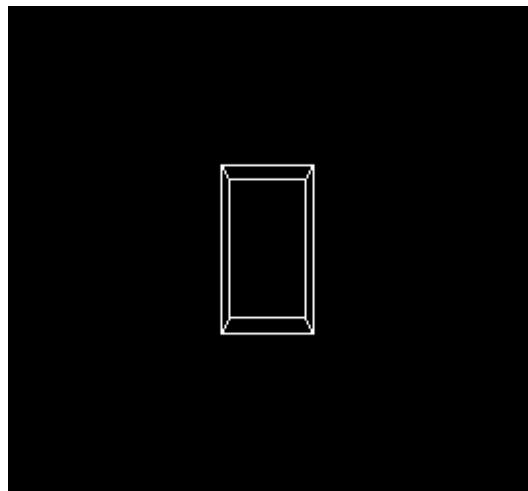
```

```

glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(500, 500);
glutInitWindowPosition(100, 100);
glutCreateWindow(argv[0]);
init();
glutDisplayFunc(display);
glutReshapeFunc(reshape);
glutMainLoop();

return 0;
} // end of int main(int, char++)

```



Resim 2-6

2.17 Vieving Transformation

Viewing transformation kameranın yerinin belirlenmesi ve modele yönlendirilmesidir. Bu kodda, viewing transformation belirlenmeden önce, program içerisinde o anda kullanılan matris **glLoadIdentity()** fonksiyonu ile identity matris olarak yüklenir. Identity matris, diagonal elemanları 1'e ve diğer tüm elemanları 0'a eşit olan bir kare matristir. (Program içerisinde o anda kullanılmakta olan matrise current matris diyelim)

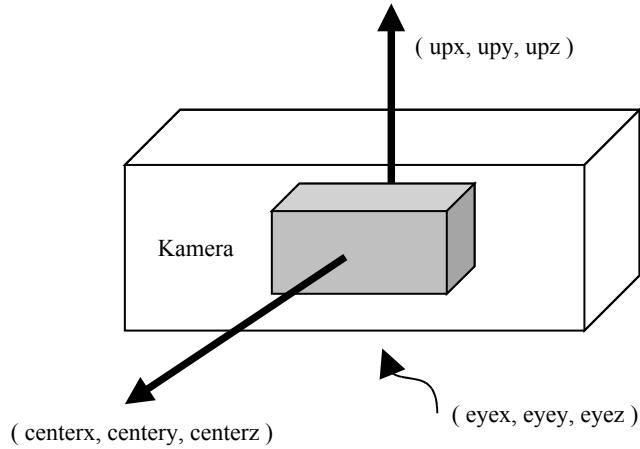
Bu aşama gereklidir, çünkü belirtilen bir matris current matris ile çarpılıp tekrar current matrise atanır. Dolayısı ile yapacağımız işlemlere dikkat etmemiz gereklidir. Eğer current matrisini identity olarak ilklemezseniz yani temizlemezseniz, bazı istenmeyen sonuçlar elde edersiniz.

Örnekte, matris ilklenmekten sonra, **gluLookAt()** fonksiyonu ile viewing transformasyonu belirlenmektedir. Fonksiyona aktarılan argümanlar, kameranın(göz koordinatları) nereye yerleştirileceğini, nereye yönlendirileceğini ve nasıl bir şekilde modeli göstereceğini belirtirler. Kodda kullanılan argümanlar kameranın (0, 0, 5) noktasında olduğunu, kameranın lensinin (0, 0, 0) noktasına yönlendirildiğini ve up-vektörü olarak (0, 1, 0) çekim yaptığını anlatmaktadır.

gluLookAt() fonksiyonunu kullanmadık, kameranın varsayılı olan değerleri kullanılmış olurdu. Varsayılı olarak, kamera koordinat sistemini merkezinde, lensi negatif z eksene çevrilmiş ve up vektör olarak (0, 1, 0) noktasına göre ayarlanmış olurdu. Örnekte ise kamerayı 5 birim, pozitif z ekseni doğrultusunda kaydırılmıştır.

```
void gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez,  
                GLdouble centerx, GLdouble centery, GLdouble centerz,  
                GLdouble upx, GLdouble upy, GLdouble upz);
```

(eyex, eyey, eyez) koordinatları kameranın yerini ayarlar. (centerx, centery, centerz) koordinatları kameranın hangi noktaya bakacağını tanımlamaktadır. Son olan koordinat parametreleri ise (upx, upy, upz) kameranın up-vektörünü belirtmektedir. Resim 2-7 bu parametrelerin, kamera üzerindeki kullanımını göstermektedir.



Resim 2-7

2.18 **glRotate***() ve **glTranslate***() Fonksiyonlarının Kullanımı

gluLookAt() fonksiyonunu kullanmadan, başka yöntemler ile bu fonksiyonu gerçekleştirebiliriz. Bu yöntem ise modelview transformasyon fonksiyonlarından olan **glRotate***() ve **glTraslate***() fonksiyonlarıdır. Bu fonksiyonlar durağan bir kameraya, üç boyutlu ortamdaki fonksiyonelliği sağlarlar. Kamerayı hareket ettirmek yerine kameranın gösterdiği modelleri hareket ettirmek de diğer bir yöntemdir. Örnek 2-3'teki koda tekrar bakacak olursanız, **display()** fonksiyonu içerisinde öbü çift ters bölümü işareteti ile pasif hale getirilmiş **glTranslatef(0.0, 0.0, -5.0)** fonksiyonun göreceksiniz. Bu fonksiyonun önündeki ters bölüm işaretlerini kaldırıp, **gluLookAt()** fonksiyonun pasif hale getirirseniz aynı etkiyi göreceksiniz. Bildiğiniz gibi **gluLookAt()** fonksiyonunu kullanmadığınızda kamera, koordinat sisteminin merkezine varsayılı olarak yerleştirilir. Koordinat sistemini, kameradan 5 birim uzağa yani negatif z eksenini doğrultusunda kaydırırsanız küpün görünümünde bir değişiklik olmayacağındır. Aşağıdaki **display()** fonksiyonu bu özeti göstermektedir.

```
void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glLoadIdentity();
    /* viewing transformation */
```

```

//gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
glTranslatef(0.0, 0.0, -5.0);
glScalef(1.0, 2.0, 1.0);
glutWireCube(1.0);
glFlush();

} // end of void display(void)

```

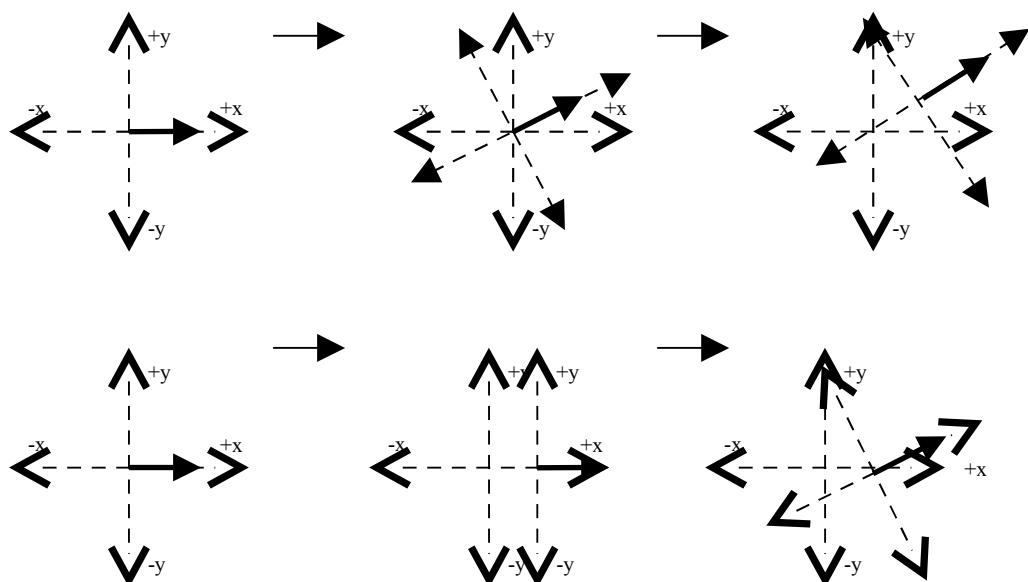
2.19 Modeling Transformation

Modeling transformasyonları, nesneleri hareket ettirerek, döndürerek, boyutunda değişiklik yaparak, bu modellerin orientasyonunda ve pozisyonunda çeşitli değişiklikler yapmamıza izin verir. Bu işlemlerin birini veya bir çوغunu birlikte kullanarak gerçekleştirebilirsiniz.

- **Translation:** Belirtilen bir eksen üzerinde nesnenin hareketini sağlar.
- **Rotating:** Bir eksen etrafında, modelin kendi çevresinde döndürme işlemidir.
- **Scalig:** Nesnenin boyutunu büyültmek veya küçültmek gibi fonksiyonelliklere sahip olan bir gerçekleştirimidir. Bu özellik sayesinde nesneye en veya boy gibi çeşitli boyut değişikliği kazandırabilirsiniz

Ekranınızda çizeceğiniz nesne için belirlemiş olduğunuz modeling transformasyon fonksiyonlarının sırası önemlidir. Örnek vermek gerekirse, Resim 2-8, translation işleminden sonra rotation işlemini ve rotation işleminden sonra translation işlemini göstermektedir. İki farklı işlemin etkileri birbirlerinden farklıdır. Farz edelim ki, koordinat sisteminin merkezine yerleşmiş olan bir ok olsun. Uygulayacağımız ilk transformasyon

İşlemi bu oku z eksene göre 30 derece döndürmektir yani rotation işlemidir. Ardından x ekseni doğrultusunda koordinat sistemini 5 birim taşıma yapalım. Eğer bu işlemi tersini yaparsak elde edeceğimiz sonuç ilkinden farklı olacaktır.



Resim 2-8

2.20 Modelview Matrisi

Modelview matrisi, nesneleri koordinat sisteminin istenilen bir yerine yerleştirmek ve onları yönetmek için koordinat sistemini tanımlar. 4' e 4'lük bir matristir vertexler ve transformasyonlar ile çarpılır. Bu çarpım işleminin sonunda, vertexlere uygulana transformasyonların sonuçlarını yansıtacak yeni bir matris üretilir.

OpenGL komutu olan **glMatrixMode()** fonksiyonu aracılığı ile modelview matrisini değiştirmek isteyebilirsiniz. Bu fonksiyonun prototipi aşağıdaki satırda belirtilmiştir

```
void glMatrixMode( GLenum mode );
```

Herhangi bir transformation işlemini çağrımadan önce, hangi matris modu üzerinde Modifikasyonlar uygulayacağınızı belirtmeniz gereklidir. Modelview modunda işlem yapmak istiyorsanız parametre olarak GL_MODELVIEW parametresini aktarın.

glMatrixMode() için aktarılacak diğer parametreler ise GL_PROJECTION ve GL_TEXTURE argümanlarıdır. Projection matrisini belirtmek için GL_PROJECTION kullanılır ve texture matrisi için ise GL_TEXTURE bunu belirtmektedir.

Bir çok durumda, bu matrisi current matris olarak ayarladıkten sonra modelview matrisini identity matris olarak ilkleyin. Bunu yapabilmek için **glLoadIdentity()** fonksiyonu kullanın.

2.21 Translation

Üç boyutlu bir dünyada, nesneyi bir yerden başka bir yere taşımak için bu işlem kullanılmaktadır. Bunu yapabilmek için **glTranslatef()** ve **glTranslated()** fonksiyonları bu işlemi gerçekleştirir.

```
void glTranslatef(GLfloat x, GLfloat y, GLfloat z);  
void glTranslated(GLdouble x, GLdouble y, GLdouble z);
```

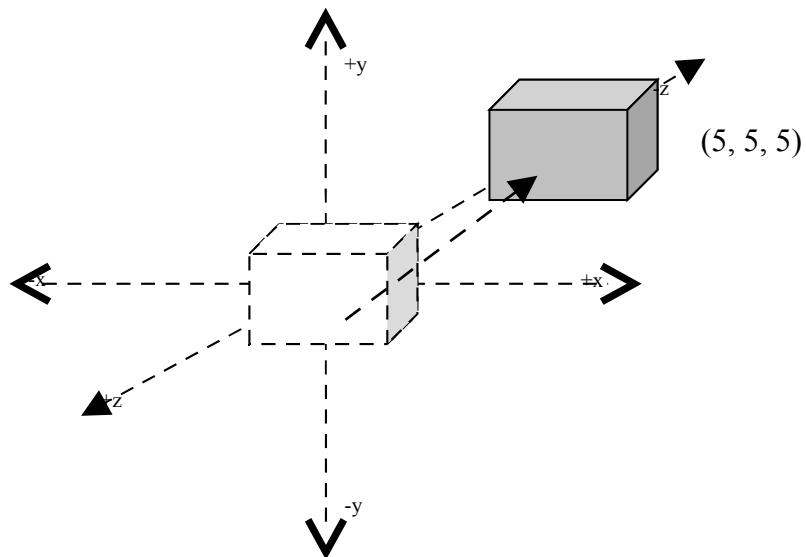
Bu iki fonksiyon arasındaki tek fark aldıkları parametre tiplerinin birbirlerinden farklı olmalıdır. **glTranslatef()**, float tipinde ve **glTranslated()** fonksiyonu double tipinde parametre alır.

Farzedin ki, bir kübü koordinat sisteminin merkezinden (5, 5, 5) noktasına taşımak istiyorsunuz. İlk olarak modelview matrisini yükleyin ve sonra ilkleyin. Aşağıdaki kod yapacağımız işlemi gerçekleştirir ve resim 2-9 bunun sonucunu gösterir.

```

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glTranslatef(5.0, 5.0, 5.0);
// kübü çiz
DrawCube();

```



Resim 2-9

2.22 Rotating

OpenGL'de rotating, **glRotate***() fonksiyonu ile gerçekleştirilmektedir.

```

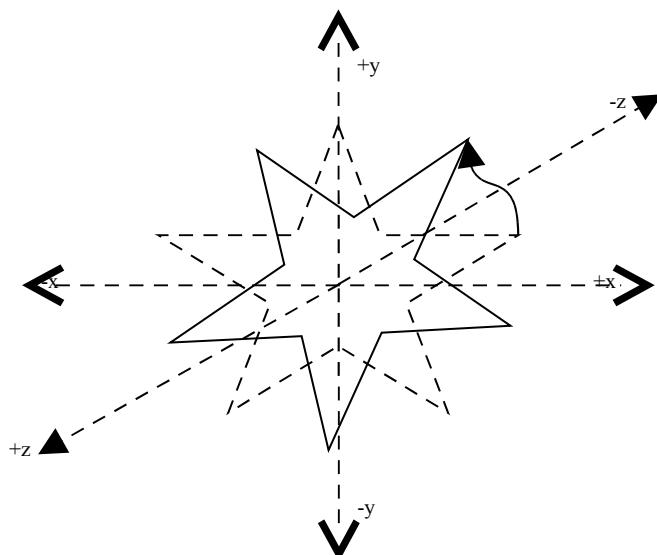
void glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z);
void glRotated(GLdouble angle, GLdouble x, GLdouble y, GLdouble z);

```

Tekrar söylemek gerekirse, **glTranslate***() fonksiyonunda olduğu gibi bu iki fonksiyon arasındaki tek fark aldıkları parametre tiplerinin birbirlerinden farklı olmalarıdır. x, y ve z parametrelerine göre, nesneyi belirli bir açı ölçüsünde bu koordinat eksenleri etrafında döndürebilirsiniz. Mesela çizmiş olduğunuz bir modeli y ekseni etrafında yelkovan yönüne, 135 derece döndürmek istiyoruz. Bunun için aşağıdaki bu işlemi bizim için yapmaktadır.

```
glRotatef(135.0, 0.0, 1.0, 0.0);
```

y argümanının aldığı 1.0 değeri, y ekseni yöndeki birim vektörü belirtmektedir. İstediğiniz eksene göre, rotate işlemini yapabilmek için sadece birim vektörünü belirtmeniz gereklidir. Resim 2-10 glRotate*() fonksiyonunun nasıl çalıştığını göstermektedir.



Resim 2-10

Eğer şekli saat yönünün tersine çevirmek istiyorsanız, açıyı negatif olarak ayarlamanz gerekmektedir. Bahsettiğimiz aynı rotating örneği için nesneyi saat yönünün tersinde 135 derece y ekseni çevresinde döndürmek istiyorsak aşağıdaki komutu yazmamız gerekmektedir.

```
glRotatef(-135.0, 0.0, 1.0, 0.0);
```

x, y, z parametrelerini kullanarak modeli istediğimiz yöne çevirebiliriz. Unutmayın ki koordinat ekseni bu dönüşten etkilenenecektir.

2.23 Scaling

Modelin boyutundaki ayarlamaları yapmak için scaling işlemi kullanılmaktadır. Nesnenin boyutları eksenlere göre uzatılabilir kısılabilir. Bunu OpenGL'in sağladığı **glScale*()** fonksiyonu ile yapabiliriz.

```
void glScalef(GLfloat x, GLfloat y, GLfloat z);  
void glScaled(GLdouble x, GLdouble y, GLdouble z);
```

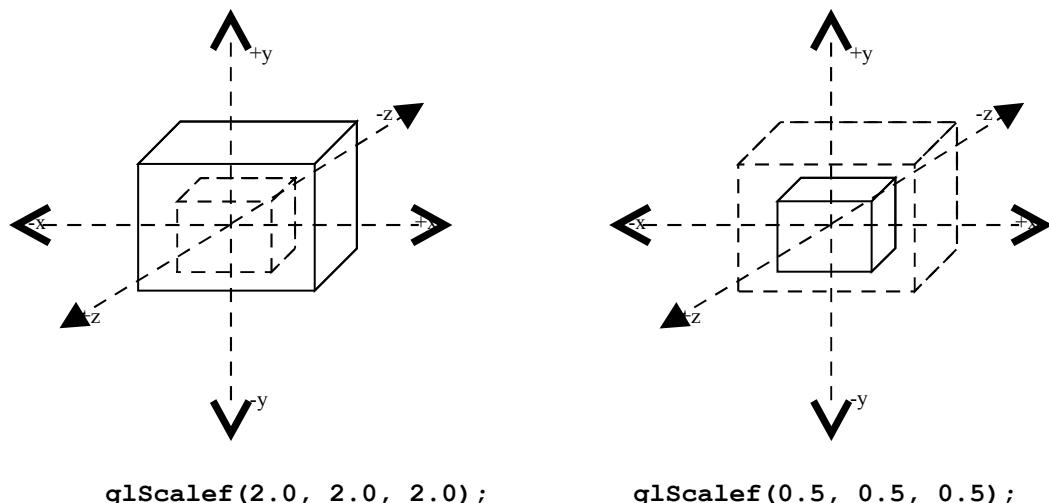
x, y, z parametrelerine geçirilen değerler her bir eksenin scale faktörünü belirler. Örnek olarak aşağıdaki satır nesnenin her boyutunu 2 katına çıkartmaktadır.

```
glScalef(2.0, 2.0, 2.0);
```

Mesela, çizdiğimiz bir kübün derinlik ve yüksekliğini değiştirmeden x eksenin üzerindeki genişliğini 2 katına çıkartmak istiyoruz.

```
glScalef(2.0, 1.0, 1.0);
```

Eğer nesneyi küçültmek istiyorsanız, kullanacağınız değerler 1'den küçük olmalıdır.



```
glScalef(2.0, 2.0, 2.0);
```

```
glScalef(0.5, 0.5, 0.5);
```

Aşağıda kübüün hacmini 8 katına çıkartan örnek gösterilmektedir.

```
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity();  
glScalef(2.0, 2.0, 2.0);  
DrawCube(); // kübü çiz
```

2.24 Matris Yığıtları

Yarattığınız modelview ve projection matrisleri, yükleme ve çarpma işlemeleri sadece buz dağının görünen yüzünü göstermektedirler. Bu matrislerin her biri matrislerin yığıtlının en üst elemanlarıdır.

Matris yığıtı bizim için ideal bir mekanizma sağlamaktadır. Geçmişteki transformasyonları hatırlayabilmek için bu yapı kullanılmaktadır. Üç tip matris yığıtı vardır.

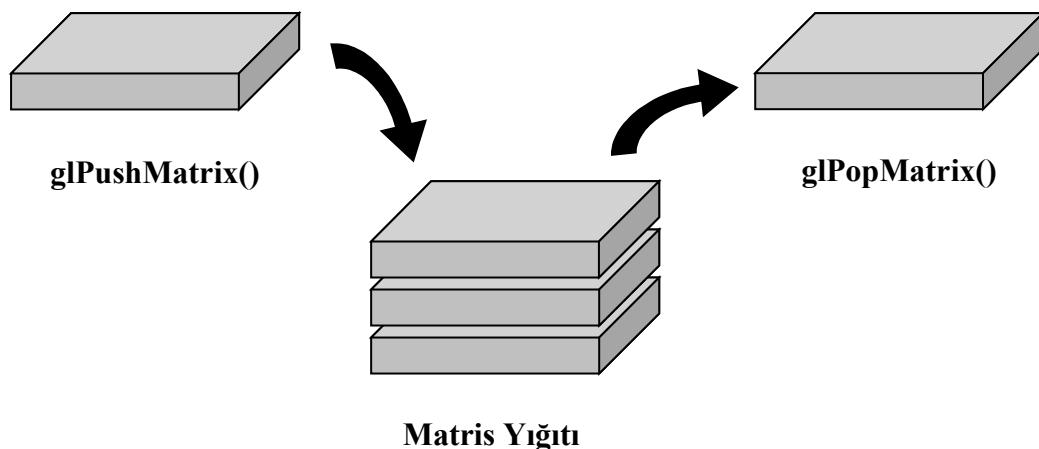
- Modelview Matris Yığıtı
- Projection Matris Yığıtı
- Texture Matris Yığıtı

Matrislerin yığıtları hiyerarşik olarak nesneleri meydana getirmek için kullanışlıdır-
kar. Bu karmaşık modeller basit olan modellerden yaratılırlar. Örneğin, kutulardan na-
sil bir robot oluşturduğunu düşünün. Eğer robottu bileşenlerine ayırsanız, robotun
bir gövdesinin, kollarının, başının ve bunun gibi bileşenlerinin olduğunu görürsünüz.

Bu ana kadar yaptığımız matris işlemleri hep current matris ile ilgiliydi. Aslında bu matris yığıtin en üst elemanıdır. Yığıt işlemlerini yerine getiren komutlar ile, yığıtin üstünde bulunan matrisi kontrol edebiliriz. Bildiğiniz gibi yığıt işlemlerinde en sıkılıkla kullanılan iki komut vardır. Bu komutlar **push()** ve **pop()** komutlarıdır. Yığıta bilgi göndermek için yığıtin en üstüne bu bilgi kopyalanır. Bu işlem **push()** işlemidir. Yığıt-
tin en üstünden bilginin atılabilmesi için **pop()** işlemi kullanılmaktadır. OpenGL bu gi-
bi komutları bize sağlamıştır. **glPushMatrix()** ve **glPopMatrix()**... **glPushMatrix()**

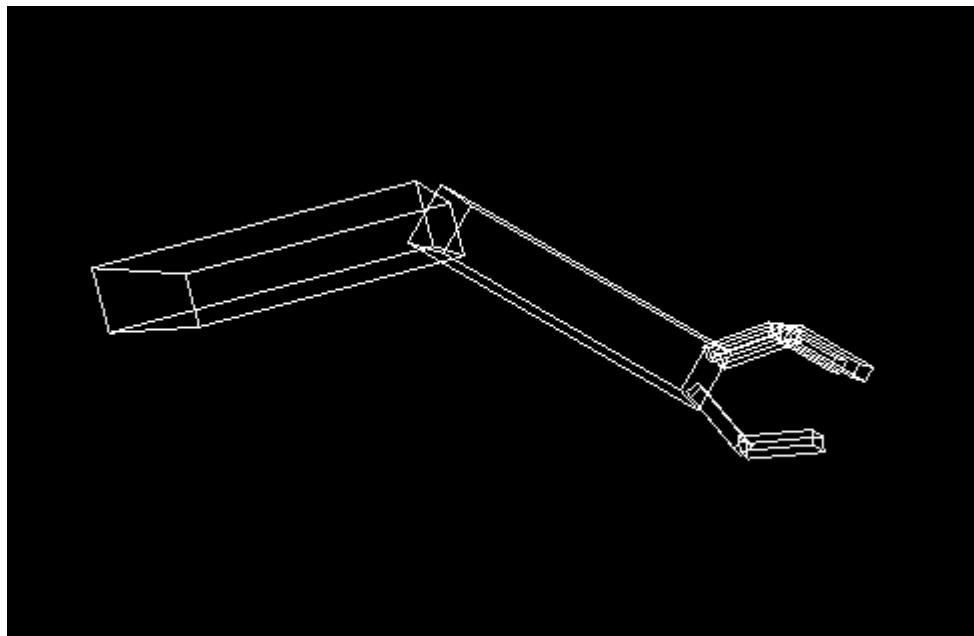
current matrisi kopyalar ve yiğitin en üstüne bu kopyalanmış bilgiyi yerleştirir.

glPopMatrix() ise yiğitin en üstündeki bilgiyi çekip çıkartır. Hatırlayın: yiğitin en üstü current matristir. Resim 2-11 yiğit matrisini gösterir.



Resim 2-11

glPushMatrix() ve **glPopMatrix()** için yapacağımız en uygun örnek, bir robot kolu olacaktır. Bu örnek sayesinde bu iki fonksiyonun ne işe yaradığını bu örnek sayesinde anlayacaksınız. Bu bölümde robot kolunun ikiden fazla segmenti vardır. Aşağıdaki resim 2-12 çizilecek olan robot kolunu göstermektedir.



Resim 2-12

Kolun segmentlerini çizebilmek için öncelikle boyutlarını değiştirebileceğiniz bir kübe ihtiyacınız olmalı, fakat ilk olarak her segmentin kendi koordinat eksenlerine yerleştirmek için uygun modelleme transformasyonlarını çağrırmak gereklidir. İlk olarak yerel koordinat sisteminin merkezi çizilen kübün merkezinde olacağından dolayı, kübün bir kenarına yerel koordinat sistemini taşımaya ihtiyacınız olacak. Diğer bir koşulda, küp pivot noktasına göre değil de merkezine göre dönecektir.

Pivot noktasını yaratmak için **glTranslate***() fonksiyonunu ve bu pivot noktasına göre **glRotate***() fonksiyonunu çağrırdıktan sonra, kübün merkezine doğru çevrimi yapın. Daha sonra küp çizilmeden önce, kübün boyutlarını değiştirin. **glScale***() fonksiyonu diğer segmentleri etkileyebilir ve bunu engellemek için **glPushMatrix** ve **glPopMatrix** fonksiyonlarını kullanmak faydalı olacaktır. Aşağıda robot kolunun ilk segmenti için, program kodunun nasıl bir şeye benzediğini görün.

```
glTranslatef(-1.0, 0.0, 0.0);
glRotatef((GLfloat) shoulder, 0.0, 0.0, 1.0);
glTranslatef(1.0, 0.0, 0.0);
```

```

glPushMatrix();
    glScalef(2.0, 0.4, 1.0);
    glutWireCube(1.0);
glPopMatrix();

```

İkinci segmenti yaratabilmek için, lokal koordinat sistemini bir sonraki pivot noktasına hareket ettirmemiz gerekiyor. Koordinat sistemi önceden döndüğüne göre, x ekseni, dönmüş olan kolun uzunluğu doğrultusunda çoktan yerleşmiş olacaktır. Bu sebepten dolayıdır ki, x ekseni doğrultusunda transformasyon yapmak lokal koordinat sistemini bir sonraki pivot noktasına hareket ettirecektir. Bu noktaya koordinat sisteminin tashındıktan sonra, ilk olarak kullanmış olduğunuz kod bloğunu yeniden kullanarak, ikinci segmenti çizebilirsiniz. Bu işlem sonsuz sayıda segmentler için geçerlidir. Örnek 2-4 deki program kodunun tümü gösterilmektedir.

Örnek 2-4

```

#include <stdlib.h>
#include <GL/glut.h>

static GLfloat shoulder = 0.0, elbow = 0.0,
               wrist1 = 0.0, finger1 = 0.0,
               b_wrist = 0.0, b_finger = 0.0;

void init(void) {

    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_FLAT);

    return;

} // end of void init(void)

void reshape(int w, int h) {

```

```

glViewport(0, 0, (GLsizei) w, (GLsizei) h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(65.0, (GLfloat) w / (GLfloat) h, 1.0, 20.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(-2.0, 0.0, -5.0);

return;

} // end of reshape(int, int)

void display(void) {

    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);

    glPushMatrix();

    glTranslatef(-1.0, 0.0, 0.0);
    glRotatef((GLfloat) shoulder, 0.0, 0.0, 1.0);
    glTranslatef(1.0, 0.0, 0.0);
    glPushMatrix();
        glScalef(2.0, 0.4, 1.0);
        glutWireCube(1.0);
    glPopMatrix();

    glTranslatef(1.0, 0.0, 0.0);
    glRotatef((GLfloat) elbow, 0.0, 0.0, 1.0);
    glTranslatef(1.0, 0.0, 0.0);
    glPushMatrix();
        glScalef(2.0, 0.4, 1.0);
        glutWireCube(1.0);
    glPopMatrix();

    glPushMatrix();
}

```

```

glTranslatef(1.0, 0.125, 0.25);
glRotatef((GLfloat)wrist1, 0.0, 0.0, 1.0);
glTranslatef(0.25, 0.0, 0.0);
glPushMatrix();
    glScalef(0.5, 0.1 , 0.25);
    glutWireCube(1.0);
glPopMatrix();

glTranslatef(0.25, 0.0, 0.0);
glRotatef((GLfloat) finger1, 0.0, 0.0, 1.0);
glTranslatef(0.25, 0.0, 0.0);
glPushMatrix();
    glScalef(0.5, 0.1 , 0.25);
    glutWireCube(1.0);
glPopMatrix();

glPopMatrix();

glPushMatrix();
glTranslatef(1.0, 0.125, -0.25);

glRotatef((GLfloat)wrist1, 0.0, 0.0, 1.0);
glTranslatef(0.25, 0.0, 0.0);
glPushMatrix();
    glScalef(0.5, 0.1, 0.25);
    glutWireCube(1.0);
glPopMatrix();

glTranslatef(0.25, 0.0, 0.0);
glRotatef((GLfloat) finger1, 0.0, 0.0, 1.0);
glTranslatef(0.25, 0.0, 0.0);
glPushMatrix();
    glScalef(0.5, 0.1, 0.25);
    glutWireCube(1.0);
glPopMatrix();

glPopMatrix();

```

```

    glPushMatrix();
    glTranslatef(1.0, -0.125, 0.0);

    glRotatef((GLfloat)b_wrist, 0.0, 0.0, 1.0);
    glTranslatef(0.25, 0.0, 0.0);
    glPushMatrix();
    glScalef(0.5, 0.1, 0.25);
    glutWireCube(1.0);
    glPopMatrix();

    glTranslatef(0.25, 0.0, 0.0);
    glRotatef((GLfloat) b_finger, 0.0, 0.0, 1.0);
    glTranslatef(0.25, 0.0, 0.0);
    glPushMatrix();
    glScalef(0.5, 0.1, 0.25);
    glutWireCube(1.0);
    glPopMatrix();
    glPopMatrix();

    glPopMatrix();

    glutSwapBuffers();

    return;
} // end of void display(void)

void keyboard(unsigned char key, int x, int y) {

    switch (key) {
        case 's':
            shoulder += 5;
            if (shoulder > 360.0) shoulder -= 360.0;
            glutPostRedisplay();
            break;
        case 'S':
            shoulder -= 5;
    }
}

```

```

        if (shoulder < 360.0) shoulder += 360.0;
        glutPostRedisplay();
        break;
    case 'e':
        elbow += 5;
        if (elbow > 360.0) elbow -= 360.0;
        glutPostRedisplay();
        break;
    case 'E':
        elbow -= 5;
        if (elbow < 360.0) elbow += 360.0;
        glutPostRedisplay();
        break;
    case 'd':
        wrist1 += 5;
        if (wrist1 > 360.0) wrist1 -= 360.0;
        glutPostRedisplay();
        break;
    case 'D':
        wrist1 -= 5;
        if (wrist1 < 360.0) wrist1 += 360.0;
        glutPostRedisplay();
        break;
    case 'f':
        finger1 += 5;
        if (finger1 > 360.0) finger1 -= 360.0;
        glutPostRedisplay();
        break;
    case 'F':
        finger1 -= 5;
        if (finger1 < 360.0) finger1 += 360.0;
        glutPostRedisplay();
        break;
    case 'v':
        b_finger += 5;
        if (b_finger > 360.0) b_finger -= 360.0;
        glutPostRedisplay();

```

```

        break;
    case 'V':
        b_finger -= 5;
        if (b_finger < 360.0) b_finger += 360.0;
        glutPostRedisplay();
        break;
    case 'c':
        b_wrst += 5;
        if (b_wrst > 360.0) b_wrst -= 360.0;
        glutPostRedisplay();
        break;
    case 'C':
        b_wrst -= 5;
        if (b_wrst < 360.0) b_wrst += 360.0;
        glutPostRedisplay();
        break;
    case 27:
        exit(0);
        break;
    default:
        break;
    } // end of switch (key)
} // end of void keyboard(unsigned char, int , int)

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 100);
    glutCreateWindow(argv[0]);
    init();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutMainLoop();
    return 0;
}

```

```
} // end of int main(int, int)
```

2.25 OpenGL’ de Işıklandırma

Konu olarak, üç boyutlu grafik kavramları açısından nesneler için ışıklandırma büyük önem taşımaktadır. Işıklandırma bilgisayar ortamında, yarattığınız nesnelere bir gerçeklik katmaktadır. Şu ana kadar işlemiş olduğumuz konularda nesnelerin nasıl yaratıldığını, bu nesneleri nasıl hareket ettirdiğimizi ve boyutlarındaki değişimini nasıl gerçekleştirdiğimizi anlatmaya çalıştık. Şimdi ise bu nesneleri ışıklandırma fonksiyonları kullanarak canlandırmaya başlayacağız.

Gerçek dünyada nesneler nasıl ışığı yansıtarak, kendi renklerini bize gösteriyorsa, ışığı kırmızı, yeşil ve mavi bileşenlerine yaklaştırarak, OpenGL ışık ve ışıklandırmayı hesaplamaktadır. Burada söylemek istenene şey, ışığın yaydığı kırmızı, yeşil ve mavi ışık miktarlarına bağlı olarak, bir ışığın yaydığı renk belirlenir. Işık bir yüzeye çarptığında, yüzey tarafından yansıtılacak olan kırmızı, yeşil ve mavi ışık tonlarının yüzdelemini belirtmek için, OpenGL yüzeyin materyalini kullanmaktadır.

OpenGL, dört ışık bileşenini kullanarak, gerçek dünyadaki ışıklandırmayı gerçekleştirmektedir.

- **Ambient Işıklandırma**, belirli bir yönden gelen ışık kaynağına benzememektedir. Bunun anlamı, ışığın hangi kaynaktan geldiğini belirlemenin imkansız olduğu anlamına gelir yani ışığın her yönden geldiğini ortaya koymaktadır.
- **Diffuse Işıklandırma**, belirli bir yönden gelmektedir, fakat ışık yüzeye çarptığında tüm yönlere eşit olarak yansıtılmaktadır. Gözünüzün hangi pozisyonda olduğunu bakılmadan, ışığın çarptığı yüzey, göze eşit olarak parlak görünür.

- **Specular Işıklandırma**, belirli bir yönden gelerek yüzeye çarpar ve önceden belirlenmiş bir yöne doğru yansımaktadır yani eşit olarak yansıtma yoktur.
- **Emissive Işıklandırma**, nesneden çıkan ışık anlamına gelmektedir. OpenGL ışıklandırma yönteminde, nesnenin yüzeyinden yansıyan renk, nesneye daha çok yoğunluk katar, fakat bu diğer ışık kaynaklarını etkilemez.

2.26 Materyal Renkler

OpenGL ışıklandırma modeli, kendisinin yansıtmakta olduğu kırmızı, yeşil ve mavi ışıkların yüzdeleri materyalin rengini etkilemektedir. Örnek olarak, kırmızı bir top, tüm gelen kırmızı ışığı yansıtır ve diğer gelmekte olan yeşil ve mavi ışığı absorbe eder. Eğer bu topu beyaz ışık altında gözlüyorsanız, topun tüm kırmızı rengi yansıtılacak ve top kırmızı olarak görülecektir. Eğer top kırmızı ışığın az olduğu bir yerde bulunuyorsa, top yine kırmızı olarak, fakat az yeşil ışığın bulunduğu yerde top bulunuyorsa, aynı top siyah olarak gözükmür. Çünkü yeşil ışık absorbe edilmektedir ve orada kırmızı ışık kaynağı yoktur, bu yüzden ışık yansıtılamaz.

İşıklarda olduğu gibi, materyallerinde farklı ambient, diffuse ve specular renklere sahiptirler. Bu renkler, materyalin ambient, diffuse ve specular yansımalarını belirler. Bir materyali ambient yansımıası, her gelen ışık kaynağının ambient bileşeni ile, diffuse yansımıası ışığın diffuse bileşeni ile ve specular yansımıası specular bileşeni ile bütünlüğe sahip olmaktadır. Ambient ve diffuse yansımaları materyalin rengini belirler. Specular yansımıza genellikle beyaz yada gridir bu yüzden yüzey üzerindeki specular olan ışıklı ve detaylı olan kısmı, orada ışık kaynağının specular yoğunluğunu olmasını sonlandırır. Eğer parlak bir kırmızı plastik topun beyaz ışık altında olduğunu düşünürseniz, bu topun büyük bir yüzeyinin kırmızı, fakat parlaklığının fazla olduğu bölgenin beyaz olduğunu göreceksiniz.

Işıklar için belirlenmiş olan renk bileşenleri, materyaller için, daha farklı bir anlama gelmektedir. Bir ışığı ele alırsak, her renk için, sayılar tam yoğunluk yüzdesine uygun düşmektedir. Işığın kırmızı, yeşil ve mavi bileşenlerinin değerinin hepsi $1.0'$ a eşit ise, ışık parlak bir beyaz ışık kaynağı olmaktadır. Eğer tüm değerler $0.5'$ e eşit olduğunda, ışık kaynağı yine beyaz olur fakat yoğunlu %50 oranında azalcaktır. Işık bileşenlerinden kırmızı ve yeşilin değeri $1.0'$ a, mavinin değeri ise $0.0'$ a eşit ise, ışık yeşil olarak görünür. Materyaller için, sayılar yansımakta ışığın bileşenler değeri ile çarpılmaktadır. Mesela materyalin kırmızı değeri $1.0'$ a, yeşil değeri $0.5'$ e ve mavi değeri $0.0'$ a eşit olsa, materyal gelen ışıklardan kırmızı ışığın %100'ünü, yeşil ışığın %50'sini ve mavi ışığın %0'ını yansıtma konusunu belirtmektedir. Diğer bir anlatım şekli ile anlatmaya çalışırsak, elimizde bir ışık kaynaklarının değerleri olsun. Bu değerler (IK, IY, IM) ve materyalin değerleri ise (MK, MY, MM) olsun. Diğer yansımaya etkilerini gözle almasak, gözün gördüğü değerler yani cismin yansıtacağı ışık değerleri (IK * MK, IY * MY, IM * MM) olmaktadır.

Benzer olarak, iki ışık kaynağı kullanıyorsanız, yani göze gönderilecek değerler (K1, Y1, M1) ve (K2, Y2, M2), OpenGL bu bileşenleri toplayacaktır (K1 + K2, Y1 + Y2, M1 + M2). Bu toplama işlemlerinden herhangi biri 1' den büyük çıkarsa toplama işlemi 1 olarak alınır.

2.27 Küçük Bir Örnek: Kırmızı Çaydanlık

Uygulamanıza, ışıklandırmayı eklemek için aşağıdaki aşamaları yapmakta faydalı olabilir:

1. Her nesnenin her verteksi için normal vektörleri tanımlayın. Bu normaller ışık kaynağıyla alakalı olarak nesnenin yerini belirlerler.
2. Bir veya birden fazla ışığı yaratın, seçin ve belirli konumlara getirin.

3. Bir ışıklandırma modelin yaratın.
4. Nesnenin materyal özelliklerini tanımlayın.

Aşağıdaki örnek 2-5 bu işlemleri gerçekleştirmektedir. Bu örnekte tek bir ışık kaynağı altında çizilmiş olan kırmızı bir çaydanlık çizilmektedir.

Örnek 2-5 Kırmızı Çaydanlık

```
#include <GL/glut.h>
#include <GL/glu.h>

GLfloat rotate_y = 0.0;

void draw_net(GLfloat size, GLint LinesX, GLint LinesZ) {

    int xc, zc;

    glBegin(GL_LINES);
    for (xc = 0; xc < LinesX; xc++)
    {
        glVertex3f(
            -size / 2.0 + xc / (GLfloat)(LinesX-1)*size,
            0.0,
            size / 2.0);
        glVertex3f(
            -size / 2.0 + xc / (GLfloat)(LinesX-1)*size,
            0.0,
            size / -2.0);
    } // end of for (xc = 0; xc < LinesX; xc++)

    for (zc = 0; zc < LinesZ; zc++)
    {
```

```

        glVertex3f(
            size / 2.0,
            0.0,
            -size / 2.0 + zc / (GLfloat)(LinesZ-1)*size);
        glVertex3f(
            size / -2.0,
            0.0,
            -size / 2.0 + zc / (GLfloat)(LinesZ-1)*size);
    } // end of for (zc = 0; zc < LinesZ; zc++)

    glEnd();

    return;
} // end of void draw_net(GLfloat, GLint, GLint)

void init(void) {

    GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};
    GLfloat light_ambient[] = {1.0, 1.0, 1.0, 1.0};
    GLfloat light_diffuse[] = {1.0, 1.0, 1.0, 1.0};
    GLfloat light_specular[] = {1.0, 1.0, 1.0, 1.0};

    GLfloat material_ambient[] = {0.23, 0.0, 0.0, 1.0};
    GLfloat material_diffuse[] = {0.44, 0.0, 0.0, 1.0};
    GLfloat material_specular[] = {0.33, 0.33, 0.52, 1.0};
    GLfloat material_emission[] = {0.0, 0.0, 0.0, 0.0};
    GLfloat material_shininess = 10;

    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_SMOOTH);

    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);

    glMaterialfv(GL_FRONT, GL_AMBIENT, material_ambient);
}

```

```

glMaterialfv(GL_FRONT, GL_DIFFUSE, material_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, material_specular);
glMaterialfv(GL_FRONT, GL_EMISSION, material_emission);
glMaterialfv(GL_FRONT, GL_SHININESS, &material_shininess);

 glEnable(GL_LIGHTING);
 glEnable(GL_LIGHT0);
 glEnable(GL_DEPTH_TEST);

 glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

 return;
} // end of init(void)

void display(void) {

 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
 glPushMatrix();

 glRotatef(rotate_y, 0.0, 1.0, 0.0);

 glPushMatrix();
 glTranslatef(0.0, 0.55, 0.0);
 glutSolidTeapot(0.75);
 glPopMatrix();

 glDisable(GL_LIGHTING);
 glColor3f(1.0, 1.0, 1.0);
 draw_net(5, 11, 11);
 glEnable(GL_LIGHTING);

 glPopMatrix();

 glutSwapBuffers();

 return;
} // end of void display(void)

```

```

void reshape(int w, int h) {

    glViewport(0, 0, (GLsizei)w, (GLsizei)h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0, (GLfloat)w/(GLfloat)h, 1.0, 20.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(0.0, 2.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

    return;
} // end of void reshape(GLfloat, GLfloat)

void rotate_teapot(void) {

    rotate_y += 0.5;
    if (rotate_y > 360.0)
        rotate_y -= 360.0;
    glutPostRedisplay();

    return;
} // end of rotate_teapot(void)

int main(int argc, char** argv) {

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Teapot");
    init();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutIdleFunc(rotate_teapot);
    glutMainLoop();
}

```

```
    return 0;  
} // end of int main(int, char**)
```

Örneği incelemeye başlayalım. Işık kaynaklarının bir çok özellikleri vardır, bunlar renk, pozisyon ve ışığın yönüdür. Bu tüm özellikleri belirlemek için kullanılan komut **glLight*()** fonksiyonudur. Fonksiyon üç argüman almaktadır. Fonksiyonun prototipi aşağıda belirtilmiştir.

```
void glLight[if]v( GLenum light, GLenum pname, Type *param );
```

İlk parametre olan *light* parametresi özelliği belirlenecek olan ışık kaynağını belirtir. Bu kaynaklar GL_LIGHT0, GL_LIGHT1, ... GL_LIGH7 olan sekiz tane ışık kaynağı mevcuttur. Işığın karakteristiğini belirleyen parametre *pname* parametresidir ve tablo 2-2 bu parametrenin alacağı değerleri listelemektedir. *pname* parametresine göre *param* parametresinin alacağı değerler vardır. Bu parametre bir vektörü gösteren bir işaretçidir.

Tablo 2-2 glLight*() Parametreleri

| Parametre | Varsayılı Değerler | Anlamı |
|-------------|----------------------|--------------------------------------|
| GL_AMBIENT | (0.0, 0.0, 0.0, 0.0) | Ambient yoğunluğu |
| GL_DIFFUSE | (1.0, 1.0, 1.0, 1.0) | Diffuse yoğunluğu (light 0 için) |
| | (0.0, 0.0, 0.0, 0.0) | (Diğer ışıklar için) |
| GL_SPECULAR | (1.0, 1.0, 1.0, 1.0) | specular yoğunluğu (light 0 için) |
| | (0.0, 0.0, 0.0, 0.0) | (Diğer ışıklar için) |
| GL_POSITION | (0.0, 0.0, 1.0, 0.0) | (x, y, z, w) pozisyon |

Gösterilecek daha bir çok parametre var fakat şu anda bizim ilgileneceğimiz parametreler uygulamalarımız için yeterli. Örneği dikkatlice incelerseniz **init()** fonksiyonu içinde ışığın ambient, diffuse, specular ve pozisyon bilgileri tanımlanmıştır.

```

GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};
GLfloat light_ambient[] = {1.0, 1.0, 1.0, 1.0};
GLfloat light_diffuse[] = {1.0, 1.0, 1.0, 1.0};
GLfloat light_specular[] = {1.0, 1.0, 1.0, 1.0};

glLightfv(GL_LIGHT0, GL_POSITION, light_position);
glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);

```

Göreceğiniz gibi, parametre değerleri için diziler tanımlanmış ve **glLightfv()** fonksiyonu ışığın özelliklerini belirlemek için tekrar tekrar çağrılmıştır. Bu özellikler ayarlandıktan sonra, ışıklandırma olayını ve ışık kaynağını aktifleştirmeye geldi. Temel olarak ışıklandırma'nın aktif hale gelebilmesi için **glEnable()** fonksiyonunu kullanmak gereklidir.

```
glEnable(GL_LIGHTING);
```

Bu işlemin ardından hangi ışık kaynağını aktif hale getirmek istediğimizi OpenGL' e anlatmak içi yine aynı fonksiyonu yani `glEnable()` fonksiyonunu kullanmamız gereklidir. Özelliklerini ayarladığımız ışık kaynağı light 0 olduğuna göre aşağıdaki kod bu ışık kaynağını aktif hale getirecektir.

```
glEnable(GL_LIGHT0);
```

İşığı aktif ettikten sonra kısaca anlatmak istediğimiz şey ışığın bulunduğu konum, yani ışığın pozisyonu ile ilgili önemli bir konuyu dile getirmek istiyoruz. ışık pozisyonu ayarlanırken kullanılan dört farklı değerler var (x, y, z, w). Bu değerleri ilk üçü yani x, y, z tahmin edeceğiniz gibi ışık kaynağının koordinatlarını belirtmektedir. Peki ya dördüncü parametre ne anlamına gelmektedir? Dördüncü parametreyi $0.0'$ a eşit olduğunda ışık kaynağının sonsuz uzaklıktan bir yerden geldiği anlamına gelmektedir ve bu sebepten dolayı ışık kaynağından yayılan ışınlar parel olarak yayılırlar. Bu akla ilk somut olarak gelen örneğin güneş ışığı olduğudur. Güneş dünyadan çok uzak olduğundan dolayı bu uzaklığa sonsuz olarak alabiliriz. Uzaklık sonsuz olunca güneşten çıkan ışınlar dünya yüzeyine parel olarak gelmektedir. Bu tip ışık kaynaklarına directional light source denmektedir. Light 0 için bu örneği komut olarak yazarsak aşağıdaki gibi kod bloğunu gösterebiliriz.

```
GLfloat light_position[] = {0.0, 0.0, 1.0, 0.0}
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

Bu yukarıdaki örnekte z – ekseninden gelen ışık kaynağı tanımlanmıştır. Dördüncü parametreyi sıfırdan farklı bir değere sahip olduğunda bu tip ışık kaynağı positional light source olarak adlandırılır. Bunun anlamı (x, y, z) noktasında konumlandırılmış olan bir ışık kaynağının bulunması anlamına gelmektedir. Bu sanki masanın üzerine konmuş olan bir lambanın bulunması gibidir.

Işık kaynağının karakteristiklerini gördükten sonra materyallerin karakteristiklerini inceleyerek ikisi arasındaki benzerlikleri veya farklılıklarını anlayacağız. En çok bilinen materyal özellikleri önceden görmüş olduğunuz ışık kaynağı özelliklerine benzemektedir. Bu özelliklerini belirlemek için kullanılan mekanizma ışıkta kullanılan mekanizma ile benzerlik taşımaktadır. Fakat bu sefer materyaller için kullanılan komut **glMaterial*()** fonksiyonudur.

```
void glMaterial[if]v( GLenum face, GLenum pname, Type *param );
```

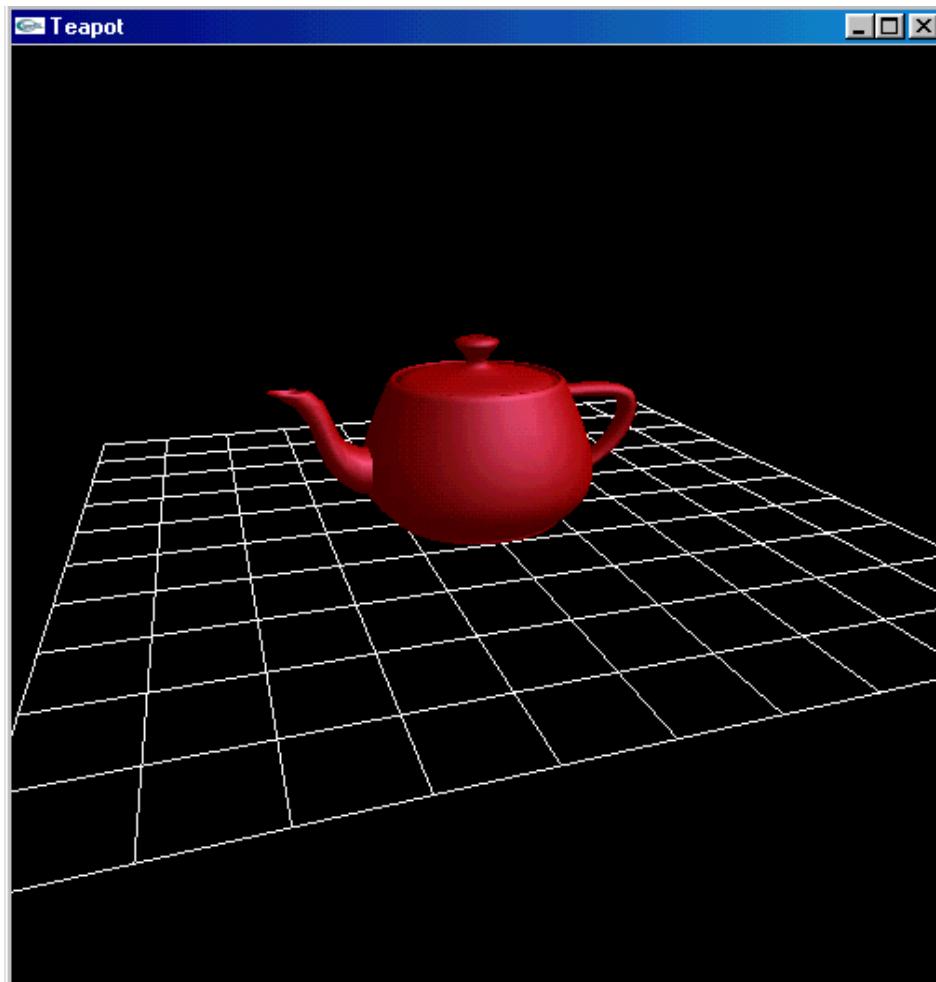
Fonksiyon sayesinde, ışıklandırmada kullanım için, materyalin özelliğini belirler. *face* parametresi, GL_FRONT, GL_BACK veya GL_FRONT_AND_BACK değerlerini alarak nesnenin yüz materyaline, materyaller için tanımlanmış olan özelliklerin uygulanması anlamına gelmektedir. Bu parametreye uygulanacak değerler ise *param* parametresi tarafından belirlenir. *param* parametresi, bir vektörü veya diziyi işaret eden bir işaretçidir. Bu *pname* parametresi için değerler tablo 2-3' te gösterilmektedir.

Tablo 2-3 glMaterial*() *pname* Parametresi

| Parametre | Varsayılı Değerler | Anlamı |
|--------------|----------------------|-------------------|
| GL_AMBIENT | (0.2, 0.2, 0.2, 1.0) | Ambient renk |
| GL_DIFFUSE | (0.8, 0.8, 0.8, 1.0) | Diffuse renk |
| GL_SPECULAR | (0.0, 0.0, 0.0, 1.0) | specular renk |
| GL_SHININESS | (0.0) | specular exponent |
| GL_EMISSION | (0.0, 0.0, 0.0, 1.0) | emissive renk |

face parametresi için GL_FRONT olarak tanımlanmış ise, GL_FRONT değeri OpenGL' e poligonun sadece ön kısmında materyal özelliklerini kullanacağı anlatılmaktadır GL_BACK olduğunda ise poligonun arka kısmında materyal özelliklerinin kullanılacağı veya GL_FRONT_AND_BACK kullanıldığında ise poligonun her iki

yüzeyinde bu materyal özelliklerinin geçerli olacağı bildirilmektedir. Program çalıştırıldığında resim 2-13 ' te çizim yapılmaktadır.



Resim 2-13

2.28 Blending

OpenGL' de renk karışımı (color blending) çizeceğiniz nesnelere saydamlık katımasına izin vermektedir. Bu saydamlık sayesinde, gerçek hayatı görmekte olduğunuz

su, çam, pencereler ve diğer nesneleri simule edebilirsiniz.

Su ana kadar nesneler üzerinde kullandığımız renk verme fonksiyonu **glColor3*()** fonksiyonuydu. Fonksiyon sadece üç parametre almaktadır. Bu parametrelerin ne olduğunu biliyorsunuz. Nesnelere saydamlık katabilmek için başka bir renk fonksiyonu kullanacağınız. Bu fonksiyon şimdije kadar kullanılan üç parametreye ek olarak dördüncü bir parametre almaktadır. Bu parametreye alfa parametresi denmektedir. Kullanacağınız fonksiyon ise **glColor4*()** fonksiyonudur. Peki ya alfa değeri ne yapar ? Alfa değeri nesnenin yeni rengini, frame buffer'ında bulunan renk ile birleştirmektedir. Frame buffer olarak söylenen şey sistemde kullanılan tüm buffer'ların bütündür. Yani color buffer, depth buffer, stencil buffer ve accumulation buffer size tek bir frame buffer'ını verebilmek için birleşirler. Buffer'lar konusuya pek ilgimiz olamayacak.

Blending işleminde, RGB bileşenleri fragmanın rengini ve alfa bileşeni ise o fragmanın katılık derecesini yani şeffaf olmama derecesini söyler. Saydam veya şeffaf olan cisimlerin şeffaf olmama derecesi saydam olmayan kilere göre daha düşüktür bu da demek oluyor ki bu cisimlerin alfa değerleri daha düşüktür.

2.29 Kaynak ve Hedef Faktörleri

Blending işlemi esnasında, gelen fragmanın renk değerleri, bu kaynak yada source olarak bilinir, hali hazırda saklı tutulan piksellerin renk değerleri, bu ise hedef yada destination olarak adlandırılmalıdır, ile bütünleştirilir. İlk olarak kaynak ve hedef faktörlerini nasıl hesaplayacağınızı belirlersiniz. Bu faktörler RGBA yani, kırmızı, yeşil, mavi ve alfa, dörtlüleridir. Bu dörtlüler kaynağın R, G, B ve A değeri ile hedefin R, G, B ve A değerleri ile çarpılır. Bunu matematiksel olarak gösterirsek kaynak ya da source' un RGBA değerleri (S_r, S_g, S_b, S_a) ve hedefin ya da destination' in RGBA değerleri ise (D_r, D_g, D_b, D_a) olsun. Son olarak blending işlemi sonucunda ortaya çıkan değerler aşağıdaki gibidir.

$$(R_s * S_r + R_d * D_r, G_s * S_g + G_d * D_g, B_s * S_b + B_d * D_b, A_s * S_a + A_d * D_a)$$

Kaynak ve hedef RGBA değerleri sırasıyla s ve d alt simge karakteri ile belirtilmektedir.

Şimdi kullanacağımız fonksiyon, kaynak ve hedef blending faktörlerini oluşturmaktadır. Kullanacağımız fonksiyonun adı **glBlendFunc()**'tur ve bu fonksiyon iki sabiti destekler. İlk hesaplanması gereken kaynak faktör ve diğeri ise hedef faktördür. Ek olarak blending işlemini aktif hale getirmek için her zaman aşağıdaki tek satırlık kodu yazmalısınız.

```
 glEnable(GL_BLEND);
```

Blending işlemini inaktif yapmak istiyorsanız bildiğiniz gibi **glDisable()** fonksiyonuna GL_BLEND parametresini aktarabilirsiniz. **glBlendFunc()** fonksiyonun prototipi aşağıdaki gibidir.

```
 void glBlendFunc( GLenum sfactor, GLenum dfactor );
```

sfactor parametresi kaynak blending faktörünün, *dfactor* parametresi ise hedef blending faktörünün nasıl hesaplanacağını belirtirler. Bu parametreler için mümkün değerler tablo 2-4 gösterilmektedir. Blending faktörleri 0 ile 1 aralığında değerler alır. Tabloda source, destination ve constant (sabit) renklerin RGBA değerleri altsimgen karakteri olan s, d, c harfleri ile gösterilmektedir.

Tablo 2-4 Source ve Destinatin Blending Faktörleri

| Sabit | İlgili Faktör | Hesaplanan Blending Faktörü |
|-----------------------------|-------------------------|---|
| GL_ZERO | source veya destination | (0, 0, 0, 0) |
| GL_ONE | source veya destination | (1, 1, 1, 1) |
| GL_DST_COLOR | source | (R _d , G _d , B _d , A _d) |
| GL_SRC_COLOR | destination | (R _s , G _s , B _s , A _s) |
| GL_ONE_MINUS_DST_COLOR | source | (1, 1, 1, 1) - (R _d , G _d , B _d , A _d) |
| GL_ONE_MINUS_SRC_COLOR | destination | (1, 1, 1, 1) - (R _s , G _s , B _s , A _s) |
| GL_SRC_ALPHA | source veya destination | (A _s , A _s , A _s , A _s) |
| GL_ONE_MINUS_SRC_ALPHA | source veya destination | (1, 1, 1, 1) - (A _s , A _s , A _s , A _s) |
| GL_DST_ALPHA | source veya destination | (A _d , A _d , A _d , A _d) |
| GL_ONE_MINUS_DST_ALPHA | source veya destination | (1, 1, 1, 1) - (A _d , A _d , A _d , A _d) |
| GL_SRC_ALPHA_SATURATE | source | (f, f, f, 1); f = min(A _s , 1 - A _d) |
| GL_CONSTANT_COLOR | source veya destination | (R _c , G _c , B _c , A _c) |
| GL_ONE_MINUS_CONSTANT_COLOR | source veya destination | (1, 1, 1, 1) - (R _c , G _c , B _c , A _c) |
| GL_CONSTANT_ALPHA | source veya destination | (A _c , A _c , A _c , A _c) |
| GL_ONE_MINUS_CONSTANT_ALPHA | source veya destination | (1, 1, 1, 1) - (A _c , A _c , A _c , A _c) |

2.30 Basit bir Örnek

Yapacağımız örnek bir Windows uygulamasıdır. Bu uygulamayı GLUT yardımı ile de yapabilirdik, fakat biraz değişiklik olsun istedik ve biraz da Windows uygulamalarında alıştırma yapmaya çalıştık. Örnek 2-6'da alfa değeri 0.75 olan ve birbirinden farklı ve kesişmiş iki üçkeni göstermektedir. Blending aktif durumdadır, kaynak blending faktörü GL_SRC_ALPHA olarak ve hedef blending faktörü ise GL_ONE_MINUS_SRC_ALPHA olarak ayarlanmıştır.

Program çalıştığında, ilk olarak sarı üçken pencerenin sol tarafında çizilir ve daha sonra çiyan renkli üçken ekranın sağ tarafına çizilerek üçkenler kesişirler. Pencerede çizilecek olan ilk üçkeni ‘l’ veya ‘r’ tuşlarına basarak belirleyebilirsiniz. ‘l’ tuşuna bastığınızda çizilecek ilk üçken sarı renkte olan ve ekranın sol tarafına bulunan üçken olacaktır, ‘r’ tuşuna basarsanız ekranın sağ tarafına çizilecek olan çiyan renkli üçken ilk olarak çizilir.

Örnek 2-6

```
#include <windows.h>
#include <gl/gl.h>
#include <gl/glu.h>
#include <gl/glaux.h>

float angle = 0.0;
HDC g_HDC;
BOOL fullScreen = FALSE;
BOOL keyPressed[256];

int left_first = GL_TRUE;

void init(void) {

    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_FLAT);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

```

    glEnable(GL_BLEND);
    return;
} // end of void initialize(void)

void resize(int width, int height) {

    glViewport(0, 0, (GLsizei) width, (GLsizei) height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (width <= height)
        gluOrtho2D(0.0, 1.0, 0.0,
                   (GLfloat)height / (GLfloat)width);
    else
        gluOrtho2D(0.0, (GLfloat)width / (GLfloat)height,
                   0.0, 1.0);
//    glMatrixMode(GL_MODELVIEW);
//    glLoadIdentity();
    return;
} // end of void resize(int, int)

void draw_leftTriangle(void) {

    glBegin(GL_TRIANGLES);
    glColor4f(1.0, 1.0, 0.0, 0.75);
    glVertex3f(0.1, 0.9, 0.0);
    glVertex3f(0.1, 0.1, 0.0);
    glVertex3f(0.7, 0.5, 0.0);
    glEnd();

    return;
} // end of void draw_leftTriangle(void)

void draw_rightTriangle(void) {

    glBegin(GL_TRIANGLES);
    glColor4f(0.0, 1.0, 1.0, 0.75);
    glVertex3f(0.9, 0.9, 0.0);

```

```

        glVertex3f(0.3, 0.5, 0.0);
        glVertex3f(0.9, 0.1, 0.0);
    glEnd();

    return;
} // end of void draw_rightTriangle(void)

void display(void) {

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    if (left_first) {
        draw_leftTriangle();
        draw_rightTriangle();
    } else {
        draw_rightTriangle();
        draw_leftTriangle();
    }

    SwapBuffers(g_HDC);
    return;
} // end of void display(void)

void SetupPixelFormat(HDC hDC) {

    int nPixelFormat;

    static PIXELFORMATDESCRIPTOR pfd = {
        sizeof(PIXELFORMATDESCRIPTOR),
        1,
        PFD_DRAW_TO_WINDOW |
        PFD_SUPPORT_OPENGL |
        PFD_DOUBLEBUFFER,
        PFD_TYPE_RGBA,
        32,
        0, 0, 0, 0, 0,
        0,

```

```

        0,
        0,
        0, 0, 0, 0,
        16,
        0,
        0,
        PFD_MAIN_PLANE,
        0,
        0, 0
    } ;

nPixelFormat = ChoosePixelFormat(hDC, &pf);

SetPixelFormat(hDC, nPixelFormat, &pf);

return;
} // end of void SetupPixelFormat(HDC)

LRESULT CALLBACK WndProc(HWND hWnd,
                        UINT message,
                        WPARAM wParam,
                        LPARAM lParam)
{
    static HGLRC hRC;
    static HDC hDC;
    char string[] = "Simple Windows Application";
    int width, height;

    switch(message) {

        case WM_CREATE:
            hDC = GetDC(hWnd);
            g_HDC = hDC;
            SetupPixelFormat(hDC);
            hRC = wglCreateContext(hDC);
            wglMakeCurrent(hDC, hRC);
    }
}

```

```

        init();
        return 0;
        break;
    case WM_CLOSE:
        wglGetCurrentContext(hDC, NULL);
        wglDeleteContext(hRC);
        PostQuitMessage(0);
        return 0;
        break;
    case WM_SIZE:
        height = HIWORD(lParam);
        width = LOWORD(lParam);
        if (height == 0) height = 1;
        resize(width, height);
        return 0;
        break;
    case WM_KEYDOWN:
        keyPressed[wParam] = TRUE;
        return 0;
        break;
    case WM_KEYUP:
        keyPressed[wParam] = FALSE;
        return 0;
        break;
    default:
        break;
    } // end of switch(message)

    return DefWindowProc(hWnd, message, wParam, lParam);
}

// end of WndProc(HWND, UINT, WPARAM, LPARAM)

int WINAPI WinMain(HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nShowCmd)
{

```

```

WNDCLASSEX windowClass;
HWND hWnd = NULL;
MSG msg;
BOOL done = FALSE;
DWORD dwExStyle;
DWORD dwStyle;
RECT windowRect;

int width = 200;
int height = 200;
int bits = 32;

windowRect.left = 0;
windowRect.right = width;
windowRect.top = 0;
windowRect.bottom = height;

windowClass.cbSize = sizeof(WNDCLASSEX);
windowClass.style = CS_HREDRAW | CS_VREDRAW;
windowClass.lpfnWndProc = WndProc;
windowClass.cbClsExtra = 0;
windowClass.cbWndExtra = 0;
windowClass.hInstance = hInstance;
windowClass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
windowClass.hCursor = LoadCursor(NULL, IDC_ARROW);
windowClass.hbrBackground = NULL;
windowClass.lpszMenuName = NULL;
windowClass.lpszClassName = "MyClass";
windowClass.hIconSm = LoadIcon(NULL, IDI_WINLOGO);

if (!RegisterClassEx(&windowClass))
    return 0;

if (MessageBox(NULL,
    "Would You Like To Run In Fullscreen Mode?",
    "Start FullScreen?", MB_YESNO|MB_ICONQUESTION)==IDYES)
    fullScreen = TRUE;

```

```

if (fullScreen) {
    DEVMODE dmScreenSettings;
    memset(&dmScreenSettings, 0, sizeof(DEVMODE));
    dmScreenSettings.dmSize = sizeof(dmScreenSettings);
    dmScreenSettings.dmPelsWidth = width;
    dmScreenSettings.dmPelsHeight = height;
    dmScreenSettings.dmBitsPerPel = bits;
    dmScreenSettings.dmFields = DM_BITSPERPEL |
                                DM_PELSWIDTH |
                                DM_PELSHEIGHT;

    if (ChangeDisplaySettings(&dmScreenSettings,
                             CDS_FULLSCREEN) != DISP_CHANGE_SUCCESSFUL) {

        MessageBox(NULL, "Display mode failed",
                  NULL, MB_OK);
        fullScreen = FALSE;
    }

} // end of if (fullScreen)

if (fullScreen) {
    dwExStyle = WS_EX_APPWINDOW;
    dwStyle = WS_POPUP;
    ShowCursor(FALSE);
} else {
    dwExStyle = WS_EX_APPWINDOW | WS_EX_WINDOWEDGE;
    dwStyle = WS_OVERLAPPEDWINDOW;
}

AdjustWindowRectEx(&windowRect,
                  dwStyle, FALSE, dwExStyle);

hWnd = CreateWindowEx(0,

```

```

    "MyClass",
    "The OpenGL Window Application",
    dwStyle | WS_CLIPCHILDREN |
    WS_CLIPSIBLINGS,
    0, 0,
    windowRect.right - windowRect.left,
    windowRect.bottom - windowRect.top,
    NULL,
    NULL,
    hInstance,
    NULL) ;

if (!hWnd)
    return 0;

ShowWindow(hWnd, SW_SHOW);
UpdateWindow(hWnd);

while(!done) {

    PeekMessage(&msg, hWnd, 0, 0, PM_REMOVE);

    if (msg.message == WM_QUIT)
        done = TRUE;
    else {

        if (keyPressed[VK_ESCAPE])
            done = TRUE;
        if(keyPressed['l'] || keyPressed['L'])
            left_first = TRUE;
        if (keyPressed['r'] || keyPressed['R'])
            left_first = FALSE;

        display();
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
}

```

```

    }

} // end of while(!done)

return (int) msg.wParam;

} // end of WinMain(HINSTANCE, HINSTANCE, LPSTR, int)

```

Çizilecek olan üçkenlerin sırası, kesişikleri bölgenin rengini etkiler. Soldaki üçken çizildiğinde, çiyan fragmanı (kaynak veya source) sarı fragman ile karıştırılır. Tabi bu sarı fragman önceden frame buffer'ın (hedef veya destination) içinde bulunmaktadır. Sağ taraftaki üçken ilk çizilirse, sarı çiya ile karıştırılacaktır. Alfa faktörünün 0.75 olmasından dolayı, kaynak için blending faktör 0.75, hedef için ise $1.0 - 0.75 = 0.25$ olacaktır.

2.31 Üç Boyutlu Nesnelerde Blending

Üç boyutlu nesneleri saydam olarak çizeceğinizde, çizdiğiniz poligonlara bağlı olarak, farklı görünümler elde edebilirsiniz. Çizim sırasını belirlerken, depth buffer'ını da düşünmeniz gerekmektedir. Depth buffer'ı pencere içinde bulunan nesnenin bir parçası ile bakış noktası arasındaki mesafeyi kontrol eder. Çizilecek olan nesne bakış noktasına yaklaştığında, bu nesne için çizilecek olan yeni renk nesne üzerinde yer alır ve bu durumda nesnenin derinlik değeri depth buffer'ı içerisinde saklı tutulur. Bu yöntem ile saklı yani gözükmeyen nesnelerin parçaları gereksiz yere çizilmez ve blending için kullanılmaz.

Aynı ekranda saydam olan bir cisim ile saydam olmayan bir cismi beraber çizmek istediğinizde, saydam olmayan bir nesnenin arkasında bulunan bir nesnenin saklı tutulan yüzeyini silebilmek için depth buffer'ını kullanırsınız. Eğer saydam olamayan bir cismin arkasında ister saydam olsun veya olmasın herhangi bir nesnenin parçası saklı tutuluyorsa, uzak olan nesnelerin elimine edebilmek için depth buffer'ına ihtiyacınız

olucaktır. Fakat saydam bir cisim bakış noktasına daha yakın olduğunda, bu cismi diğer saydam olmayan cisimler ile blend etmek zorundasınız. Bunu yapabilmek için depth buffer'ını read-only modunda kullanmak gereklidir. Çizilecek üç boyutlu nesnelerin çizim sırasında önemlidir. O zaman ilk olarak çizilecek olan nesneler saydam olmayan cisimlerdir tabi bu sırada depth buffer'ı normal olarak işlemektedir. Bu depth buffer'ını read-only moduna getirerek, derinlik değerlerini saklayın. Saydam nesneler çizildiğinde, onların derinlik değerleri saydam olmayan nesneler tarafından üretilen değerler ile kıyaslanır ve böylece saydam olan nesneler çizilmez tabi bu nesneler saydam olmayan nesnelerin arkasında ise... Eğer saydam nesneler bakış noktasına yakın olduğunda, saydam olmayan nesneler saklamazlar, çünkü depth buffer değerleri değişmemiştir. Bunun yerine bu nesneler saydam olmayan nesneler ile blend edilmiştir. Depth buffer'ını yazılabilir olup olmadığını kontrol edebilmek için, **glDepthMask()** fonksiyonu kullanılır.

```
void glDepthMask(GLboolean flag);
```

flag parametresine, GL_FALSE değerini atarsanız, buffer read-only modunda olacaktır, fakat GL_TRUE olarak atarsanız yazılabilir bir buffer olacaktır.

Örnek 2-7, bize bu anlattıklarımızın küçük bir uygulamasını gösterecek bir program kodunu anlatmaktadır. Ekran içerisinde bir grid üzerinde bulunan saydam ve kırmızı renkli bir küp ile onun biraz ötesinde bulunan mavi renkli bir küre çizilmiştir ve bu çizilen nesneler y ekseni etrafında dönmektedir.

Örnek 2-7

```
#include <stdlib.h>
#include <GL/glut.h>
#include <GL/glu.h>

GLfloat cube_ambient[] = {0.23, 0.0, 0.0, 1.0};
GLfloat cube_diffuse[] = {0.44, 0.0, 0.0, 1.0};
```

```

GLfloat cube_specular[] = {0.33, 0.33, 0.52, 1.0};
GLfloat cube_emission[] = {0.0, 0.0, 0.0, 0.0};
GLfloat cube_shininess = 50;

GLfloat sphere_ambient[] = {0.0, 0.0, 0.39, 1.0};
GLfloat sphere_diffuse[] = {0.0, 0.47, 0.5, 1.0};
GLfloat sphere_specular[] = {0.0, 0.64, 0.34, 1.0};
GLfloat sphere_emission[] = {0.0, 0.0, 0.2, 0.0};
GLfloat sphere_shininess = 38;

GLfloat rotate_y = 0.0;

void draw_net(GLfloat size, GLint LinesX, GLint LinesZ) {

    int xc, zc;

    glBegin(GL_LINES);
        for (xc = 0; xc < LinesX; xc++)
        {
            glVertex3f(
                -size / 2.0 + xc / (GLfloat)(LinesX-1)*size,
                0.0,
                size / 2.0);
            glVertex3f(
                -size / 2.0 + xc / (GLfloat)(LinesX-1)*size,
                0.0,
                size / -2.0);
        } // end of for (xc = 0; xc < LinesX; xc++)

        for (zc = 0; zc < LinesZ; zc++)
        {
            glVertex3f(
                size / 2.0,
                0.0,
                -size / 2.0 + zc / (GLfloat)(LinesZ-1)*size);
            glVertex3f(
                size / -2.0,

```

```

        0.0,
        -size / 2.0 + zc / (GLfloat)(LinesZ-1)*size);
    } // end of for (zc = 0; zc < LinesZ; zc++)

    glEnd();

    return;
} // end of void draw_net(GLfloat, GLint, GLint)

void init(void) {

    GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};
    GLfloat light_ambient[] = {1.0, 1.0, 1.0, 1.0};
    GLfloat light_diffuse[] = {1.0, 1.0, 1.0, 1.0};
    GLfloat light_specular[] = {1.0, 1.0, 1.0, 1.0};

    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_SMOOTH);

    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_DEPTH_TEST);

    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

    return;
} // end of init(void)

void reshape(int w, int h) {

    glViewport(0, 0, (GLsizei)w, (GLsizei)h);
    glMatrixMode(GL_PROJECTION);
}

```

```

glLoadIdentity();
gluPerspective(60.0, (GLfloat)w/(GLfloat)h, 1.0, 20.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(0.0, 2.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

return;
} // end of void reshape(GLfloat, GLfloat)

void display(void) {

    glClear(GL_COLOR_BUFFER_BIT |
             GL_DEPTH_BUFFER_BIT);

    glPushMatrix();

    glRotatef(rotate_y, 0.0, 1.0, 0.0);

    glPushMatrix();
    glTranslatef(1.0, 0.5, 0.0);
    glMaterialfv(GL_FRONT, GL_AMBIENT,
                  sphere_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE,
                  sphere_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR,
                  sphere_specular);
    glMaterialfv(GL_FRONT, GL_EMISSION,
                  sphere_emission);
    glMaterialfv(GL_FRONT, GL_SHININESS,
                  &sphere_shininess);
    glutSolidSphere(0.5, 16, 16);
    glPopMatrix();

    glPushMatrix();
    glTranslatef(-1.0, 0.5, 0.0);
    glMaterialfv(GL_FRONT, GL_AMBIENT,

```

```

            cube_ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE,
            cube_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR,
            cube_specular);
glMaterialfv(GL_FRONT, GL_EMISSION,
            cube_emission);
glMaterialfv(GL_FRONT, GL_SHININESS,
            &cube_shininess);
 glEnable(GL_BLEND);
 glDepthMask(GL_FALSE);
 glBlendFunc(GL_SRC_ALPHA, GL_ONE);
 glutSolidCube(1.0);
 glDepthMask(GL_TRUE);
 glDisable(GL_BLEND);
 glPopMatrix();

glDisable(GL_LIGHTING);
	glColor3f(1.0, 1.0, 1.0);
draw_net(5, 11, 11);
 glEnable(GL_LIGHTING);

glPopMatrix();

glutSwapBuffers();

return;
} // end of void display(void)

void rotate(void) {
    rotate_y += 0.5;

    if (rotate_y > 360.0)
        rotate_y -= 360.0;
}

```

```

    glutPostRedisplay();

    return;
} // end of rotate_teapot(void)

void keyboard(unsigned char key, int x, int y) {

    switch(key) {
        case 'R':
        case 'r':
            rotate_y = 0;
            glutPostRedisplay();
            break;
        case 27:
            exit(0);
    }

    return;
} // end of void keyboard(unsigned char, int, int)

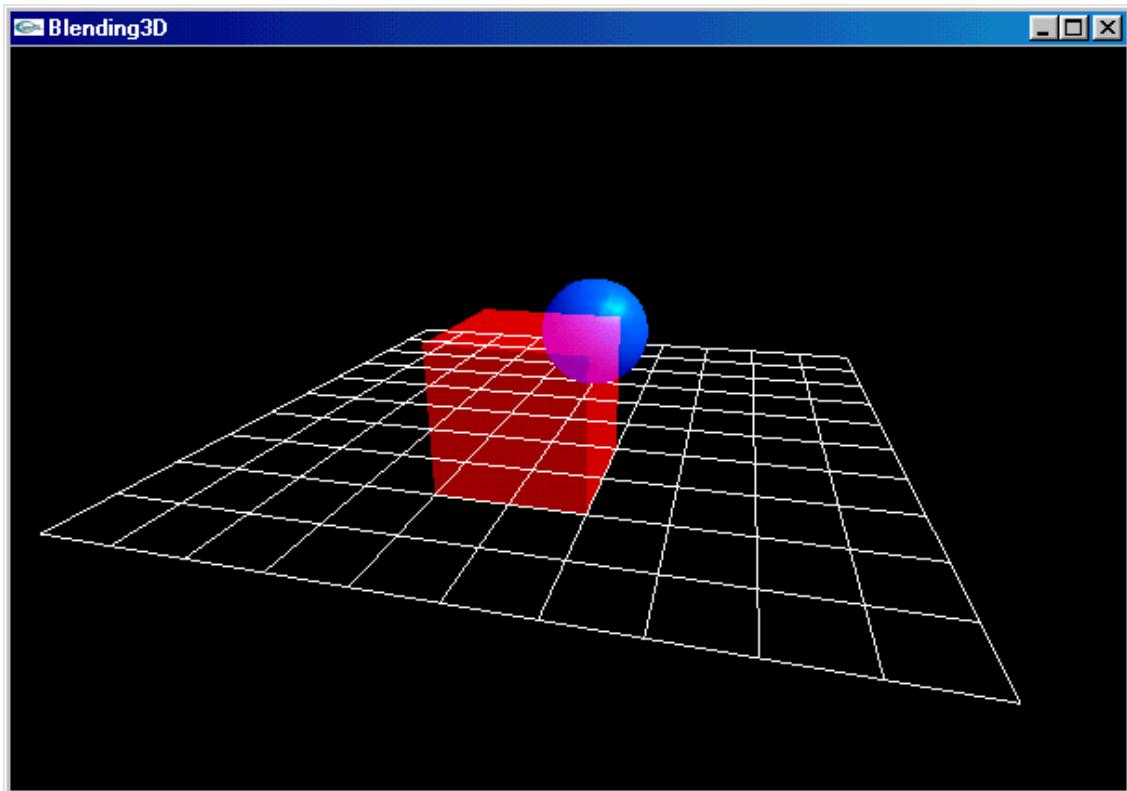
int main(int argc, char** argv) {

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(600, 400);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Teapot");
    init();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutIdleFunc(rotate);
    glutMainLoop();

    return 0;
} // end of int main(int, char**)

```

Aşağıdaki resim 2-14, Örnek 2-7' deki kodu çalıştırıldığınızda ortaya çıkacak sonucu göstermektedir.

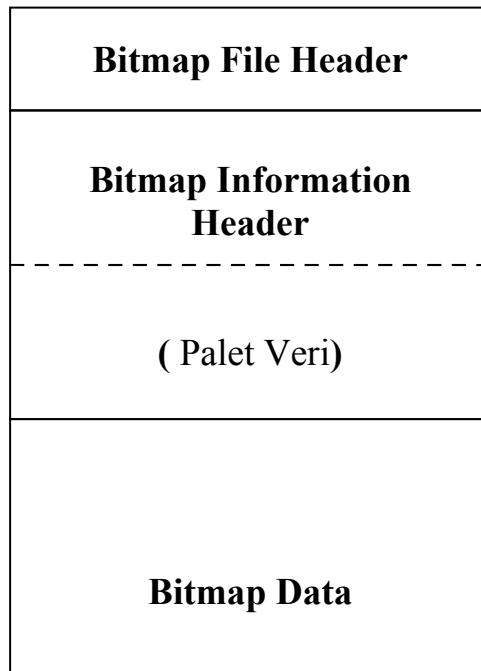


Resim 2-14

2.32 Bitmap Dosyaları

İleride açıklayacağımız texture mapping' de, kullanacağımız resim dosyalarının adı bitmap resim dosyalarıdır. Bu dosyaların uzantıları bmp olarak bilinmektedir. Bitmap dosyalarında yada BMP dosyalarındaki en büyük şey, Windows işletim sistemini kullanan bir kişinin bu dosyaları yaratabilir, içeriğini değiştirebilir ve gözlenebilir olmasıdır. BMP dosyaları herhangi bir sıkıştırma şemalarını kullanmazlar bu sebepten dolayı boyutlarında küçük bir büyümeye gözükebilir. Fakat sıkıştırma şemasından yoksun olmanın anlamı bu dosya formatının kolaylıkla okunabilir ve kullanılabilir olmasıdır.

Resim 2-15 bir BMP dosyasının yapısını göstermektedir. Bitmap file header, Bitmap information header ve bitmap data...



Resim 2-15

Bitmap dosya Bitmap File Header dosyasını yüklemek için kullanılan veri yapısı BITMAPFILEHEADER' dır ve bu yapı aşağıda gösterilmektedir.

```
typedef struct tagBITMAPFILEHEADER
{
    WORD   bfType;      // specifies the file type; must be BM (0x4D42)
    DWORD  bfSize;     // specifies the size in bytes of BMP file
    WORD   bfReserved1; // reserved; must be zero
    WORD   bfReserved2; // reserved; must be zero
    DWORD  bOffBits;   // specifies the offset, in bytes, from
                      // BITMAPFILEHEADER structure to the bitmap bits
} BITMAPFILEHEADER;
```

Bu yapı ile ilgili olarak, bir bmp dosyası yükleyeceğiniz zaman dosyanın uzantısı-

na dikkat etmeniz gerektidir. Bunu, dosyanın BITMAPFILEHEADER yapısının bir üyesine bakarak yani *bfType*' e bakarak dosyanın uzantısının bmp olup olmadığını anlayabilirsiniz. Bu üye bitmap dosyaları için her zaman 0x4D42 değerine eşittir. Bu sayıyı önceden Windows proglamlaması esnasında görmüştünüz. Bir bmp uzantılı dosyayı açarken bu sayı ile kontrol yapılmaktadır. *bfSize*, bize dosyanın boyutunu, *bOffBits* ise bize dosyanın başından itibaren Bitmap Data'nın nereden başladığını içeren bir offset bilgisini gösterir.

Bir sonraki inceleyeceğimiz veri bölgesi, her BMP dosyasına özgü olan bilgileri içerir. Bu bölge, BMP dosyasının 8-bitlik ve bir palet verisine sahip olmasına göre bir veya iki parçaaya ayrılmaktadır. Biz palet verisi ile ilgilenmeyeceğimiz için sadece BITMAPINFOHEADER yapısını okuyacağız.

```
typedef struct tagBITMAPINFOHEADER
{
    DWORD biSize; // number of bytes required by the structure
    LONG biWidth; // width of the bitmap, in pixels
    LONG biHeight; // height of the bitmap, in pixels
    WORD biPlanes; // number of color planes, must be 1
    WORD biBitCount; // number of bits per pixel must be 1, 4, 8,
                     // 16, 24, 32
    DWORD biCompression; // type of compression
    DWORD biSizeImage; // size of image in bytes
    LONG biXPelsPerMeter; // number of pixels per meter in x-axis
    LONG biYPelsPerMeter; // number of pixels per meter in y-axis
    DWORD biClrUsed; // number of colors used by the bitmap
    DWORD biClrImportant; // number of colors that are important
} BITMAPINFOHEADER;
```

Bu yapının yanındaki açıklamaları okumak kendini açıklayacak kadar yeterlidir. Burada önemli olan bir BMP dosyasını nasıl belleğe yükleyeceğimizdir. BMP dosyasının Bitmap Data kısmında kırmızı ve mavi renk değerlerinin yerleri değişmiştir yani dosya bildığınız gibi kırmızı, yeşil ve mavi olarak sıralı değilde mavi, yeşil kırmızı ola-

rak okunmalı ve kırmızı ile mavi değerlerin yerlerinin değişmesi gerekmektedir. Bu işlem, dosya okunurken yapılmadır. Bir bitmap dosyası yaratılacaksa bu işlemi tersinden yapmak yani öncelikle mavi sonra yeşil ve en son olarak kırmızı renk değeri dosyaya yazılmalıdır çünkü bitmap dosya yapısına uymamız gereklidir. Örnek 2-8' de 24-bitlik bir bitmap dosyasını yükleye ve bitmap dosyasını oluşturan program kodu ve onun başlık dosyası bulunmaktadır.

Örnek 2-8 Bitmap.h ve Bitmap.c

```
// Bitmap.h

#ifndef _BITMAP_H
#define _BITMAP_H
#include <windows.h>
typedef unsigned char *BITMAPIMAGE;
#define BITMAP_ID 0x4D42
#define FILE_NOT_OPENED_TO_WRITE -12
extern BITMAPIMAGE
Bitmap_LoadBitmapFile(const char*, const BITMAPINFOHEADER*);
extern int
Bitmap_WriteBitmapFile(const char*, LONG, LONG, BITMAPIMAGE);
#endif

// Bitmap.c

#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "Bitmap.h"

BITMAPIMAGE
```

```

Bitmap_LoadBitmapFile(const char* filename,
                      const BITMAPINFOHEADER* bitmapInfoHeader)
{

    FILE *filePtr;
    BITMAPFILEHEADER bitmapFileHeader;
    BITMAPIMAGE bitmapImage;
    DWORD imageIdx;
    unsigned char tempRGB;

    if ((filePtr = fopen(filename, "rb")) == NULL) {
        MessageBox(NULL, "File not found.", "Error", MB_OK);
        return NULL;
    }

    fread(&bitmapFileHeader,
          sizeof(BITMAPFILEHEADER), 1, filePtr);

    if (bitmapFileHeader.bfType != BITMAP_ID) {
        MessageBox(NULL,
                  "Invalid file extention.",
                  "Error", MB_OK);
        fclose(filePtr);
        return NULL;
    }

    fread((BITMAPINFOHEADER*)bitmapInfoHeader,
          sizeof(BITMAPINFOHEADER), 1, filePtr);

    fseek(filePtr, bitmapFileHeader.bfOffBits, SEEK_SET);

    bitmapImage = (BITMAPIMAGE)
                  malloc(bitmapInfoHeader->biSizeImage);

    if (!bitmapImage) {
        MessageBox(NULL, "Out of memory", "Error", MB_OK);
        fclose(filePtr);

```

```

        return NULL;
    }

    fread((BITMAPIMAGE)bitmapImage,
           sizeof(*bitmapImage),
           bitmapInfoHeader->biSizeImage, filePtr);

    if (!bitmapImage) {
        MessageBox(NULL, "File not read.", "Error", MB_OK);
        free(bitmapImage);
        fclose(filePtr);
        return NULL;
    }

    for(imageIdx = 0;
        imageIdx < bitmapInfoHeader->biSizeImage;
        imageIdx+=3)
    {
        tempRGB = bitmapImage[imageIdx];
        bitmapImage[imageIdx] = bitmapImage[imageIdx + 2];
        bitmapImage[imageIdx + 2] = tempRGB;
    }

    fclose(filePtr);

    return bitmapImage;
}

int
Bitmap_WriteBitmapFile(const char* filename,
                      LONG width, LONG height,
                      BITMAPIMAGE imageData)
{
    FILE *filePtr;
    BITMAPFILEHEADER bitmapFileHeader;
    BITMAPINFOHEADER bitmapInfoHeader;

```

```

DWORD imageIdx;
unsigned char tempRGB;

if ((filePtr = fopen(filename, "wb")) == NULL)
    return FILE_NOT_OPENED_TO_WRITE;

bitmapFileHeader.bfSize = sizeof(BITMAPFILEHEADER);
bitmapFileHeader.bfType = BITMAP_ID;
bitmapFileHeader.bfReserved1 = 0;
bitmapFileHeader.bfReserved2 = 0;
bitmapFileHeader.bfOffBits = sizeof(BITMAPFILEHEADER) +
    sizeof(BITMAPINFOHEADER);
bitmapInfoHeader.biSize = sizeof(BITMAPINFOHEADER);
bitmapInfoHeader.biPlanes = 1;
bitmapInfoHeader.biBitCount = 24;
bitmapInfoHeader.biCompression = BI_RGB;
bitmapInfoHeader.biSizeImage = width * height * 3;
bitmapInfoHeader.biXPelsPerMeter = 0;
bitmapInfoHeader.biYPelsPerMeter = 0;
bitmapInfoHeader.biClrImportant = 0;
bitmapInfoHeader.biWidth = width;
bitmapInfoHeader.biHeight = height;

for(imageIdx = 0;
    imageIdx < bitmapInfoHeader.biSizeImage;
    imageIdx += 3)
{
    tempRGB = imageData[imageIdx];
    imageData[imageIdx] = imageData[imageIdx + 2];
    imageData[imageIdx + 2] = tempRGB;
}

fwrite((BITMAPFILEHEADER*)&bitmapFileHeader,
       1, sizeof(BITMAPFILEHEADER), filePtr);

fwrite((BITMAPINFOHEADER*)&bitmapInfoHeader,
       1, sizeof(BITMAPINFOHEADER), filePtr);

```

```

        fwrite((BITMAPIMAGE) imageData,
               sizeof(*imageData),
               bitmapInfoHeader.biSizeImage, filePtr);

    fclose(filePtr);
    return 0;
}

```

2.33 Texture Mapping

Texture mapping konusu OpenGL' de çok geniş bir konudur, bu yüzden sadece gerekli olan kısımları inceleyeceğiz. Texture mapping, poligonlar üzerine resimleri yapıştırmanızı sağlamaktadır. Mesela, bir küre yarattınız ve bu küre üstüne 3 boyutlu olarak bir dünya resmi yağıstırabilirsiniz. Texture mapping' i tüm 3 boyutlu oyunlarda görebilirsiniz ve bu method oyulara daha çok gerçeklik katmaktadır.

Texture map'leri veriyi tutan kare dizilerinden oluşmaktadır. Her veri parçasına *texel* denmektedir. Bunlar veriyi taşıyan kare dizileri plmalarına karşın, texture map'leri karesel olmayan nesneler üzerine map edilebilirler, mesela silindirler ve küreler gibi...

Tasarımcılar genellikle, kendi grafik çalışmalarında iki boyutlu texture mapping yöntemini kullanırlar. Fakat tek boyutlu ve üç boyutlu texture kaplamalar da mevcuttur, fakat biz bunlar ile ilgilenmeyeceğiz. Konu olarak basit bir şekilde iki boyutlu texture mappingi inceleyeceğiz.

2.34 İki Boyutlu Texture

Bir dosyadan, texture mapping için kullanacağınız bir dosyayı belleğe yükledikten sonra, bunu texture map olarak tanıtmaz gerekmektedir. Texture map' in boyutları bu işi yapabilmeniz için hangi fonksiyonları kullanacağınızı belirtmektedir. Eğer 2 bo-

yutlu bir texture map işlemi yapmak istiyorsanız **glTexImage2D()** fonksiyonunu kullanın. Bir boyutlu texture map işlemi yapmak istiyorsanız **glTexImage1D()** fonksiyonunu, üç boyutlu olarak işlem yapmak istiyorsanız **glTexImage3D()** fonksiyonunu kullanabilirsiniz.

```
void glTexImage2D(GLenum target, GLint level, GLint internalFormat,  
                  GLsizei width, GLsizei height, GLint border,  
                  GLenum type, const GLvoid* texels);
```

target parametresi **GL_TEXTURE_2D** veya **GL_PROXY_TEXTURE_2D** olarak ayarlanabilir. Fakat bizim için bu parametreyi **GL_TEXTURE_2D** yapmak daha uygundur. *level* parametresi texture map' in çözünürlüğünü belirler. Sadece tek bir çözünürlük kullanmak istiyorsanız bu parametreyi 0 yapın. Şu an için bir çok çözünürlük modunun kullanımını tartışmayacağız. *internalFormat* parametresi 1' den 4' e kadar veya size gösteremeyeceğimiz 38 sabit değerden birini kullanan sayıya eşittir. Bizim sadece kullanabileceklerimiz **GL_LUMINANCE**, **GL_LUMINANCE_ALPHA**, **GL_RGB** ve **GL_RGBA** olmaktadır. *width* ve *height* parametreleri texture map' in genişlik ve boyutunu belirlerler. Bu değerler 2' nin üssü bir değer olmalıdır. *border* parametresi texture çevresinde bir sınırın olup olmadığını bildirmektedir. Sınır var ise bu değer 1, yok ise değer 0 olmalıdır. *format* parametresi, texture map' in verisinin formatını tanımlar ve bu format **GL_COLOR_INDEX**, **GL_RGB**, **GL_RGBA**, **GL_RED**, **GL_GREEN**,

GL_BLUE, **GL_ALPHA**, **GL_LUMINANCE** veya **GL_LUMINANCE_ALPHA** olabilir. *type* parametresi ise bu verinin tipini belirtir ve **GL_BYTE**, **GL_UNSIGNED_BYTE**, **GL_SHORT**, **GL_UNSIGNED_SHORT**, **GL_INT**, **GL_UNSIGNED_INT**, **GL_FLOAT** veya **GL_UNSIGNED_FLOAT** olabilir. Son parametre olan *texel* parametresi bir işaretçidir. Bu işaretçi texture mapping için kullanacağınız resmi işaret eder. Mesela bir RGBA resmi yüklediniz ve onu gösteren işaretçi ise *textureData* olsun. Bu dizinin boyutu *textureWidth* ve *textureHeight* olsun. **glTexImage2D()** fonksiyonu ise aşağıdaki gibi kullanılabilir.

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, textureWidth, textureHeight,  
0, GL_RGBA, GL_UNSIGNED_BYTE, textureData);
```

2.35 Texture Nesneleri

Texture nesneleri, texture verilerini saklamanıza ve onları kullanmak için hazır durumda bekletmenizi sağlamaktadır. Bu nesneleri kullanarak, bir kerede bir çok texture’yi belleğe yükleyebilir ve çizim esnasında bunların herhangi birine referans verebilirisiniz.

Texture nesnelerini kullanamak için ilk yapacağınız şey bir texture ismi yaratmanızdır. Texture isimleri sıfırdan farklı işaretsiz bir tamsayılardır. Aynı texture ismini iki kere kullanmamak için **glGenTextures()** fonksiyonunu kullanın.

```
void glGenTextures(GLsizei n, GLuint *textureNames);
```

parametre *n*, kaç tane texture ismi yaratılacağını bildirmektedir ve bu isimle bir dizinin içine yerleştirilecek olan bir *textureNames* parametresinin gösterdiği dizi olacaktır. Örnek olarak vermek gerkirse, texture isimlerini saklayacağınız bir 3 elemanlı diziniz mevcutsa, fonksiyon aşağıdaki gibi kullanılacaktır.

```
unsigned int textureNames[3];  
...  
glGenTextures(3, textureNames);
```

Texture isimlerini yarattıktan sonra bu isimlerin texture verilerine verilmesi veya bağlanması gerekmektedir. Bunu **glBindTexture()** fonksiyonu ile gerçekleştirebilir.

```
void glBindTexture(GLenum target, GLuint textureName);
```

Bu fonksiyonu ilk olarak kullandığınızda, yeni bir texture nesnesi varsayılı değerle-re sahip olarak yaratılır. OpenGL bu isim bağlama işleminden sonra texture nesnesinde-

ki varsayılı değerleri yeniden tanımlayacaktır. *target* parametresi bu fonksiyon için GL_TEXTURE_1D, GL_TEXTURE_2D veya GL_TEXTURE_3D olabilir. Bu isim bağlama işleminden sonra texture nesnesini o an kullanılabilecek texture durumuna getirmek için **glBindTexture()** fonksiyonunu kullanın. Mesela birkaç texture nesnesi yarattığınızı düşünelim ve bu nesnelerin texture verilerini ve texture özelliklerini bu nesnelere bağladınız. Ekranda poligonları çizeceğiniz zaman, **glBindTexture()** ile texture'ü belirleyerek, OpenGL' e belirlemiş olduğunuz texture' ü söyleyebilirsiniz. Aşağıdaki kod parçası, texture özellikleri ayarlandıktan sonra, ikinci texture nesnesini current texture olacağını, OpenGL' e anlatmaktadır.

```
unsigned int textureNames[3];  
...  
 glGenTextures(3, textureNames);  
...  
 glBindTexture(GL_TEXTURE_2D, textureNames[1]);  
 // texture verisi ve özelliklerini ayarla  
 glBindTexture(GL_TEXTURE_2D, textureNames[1]);  
 // nesneyi çiz
```

2.36 Texture Filtreleme

Texture resimleri, poligonlara uygulandıktan sonra, biçim bozuk hale dönüşür ve sonuç olarak küçük bir piksel texel' in sadece küçük bir parçasını veya texel' leri gösterebilir. Resmin son hali için, OpenGL' e bu pikselleri ve texel' lerin nasıl hesaplanacağını texture filtreleme kullanarak anlatabilirsiniz.

Texture filtrelemede, magnification, bir pikselin bir texel parçasını göstermesi anlamını taşımaktadır ve minification ise bir pikselin texel' leri göstermesi anlamına gelir. OpenGL' e bu her iki durumu nasıl ele almak istediğiniz **glTexParameter()** fonksiyonunu çağırarak anlatabilirsiniz.

```
void glTexParameter(GLenum target, GLenum pname, GLint param);
```

target parametresi kullandığınız texture boyutuna bağlıdır ve GL_TEXTURE_1D, GL_TEXTURE_2D veya GL_TEXTURE_3D değerleri alabilir. Magnification filtrelemeye *pname* parametresi GL_TEXTURE_MAG_FILTER değerini ve minification filtrelemeye *pname* parametresi GL_TEXTURE_MIN_FILTER değerini almaktadır. *param* parametresinin alacağı değerler tablo 2-5' te gösterilmektedir.

Tablo 2-5 Texture Filtre Değerleri

| Filtre | Açıklama |
|---------------------------|---|
| GL_NEAREST | Texture edilecek olan pikselin merkezine en yakın olan texel' i kullanır. |
| GL_LINEAR merkezi- | Çizilecek olan pikselin ne en yakın olan texel'lerin ağırlıklı ortalamasını kullanır. |
| GL_NEAREST_MIPMAP_NEAREST | GL_NEAREST filtrelemeyi ve poligon çözünürlüğünne en yakın imajı kullanır. |
| GL_NEAREST_MIPMAP_LINEAR | GL_LINEAR filtrelemeyi ve poligon çözünürlüğünne en yakın imajı kullanır. |
| GL_LINEAR_MIPMAP_NEAREST | GL_NEAREST filtrelemeyi ve poligon çözünürlüğünne en yakın olan iki mipmap' ler arasındaki ağırlıklı ortalamayı kullanır. |
| GL_LINEAR_MIPMAP_LINEAR | GL_LINEAR filtrelemeyi ve poligon çözünürlüğünne en yakın olan iki mipmap' ler arasındaki ağırlıklı ortalamayı kullanır. |

2.37 Texture Fonksiyonları

OpenGL, texture fonksiyonları aracılığı ile texture map renklerinin davranışını belirler. Her texture için, **glTexEnv()** fonksiyonu ile dört tane texture fonksiyonundan seçim yapabilirsiniz.

```
void glTexEnv(GLenum target, GLenum pname, GLint param);
```

target parametresi GL_TEXTURE_ENV değerine eşit olmalıdır. *pname* parametresine ise GL_TEXTURE_ENV_MODE değerini aktararak, frame buffer'ındaki renkler ile texture'lerin nasıl birleştirileceğini belirleyebilirsiniz. *param* parametresi için kullanılacak değerler tablo 2-6'da belirtilmiştir.

Tablo 2-6

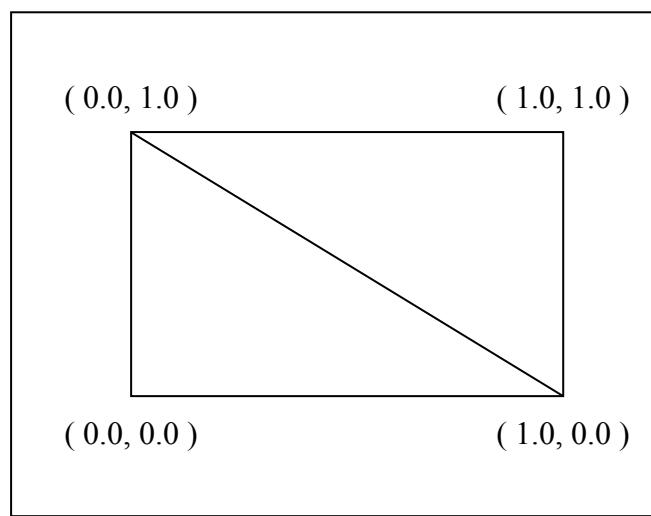
| Mod | Açıklama |
|-------------|--|
| GL_BLEND | Texture rengi piksel rengi ile çarpılır ve sabit bir renk ile birleştirilir. |
| GL_DECAL | Texture, mevcut olan pikseller ile yer değiştirir. |
| GL_MODULATE | Texture rengi piksel rengi ile çarpılır. |

glTexEnv()'ini varsayılı değeri GL_MODULATE' tir. Aşağıdaki tek satırlık kod texture fonksiyonunu GL_DECAL'e ayarlamaktadır.

```
glTexEnv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
```

Ekranınızı çizdiğinizde, her verteksin texture koordinatlarını belirtmeniz gereklidir.

Texture koordinatları, bir nesneye ait olan texture map' in her bir texel' i nereye konumlanacağını belirtmek için kullanılır. Bir texture koordinatı olan (0, 0) texture ' un sol alt köşesini, (1, 1) ise sağ üst köşesini belirtir. İki boyutlu texture' lerde bu koordinatlar (s, f) formundadır. Burada s ve f değerleri 0' dan 1' e doğru değişir. Resim 2-16 poligonun her vertexi için texture koordinatlarını göstermiştir. Bu koordinatlar her vertex render edilmeden önce, kendileri ile ilişkilendirilmiş olan texture koordinatları belirlenmelidir.



Resim 2-16

Texture koordinatlarını **glTexCoord*()** fonksiyonu ile belirlersiniz. Biz konu içerisinde iki boyutlu texture mapping yapacağımızdan dolayı bu fonksiyonun aşağıdaki iki boyutlu olanını kullanacağız.

```
void glTexCoord2f(GLfloat s, GLfloat t);
```

Aşağıda bu fonksiyonun basit bir kullanımını göstermektedir.

```
glBegin(GL_QUADS);
    glTexCoord2f(0.0, 0.0); glVertex3f(-0.5, 0.5, 0.5);
    glTexCoord2f(1.0, 0.0); glVertex3f(0.5, 0.5, 0.5);
    glTexCoord2f(1.0, 1.0); glVertex3f(0.5, 0.5, -0.5);
    glTexCoord2f(0.0, 1.0); glVertex3f(-0.5, 0.5, -0.5);
glEnd();
```

```

glTexCoord2f(0.0, 1.0); glVertex3f(-0.5, 0.5, -0.5);
glEnd();

```

Texture mapping konusu bu kadar... Örnek ise Windows platformunda gerçekleşti- rilmiş olan dönen küpün bir benzeri vardır fakat bu örnek GLUT kütüphanesi fonksiyonları kullanılarak yapılmıştır. Bu kübün her bir yüzeyine bir bitmap dosyasından yüklenmiş resim yapıştırılmıştır. Resim ise satranç tahtasına benzeyen bir resim kübüne her yüzeyi texture kaplama yapılmıştır. Kübü, farenizin sol tuşuna basık tutarak döndürebilirsiniz. Fareyi hareket halinde iken serbest bıraktığınızda küb sabit bir hızda dönmeye devam edecektir. Örnek 2-9' u inceleyiniz.

Örnek 2-9

```

#include <windows.h>
#include <stdlib.h>
#include <GL/glut.h>

#include "Bitmap.h"
#include "HiResTimer.h"

#define BITMAP_ID 0x4D42

BITMAPINFOHEADER bitmapInfoHeader;
BITMAPIMAGE bitmapData = NULL;

unsigned int texture;

int last_x, last_y;
GLfloat rotationX = 0.0,
        rotationY = 0.0,
        timeElapsed = 0.0;

GLfloat dx = 0.0, dy = 0.0;

HiResTimer timer = NULL;

```

```

void DrawTextureCube(GLfloat xPos,
                     GLfloat yPos,
                     GLfloat zPos)
{
    glPushMatrix();
    glTranslatef(xPos, yPos, zPos);
    glBegin(GL_QUADS); // top face
    glTexCoord2f(0.0, 0.0);
    glVertex3f(-0.5, 0.5, 0.5);
    glTexCoord2f(1.0, 0.0);
    glVertex3f(0.5, 0.5, 0.5);
    glTexCoord2f(1.0, 1.0);
    glVertex3f(0.5, 0.5, -0.5);
    glTexCoord2f(0.0, 1.0);
    glVertex3f(-0.5, 0.5, -0.5);
    glEnd();

    glBegin(GL_QUADS); // front face
    glTexCoord2f(0.0, 0.0);
    glVertex3f(0.5, -0.5, 0.5);
    glTexCoord2f(1.0, 0.0);
    glVertex3f(0.5, 0.5, 0.5);
    glTexCoord2f(1.0, 1.0);
    glVertex3f(-0.5, 0.5, 0.5);
    glTexCoord2f(0.0, 1.0);
    glVertex3f(-0.5, -0.5, 0.5);
    glEnd();

    glBegin(GL_QUADS); // right face
    glTexCoord2f(0.0, 0.0);
    glVertex3f(0.5, 0.5, -0.5);
    glTexCoord2f(1.0, 0.0);
    glVertex3f(0.5, 0.5, 0.5);
    glTexCoord2f(1.0, 1.0);
    glVertex3f(0.5, -0.5, 0.5);
}

```

```

        glTexCoord2f(0.0, 1.0);
        glVertex3f(0.5, -0.5, -0.5);
    glEnd();

    glBegin(GL_QUADS); // left face
        glTexCoord2f(0.0, 0.0);
        glVertex3f(-0.5, -0.5, 0.5);
        glTexCoord2f(1.0, 0.0);
        glVertex3f(-0.5, 0.5, 0.5);
        glTexCoord2f(1.0, 1.0);
        glVertex3f(-0.5, 0.5, -0.5);
        glTexCoord2f(0.0, 1.0);
        glVertex3f(-0.5, -0.5, -0.5);
    glEnd();

    glBegin(GL_QUADS); // bottom face
        glTexCoord2f(0.0, 0.0);
        glVertex3f(0.5, -0.5, 0.5);
        glTexCoord2f(1.0, 0.0);
        glVertex3f(-0.5, -0.5, 0.5);
        glTexCoord2f(1.0, 1.0);
        glVertex3f(-0.5, -0.5, -0.5);
        glTexCoord2f(0.0, 1.0);
        glVertex3f(0.5, -0.5, -0.5);
    glEnd();

    glBegin(GL_QUADS); // back face
        glTexCoord2f(0.0, 0.0);
        glVertex3f(0.5, 0.5, -0.5);
        glTexCoord2f(1.0, 0.0);
        glVertex3f(0.5, -0.5, -0.5);
        glTexCoord2f(1.0, 1.0);
        glVertex3f(-0.5, -0.5, -0.5);
        glTexCoord2f(0.0, 1.0);
        glVertex3f(-0.5, 0.5, -0.5);
    glEnd();
glPopMatrix();

```

```

        return;
    } // end of void DrawTextureCube(GLfloat, GLfloat, GLfloat)

void init(void) {

    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_SMOOTH);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);
    glFrontFace(GL_CCW);
    glEnable(GL_TEXTURE_2D);

    bitmapData =
        Bitmap_LoadBitmapFile("Checkerboard_64.bmp",
                               &bitmapInfoHeader);

    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    glGenTextures(1, &texture);
    glBindTexture(GL_TEXTURE_2D, texture);

    glTexParameteri(GL_TEXTURE_2D,
                    GL_TEXTURE_MAG_FILTER,
                    GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D,
                    GL_TEXTURE_MIN_FILTER,
                    GL_NEAREST);

    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB,
                bitmapInfoHeader.biWidth,
                bitmapInfoHeader.biHeight, 0,
                GL_RGB, GL_UNSIGNED_BYTE, bitmapData);

    return;
} // end of void initialize(void)

void reshape(int width, int height) {

```

```

glViewport(0, 0, width, height);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(60.0,
               (GLfloat)width/(GLfloat)height,
               1.0, 20.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

return;
} // end of void resize(int, int)
void display(void) {

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glPushMatrix();
    glEnable(GL_TEXTURE_2D);
    glTexEnvf(GL_TEXTURE_ENV,
               GL_TEXTURE_ENV_MODE, GL_REPLACE);
    glBindTexture(GL_TEXTURE_2D, texture);
    glTranslatef(0.0, 0.0, -3.0);
    glRotatef(rotationY, 0.0, 1.0, 0.0);
    glRotatef(rotationX, 1.0, 0.0, 0.0);
    DrawTextureCube(0.0, 0.0, 0.0);
    glDisable(GL_TEXTURE_2D);
    glPopMatrix();
    glFlush();
    glutSwapBuffers();
    return;
} // end of void display(void)

void spin(void) {

    rotationX += dx;
    rotationY += dy;
}

```

```

    if (rotationX > 360.0)
        rotationX -= 360.0;

    if (rotationY > 360.0)
        rotationY -= 360.0;

    glutPostRedisplay();
    return;
} // end of void spin(void)

void mouse(int button, int button_state,
           int x, int y)
{
    if (button == GLUT_LEFT_BUTTON &&
        button_state == GLUT_DOWN) {
        glutIdleFunc(NULL);
        if (last_x == x)
            dx = 0.0;
        if (last_y == y)
            dy = 0.0;
        last_x = x;
        last_y = y;
    } // end of if statement
    else
        glutIdleFunc(spin);

    return;
} // end of void mouse(int, int, int, int)

void motion(int x, int y) {
    timeElapsed = HiResTimer_GetElapsed_Seconds(timer);
    rotationX += (float) (y - last_y);
    rotationY += (float) (x - last_x);
}

```

```

dx = (float)(y - last_y) / (1000*timeElapsed);
dy = (float)(x - last_x) / (1000*timeElapsed);

if (rotationX > 360.0)
    rotationX -= 360.0;

if (rotationY > 360.0)
    rotationY -= 360.0;

last_x = x;
last_y = y;

glutPostRedisplay();

return;
} // end of void motion(int, int)

void keyboard(unsigned char key, int x, int y) {

switch(key) {
    case 27:
    case 'q':
        exit(0);
    default:
        break;
} // end of switch(key)

return;
} // end of void keyboard(unsigned char, int, int)

int on_exit(void) {

    free(bitmapData);
    bitmapData = NULL;
    free(timer);
    timer = NULL;
}

```

```

        return 0;
    } // end of void on_exit(void)

int main(int argc, char** argv) {

    timer = HiResTimer_Create();

    if (!HiResTime_Init(timer)) {
        MessageBox(NULL, "Timer initialization failed",
                  0, 0);
        free(timer);
        timer = NULL;
        exit(-1);
    } // end of if (!HiResTime_Init(timer))

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Texture Mapping Application");
    init();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutMotionFunc( motion );
    glutMouseFunc( mouse );

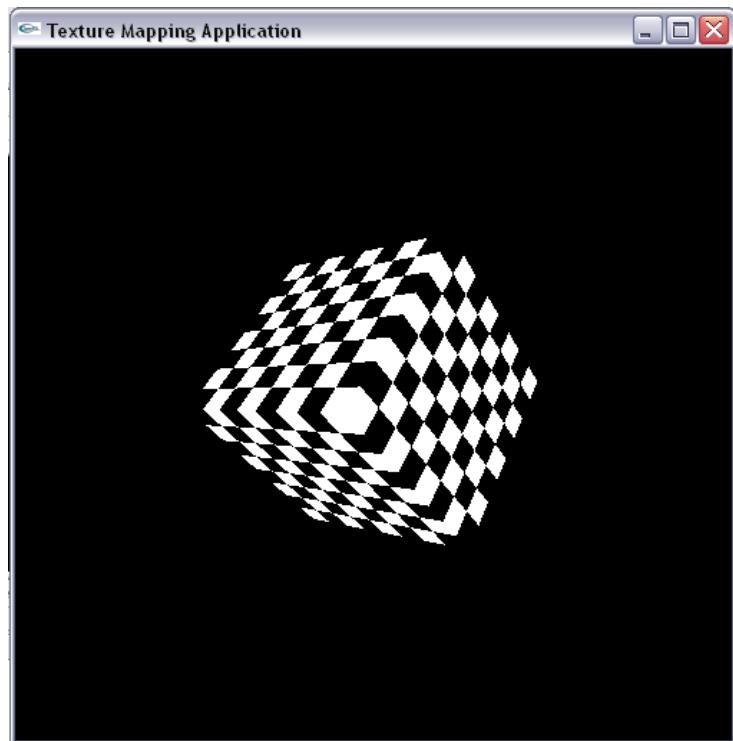
    _onexit(on_exit);

    glutMainLoop();

    return 0;
} // end of int main(int, char**)

```

Resim 2-17, bu kodu çalıştıracağınız zaman, texture mapping ile kaplanmış olan kükü göstermektedir.



Resim 2-17

Kodu inceledikten sonra, kod içerisinde birkaç bilinmedik fonksiyon göreceksiniz. Bunlardan biri **glPixelStorei()** fonksiyonudur. Bu fonksiyonu açıklamadan önce birkaç söylenecek şey var. Geliştirmiş olduğunuz bir projeyi bir makineden diğer bir makineye taşıyorken, projeniz daha yavaş çalışır. Bununla ilgili bir çok faktör olmasına rağmen, bu faktörlerden ilgineceğimiz sadece biri olan hafızadaki alignment' tır. Bazı makineler-
de veri aktarımı daha hızlı yapısın diye, veri 2, 4 veya 8 byte'lık sınırlar çerçevesinde memoride doğru bir pozisyonda konumlanır. Bu durumda, **glPixelStorei()** fonksiyonunu kullanarak, veri pozisyonunu (data alignment) kontrol edebilirsiniz.

```
void glPixelStorei(GLenum pname, Type param);
```

Fonksiyonu kullanmak için, *pname* parametresi GL_PACK_ALIGNMENT veya GL_UNPACK_ALIGNMENT olabilir. *param* parametresi 1, 2, 4 veya 8 olabilir. *pname* parametresini GL_PACK_ALIGNMENT olarak belirlediğinizde, OpenGL' e ve-

rinin nasıl paketleneceğini anlatırsınız. GL_UNPACK_ALIGNMENT parametresini kullandığımızda, OpenGL' e verinin hafızada nasıl paketlenmeyeceği anlatılır.

İlk olarak gördüğünüz diğer fonksiyonlar zaman ile ilgili olan fonksiyonlardır. HiResTimer.h başlık dosyası bu fonksiyonların tanımlamasını yapmaktadır. Zaman ile ilgili fonksiyonları niçin kullandık ve bu fonksiyonların ne işe yaradığını anlatalım.

Zaman, dünyada bir niteliktir, kendi oyun dünyanızda ise bu zaman kavramı bir önem taşımaktadır. Zamanı dikkatli bir şekilde izleyebilmek için bir timer kullanırsınız. Bu timer fiziksel hesaplamalara daha çok gerçeklik kazandırır ve hesaplamaları, çizim esnasındaki her çizilen çerçeveden (frame) bağımsız bir şekilde takip edebilirsiniz.

İki tip timer kullanabilirsiniz: Yüksek performansa sahip olan bir timer veya Windows' un mesaj olarak gönderdiği timer, WM_TIMER. Bu ikisi arasındaki en önemli fark hız ve doğruluktur. Biz yüksek performansa sahip olan timer' ı kullanıcaz çünkü hız ve doğruluğa sahiptir. Eğer bilgisayarınız bunu desteklemiyorsa, bunun yerine WM_TIMER' ı kullanın. Yüksek performansta timer' ı kullanabilmek için makinenizde bir saniyede kaç clock atıldığını belirlemeniz gerekmektedir. Bundan sonra o anki zamanı öğrenip, uygulamanızın ne zaman başladığını gösteren bir değişkenin içine bu zamanı saklaması gereklidir. Bu her bir taslağı gerçekleştirmek için **QueryPerformanceFrequency()** ve **QueryPerformanceCounter()** fonksiyonlarını kullanmalısınız.

```
BOOL QueryPerformanceFrequency(LARGE_INTEGER *lpFrequency);  
BOOL QueryPerformanceCounter(LARGE_INTEGER *lpPerformanceCount);
```

QueryPerformanceFrequency() fonksiyonu bir saniyede kaç clock atıldığını göstermektedir. Clock sayısını *lpFrequency* işaretçisi aracılığı ile elde ederiz. Donanımınız bu fonksiyonu desteklemiyorsa, bu işaretçinin göstereceği değer sıfır olabilir ve fonksiyondan dönen değer sıfır' dır. Eğer donanımınız bu fonksiyonu destekliyorsa fonksiyondan dönen değer sıfırdan farklı olacaktır. **QueryPerformanceCounter()** ise o anki

timer' in current değerini dönderir ve dönen değer *lpPerformanceCount* işaretçisi tarafından gösterilmektedir. Bu fonksiyonun donanım tarafında desteklenip desteklenmemesi gibi durumunda fonksiyonun izlediği tutum, **QueryPerformanceFrequency()** fonksiyonunun izlediği durumdan farksız olacaktır. Her iki fonksiyon da aynı parametre tiplerini almaktadır. Bu parametre tipi 64-bitlik integer olarak ifade edilir ve ismi **LARGE_INTEGER**'dır.

```
typedef union _LARGE_INTEGER {
    struct {
        DWORD LowPart;
        LONG HighPart;
    };
    LONGLONG QuadPart;
} LARGE_INTEGER;
```

LowPart alanı 32-bitlik low-order parçasını ve HighPart ise 32-bitlik high-order parçasını belirtmektedir. QuadPart ise 64-bitlik işaretli bir tam sayıyı gösterir. Dikkat edeceğiniz gibi bu veri yapısı bir union'dur.

Örnek 2-10, HiResTimer.h başlık dosyasını ve HiResTimer.c ise bu başlık dosyaındaki fonksiyonların gerçekleştirimlerini göstermektedir.

Örnek 2-10

```
// HiResTimer.h

#ifndef _HIRESTIMER_H
#define _HIRESTIMER_H

#define WIN32_LEAN_MEAN

#include <windows.h>

struct _HIRESTIMER;
typedef struct _HIRESTIMER* HiResTimer;
```

```

extern HiResTimer HiResTimer_Create(void);
extern void HiResTimer_Destroy(HiResTimer*);
extern BOOL HiResTime_Init(HiResTimer);
extern float HiResTimer_GetElapsed_Seconds(HiResTimer);
extern float HiResTimer_GetFPS(HiResTimer, unsigned int);
extern float HiResTimer_LockFPS(HiResTimer, unsigned char);

#endif
// HiResTimer.c

#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include "HiResTimer.h"

struct _HIRESTIMER {
    LARGE_INTEGER m_StartTime;
    LARGE_INTEGER m_ticksPerSecond;
    LARGE_INTEGER s_lastTime_GetElapsed_Seconds ;
    LARGE_INTEGER s_lastTime_GetFPS;
    LARGE_INTEGER s_lastTime_LockFPS;
};

HiResTimer HiResTimer_Create(void) {

    HiResTimer _this;
    _this = (HiResTimer)
        malloc(sizeof(struct _HIRESTIMER));

    if (!_this) {
        MessageBox(NULL, "\tOut of memory", "Error", 0);
        return NULL;
    } // end of if (!_this)

    return _this;
} /* end of HiResTimer HiResTimer_Create(LARGE_INTEGER,
                                         LARGE_INTEGER)

```

```

*/



void HiResTimer_Destroy(HiResTimer *_this) {

    free(*_this);
    *_this = NULL;

    return;
} // end of void HiResTimer_Destroy(HiResTimer*)

BOOL HiResTime_Init(HiResTimer _this) {

    if (!QueryPerformanceFrequency(
        &(_this->m_ticksPerSecond)))
        return FALSE;

    else {
        QueryPerformanceCounter(&(_this->m_StartTime));
        _this->s_lastTime_GetElapsed_Seconds =
            _this->m_StartTime;
        _this->s_lastTime_GetFPS = _this->m_StartTime;
        _this->s_lastTime_LockFPS = _this->m_StartTime;
        return TRUE;
    }
} // end of BOOL HiResTime_Init(HiResTimer)

float HiResTimer_GetElapsed_Seconds(HiResTimer _this) {

    LARGE_INTEGER currentTime;
    float seconds;

    QueryPerformanceCounter(&currentTime);
    seconds = ((float)currentTime.QuadPart -
    (float)_this->s_lastTime_GetElapsed_Seconds.QuadPart) /
    (float)_this->m_ticksPerSecond.QuadPart;

    _this->s_lastTime_GetElapsed_Seconds = currentTime;
    return seconds;
}

```

```

} // end of float HiResTimer_GetElapsed_Seconds(HiResTimer)

float HiResTimer_GetFPS(HiResTimer _this,
                        unsigned int elapsedFrames)
{

    LARGE_INTEGER currentTime;
    float fps;

    QueryPerformanceCounter(&currentTime);

    fps = (float)elapsedFrames*
        (float)_this->m_ticksPerSecond.QuadPart /
        ((float)currentTime.QuadPart -
         (float)_this->s_lastTime_GetFPS.QuadPart);

    _this->s_lastTime_GetFPS = currentTime;

    return fps;
} /* end of float HiResTimer_GetFPS(HiResTimer,
                                    unsigned int)

 */

float HiresTimer_LockFPS(HiResTimer _this,
                         unsigned char targetFPS)
{
    LARGE_INTEGER currentTime;
    float fps;

    if (targetFPS == 0) targetFPS = 1;

    do {

        QueryPerformanceCounter(&currentTime);
        fps = (float) _this->m_ticksPerSecond.QuadPart /

```

```

        ((float)(currentTime.QuadPart -
        _this->s_lastTime_LockFPS.QuadPart));
    }

    }while(fps > (float)targetFPS);

    _this->s_lastTime_LockFPS = currentTime;

    return fps;
} /* end of float HiresTimer_LockFPS(HiResTimer,
                                         unsigned char)
 */

```

HiResTimer_Create() fonksiyonu sizin için bir timer yaratır ve döndüreceği değer HiResTimer olan bir işaretçi dönderecektir. Bu işaretçi yaratılan timer'ı göstermektedir. **HiResTimer_Destroy()** ise yarattığınız timer'ı yok eder, alacağı parametre ise bu yok edilecek olan timer'ın işaretçisidir. Bu işaretçinin gösterdiği timer'ın içi boşaltılıp işaretçiye NULL değeri atanmaktadır.

HiResTime_Init() fonksiyonu timer'ı ilklemektedir. Bu ilklemeye işleminde, bir saniyede atılan clock sayısı ve o anki başlangıç clock değeri bulunur.

HiResTimer_GetElapsed_Seconds() fonksiyonu, bize fonksiyonun çağrılmamasından bu yana ne kadar saniyenin geçtiğini döndürmektedir. Dönüş değeri float tipindedir.

HiResTimer_GetFPS(), *elapsedFrames* parametresi üzerinden ortalama ne kadar frame'ın gösterildiğini söylemektedir. **HiresTimer_LockFPS()** fonksiyonu ise istenilen frame sayısı kadar frame oranını kilitleyerek bekletir. Bu istenilen frame sayısı *targetFPS* parametresi ile fonksiyona aktarılmaktadır.

OpenGL TANIMI Uygulama

1. Giriş Günümüzde yazılım ve donanımın gelişmesi ile birlikte bilgisayar grafikleri alanında oldukça önemli gelişmeler kaydedilmektedir. Bu gelişmelere paralel olarak yazılım geliştirme kütüphaneleri de ortaya çıkmaktadır. OpenGL (Open Graphic Library) de gelişmiş grafik uygulamaları için geliştirilen bir kütüphanedir. OpenGL grafik kütüphanesi, grafik donanımına bir yazılım arayüzüdür. 2D ya da 3D nesnelerden oluşan hareket eden görüntüler üreten, interaktif, çok çeşitli bilgisayar platformlarında çalışabilecek programlar yaratmamıza olanak sağlar. OpenGL, donanım-bağımsız bir arayözdür. Görüntüde bulunan nesneleri tanımlamak ve bu nesneler üzerinde gerek duyulan işlemleri gerçekleştirmek için yaklaşık 700 farklı komut içerir. (Bu komutlardan yaklaşık 650 tanesi saf OpenGL komutu ve yaklaşık 50 tanesi ise OpenGL Utility Library de bulunmaktadır [1]. OpenGL komut yorumlama modeli, istemci/sunucu (client/server) modelidir. Uygulama kodu (istemci), komut çağrılarını yapar. Bu komutlar, OpenGL (sunucu) tarafından yorumlanır ve işlenir. OpenGL, grafik programını koşan ve yarattığımız grafiği görüntüleyen bilgisayarlar farklı olduğunda bile etkili çalışacak şekilde tasarlanmıştır. Bu durum, istemci ve sunucunun farklı tip bilgisayarlar olabileceği bir bilgisayar ağı ortamında geçerlidir. OpenGL 'in donanım-bağımsız olmasının nedeni, pencere işlemlerini (windowing task-ekranda bir pencere oluşturmak gibi) yapan ya da kullanıcıdan girdi alan herhangi bir komutunun bulunmamasıdır. Belirtilen bu işleri gerçekleştirmek için işletim sisteminin mevcut özellikleri kullanılır. Ancak işletim sisteminde pencere işlemlerini gerçekleştirmek karmaşık işlemler içerdiginden tüm bu işlevleri barındıran ve işletim sistemine özel olarak yazılmış GLUT (Graphic Library Utility) kütüphaneleri bulunmaktadır. OpenGL, 3D nesneleri tanımlamak için yüksek-seviye komutlar içermez. Bunun yerine; nokta, doğru ve poligon gibi alt-seviye geometrik primitifleri

kullanarak nesneleri tanımlamamıza olanak sağlar. Primitifler, bir ya da daha fazla vertex ile tanımlanır. Bir nokta, bir doğrunun başlangıç ve bitiş noktaları, bir poligonun iki kenarının kesiştiği köşe noktası, vertex 'ler yardımıyla belirlenir.

2. OpenGL

2.1. OpenGL'in Avantajları

OpenGL kullanarak grafikler oluşturmanın avantajları aşağıda sıralanmıştır.

- Tüm OpenGL uygulamaları, işletim sistemi ne olursa olsun, OpenGL API uyumlu donanımlar üzerinde mükemmel görsel sonuçlar üretebilir.
- Grafik donanımlarının yeni gelişmiş özellikleri, OpenGL tarafından, geliştirme mekanizması (extension mechanism) sayesinde kullanılabilir. Yani OpenGL, donanımaya özel, gelişmiş özellikleri kullanmak için API fonksiyonları içerebilir.
- OpenGL temelli grafik uygulamaları, çok çeşitli sistemler üzerinde koşulabilir. (tüketici elektroniği – consumer electronics, PC, iş istasyonu – workstation, süper bilgisayarlar gibi)
- OpenGL grafik kütüphanesi kullanılarak, çok daha az bir kod satırıyla daha yüksek performansa sahip uygulamalar geliştirmek mümkündür.
- OpenGL grafik kütüphanesine dair teknik bilgi içeren birçok kaynak mevcuttur.(internet, kitaplar, vs.)
- Birçok programlama dili (C, C++, Fortran, Ada, Java gibi) OpenGL tabanlı uygulama geliştirmemize olanak sağlar.

2.2. OpenGL Utility (GLUT)

OpenGL platformdan bağımsız olduğu için bazı işlemler bu kitaplık ile yapılamaz. Örneğin kullanıcıdan veri almak, bir pencere çizdirmek gibi işler hep kullanılan pencere yöneticisi ve işletim sisteme bağlıdır. Bu yüzden bir an için OpenGL'in platform bağımlı olduğu düşünülebilir. Çünkü çalışma penceresini her pencere yöneticisinde (her ortamda) farklı çizdirecek bir canlandırma programı yazmak demek her bilgisayarda çalışacak ayrı pencere açma kodu yazmak demektir. Bu ise OpenGL'in doğasına aykırıdır. Bu

gibi sorunları aşmak için OpenGL Araç Kiti (GLUT - OpenGL Utility Toolkit) kullanılmaktadır. GLUT, birçok işletim sisteme aktarılmış bir kitaplıktır. Amacı OpenGL programlarının pencerelerini oluşturmak, klavye ve fareden veri almak gibi ihtiyaçları karşılamaktır. GLUT olmadan da OpenGL programlama yapılabilir, örneğin Linux'ta kullanılan X-Window sistemin kendi işlevleri kullanılarak pencere çizdirilebilir fakat bu kod sadece X-Window'da çalışır. Kod Windows'a götürülp derlendiğinde çalışmaz, çünkü Windows'da X-Window işlevleri yoktur. Benzer şekilde Windows tabanlı işletim sistemlerinin de kendilerine has pencere oluşturma işlevleri vardır. Bu yüzden bu deneyde GLUT kitaplığı kullanılarak klavye ve fare için işletim sisteminde bağımsız giriş/çıkış işlemleri yapılması sağlanmıştır. 2.3.OpenGL Söz dizimi OpenGL komutları, gl öneki ile başlarlar. (örnek; glClearColor()). Benzer şekilde OpenGL tarafından tanımlı sabitler de GL_ öneki ile başlarlar ve kelimeler birbirinden _ ile ayrılacak şekilde büyük harfle yazılırlar. (örnek; GL_COLOR_BUFFER_BIT). glColor3f komutundaki 3 sayısı da 3 parametre alacağı anlamına gelmektedir. f ise verilen parametrelerin float olacağı anlamına gelmektedir. glVertex2i(1,3); ya da glVertex2f(1.0,3.0); gibi

- OpenGL temelli grafik uygulamaları, çok çeşitli sistemler üzerinde koşulabilir. (tüketici elektroniği – consumer electronics, PC, iş istasyonu – workstation, süper bilgisayarlar gibi)
- OpenGL grafik kütüphanesi kullanılarak, çok daha az bir kod satırıyla daha yüksek performansa sahip uygulamalar geliştirmek mümkündür.
- OpenGL grafik kütüphanesine dair teknik bilgi içeren birçok kaynak mevcuttur.(internet, kitaplar, vs.)
- Birçok programlama dili (C, C++, Fortran, Ada, Java gibi) OpenGL tabanlı uygulama geliştirmemize olanak sağlar.

2.2.Open GL Utility (GLUT)

OpenGL platformdan bağımsız olduğu için bazı işlemler bu kitaplık ile yapılamaz. Örneğin kullanıcının veri almak, bir pencere çizdirmek gibi işler hep kullanılan pencere yöneticisi ve işletim sistemine bağlıdır. Bu yüzden bir an için OpenGL'in platform bağımlı olduğu düşünülebilir. Çünkü çalışma penceresini her pencere yöneticisinde (her ortamda) farklı çizdirecek bir canlandırma programı yazmak demek her bilgisayarda çalışacak ayrı pencere açma kodu yazmak demektir. Bu ise OpenGL'in doğasına aykırıdır. Bu gibi sorunları aşmak için OpenGL Araç Kiti (GLUT - OpenGL Utility Toolkit) kullanılmaktadır.

GLUT, birçok işletim sisteme aktarılmış bir kitaplıktır. Amacı OpenGL programlarının pencerelerini oluşturmak, klavye ve fareden veri almak gibi ihtiyaçları karşılamaktır.

GLUT olmadan da OpenGL programlama yapılabilir, örneğin Linux'ta kullanılan X-Window sistemin kendi işlevleri kullanılarak pencere çizdirilebilir fakat bu kod sadece X-Window'da çalışır. Kod Windows'a götürülüp derlendiğinde çalışmaz, çünkü Windows'da X-Window işlevleri yoktur. Benzer şekilde Windows tabanlı işletim sistemlerinin de kendilerine has pencere oluşturma işlevleri vardır.

Bu yüzden bu deneyde GLUT kitaplığı kullanılarak klavye ve fare için işletim sisteminden bağımsız giriş/çıkış işlemleri yapılması sağlanmıştır.

2.3.OpenGL Söz dizimi

OpenGL komutları, gl öneki ile başlarlar. (örnek; glClearColor()). Benzer şekilde OpenGL tarafından tanımlı sabitler de GL_ öneki ile başlarlar ve kelimeler birbirinden _ ile ayrılacak şekilde büyük harfle yazılırlar. (örnek; GL_COLOR_BUFFER_BIT).

glColor3f komutundaki 3 sayısı da 3 parametre alacağı anlamına gelmektedir. f ise verilen parametrelerin float olacağı anlamına gelmektedir.

glVertex2i(1,3); ya da
glVertex2f(1.0,3.0); gibi

2.4.İlk OpenGL Programı

```
#include <GL/glut.h>
#include <stdlib.h>

void ayarlar(void)
{
    glClearColor(0.0,0.0,0.0,0.0);
```

```

glShadeModel(GL_FLAT);
glOrtho(-2.0, 2.0, -2.0, 2.0, -1.0, 1.0);

}

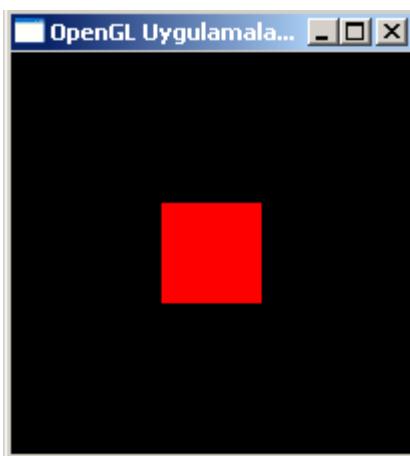
void gosterim(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);

    glBegin(GL_POLYGON);
        glVertex2f(-0.5, -0.5);
        glVertex2f(-0.5, 0.5);
        glVertex2f(0.5, 0.5);
        glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}

int main(int argc,char ** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB );
    glutInitWindowPosition(0,0);
    glutInitWindowSize(200,200);
    glutCreateWindow("OpenGL Uygulamaları-I");
    ayarlar();
    glutDisplayFunc(gosterim);
    glutMainLoop();
    return 0;
}

```

Programda ilk olarak bir pencere yaratılmakta daha sonra da gösterim fonksiyonu ayarlanmaktadır. Gösterim fonksiyonu içerisinde de her defasında çizilecek olan grafik çizilmektedir. Bu hali ile verilen kod basit bir OpenGL programının iskeletini oluşturmaktadır. Program çalıştırıldığında elde edilen ekran görüntüsü Şekil 1'de verilmiştir.



Şekil 1. OpenGL ile çizilmiş basit bir şekil

2.5.OpenGL ile Birden Fazla Şekil Çizimi

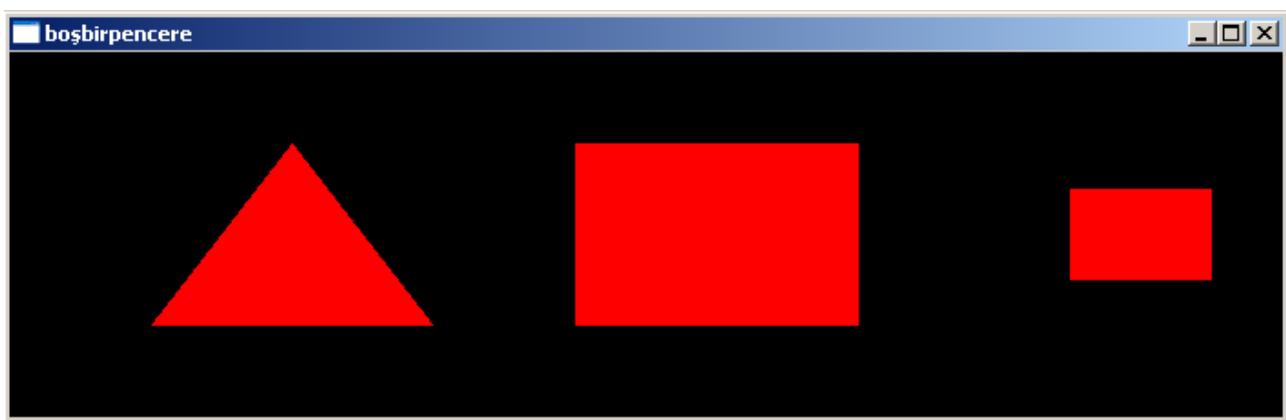
Birden fazla şekil çizmek için yukarıda verilen iskelet program üzerinde sadece gosterim isimli fonksiyon aşağıdaki şekilde değiştirilir.

```
void gosterim(void)
{
    glLoadIdentity();
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);
    glOrtho(-2.0, 7.0, -2.0, 2.0, -1.0, 1.0);
    glBegin(GL_TRIANGLES);
        glVertex3f( 0.0f, 1.0f, 0.0f);
        glVertex3f(-1.0f,-1.0f, 0.0f);
        glVertex3f( 1.0f,-1.0f, 0.0f);
    glEnd();

    glTranslatef(3.0f,0.0f,0.0f);
    glBegin(GL_QUADS);
        glVertex3f(-1.0f, 1.0f, 0.0f);
        glVertex3f( 1.0f, 1.0f, 0.0f);
        glVertex3f( 1.0f,-1.0f, 0.0f);
        glVertex3f(-1.0f,-1.0f, 0.0f);
    glEnd();

    glTranslatef(3.0f,0.0f,0.0f);
    glBegin(GL_POLYGON);
        glVertex2f(-0.5, -0.5);
        glVertex2f(-0.5, 0.5);
        glVertex2f(0.5, 0.5);
        glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}
```

Verilen programın ekran görüntüsü Şekil 2'de verilmiştir.



Şekil 2. OpenGL ile birden çok şekil çizmek

Tartışma Sorusu-1 : Gösterim fonksiyonunda kullanılan *glLoadIdentity()*; fonksiyonunun işlevi nedir? Neden kullanılmıştır? Bu fonksiyon kullanılmamasıydı ekran görüntüsü nasıl olurdu? Tartışınız...

2.6. OpenGL ile Farklı Renklerle Çalışmak

Verilen taslak program üzerinde gösterim fonksiyonu aşağıdaki gibi değiştirilerek çizilen şekiller

3. Deneye Hazırlık

Bu bölüm, deneye gelmeden önce her öğrenci tarafından yapılması gereken maddeleri içermektedir.

1. Deneye gelmeden önce Dev C++ programı bilgisayara kurulmalı ve temel düzeyde programın kullanılışı öğrenilmelidir. Programı su adresinden (<http://www.bloodshed.net/download.html>) indirilebilir.
2. Ek-1'de verilen adımlar takip edilerek Dev C++ programına OpenGL ve GLUT kütüphaneleri kurulmalıdır.
3. Deney foyü dikkatlice okunmalı ve Deneye hazırlık soruları cevaplanmalıdır.
4. Ek-2'de verilen örnek program yazılarak koşulmalı ve programın çalışma mantığı anlaşılmalıdır.

4. Deneyin Yapılışı

5. Deney Soruları

6. Kaynakça

Addison Wesley, “OpenGL Programming Guide”, 6th Edition, 2008.

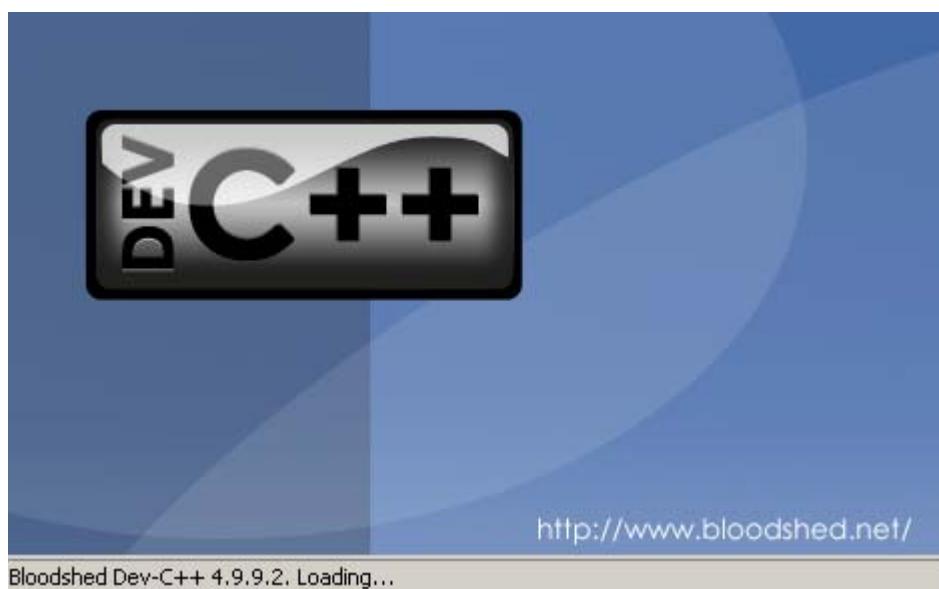
Yapılacaklar

- Ek 1 de dev c++ a glut un kurulusu anlatılacak
- Ek-2 de tüm yazılan kodlar anlatılacak.
- Deneyin yapılışı adım adım anlatılacak
- Deney soruları verilecek

7. Ek 1

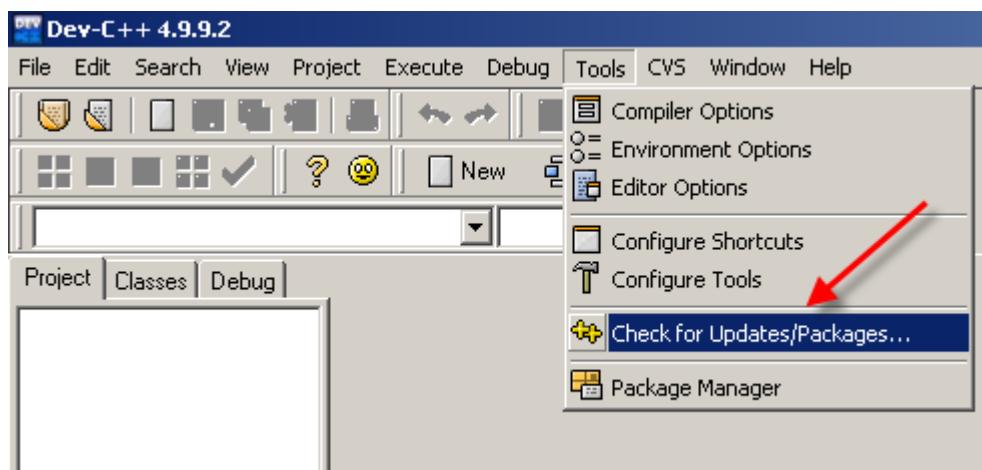
7.1.Adım 1

Dev C++ programını kurun

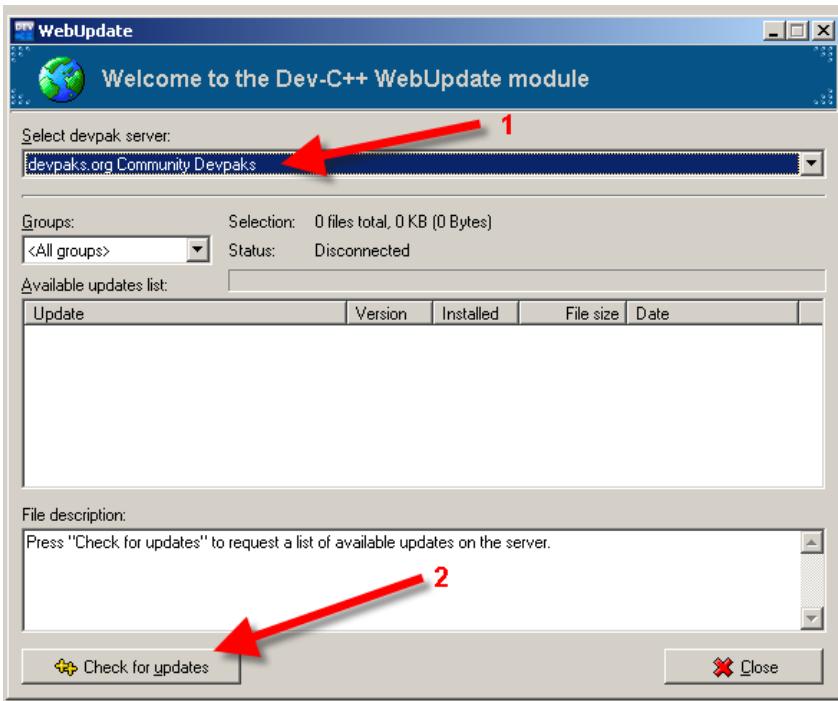


7.2.Adım 2

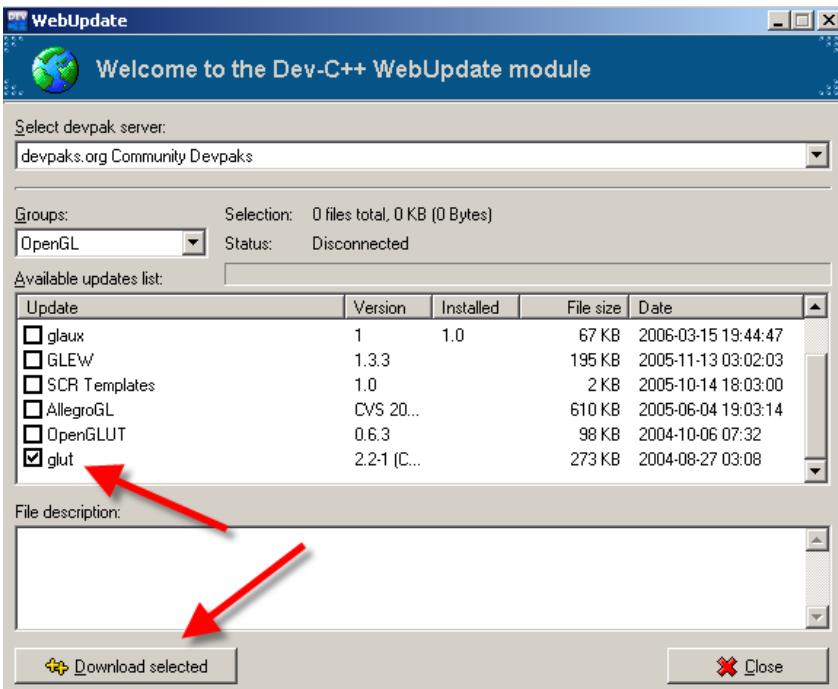
Tools / Check for Updates/Packages...



7.3.Adım 3



7.4.Adım 4

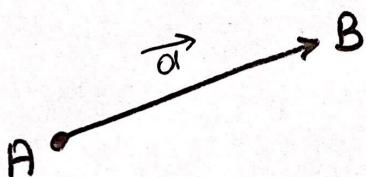


①

Tel Kafes (Wireframe) Grafikleri:

Vektör: Doğrultusu (yönü) ve uzunluğu (büyüklüğü) belirli olan bir doğru parçası.

Başlangıç noktası A, bitim noktası B olan \overrightarrow{AB} doğru parçasına yönlü doğru parçası denir. Bu vektör \vec{a} ile gösterilir. Ok vektörün yönünü gösterir.



Bir vektör çok çeşitli sekillerde gösterilebilir. En çok kullanılan gösterimler (\vec{a}) veya koyu harf (a) şeklindeki gösterimlerdir.

Vektörün bileşenleriyle gösteriminde ise genellikle sıralı n-lı kullanılır.

$$\vec{a} = (a_1, a_2, \dots, a_n)$$

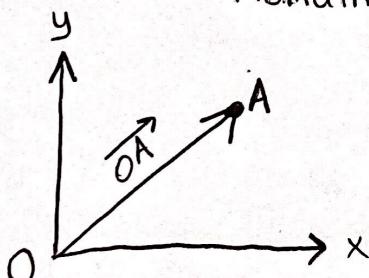
Vektörün bileşenleri satır veya sütun dizimleri şeklinde de yazılabilir.

$$\vec{a} = [a_1 \ a_2 \ \dots \ a_n] \text{ ya da } \vec{a} =$$

$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$$

$\vec{i}_1 = (1, 0, 0, \dots, 0)$, $\vec{i}_2 = (0, 1, 0, \dots, 0)$, \dots , $\vec{i}_n = (0, 0, \dots, 0, 1)$ vektörleri olmak üzere. $\vec{a} = a_1 \vec{i}_1 + a_2 \vec{i}_2 + \dots + a_n \vec{i}_n$ vektör gösterimi birim vektör gösterimidir.

Konum (yer) vektörü

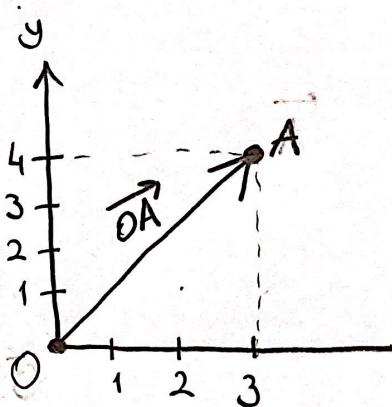


Başlangıç noktası orijin olan vektörlere konum (yer) vektörü denir. Eğer vektör orijinde değilse vektörün uzunluğunu ve yönünü değiştirmeden orijine taşıyabiliriz.

②

Başlangıç noktası $O = (0,0)$, bitiş noktası $A = (3,4)$ olan
iki boyutlu bir vektörü as. şekilde gösterebiliriz.

$$\overrightarrow{OA} = \vec{A} - \vec{O} = (3,4) - (0,0) = (3,4)$$



Bir vektörün normu: \vec{A} vektörünün normu (uzunluğu) $\|\vec{A}\|$
veya $|A|$ ile gösterilir.

$$\vec{A} = (a_1, a_2, a_3) = a_1 \vec{i} + a_2 \vec{j} + a_3 \vec{k} \text{ iin}$$

$$\|\vec{A}\| = \sqrt{a_1^2 + a_2^2 + a_3^2}$$

2B iin Dönüşümler

- Öteleme (yer degistirme, tasima) (translation)
- Döndürme (rotation)
- Ölçeklendirme (scaling)
- Meyillendirme (shearing)
- Yansıtma (reflection)

Standart
dönüşümler

Örnek: $\vec{v} = [x, y]$ olsun. T bir dönüşüm matrisi olsan.

$$\underbrace{[x, y]_{1 \times 2}}_{\text{Konum vektörü}} \cdot \underbrace{\begin{pmatrix} a & b \\ c & d \end{pmatrix}_{2 \times 2}}_{T \text{ dönüşüm matrisi}} = \underbrace{[]_{1 \times 2}}_{\text{Dönüştürülmiş konum vektörü } \vec{v}^*}$$

(3)

1) Ölçeklendirme (Scaling)

$T = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ dönüşüm matrisi olmak üzere a ve d pozitif ve birbirine eşit ve $b=c=0$ olmak üzere ölçeklendirme (dengeleştirilmiş ölçeklendirme) (balanced scaling) için kullanılacak olan matris

$$T = \begin{bmatrix} a & 0 \\ 0 & a \end{bmatrix}, a > 0$$

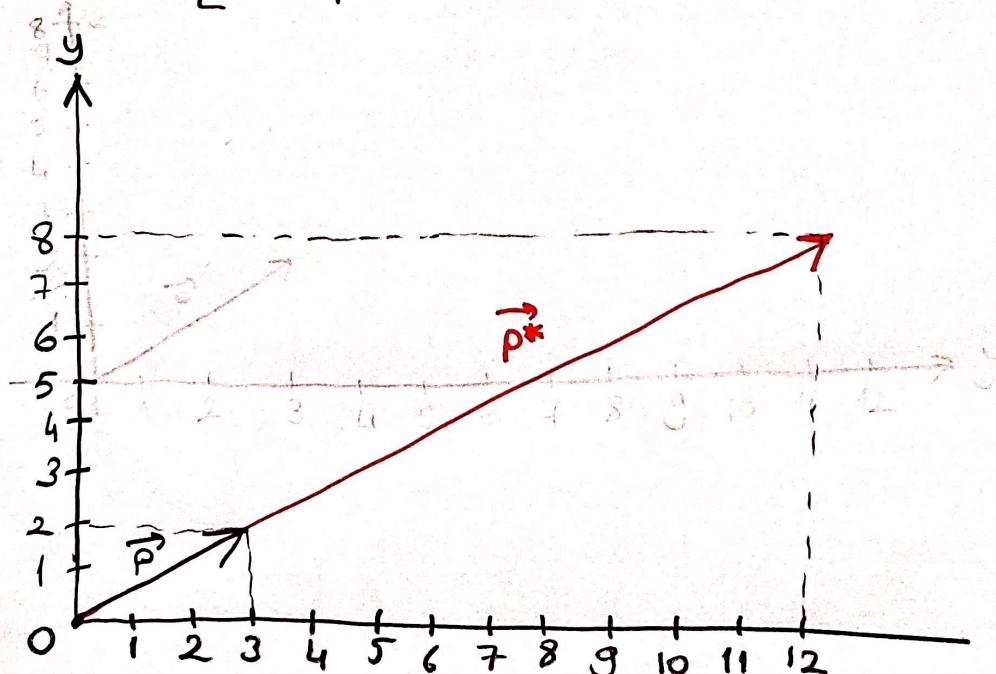
Örneğin; $\vec{p} = (3, 2)$ konum vektörü için

$$(3 \ 2) \begin{bmatrix} a & 0 \\ 0 & a \end{bmatrix} = (3a \ 2a) = \vec{p}^*$$

$T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ ise konum vektörü değişmez.

$\vec{p} = (3 \ 2)$ konum vektörüne $T = \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}$ dönüşüm matrisi uygulanırsa;

$$(3 \ 2) \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix} = (12 \ 8)$$



Örn: $T = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ döndürüm matrisinde $a+b=c+d=0$ faktörüne göre
 a ve d farklı pozitif sayılar ise bu dönüşüm etkisi
 dengelememis ölçütleridir (unbalanced scaling) oluşturur.

Örn: 1) $[x \ y]$ noktasını x yönünde 2 birim uzatın
 (ölaçleyin).

$$[x \ y] \cdot \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} = [2x \ y]$$

2) $[x \ y]$ noktasını y yönünde $\frac{1}{3}$ birim kısaltın (ölaçleyin).

$$[x \ y] \cdot \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{3} \end{bmatrix} = [x \ \frac{1}{3}y]$$

$$T = \begin{pmatrix} a & 0 \\ 0 & d \end{pmatrix} \text{ için}$$

$a > 0$ ise x eksenine göre uzatma

$a < 1$ ise x eksenine göre kısaltma

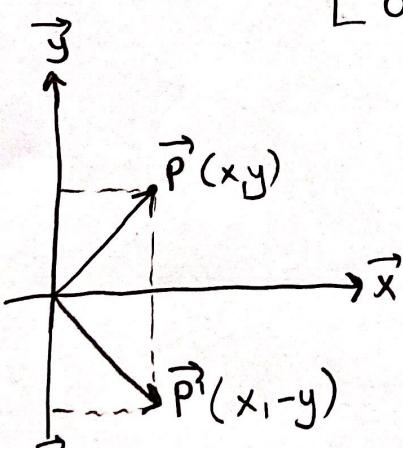
$d > 0$ ise y eksenine göre uzatma

$d < 1$ ise y eksenine göre kısaltma

2) Yansıtma (Reflection)

2 boyutta bir konum vektörünün x -eksenine göre
 yansıması

$$T = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \text{ matrisi ile tensil edilir.}$$



$$[x \ y] \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = [x \ -y]$$

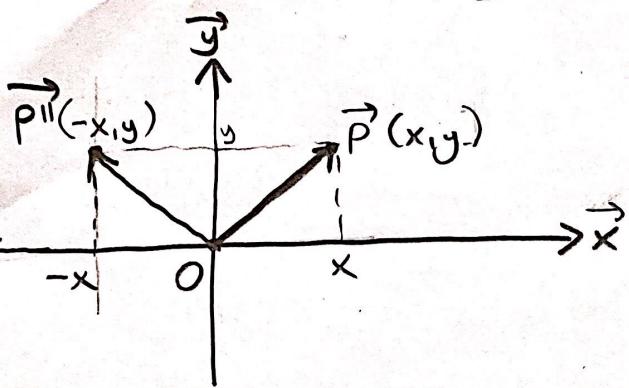
$d = -1$ ise x eksenine göre yansıtma

$$T = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

(5)

2D de bir konum vektörünün y - eksenine göre yansıması

$T = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$ matrisi ile tamsil edilir.

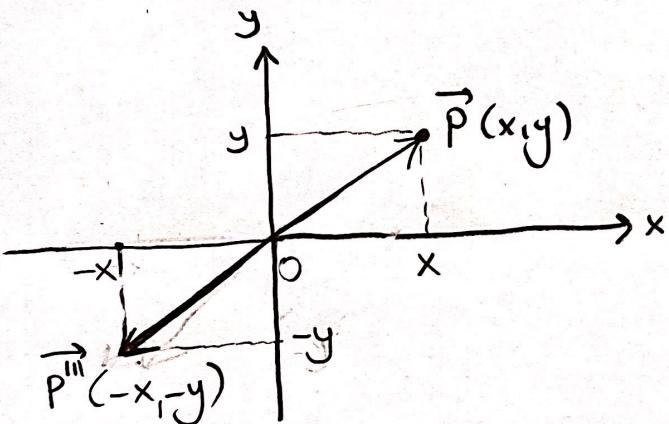


$$[x \ y] \cdot \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} = [-x \ y]$$

$a = -1$ ise y eksenine göre yansıma

2D de orijine göre yansıma, $a = -1, d = -1$ ise orijine göre yansıma yapılır.

$$T = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$



$$[x \ y] \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} = [-x \ -y]$$

3) Meyillendirme (Shearing)

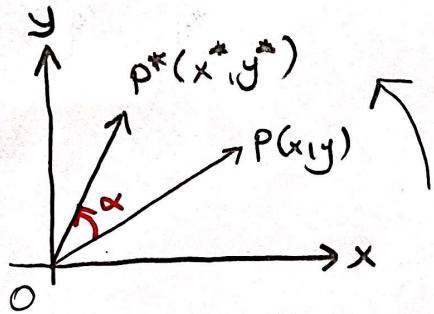
$[x \ y]$ konum vektörünü x yönünde 3 birim meyllendirmek demek y yi sabit tutup x i taşımak demek

$$[x \ y] \cdot \begin{bmatrix} 1 & 0 \\ c & 1 \end{bmatrix} = [x+cy \ y]$$

$$[x \ y] \cdot \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix} = [x+3y \ y]$$

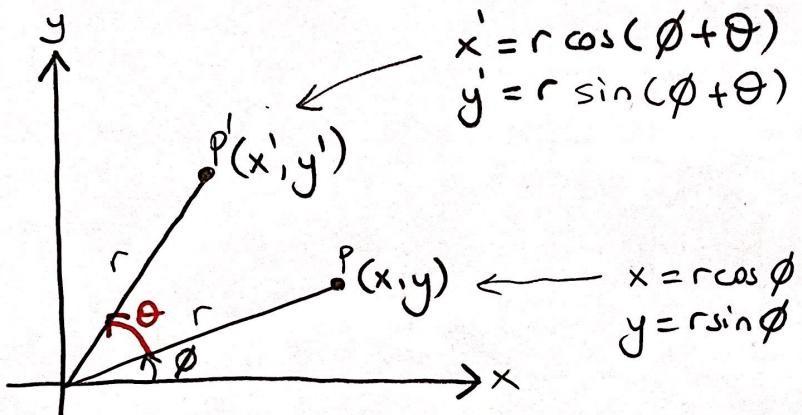
(6)

4) 2D de Dönüşme (Rotation)



Pozitif yönde yani saat yönünün tersine döndürme yapıyoruz.

Orjin etrafında pozitif yönde θ açısı ~~la~~ döndürülüğünde,



$$x' = r \cos(\phi + \theta) = r(\cos \phi \cos \theta - \sin \phi \sin \theta) = x \cos \theta - y \sin \theta$$

$$y' = r \sin(\phi + \theta) = r(\sin \phi \cos \theta + \cos \phi \sin \theta) = x \sin \theta + y \cos \theta$$

$$T = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \quad \text{Dönmeye matrisi}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix}_{2 \times 1} = \underbrace{\begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}}_{2 \times 2} \begin{bmatrix} x \\ y \end{bmatrix}_{2 \times 1}$$

Dönmeye matrisi

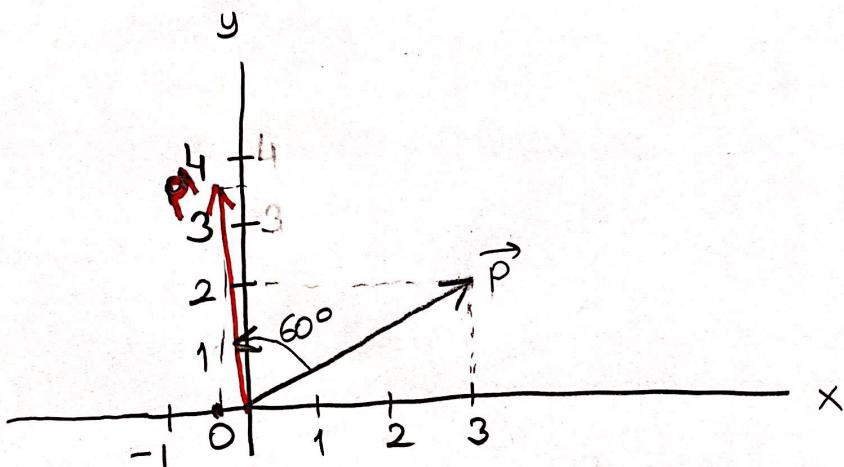
Örn: $P[3 \ 2]$ konum vektörünü pozitif yönde 60° döndürülüğünde olusacak olan yeni konum vektörünü bulunuz.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos 60 & -\sin 60 \\ \sin 60 & \cos 60 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & -\frac{\sqrt{3}}{2} \\ \frac{\sqrt{3}}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

(7)

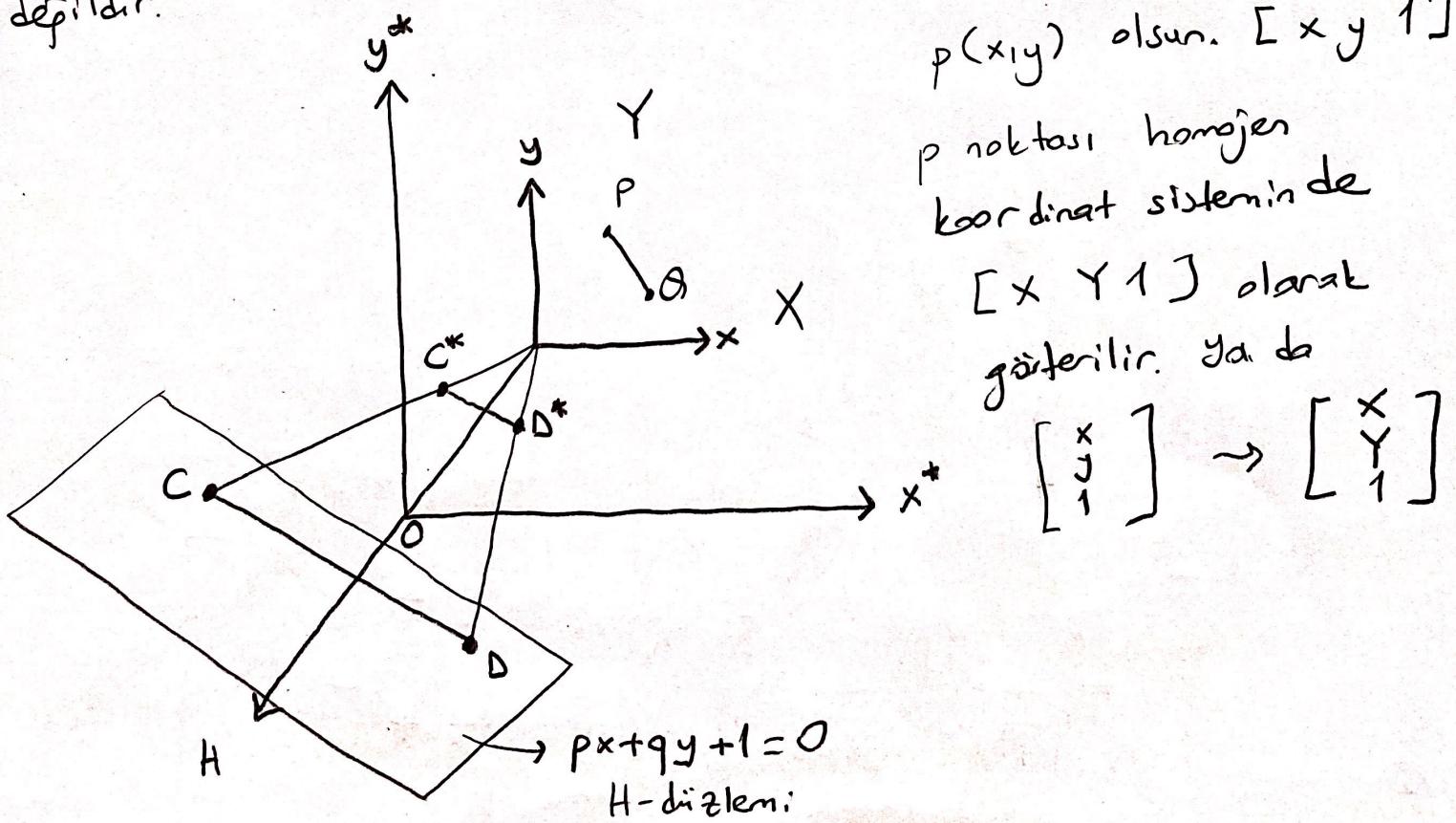
$$= \begin{bmatrix} 3 \cdot \frac{1}{2} - 2 \cdot \frac{\sqrt{3}}{2} \\ 3 \cdot \frac{\sqrt{3}}{2} + 2 \cdot \frac{1}{2} \end{bmatrix} = \begin{bmatrix} 1,5 - 2 \cdot 0,866 \\ 3 \cdot 0,866 + 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1,5 - 1,732 \\ 2,598 + 1 \end{bmatrix} = \begin{bmatrix} -0,23 \\ 3,6 \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$



2D de Homojen Koordinatlar

Homojen koordinatlarda bir p noktasının singelenisi tek
degildir.



$\text{p noktası } (2,3) \text{ olsun. } [x \ y \ 1] = [2 \ 3 \ 1]$ ⑧

 $[x \ y \ H] = [2 \ 3 \ 1]$
 $[x \ y \ 1/2] = [1 \ 3/2 \ 1/2]$
 $[x \ y \ 1/4] = [1/2 \ 3/4 \ 1/4]$
 $[x \ y \ 1/8] = [1/4 \ 3/8 \ 1/8]$
 $[x \ y \ 2] = [4 \ 6 \ 2]$

⋮

$H=0$ ise p noktası sonsuzda bir nokta tensil eder.

$[x \ y \ 0] = [\infty \ \infty \ 1]$

$\underset{H}{\circlearrowleft}$

$\underbrace{[x \ y \ H]}_{C \text{ noktası}} = \underbrace{[x \ y \ 1]}_{P \text{ noktası}} \cdot \begin{bmatrix} 1 & 0 & p \\ 0 & 1 & q \\ 0 & 0 & 1 \end{bmatrix}^{\begin{array}{l} x \\ y \\ t \end{array}} = \underbrace{[x^* \ y^* \ 1]}_{C^* \text{ noktası}}$

$x = x$

$y = y$

$H = px + qy + 1$

$[x \ y \ H] = [x \ y \ px + qy + 1] = \begin{bmatrix} x & y & 1 \\ \hline px + qy + 1 & px + qy + 1 & 1 \end{bmatrix}$

(3)

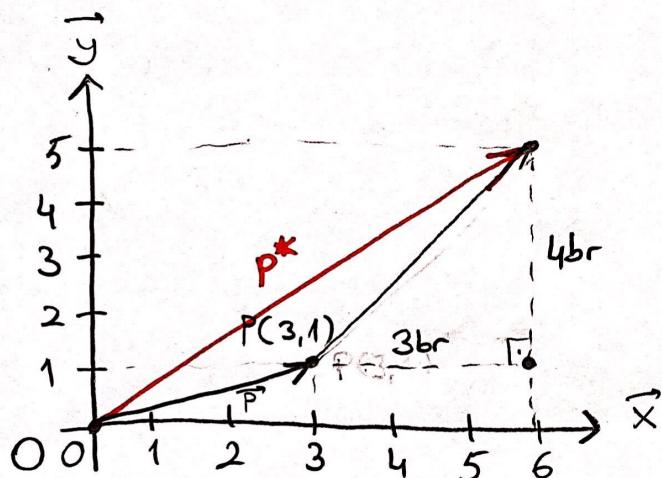
1) 2D'de Öteleme (Yer değiştirmme) (Translation)

$\vec{P} = \overrightarrow{OP} = (x, y)$ olsun. x yönünde 3 birim, y yönünde

4 birim ötelemesi demek;

$$\left. \begin{array}{l} x^* = x + 3 \\ y^* = y + 4 \end{array} \right\} \quad \begin{array}{l} \vec{P}^* = [x^*, y^*] \text{ satır vektör} \\ P^* = \begin{bmatrix} x^* \\ y^* \end{bmatrix} \text{ sütun vektör} \end{array}$$

Örneğin $\vec{P} = [3, 1]$ olsun. x yönünde 3 birim, y yönünde
4 birim ötelendiğinde \vec{P} vektörü



$$\begin{aligned} [x^* \ y^*] &= [3 \ 1] + [3 \ 4] \\ &= [6 \ 5] \end{aligned}$$

$P = [x, y]$ konum vektörünün;

x yönünde 2 birim

y yönünde 3 birim ötelemesini istiyoruz.

$$\left. \begin{array}{l} x' = x + 2 \\ y' = y + 3 \end{array} \right\} \text{ Bunu elde etmek istiyoruz.}$$

$\begin{bmatrix} x \\ y \end{bmatrix}$ sütun vektör ise $T = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix}$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix}}_T \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$[x \ y]$ satır vektör ise $T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 3 & 1 \end{bmatrix}$ ve (10)

$$[x' \ y' \ 1] = [x \ y \ 1] \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 3 & 1 \end{bmatrix}}_T$$

Bilgisayar Grafiklerinin Gelişimi

- MIT doktora tezi (Ivan Sutherland – 1960’lar)
“Sketchpad: a man-machine graphical communication system” ve CAD-CAM sistemlerinin kullanılmaya başlaması
- SIGGRAPH grubu 1969 (Çalışmaları devam ediyor)
- 3D Core Graphics System (1977) kullanılan aygıtta bağımsız geliştirilebilen grafik programlamaya elverişli ilk standart
- 1980'lere kadar donanımlar pahalı ve grafik uygulamaları az

SATIR SATIR GÖRÜNTÜLEME gelişimin başlangıcı
ve GUI VRAM(Video Access Memory) ve fractalların gelişimi(1983)

- 1985'te **GKS(Graphical Kernel System)** (1985)
- MIT'de Unix altında çalışan Windows için X library
- GKS (The Graphical Kernel **System**) 2D'den 3D'ye geçiş (1988)
- PHIGS (Programmer's Hierarchical Interactive Graphics System) ANSI ve ISO standarı haline geldi.
- Adobe'ın PostScript, SGI'in GL uygulamaları (1988)
- GL uygulamasından türetilen OPENGL (1993)
- Apple QuickDraw 3D ve Microsoft Direct 3D (1995)

Vektörel görüntüleme mimarisi
-rasgele tarama-

ve

satır satır görüntüleme mimarisinin
karşılaştırılması

Vektörel görüntüleme mi, yoksa satır satır görüntüleme mi kullanılacağına karar verilmeli.....

Daire çizimi ile karşılaştırılabilir

Vektörel Görüntüleme Mimarisi

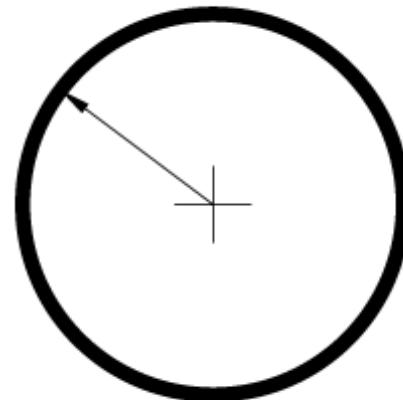
Vektör tabanlı grafik elemanları:

- Matematiksel bir formül
- Ek özellikler

Daire çizimi için dairenin merkezinin koordinatları ve çapı yeterli.
Ek özellikler çizgi ölçüsü, rengini...vb

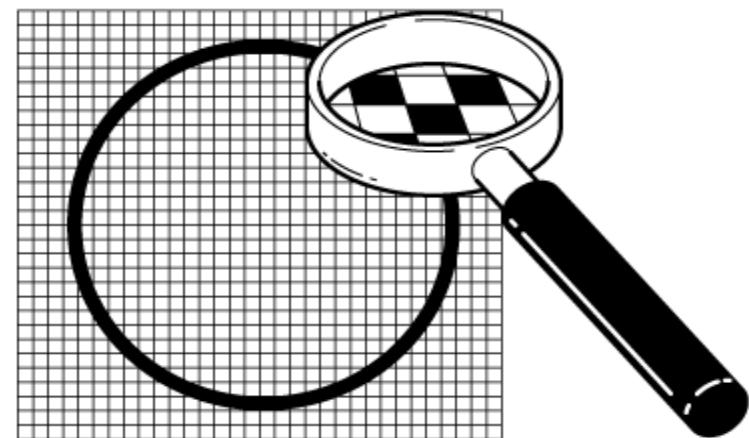
Kısaca; bir yer belirlemek için ayrık
doğrular veya noktalar tanımlanır

Yani; her bir nesne için bir yer tanımı
Yapılması gerekmektedir.

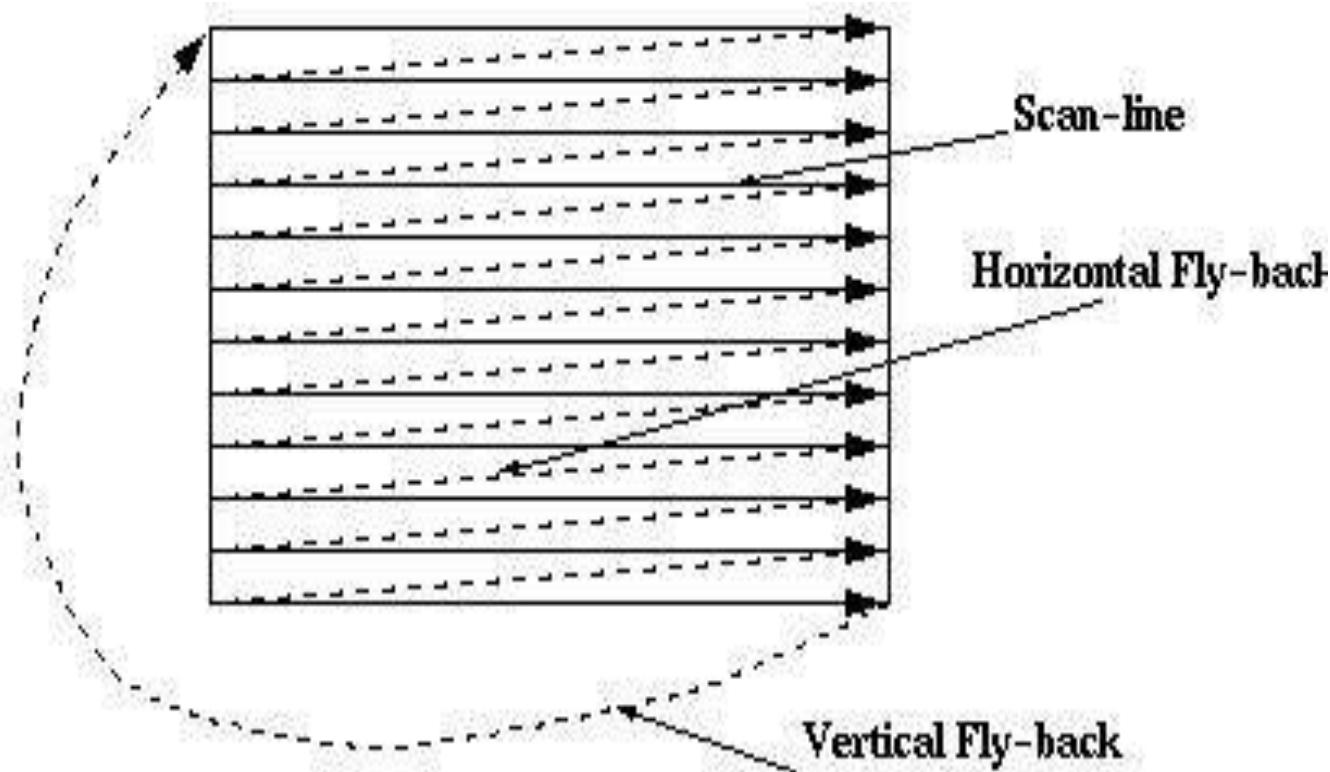


Satır Satır Görüntüleme

- Grafik elemanları, piksel olarak adlandırılan noktaların matris şeklindeki dizilimi ile tanımlanır.
- Piksellerin özellikleri rengi, transparanlığı ...vb farklı karakterler gösterir
- Kısaca; çalışma alanı düzenli bir hücre dizisi şeklinde bölünür - dizi en üst köşedeki hücreden başlayarak satır satır tarama ile devam eder - her bir hücrenin bir değeri vardır.



Satır Satır Görüntüleme



Satır Satır ya da Vektör Görüntüleme?

- Renk bilgisi noktadan noktaya değişiyorsa, **piksel tabanlı** görüntüleme, yani **satır satır görüntüleme** daha uygundur; örneğin **fotoğraflar**....
- Satır satır görüntüleme düşük maliyet (ucuz bellek)
- yoğun renk ve desen çeşitliliği (3D şekillerde) -
yenilenme prosesinin imajın karmaşıklığından
bağımsızlığı - geliştirilen donanımlar daha basit-
sistemler arasındaki uyumluluk
- **Vektör tabanlı görüntüleme**, çizgi, dikdörtgen,
metin...vb. içeren grafiklerin yaratılmasında daha
uygundur. Bu grafiklerin boyutu kalite kaybı
olmadan değiştirilebilir- Disk ihtiyaçları daha
düşüktür ve tarama dönüşümü gerektirmez

Vektör ya da Satır Satır Görüntüleme?

İki görüntüleme arasındaki en temel fark:

- Vektör görüntülemede bir noktadan diğer bir noktaya düz ve sürekli bir çizgi çizilebilir
- Satır satır görüntülemede doğru ve çokgenlerin sınır noktalarına piksel olarak yaklaşım, düz yerine tırtıklı bir görüntü elde edilmesine neden olabilir.

Vektör tabanlı mı? Piksel tabanlı mı?

İki çeşit grafik programı vardır:

- Vektör tabanlı programlar - CoreDraw, Adobe Illustrator - Macromedia Freehand ,Flash ...
- Satır satır görüntüleme programları (piksel) - Adobe Photoshop - Jasc Paintshop Pro ...
- Grafiğin nerede ve nasıl kullanılacağına göre bu iki programdan biri seçilir
- Bu programların birbirlerine göre farklı üstünlükleri vardır.

Ödev 1

Soru 1: Günümüzde yaygın olarak kullanılan vektör tabanlı (vector scanning) ve piksel tabanlı (raster scanning) çizimler gerçekleştiren programları araştırın ve bu programların temel özelliklerini yazın.

Görüntüleme Kalitesi

Vektör ve satır satır görüntüleme programları kalite açısından farklılıklar gösterir.

- Vektörler her zaman en iyi kalite ile simgelenirler. Formül için görünmez koordinatlarının dışında sabit noktaları göstermezler.
- Satır satır görüntülemenin kalitesi mesafenin nasıl ayarlandığına bağlıdır. Bu da sistemin piksel yapısından kaynaklanır. Belirli yakınlaşmalarda tekil pikseller görünür hale gelir.

Her iki tipteki programların diğer üstünlükleri

- Vektör dosyaları esnekdir - çizgi uzatılabilir - çap değiştirilebilir - elemanlar yeni bir düzene göre kolayca değiştirilebilir; kısaca görüntü değişiminde çözünürlük ve donanımdan bağımsızdır.
Teknik göstirimlerde kolaylık sağlar – kullanılan hazır grafikler istenilen boyuta, renge..vb. ayarlanabilir
Vektör görüntü ekrana yansıtılırken daha az ekran kartı kullanılır
- Satır satır görüntüleme dosyalarını düzenlemek de kolaydır - silgi fonksiyonu kullanılarak istenilen yer silinebilir.

Dosya Boyutu ve Taşınabilirlik

- Satır satır görüntüleme, vektörel görüntülemeden daha fazla hafızaya gerektirir - dosya boyutları özellikle Internet'te yayınlanan dosyalar için önemlidir.
- Vektörel görüntüleme satır satır görüntülemeye çevrilebilir - çok kolay değildir - satır satır görüntülemenin kalitesi bazı durumlarda değiştirmede ve çevirmede sorun yaratabilir.

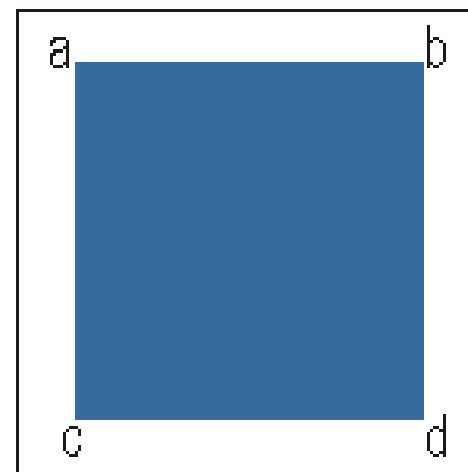
Sonuç

Çizilecek grafiğin özelliklerine göre uygun yöntem belirlenmelidir.

- Grafik belirli bir ölçüye bağlı değil ise ya da dosya boyutu önemsizse **satır satır görüntüleme** yöntemi seçilebilir.
- Grafik farklı sanal ortamlarda kullanılacaksa, sık sık ölçektekleme yapılıyorsa ya da çift arşivleme istenmiyorsa, **vektörel görüntülemeyi** kullanmak uygundur.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| a | | | | | | | | |
| b | | ■ | ■ | ■ | ■ | ■ | ■ | |
| c | ■ | ■ | ■ | ■ | ■ | ■ | ■ | |
| d | ■ | ■ | ■ | ■ | ■ | ■ | ■ | |
| e | ■ | ■ | ■ | ■ | ■ | ■ | ■ | |
| f | ■ | ■ | ■ | ■ | ■ | ■ | ■ | |
| g | ■ | ■ | ■ | ■ | ■ | ■ | ■ | |
| h | | | | | | | | |

Satır satır görüntüleme



Vektör görüntüleme

EKRAN KARTLARI

ve

GRAFİK KARTLARININ

GELİŞİMİ

Ekrandaki Görüntü Nasıl Oluşur?

- Monitörde görüntü çok küçük noktalardan oluşur.
- Bu noktalar görüntünün en küçük birimi olan piksellerdir.
- Her pikselin kendine ait renk ve yoğunluk bilgileri vardır.

Kısaca;

- Ekranın bağımsız olarak kontrol edilebilir en küçük parçası pikseldir.
- Piksellerden binlercesi bir araya gelerek ekrandaki görüntü oluşur.

Çözünürlük

- Görüntü kalitesini belirleyen en önemli faktördür.
- Ekrandaki görüntünün kaç pikselden oluşacağını belirler; yatay ve dikey piksel cinsinden belirtilir (800x600,1024x768 gibi).
- Çözünürlük arttıkça birbirinden bağımsız olarak kontrol edilebilen daha çok piksel görüntüyü oluşturur; böylece görüntü kalitesi de yükselir.
- Windows 95 ile gelen "scalable screen objects" teknolojisi ile çözünürlük arttıkça ekrandaki kullanılabilir alan da artar.
- Ekranın çözünürlüğü ne olursa olsun, nesneleri oluşturan piksel sayısı değişmez.
- Çözünürlük arttıkça pikseller de küçülür; böylece nesneler daha az yer kaplar. Ayrıca; masaüstündeki kullanılabilir alan çözünürlükle doğru orantılı olarak artar.

Renk Derinliği

- Piksellerin alabileceği tüm renkler **kırmızı**, **yeşil** ve **mavi**den türetilir.
- Renk derinliği bu renklerin miktarını belirler.
- Renk derinliği ne kadar artarsa her pikselin alabileceği renk sayısı artar ve renkler gerçeğe daha yakın olur.
- Renk derinliği **bit** cinsinden değerlendirilir. **8 bit** kullanıldığında bu bitlerden $2^8 = 256$ kombinasyon üretilir. Aynı şekilde **8 bit renk derinliğinde** de her **piksel** için **256** renk kullanılabilir.
- Ekrandaki görüntüyü gerçek gibi göstermek için kullanılan **üç rengin** de (kırmızı, yeşil ve mavi) **256`şar tonu** gereklidir, bu da renk başına 8 bitten 24 bit yapar. Bu mod **True Color** (Gerçek Renk) olarak adlandırılır.
- Fakat güncel ekran kartları görüntü belleğini kullanma yöntemleri yüzünden 32 bite ihtiyaç duyarlar. Kalan 8 bit **alpha kanalı** (piksellerin saydamlık bilgisini tutar) için kullanılır.

Çözünürlük

- Çözünürlük arttıkça yükselen görüntü kalitesinin de bir bedeli var tabii ki: Çözünürlük yükseldikçe kontrol edilmesi gereken piksel sayısı ve dolayısıyla da gerekli işlem gücü, ayrıca bu piksellerin bilgilerini tutmak için gerekli bellek miktarıyla onların transferi için gereken bellek bant genişliği artar. Bu yüzden de performans düşer. Kullanmak istediğiniz çözünürlüğü hem ekran kartınız desteklemeli, hem de monitörünüz fiziksel olarak gerekli sayıda pikseli ekranda oluşturabilmeli.

3D Core Graphics System

- SIGGRAPH (Graphics Standards Planning Committee (GSPC)) tarafından geliştirilmiş **aygıttan-bağımsız** görüntüleme paketidir.
- Taşınabilir grafik programlamaya uygun **ilk standart pakettir**.
- Aygıta bağımlı ve düşük seviyeli standarttan; aygıttan bağımsız ve taşınabilir grafik programlamaya elverişli standarda geçiş, görüntüleme cihazlarında büyük bir gelişmedir.
- Modelleme gibi bazı özel uygulama yazılımları da paket üzerinden çalıştırıldı. **“Core”** eki buradan gelmektedir.
- Hem 2D hem de 3D’ de çalışabilir
- Grafik tasarımda kullanılmayan mantıksal girdi araçları da tanınarak paralel ve perspektif görüntüleme işlemleri gerçekleştirilir.
- **En önemli özellik:** 2D görüntü modellemenin 3D görüntü modellemenin bir alt kümesi olarak standartlaşmasıdır.

GKS - Graphics Kernel System

- Siggraph Komitesi Avrupa'da Core sistemini temel alan, fakat çok daha gelişmiş GKS' i (Graphical Kernel System) çıkardı ve hemen ISO standarı oldu.
- Core'ın 2D uyarlaması olarak iki boyutta çizme imkanı veren düşük seviyeli çizim oluşmasında önemli rol oynamıştır (örneğin; elektrik devrelerin tasarlanması)
- Core tasarım olarak 3D kullanıcıları desteklerken, GKS daha ucuz ve yaygın depolama tüpü ve görüntü aygıtlarını hedefliyordu.
- Hem GKS, hem de Core vektörel görüntülemede standart olurlarken, satır taramalı görüntüleme özellikleri çok rağbet görmedi.
- Sonradan bir standart olacak PHIGS (Programmer's Hierarchical Interactive Graphics System) GKS ve Core uygulamalarını temel almıştır.

Görüntüleme sistemlerinden bağımsız
uygulama modelleri kullanılarak
geliştirilen, **Open- GL** gibi uygulama
programlarının günümüz grafik teknolojisinin
gelişimindeki rolü nedir?

kısaca Open-GL....

- Grafik donanımı için tanımlanmış bir **yazılım arayüzü**dür.
- Bir programlama dili değildir.
- Windows API 'lerinin bir dosyaya veya Internet'e erişmek için kullanıldığına benzer olarak, **OpenGL API** , bir oyun programı yazmak için programlama dili tarafından kullanılır.
- OpenGL fonksiyon çağrılarına cevap veren bir yazılım kütüphanesi ya da bir donanıma sürücü şeklinde tanımlanabilir.
- Oldukça hızlı olduğundan donanım sürücü olarak günümüz PC'lerinde çok yaygın kullanılmaktadır.
- Yazılım kütüphanesi olarak yordamsal çalışan grafik API 'dır.

- Open GL komutları ile nokta , çizgi , çokgen gibi temel grafik öğelerini 3 boyutlu çizmek mümkündür.
- OpenGL ile, ilave olarak, ışıklandırma ve gölgeleme , doku kaplama , karıştırma (blending) , geçirgenlik animasyon ve daha birçok özel efekt yapılabilir.
- **Sonuç olarak:** Open GL gibi geliştirilen uygulama programları ile günümüzde grafik teknolojisi çok gelişmiştir.
- Grafik teknolojileri bilgisayar oyunları, sinemalar, animasyonlar, tanıtımalar, cep telefonları v.b. gibi çok çeşitli alanlarda kullanılmaktadır.
- **Böylece;** grafik teknolojisi günümüz teknolojilerinin vazgeçilmezlerinden biridir.

- OpenGL kütüphaneleri (**libraries**) grafik donanımının yönetimini yazılım geliştiricilerine geçirir.
- OpenGL kütüphanesi, eski IRIS-GL kitaplığının tersine işletim sisteminden ve işletim sisteminin çalıştığı platformdan bağımsızdır.
- OpenGL ile , grafik kartının modeli veya işlemci mimarisinden, yani **donanımdan bağımsız** programlama gerçekleştirilir.
- Ağlar üzerinden istemci- sunucu ilişkisi yapılandırılarak bu tür uygulama modelleri kullanıldığında, sunucu ve istemciler farklı ortamlarda uyumlu çalışacaklardır.
- Kolay kullanımı ve "taşınabilirlik" özellikleri yüzünden OpenGL oldukça fazla tercih edilen bir yazılım aracıdır.
- OpenGL kütüphanesinden **C, C++, C# FORTRAN, Python, Perl** ve **Java** programlama dilleri kullanılarak yararlanılabilir.

- OpenGL ve benzeri uygulama programları grafik uygulamayı çok kolay hale getirmiştir.
- OpenGL **platformdan bağımsız** bir API' dir.
- Arka arkaya çalıştırılan komut akışları gerçekleştiren bir **arayüz** olarak tasarlanmıştır. Kaliteyi sağlamak için de “windowing task” ve “obtaining user input” gibi komutlar içermez; onun yerine kullandığı platformun OpenGL'e sağladığı **“windowing”** ve **“input”** hizmetlerinden yararlanır.
- 3D nesneler oluşturmak için de farklı komutlar içermez. Fakat sağladığı komutlarla bir otomobil, uçak veya molekül yüzeyi oluşturabilir.
- Daha iyi bir grafik sistemi OpenGL' in temel yapısı üzerine yapılandırılarak kurulabilir.
- OpenGL Utility Library (GLU) birçok yüksek düzey modelleme özelliklerine, yüzeylere, NURBS eğri ve yüzey oluşturma özelliklerine sahiptir.



OpenGL Syntax

- ❑ Functions have prefix **gl** and initial capital letters for each word
glClearColor(), glEnable(), glPushMatrix() ...
- ❑ **glu** for **GLU** functions
gluLookAt(), gluPerspective() ...
- ❑ constants begin with **GL_**, use all capital letters
**GL_COLOR_BUFFER_BIT, GL_PROJECTION,
GL_MODELVIEW ...**
- ❑ Extra letters in some commands indicate the number and type of variables
glColor3f(), glVertex3f() ...
- ❑ OpenGL data types
GLfloat, GLdouble, GLint, GLenum, ...



OpenGL function format

belongs to GL library

function name

dimensions

glVertex3f(x, y, z)

x, y, z are floats

The diagram illustrates the OpenGL function format for glVertex3f. A black arrow points from the text "function name" to the word "glVertex3f". A red arrow points from "dimensions" to the three parameters "x, y, z". A blue arrow points from "belongs to GL library" to the "gl" prefix in "glVertex3f". Below the first function, another instance of glVertex3f is shown with a yellow arrow pointing from the text "p" to the parameter "p" in the parentheses, indicating that p is a pointer to an array.

glVertex3fv(p)

p is a pointer to an array



OpenGL Syntax Examples

Setting the current color using `glColor`

- ❑ Colors may have 3 components (RGB) or 4 components (RGBA). Think of A (or `alpha`) as `opacity`.
- ❑ Floating point - color component values range from 0 to 1

```
glColor3f(0.0, 0.5, 1.0);
```

This is 0% Red, 50% Green, 100% Blue;

```
glColor4f(0.0, 0.5, 1.0, 0.3);
```

This is 0% Red, 50% Green, 100% Blue, 30% Opacity

```
GLfloat color[4] = { 0.0, 0.5, 1.0, 0.3 };  
glColor4fv(color);
```

0% Red, 50% Green, 100% Blue, 30% Opacity



OpenGL Syntax Examples

- ❑ Unsigned byte – color component values range from 0 to 255 (same as C's unsigned char).

```
glColor3ub (0, 127, 255);
```

This is: 0% Red, 50% Green, 100% Blue

```
glColor4ub (0, 127, 255, 76);
```

This is 0% Red, 50% Green, 100% Blue, 30% Opacity



Windowing with OpenGL

- OpenGL is independent of any specific window system
- GLUT provide a portable API for creating window and interacting with I/O devices



GLUT

Developed by Mark Kilgard

- Hides the complexities of differing window system APIs
 - Default user interface for class projects
- Glut routines have prefix **glut**
 - `glutCreateWindow()` ...
- Has very limited GUI interface
- **Glui** is the C++ extension of glut that provides buttons, checkboxes, radio buttons, etc.



Glut Routines

- **Initialization:** `glutInit()` processes (and removes) command line arguments that may be of interest to glut and the window system and does general initialization of Glut and OpenGL
 - Must be called before any other glut routines
- **Display Mode:** The next procedure, `glutInitDisplayMode()`, performs initializations informing OpenGL how to set up the frame buffer.

| Display Mode | Meaning |
|--------------|---------|
|--------------|---------|

- | | |
|--|---------------------------------------|
| • <code>GLUT_RGB</code> | Use RGB colors |
| • <code>GLUT_RGBA</code> | Use RGB plus alpha (for transparency) |
| • <code>GLUT_DOUBLE</code> | Use double buffering (recommended) |
| • <code>GLUT_SINGLE</code> recommended) | Use single buffering (not |
| • <code>GLUT_DEPTH</code> removal.) | Use depth buffer (for hidden surface |



Glut Routines

□ Window Setup

`glutInitWindowSize(int width, int height)`

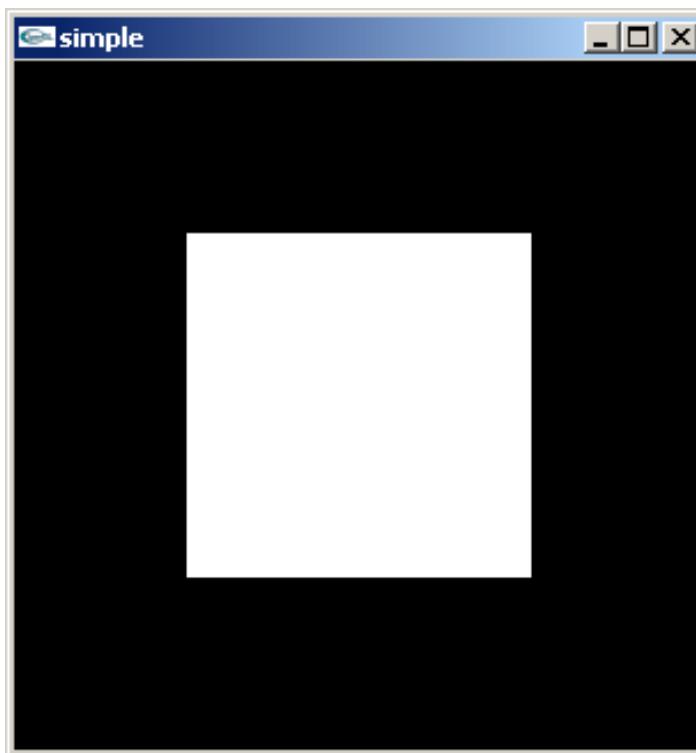
`glutInitWindowPosition(int x, int y)`

`glutCreateWindow(char* title)`



A Simple Program

Generate a square on a solid background



simple.c



```
#include <GL/glut.h>
void mydisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
        glVertex2f(-0.5, -0.5);
        glVertex2f(-0.5, 0.5);
        glVertex2f(0.5, 0.5);
        glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("simple");
    glutDisplayFunc(mydisplay);
    init();
    glutMainLoop();
}
```



The function main()

```
#include <GL/glut.h>           ← includes gl.h  
                                and glu.h  
  
int main(int argc, char** argv)  
{  
    glutInit(&argc, argv);  
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);  
    glutInitWindowSize(500, 500);  
    glutInitWindowPosition(0, 0);  
    glutCreateWindow("simple");  
    glutDisplayFunc(mydisplay);  
  
    init();  
    ← set OpenGL state  
    glutMainLoop();  
}
```

define window properties

display callback

enter event loop

The program goes into a infinite loop waiting for events



The function init()

```
void init()  
{  
    glClearColor (0.0, 0.0, 0.0, 1.0);  
  
    glColor3f(1.0, 1.0, 1.0);      ← fill/draw with white  
  
    glMatrixMode (GL_PROJECTION);  
    glLoadIdentity ();  
    gluOrtho2D(-1.0, 1.0, -1.0, 1.0);  
}
```

black clear color

opaque window

fill/draw with white

viewport



Callback functions

- Most of window-based programs are **event-driven**
 - Even driven means do nothing until an event happens, and then execute some pre-defined functions
- **Events** – key press, mouse button press and release, window resize, etc.



Callbacks

- ❑ Virtually all interactive graphics programs are event driven
- ❑ Glut uses callbacks to handle events
 - Windows system invokes a particular procedure when an event of particular type occurs.
 - MOST IMPORTANT: display event
 - Signaled when window first displays and whenever portions of the window reveals from blocking window
 - **glutDisplayFunc (void (*func) (void))** registers the display callback function
- Running the program: **glutMainLoop ()**
 - Main event loop. Never exit()



More Callbacks

- `glutReshapeFunc(void (*func)(int w, int h))` indicates what action should be taken when the window is resized.
- `glutKeyboardFunc(void (*func)(unsigned char key, int x, int y))` and `glutMouseFunc(void (*func)(int button, int state, int x, int y))` allow you to link a keyboard key or a mouse button with a routine that's invoked when the key or mouse button is pressed or released.
- `glutMotionFunc(void (*func)(int x, int y))` registers a routine to call back when the mouse is moved while a mouse button is also pressed.
- `glutMouseFunc(void (*func)(int button, int state, int x, int y))` registers a function that's to be executed if a mouse button event occurs. The argument button can be GLUT_LEFT_BUTTON or GLUT_RIGHT_BUTTON. The argument state can be GLUT_UP or GLUT_DOWN. The arguments x and y indicated the mouse cursor position when the button was clicked.



glutDisplayFunc(void (*func)(void))

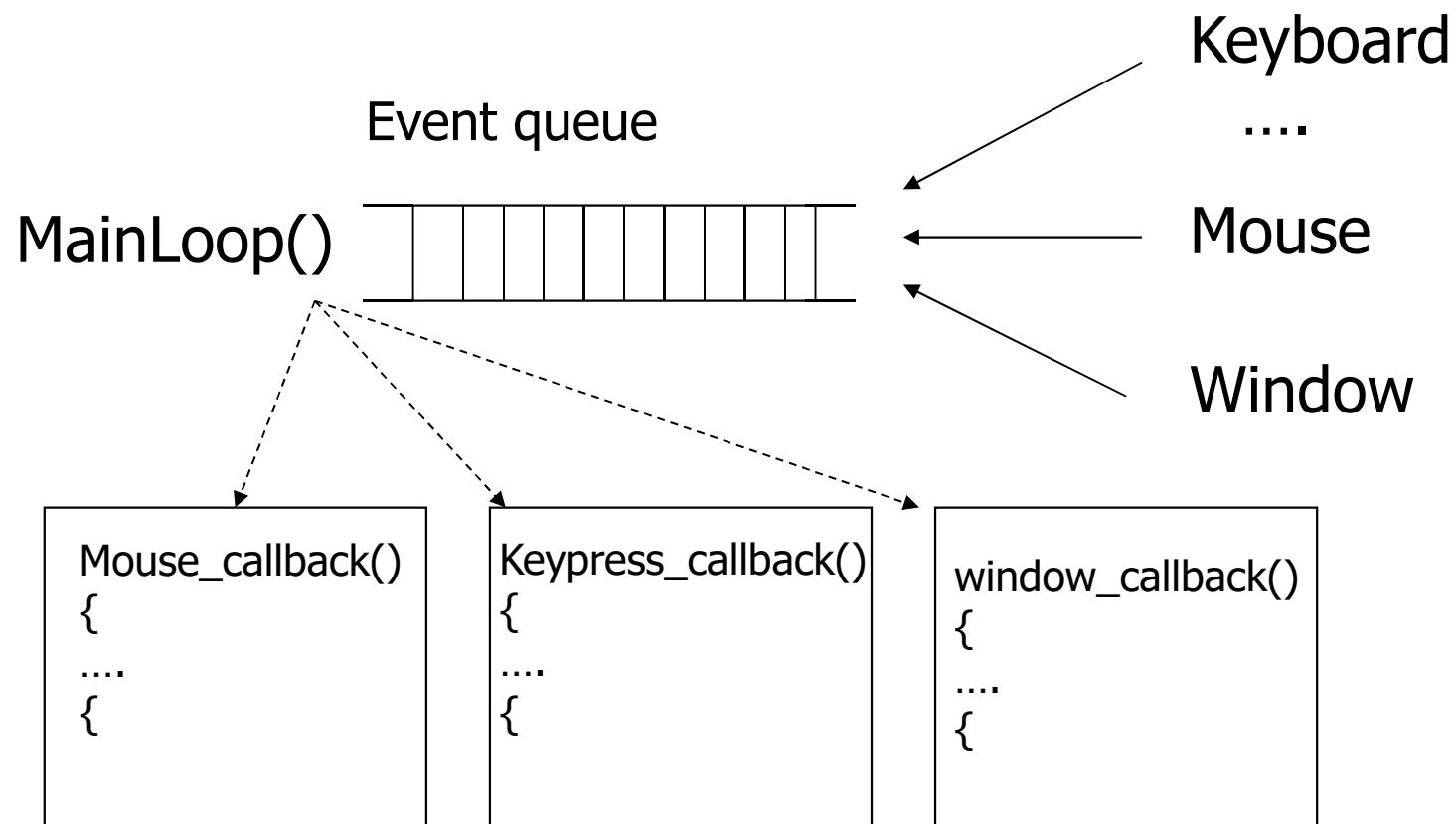
```
int main(int argc, char** argv)
{
    ...
    glutDisplayFunc(mymdisplay);
    ...
}
```



void display() – the function you provide. It contains all the OpenGL drawing function calls and will be called when pixels in the window need to be refreshed.



Event Queue





glut Functions

- **glutKeyboardFunc()** – register the callback that will be called when a key is pressed
- **glutMouseFunc()** – register the callback that will be called when a mouse button is pressed
- **glutMotionFunc()** – register the callback that will be called when the mouse is in motion while a button is pressed
- **glutIdleFunc()** – register the callback that will be called when nothing is going on (no event)



OpenGL Drawing

Steps in the display function

- ❑ Clear the window
- ❑ Set drawing attributes
- ❑ Send drawing commands
- ❑ Flush the buffer



Clear the Window

- **glClear(GL_COLOR_BUFFER_BIT)**
 - ❖ Clears the frame buffer by overwriting it with the background color.
- Background color is a state set by

glClearColor(GLfloat r, GLfloat g, GLfloat b, GLfloat a) in the **init()**.



Drawing Attributes: Color

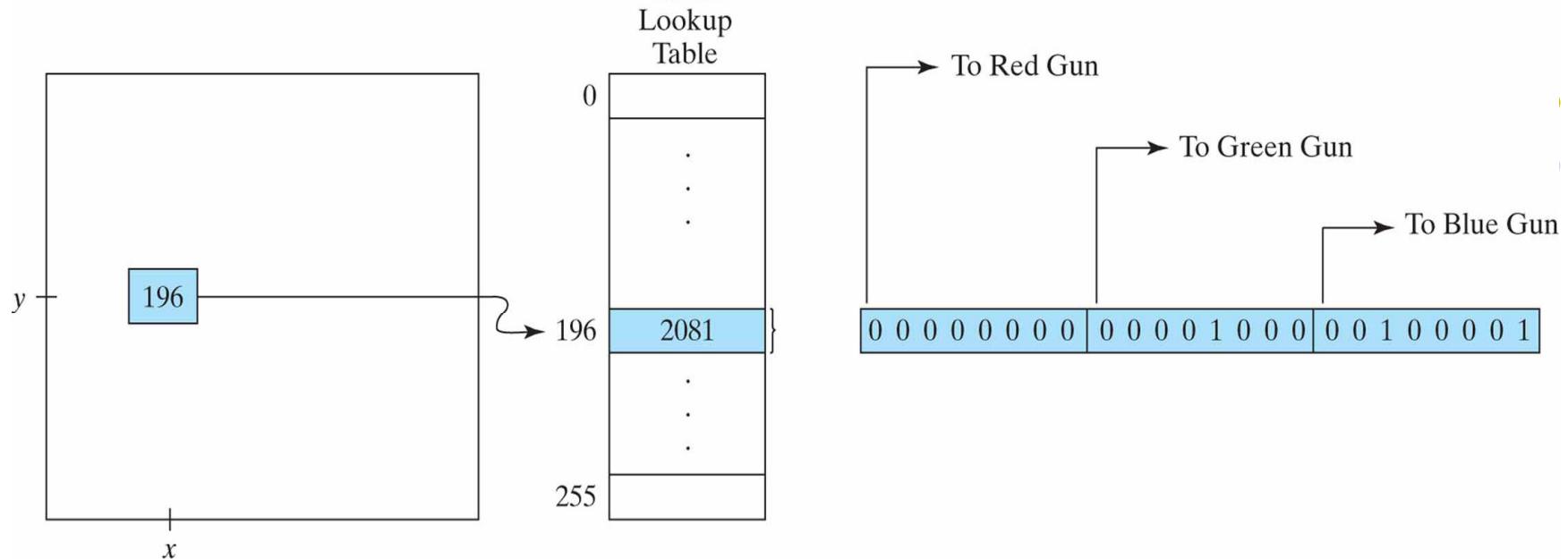
- `glColor3f(GLfloat r, GLfloat g, GLfloat b)`
sets the drawing color
`glColor3d()`, `glColor3ui()` can also be used
- OpenGL is a state machine
 - ❖ Once set, the attribute applies to all subsequent defined objects until it is set to some other value
`glColor3fv()` takes a flat array as input



TABLE 5 - 1

The eight RGB color codes for a 3-bit-per-pixel frame buffer

| Color Code | Stored Color Values in Frame Buffer | | | Displayed Color |
|------------|--|-------|------|-----------------|
| | RED | GREEN | BLUE | |
| 0 | 0 | 0 | 0 | Black |
| 1 | 0 | 0 | 1 | Blue |
| 2 | 0 | 1 | 0 | Green |
| 3 | 0 | 1 | 1 | Cyan |
| 4 | 1 | 0 | 0 | Red |
| 5 | 1 | 0 | 1 | Magenta |
| 6 | 1 | 1 | 0 | Yellow |
| 7 | 1 | 1 | 1 | White |

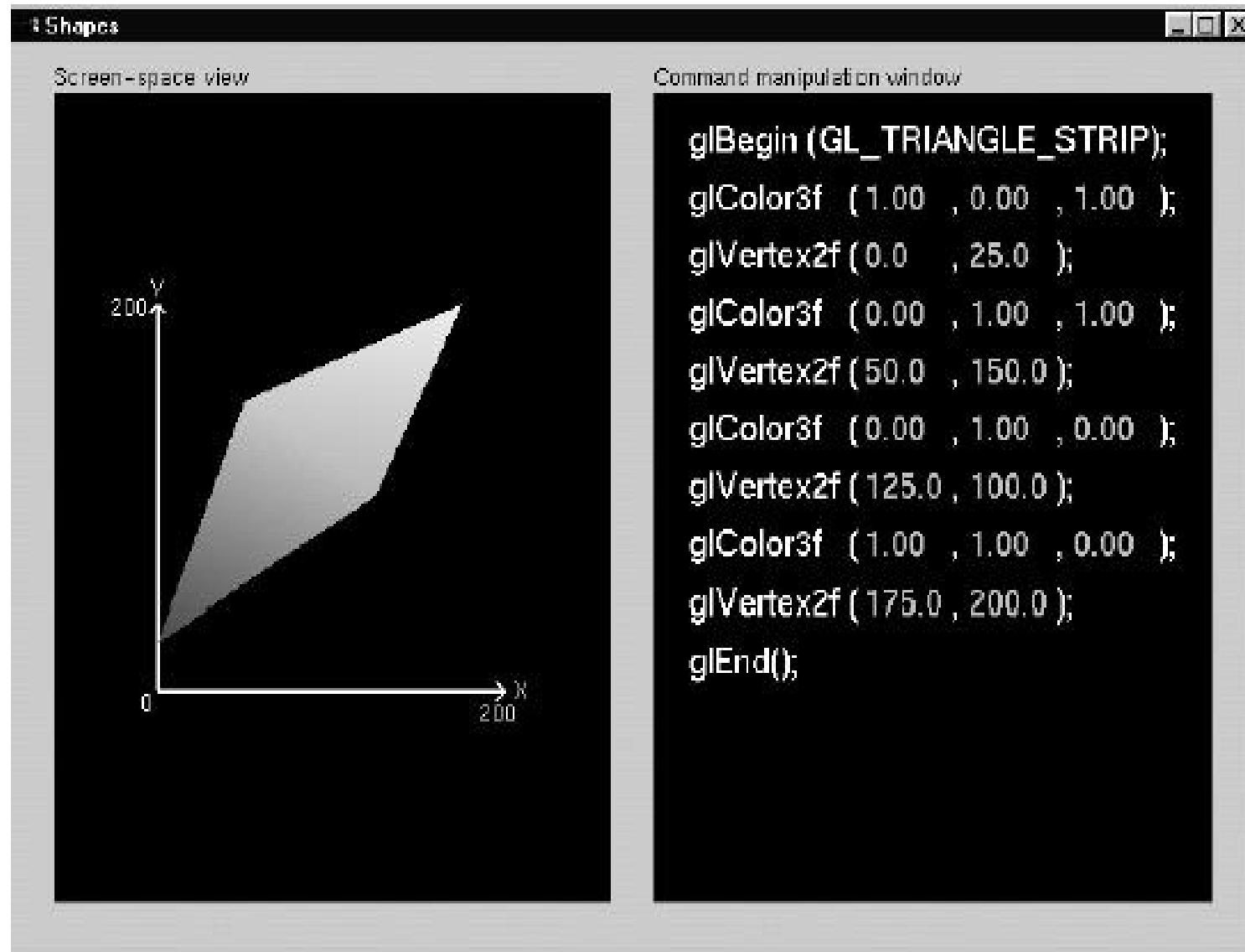


Copyright ©2011 Pearson Education, publishing as Prentice Hall

A **color lookup table** with **24 bits per entry** that is accessed from a frame buffer with **8 bits per pixel**.

A value of 196 stored at pixel position (x, y) references the location in this table containing the hexadecimal value 0x0821 (a decimal value of 2081).

Each **8-bit segment** of this entry controls the intensity level of one of the three electron guns in an **RGB monitor**.





Drawing Commands

□ Simple Objects

glRectf()

□ Complex Objects

- Use construct

glBegin(mode) and
glEnd() and a list of
vertices in between

- **glBegin(mode)**
glVertex(v0);
glVertex(v1);
...
glEnd();

v0 v1 v2 v3 v4
v1 v2 v3 v4
GL_POINTS

v0 v1 v2 v3 v4 v5 v6 v7
GL_LINES

v0 v1 v2 v3 v4 v5 v6 v7
GL_LINE_STRIP

v0 v1 v2 v3 v4 v5 v6 v7
GL_LINE_LOOP

v0 v1 v2 v3 v4 v5
GL_TRIANGLES

v0 v1 v2 v3 v4 v5 v6 v7
GL_TRIANGLE_STRIP

v0 v1 v2 v3 v4 v5 v6 v7
GL_TRIANGLE_FAN

v0 v1 v2 v3 v4 v5 v6 v7
GL_QUADS

v0 v1 v2 v3 v4 v5 v6 v7
GL_QUAD_STRIP

v0 v1 v2 v3 v4 v5 v6 v7
GL_POLYGON



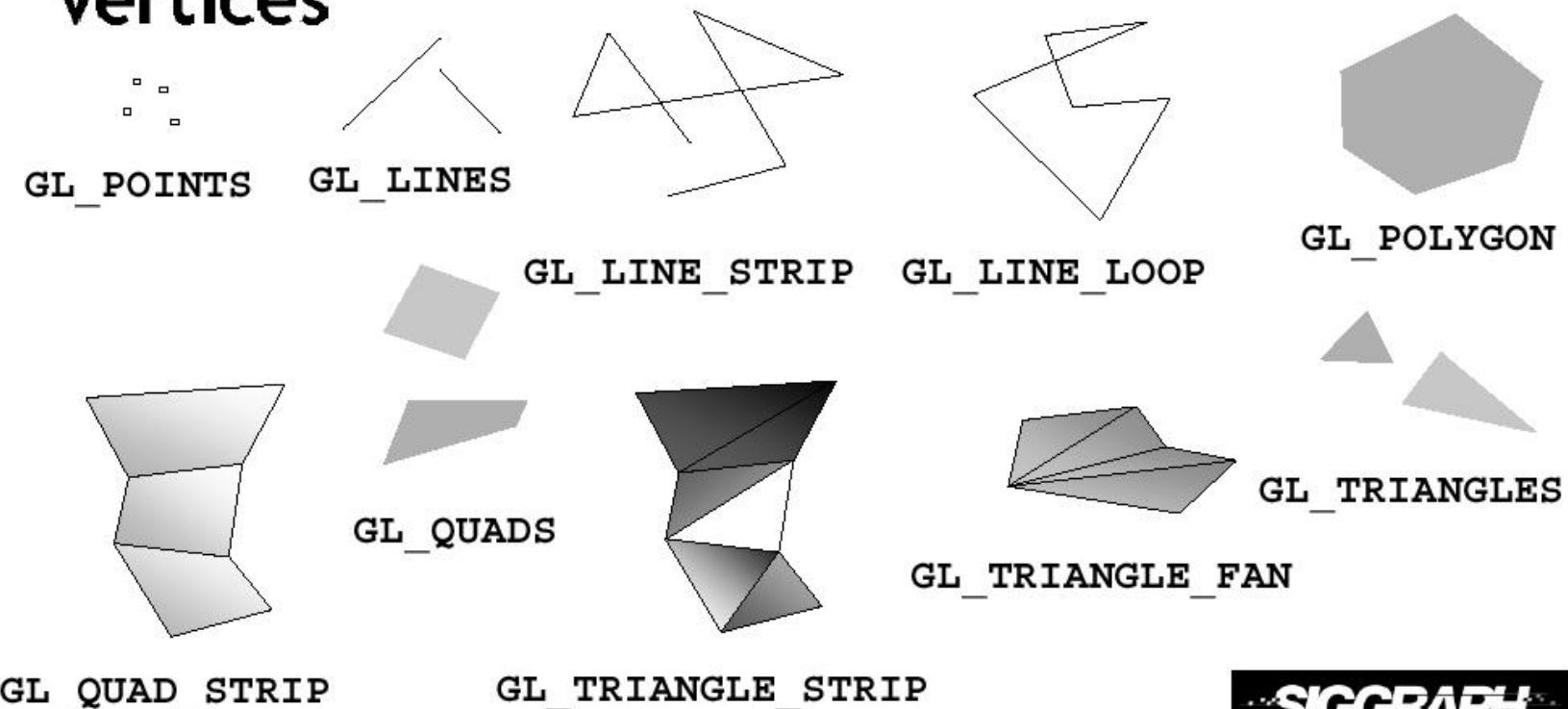
Drawing Attributes

- Besides `glVertex()` commands, other attributes commands can also be used between `glBegin()` and `glEnd()`, e.g. `glColor3f()`.
- There are more drawing attributes than color
 - Point size: `glPointSize()`
 - Line width: `glLineWidth()`
 - Dash or dotted line: `glLineStipple()`
 - Polygon pattern: `glPolygonStipple()`
 - ...



Primitive Types

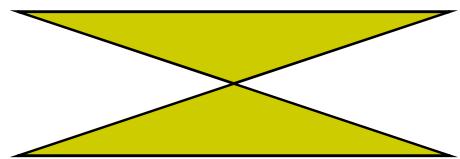
All geometric primitives are specified by vertices





Polygon Issues

- OpenGL will only display polygons correctly that are
 - Simple: edges cannot cross
 - Convex: All points on line segment between two points in a polygon are also in the polygon
 - Flat: all vertices are in the same plane
- User program can check if above true
 - OpenGL will produce output if these conditions are violated but it may not be what is desired
- Triangles satisfy all conditions



nonsimple polygon



nonconvex polygon

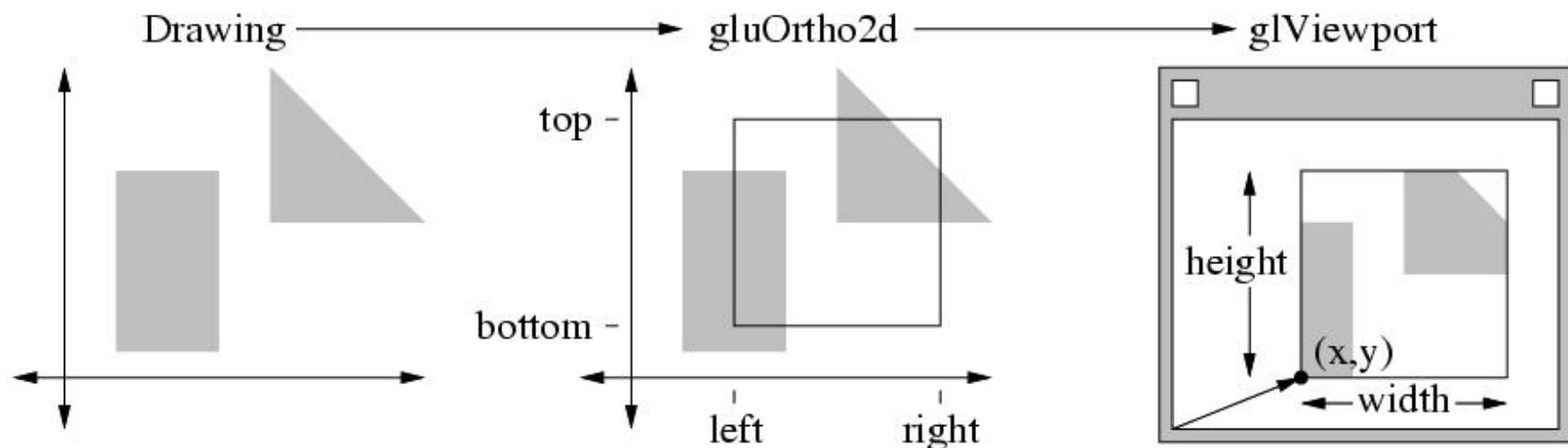


Display Callback

```
void mydisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
        glVertex2f(-0.5, -0.5);
        glVertex2f(-0.5, 0.5);
        glVertex2f(0.5, 0.5);
        glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}
```



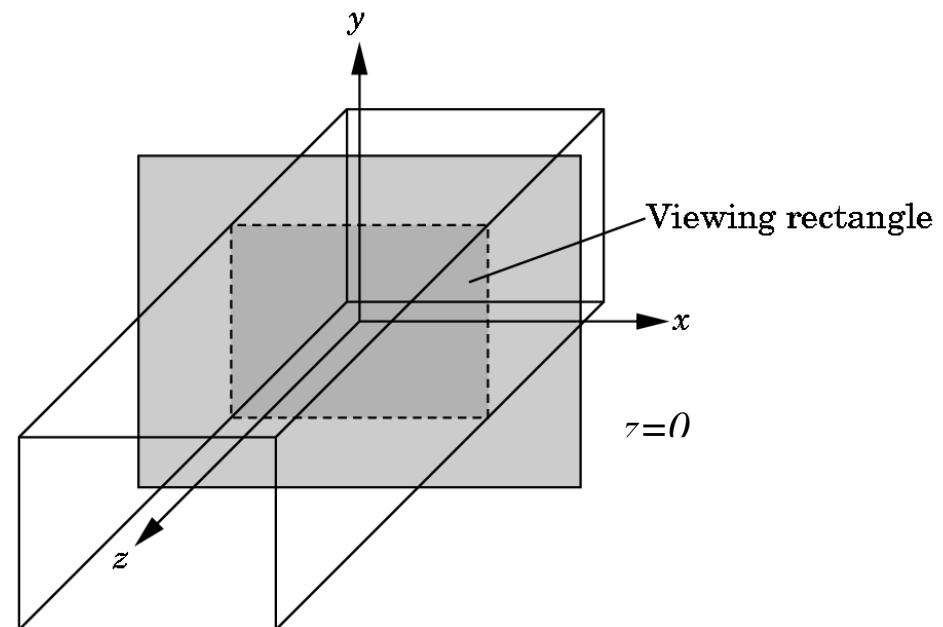
Projection and Viewport





Orthographic projection

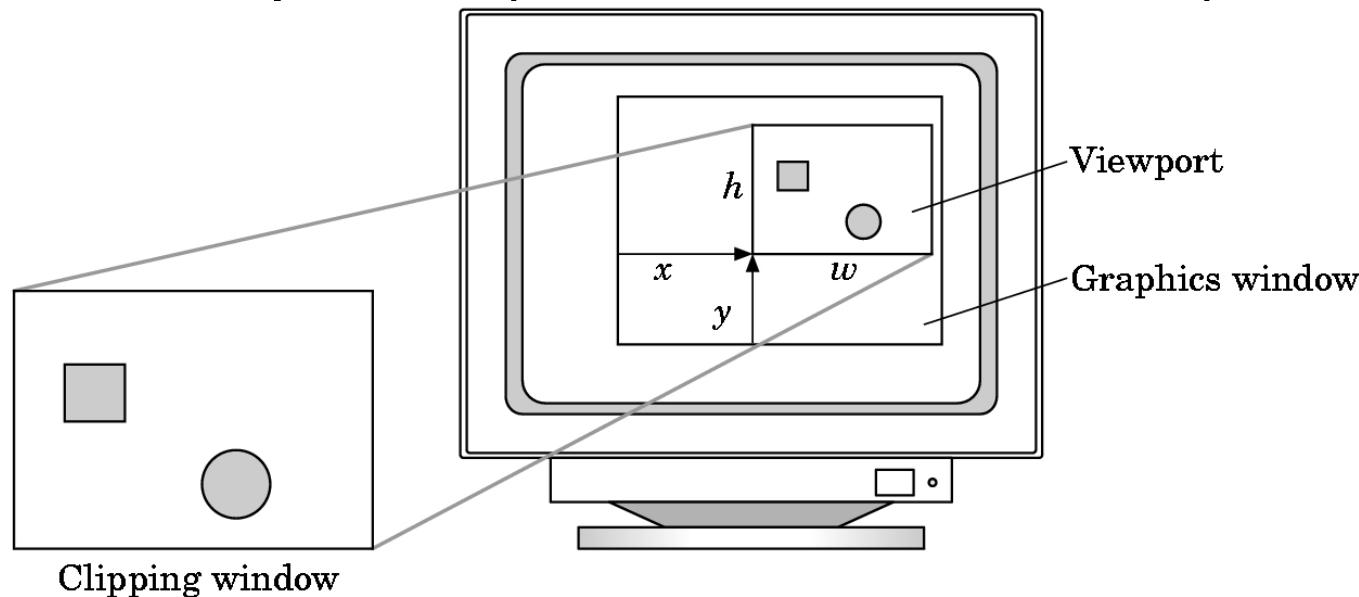
- ❑ Orthographic projection used for 2D drawing,
Perspective project often used for 3D drawing
- ❑ 2D Viewing: Orthographic View
 - **gluOrtho2D (left, right, bottom, top)**
 - Specifies the coordinates of 2D region to be projected into the viewport.
 - Any drawing outside the region will be automatically clipped away.





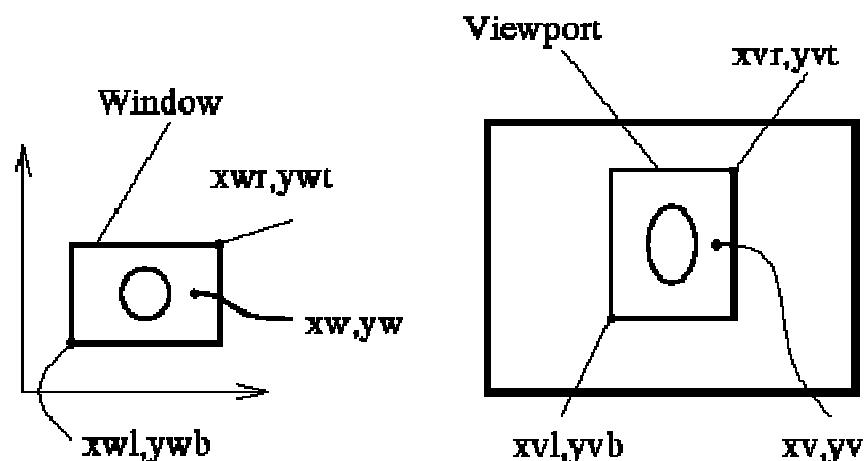
Viewports

- Do not have to use the entire window for the image: **glViewport (x, y, w, h)**
- Values in pixels (screen coordinates)





Window to Viewport mapping



Aspect Ratio: Height/Width
If the aspect ratio of the window
Is different from that of the
viewport, the picture will be
distorted.

2D de Tüm nesne üzerinde Ölçekleme

Eğer $P[x, y]$ satır vektör ise $T = \begin{pmatrix} a & b & p \\ c & d & q \\ m & n & \boxed{s} \end{pmatrix}$ ya da

eğer $P[\begin{smallmatrix} x \\ y \end{smallmatrix}]$ sütun vektör ise $T = \begin{pmatrix} a & c & m \\ b & d & n \\ p & q & \boxed{s} \end{pmatrix}$

m, x yönünde yer değiştirmeye ; n y yönünde yer değiştirmeye ;
 a, b, c, d ölçeklendirme, negatiflendirme, yansıma, döndürmenin
koordinatları ; p, q cisim/noktanın homojen koordinatlardaki
koordinat düzlemi ya da onu simgeleyen düzlem.

Örnek (Tüm nesne üzerinde ölçekleme) : $P(1, 2)$ ve $Q(-1, 1)$
noktalarından oluşan doğruyu 3 birim ölçekleyin.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{3} \end{pmatrix} \cdot \begin{pmatrix} 1 & -1 \\ 2 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ 2 & 1 \\ \frac{1}{3} & \frac{1}{3} \end{pmatrix} = \begin{pmatrix} 3 & -3 \\ 6 & 3 \\ 1 & 1 \end{pmatrix}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 2 & 1 \\ \frac{1}{3} & \frac{1}{3} \end{bmatrix} \quad \begin{bmatrix} x^* \\ y^* \\ 1 \end{bmatrix} = \begin{pmatrix} 3 & -3 \\ 6 & 3 \\ 1 & 1 \end{pmatrix}$$

2D de Ters Dönüşümler

Ölçekleme: $T^{-1}(sx, sy) = T\left(\frac{1}{sx}, \frac{1}{sy}\right)$

$$T^{-1} = \begin{pmatrix} \frac{1}{sx} & 0 & 0 \\ 0 & \frac{1}{sy} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Öteleme (Yer değiştirmeye): $T^{-1}(tx, ty) = T(-tx, -ty)$

$$T^{-1} = \begin{pmatrix} 1 & 0 & -tx \\ 0 & 1 & -ty \\ 0 & 0 & 1 \end{pmatrix}$$

Döndürme: $T^{-1}(\theta) = T(-\theta)$

$$T^{-1} = \begin{pmatrix} \cos(-\theta) & -\sin(-\theta) & 0 \\ \sin(-\theta) & \cos(-\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\cos(-\theta) = \cos\theta$$

$$\sin(-\theta) = -\sin\theta$$

Dönüşüm Sırası: p , sütun matris olsun. M_1, M_2, M_3, M_4 dönüşüm matrisleri olsun.

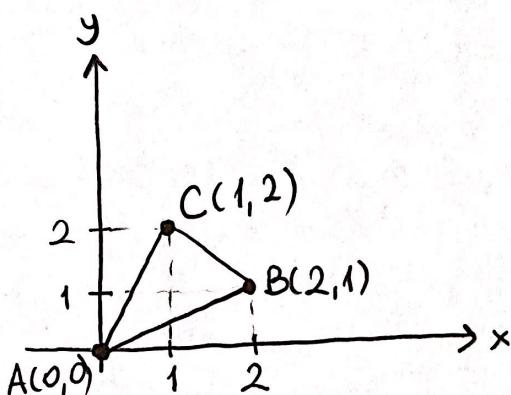
$$p^* = M_4 \cdot M_3 \cdot M_2 \cdot M_1 \cdot p = \left(M_4 \left(M_3 \left(M_2 \left(M_1 p \right) \right) \right) \right)$$

\downarrow
M₁ matrisi ilk uygulanacak olan matris

Yukarıdaki eşitliğin transpozisi alınacak olursa;

$$p^{*T} = p^T \cdot M_1^T \cdot M_2^T \cdot M_3^T \cdot M_4^T$$

Örnek: A(0,0), B(2,1), C(1,2) noktalarından oluşan bir üçgen veriliyor. Cismi (üçgeni) B noktasına göre x ve y yönlerinde 2 birim ölçekleyin.



1. adım: B noktasına göre dediği için B noktası orijine taşınır. (T_1)

2. adım: Belirtilen ölçekleme yapılır. (T_2)

3. adım: B'yi eski yerine (ilk koordinatlarına) taşıriz. (T_3)

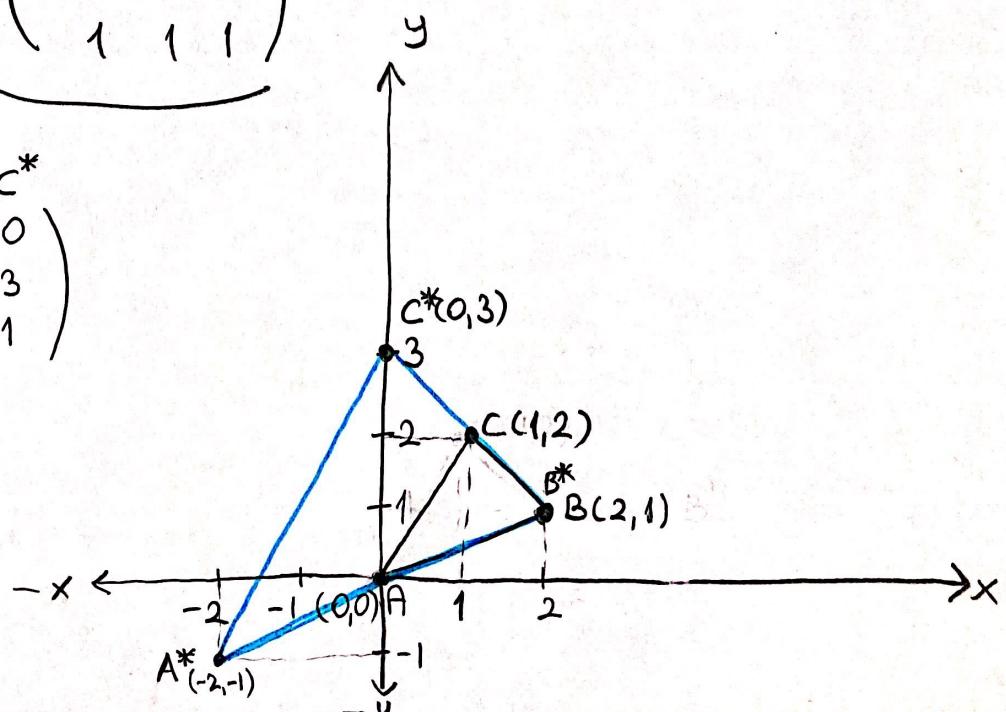
$$T_3 \cdot T_2 \cdot T_1 \quad \xleftarrow{\text{matris çarpım yönü}}$$

$$\begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & -2 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 2 & 1 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} -2 & 0 & -1 \\ -1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} -4 & 0 & -2 \\ -2 & 0 & 2 \\ 1 & 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} A^* & B^* & C^* \\ -2 & 2 & 0 \\ -1 & 1 & 3 \\ 1 & 1 & 1 \end{pmatrix}$$



2. çözüm: Tüm nesne üzerinde ölçekteme yapılabilir. Çünkü $m=n=2$, x ve y yönünde aynı ölçekteme yapılması istenmiş.

$$(T_3 \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}) \cdot (T_2 \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/2 \end{pmatrix}) \cdot (T_1 \begin{pmatrix} 1 & 0 & -2 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix}) \cdot \begin{pmatrix} A & B & C \\ 0 & 2 & 1 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{pmatrix}$$

$\delta = \frac{1}{2}$ alınır

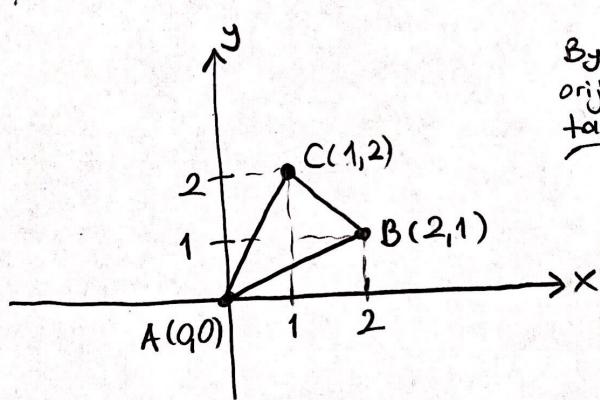
matris çarpım yönü

B noktasına göre ölçekteme dediği için B degismeyecek.

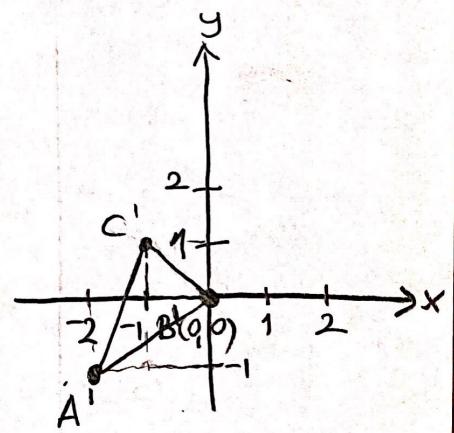
Günkü B ye göre hesaplarken önce B yi orijine taşıriz. Böylece B nin koordinatları $(0,0)$ olur. Ölçekleyip eski haline alınca tekrar $(2,1)$ olacak yani son durumda B nin koordinatları değişmemiş olacak.

1. adım: B noktası orijini taşırı (T_1)

$$\left(T_1 \begin{pmatrix} 1 & 0 & -2 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix} \right) \cdot \begin{pmatrix} A & B & C \\ 0 & 2 & 1 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} A' & B' & C' \\ -2 & 0 & -1 \\ -1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

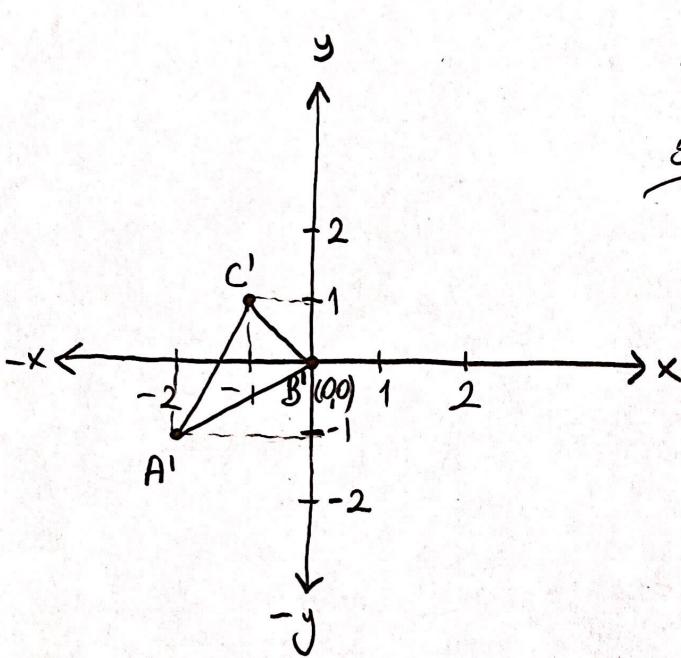


Byi.
orijine
taşı

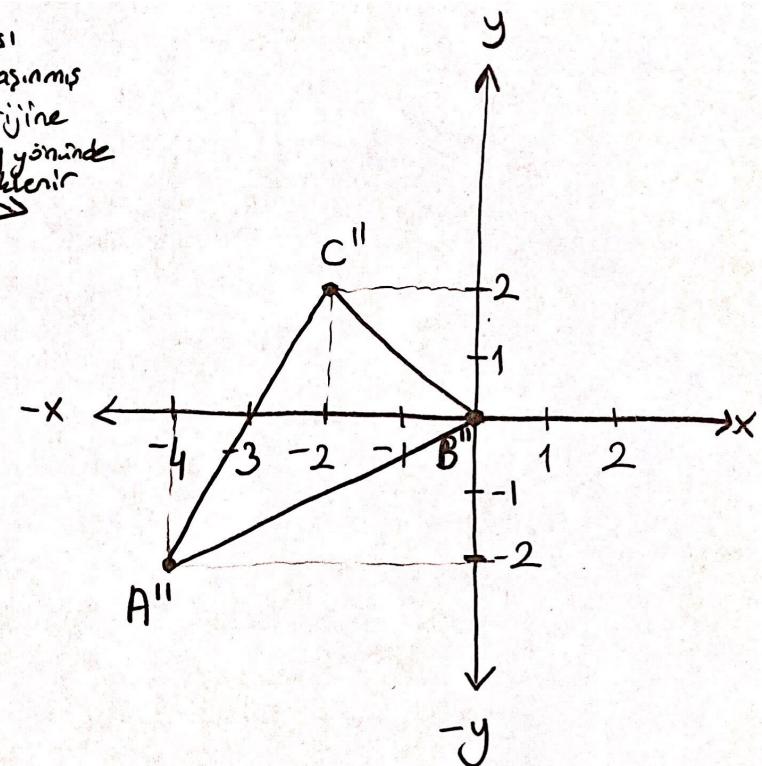


2. adım: Orijine göre x ve y yönünde 2 birim ölçekleyin. (T_2)

$$\left(T_2 \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \right) \left(\begin{pmatrix} A' & B' & C' \\ -2 & 0 & -1 \\ -1 & 0 & 1 \end{pmatrix} \right) = \begin{pmatrix} A'' & B'' & C'' \\ -4 & 0 & -2 \\ -2 & 0 & 2 \\ 1 & 1 & 1 \end{pmatrix}$$

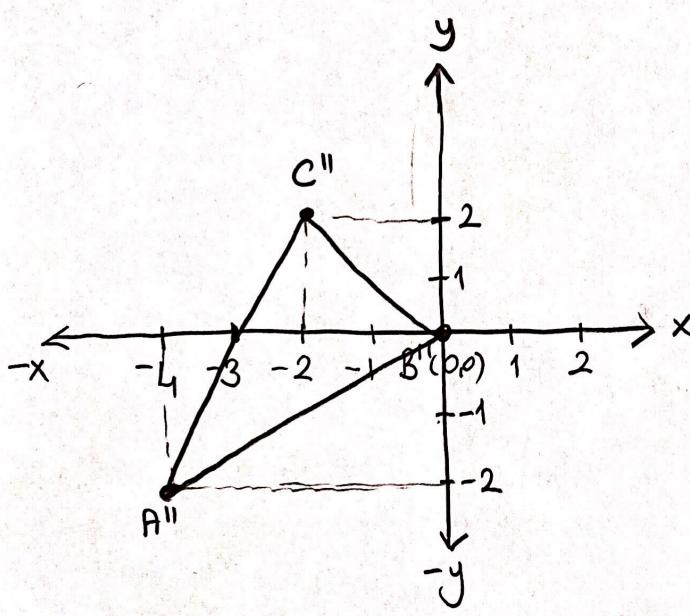


B noktası
orijine taşınır
üçgen orijine
göre x ve y yönünde
2 katı uzaklaşır

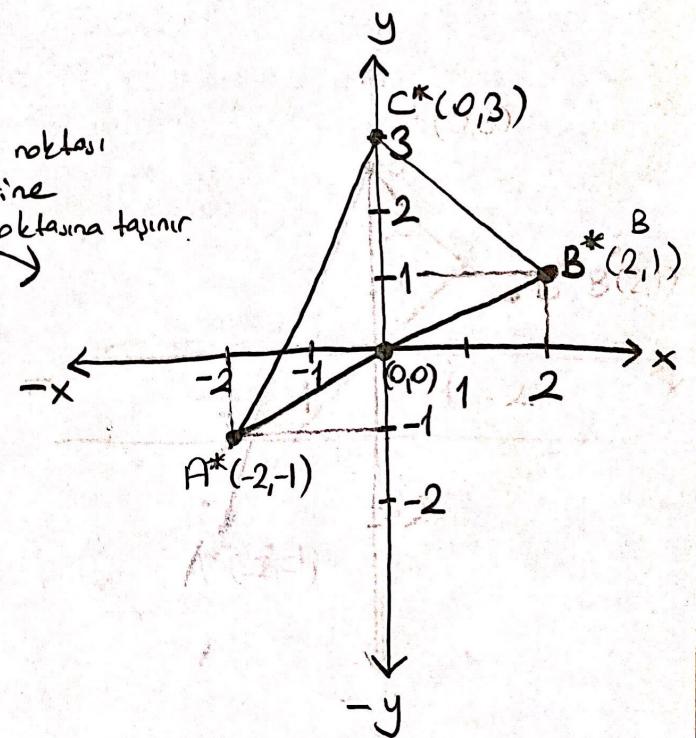


3. adım: Orijindeki B noktası eski yerine (ilk koordinatlarına) taşınır. (T_3)
(B'' noktası)

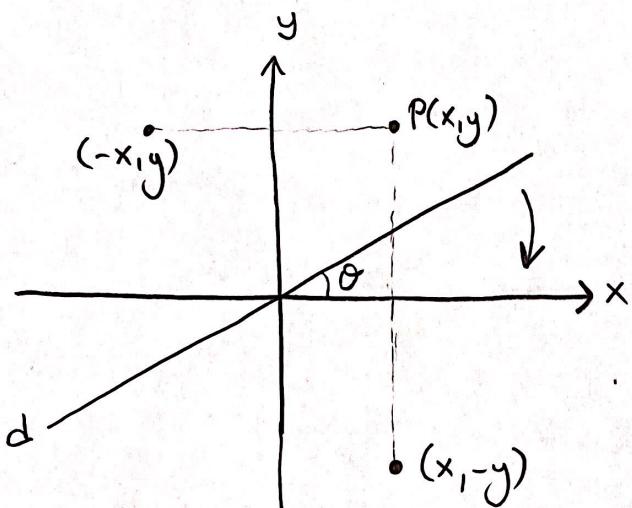
$$\begin{pmatrix} T_3 \\ 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} A'' & B'' & C'' \\ -4 & 0 & -2 \\ -2 & 0 & 2 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} A^* & B^* & C^* \\ -2 & 2 & 0 \\ -1 & 1 & 3 \\ 1 & 1 & 1 \end{pmatrix}$$



$B''(0, 0)$ noktası
eski yerine
(2, 1) noktası
taşınır



Bir p noktasının orijinden geçen doğruya (eksen) göre yansıtılması



1. Adım: d doğrusu x eksenile çakıştırılır. Saat yönünde (negatif yönde) θ açısı kadar döndürülür. x eksen ile çakıştırma için döndürme yapıyoruz.

$$T_1 = \begin{pmatrix} \cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{pmatrix}$$

$$T_1 = \begin{pmatrix} \cos \theta & -\sin \theta \\ -\sin \theta & \cos \theta \end{pmatrix}$$

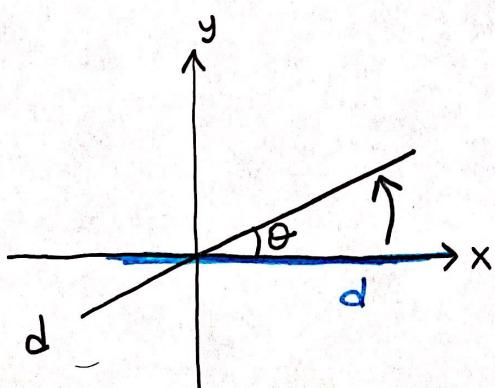
x 'e göre yansıma: $\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ -y \end{pmatrix}$

y 'ye göre yansıma: $\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -x \\ y \end{pmatrix}$

2. Adım: x eksenine göre yansıma için $T_2 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$

3. Adım: d doğrusu eski yerine taşınır.

Saat yönünün tersi yönünde (pozitif yönde) θ açısı kadar döndürülür.



$$T_3 = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$$

$$T_3 \cdot T_2 \cdot T_1 \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x^* \\ y^* \end{pmatrix}$$

$$T_3 \cdot T_2 \cdot T_1 \cdot P = P^*$$

Örnek: P(1,2) noktasının x eksenile 45° lik açı yapan bir d doğrusuna göre yansıtılmasından sonra oluşan yeni noktasının koordinatlarını bulunuz.

T_1 : d doğrusu x ile çakıştırılır. Saat yönünde 45° döndürülür.

$$T_1 = \begin{pmatrix} \cos(-45) & -\sin(-45) \\ \sin(-45) & \cos(-45) \end{pmatrix} = \begin{pmatrix} \cos 45 & \sin 45 \\ -\sin 45 & \cos 45 \end{pmatrix} = \begin{pmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{pmatrix}$$

T_2 : x eksenile göre yansıtma yapılır: $T_2 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$

T_3 : d doğrusu eksi y eksenine tasınır. Saat yönünün tersi yönünde 45° döndürülür.

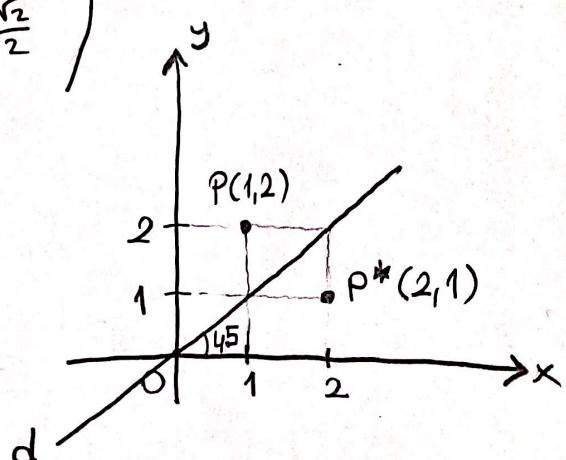
$$T_3 = \begin{pmatrix} \cos 45 & -\sin 45 \\ \sin 45 & \cos 45 \end{pmatrix} = \begin{pmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{pmatrix}$$

Öyleyse;

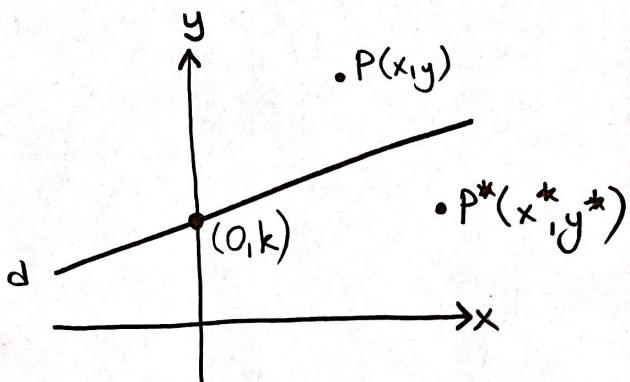
$$\begin{pmatrix} x^* \\ y^* \end{pmatrix} = T_3 \cdot T_2 \cdot T_1 \cdot \underbrace{\begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}}_{P}$$

$$\begin{pmatrix} x^* \\ y^* \end{pmatrix} = \begin{pmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

$$\begin{pmatrix} x^* \\ y^* \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$



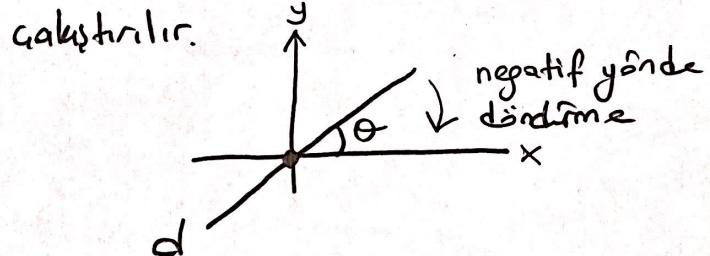
Bir p noktasını herhangi bir doğruga (eksen) göre yansıtma



1. Adım: d doğrusu orijine tasınır.

$$T_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -k \\ 0 & 0 & 1 \end{pmatrix}$$

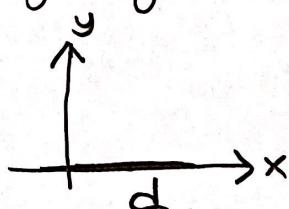
2. Adım: d doğrusu x eksenile
çakıştırılır.



$$T_2 = \begin{pmatrix} \cos(-\theta) & -\sin(-\theta) & 0 \\ \sin(-\theta) & \cos(-\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

3. Adım: P noktası x eksenine göre yansıtılır.

$$T_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

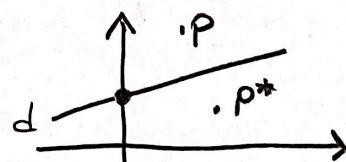


4. Adım: d doğrusu θ aksı kadar pozitif yönde döndürülür.

$$T_4 = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

5. Adım: d doğrusu k birim kadar eski yerine tasınır (öteleşenir).

$$T_5 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & k \\ 0 & 0 & 1 \end{pmatrix}$$

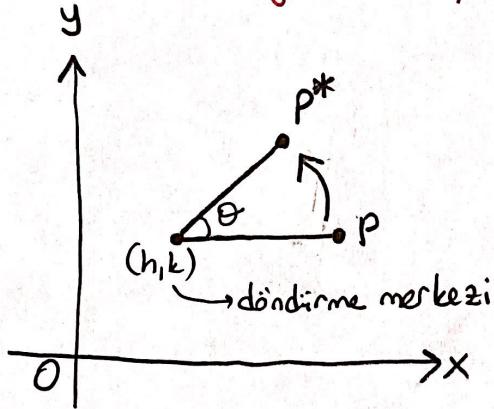


$$\begin{pmatrix} x^* \\ y^* \\ 1 \end{pmatrix} = T_5 \cdot T_4 \cdot T_3 \cdot T_2 \cdot T_1 \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

yer deşifirme oldugu
için durum sektörü
böyle yazılır.

$$P^* = T_5 \cdot T_4 \cdot T_3 \cdot T_2 \cdot T_1 \cdot P$$

Herhangi bir p noktasına göre döndürme



Döndürme yapılıcada OPR olabilir.

PQR olabilir.

ABC olabilir.

Önenli olan neye göre döndürme yapacağımıza.

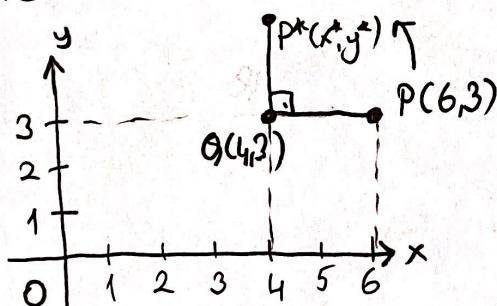
1. Adım: Döndürme merkezi orijine tasınır. (T_1)

2. Adım: θ açısı kadar pozitif yönde (saat yönünün tersi) döndürme yapılır. (T_2)

3. Adım: Döndürme merkezi eski haline getirilir. (T_3)

$$T_3 \cdot T_2 \cdot T_1 \cdot \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} x_1^* & x_2^* & x_3^* \\ y_1^* & y_2^* & y_3^* \\ 1 & 1 & 1 \end{pmatrix}$$

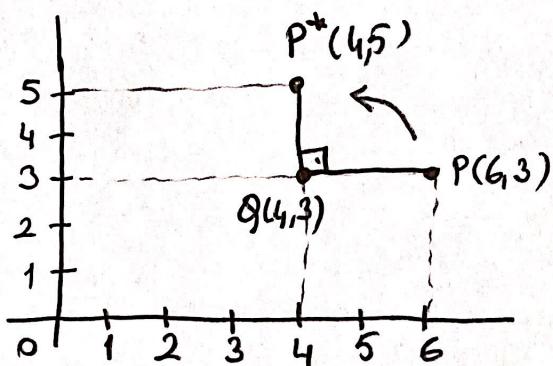
Örnek: $P(6,3)$ noktasının $Q(4,3)$ noktasına göre 90° döndürülmesi sonunda P^* noktasının koordinatlarını bulunuz.



$$\underbrace{\left(\begin{array}{ccc} 1 & 0 & 4 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{array} \right) \cdot \left(\begin{array}{ccc} \cos 90^\circ & -\sin 90^\circ & 0 \\ \sin 90^\circ & \cos 90^\circ & 0 \\ 0 & 0 & 1 \end{array} \right) \cdot \left(\begin{array}{ccc} 1 & 0 & -4 \\ 0 & 1 & -3 \\ 0 & 0 & 1 \end{array} \right)}_{''} \cdot \left(\begin{array}{ccc} 6 & 4 & 1 \\ 3 & 3 & 1 \\ 1 & 1 & 1 \end{array} \right) = \left(\begin{array}{ccc} P^* & Q & \\ x^* & y & \\ 1 & 1 & \end{array} \right)$$

$$\left(\begin{array}{ccc} 1 & 0 & 4 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{array} \right) \cdot \underbrace{\left(\begin{array}{ccc} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{array} \right)}_{P^* \text{ } Q} \cdot \left(\begin{array}{ccc} 2 & 0 & \\ 0 & 0 & \\ 1 & 1 & \end{array} \right) = \left(\begin{array}{ccc} 4 & 4 & \\ 5 & 3 & \\ 1 & 1 & \end{array} \right)$$

$P(6,3)$ noktası $Q(4,3)$ noktasına göre 90° döndürildiğinde $P^*(4,5)$ noktası elde edilir.



Örnek: $A(3,1)$ bir noktası ve $T = \begin{pmatrix} -1 & -3 & 5 \\ 2 & 2 & -1 \\ -2 & 1 & -1/2 \end{pmatrix}$ dönüşümü olsun.

A noktasına T dönüşümü uygulandıktan sonra A^* noktasının koordinatlarını bulunuz. (Homojen koordinatlara örnek)

$$\begin{pmatrix} -1 & -3 & 5 \\ 2 & 2 & -1 \\ -2 & 1 & -1/2 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ 7 \\ -\frac{11}{2} \end{pmatrix} = \begin{pmatrix} X \\ Y \\ H \end{pmatrix}$$

$$\begin{pmatrix} X \\ Y \\ H \end{pmatrix} = \begin{pmatrix} -\frac{1}{7} \\ -\frac{11}{2} \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{2}{11} \\ -\frac{14}{11} \\ 1 \end{pmatrix} = \begin{pmatrix} X^* \\ Y^* \\ 1 \end{pmatrix}$$

H yi 1 yapmanız gereklidir.

X ve Y yi $-\frac{11}{2}$ ye böldük.

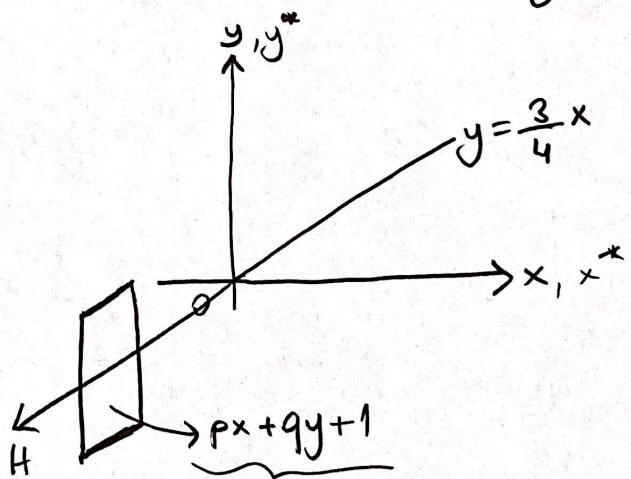
$A^*(\frac{2}{11}, -\frac{14}{11})$ noktasının homojen koordinat sistemindeki $\begin{pmatrix} X \\ Y \\ H \end{pmatrix}$

şüyleden biri $\begin{pmatrix} -\frac{1}{7} \\ -\frac{11}{2} \\ 1 \end{pmatrix}$ dir. $A^*(x^*, y^*)$ noktasının homojen

koordinat sisteminde sonsuz şüylediği vardır. Örneğin; $\begin{pmatrix} -2 \\ 14 \\ 11 \end{pmatrix}$ ve

$\begin{pmatrix} 2 \\ -14 \\ 11 \end{pmatrix}$ bu şüyleden iki tanesidir.

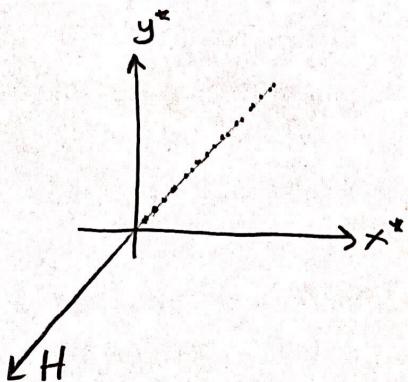
Homojen koordinat sistemlerinde $H=0$ ise $ay-bx=0$ dğrnesi sonsuzda bir nokta gösterir.



düzlende x, y sabit
olarak H sonsuz değeri ile
sonsuz nokta yazabilirim.

② $H=0$ düzlemini göstereneyi^z
o sonsuzda bir noktadır.

$$\begin{aligned}
 \begin{bmatrix} x \\ y \\ H \end{bmatrix} &= \begin{bmatrix} 4 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \\ 1 \end{bmatrix} \\
 &= \begin{bmatrix} 4 \\ 3 \\ 1/2 \end{bmatrix} = \begin{bmatrix} 8 \\ 6 \\ 1 \end{bmatrix} \\
 &= \begin{bmatrix} 4 \\ 3 \\ 1/200 \end{bmatrix} = \begin{bmatrix} 800 \\ 600 \\ 1 \end{bmatrix} \\
 &= \begin{bmatrix} 4 \\ 3 \\ 1/2000 \end{bmatrix} = \begin{bmatrix} 8000 \\ 6000 \\ 1 \end{bmatrix} \\
 &= \begin{bmatrix} 4 \\ 3 \\ 0 \end{bmatrix} = \begin{bmatrix} \infty \\ \infty \\ 1 \end{bmatrix}
 \end{aligned}$$



3D de Dönüşümler

Ölçeklene: $T = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Yer değiştirmeye (öteleme): $\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Döndürme: Her dönmeye, 3 eksen etrafında 3 farklı saat yönünün tersine dönüş açısının birleşimi olarak temsil edilebilir, yani

yz düzleminde x eksenine Ψ açısı kadar,

xz düzleminde y eksenine Θ açısı kadar,

xy düzleminde z eksenine ϕ açısı kadar.

$$R_x: R_{yz}(\Psi): x$$
 eksenine göre Ψ açısı kadar döndürme matrisi: $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \Psi & -\sin \Psi & 0 \\ 0 & \sin \Psi & \cos \Psi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

$R_y: R_{zx}(\Theta): y$ eksenine göre Θ açısı kadar döndürme: $\begin{pmatrix} \cos \Theta & 0 & \sin \Theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \Theta & 0 & \cos \Theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

$R_z: R_{xy}(\phi): z$ eksenine göre ϕ açısı kadar döndürme: $\begin{pmatrix} \cos \phi & -\sin \phi & 0 & 0 \\ \sin \phi & \cos \phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Döndürme eksenlere göre yapılır. Döndürme yapmadan önce mutlaka cisimin origine taşıınmalıdır.

3D de Dönüşümlerin Tersi

(Dönüşüm Matrikslerinin Tersi)

Ölçeklene: $T^{-1} = \begin{pmatrix} 1/s_x & 0 & 0 & 0 \\ 0 & 1/s_y & 0 & 0 \\ 0 & 0 & 1/s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

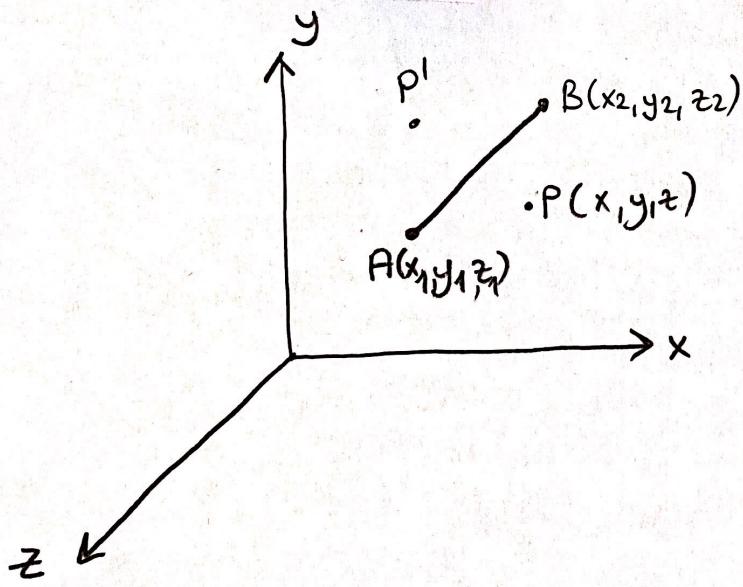
Öteleme: $T^{-1} = \begin{pmatrix} 1 & 0 & 0 & -dx \\ 0 & 1 & 0 & -dy \\ 0 & 0 & 1 & -dz \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Döndürme: x eksenine göre: $R_{yz}^{-1}(\psi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\psi & \sin\psi & 0 \\ 0 & -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

y eksenine göre: $R_{zx}^{-1}(\theta) = \begin{pmatrix} \cos\theta & 0 & -\sin\theta & 1 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

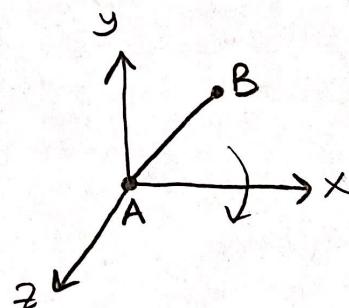
z eksenine göre $R_{xy}^{-1}(\phi) = \begin{pmatrix} \cos\phi & \sin\phi & 0 & 0 \\ -\sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

3D de Keyfi Bir Eksen Etrafında Döndürme



1. Adım: A noktası orijine taşınır.

$$T_1 = \begin{pmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



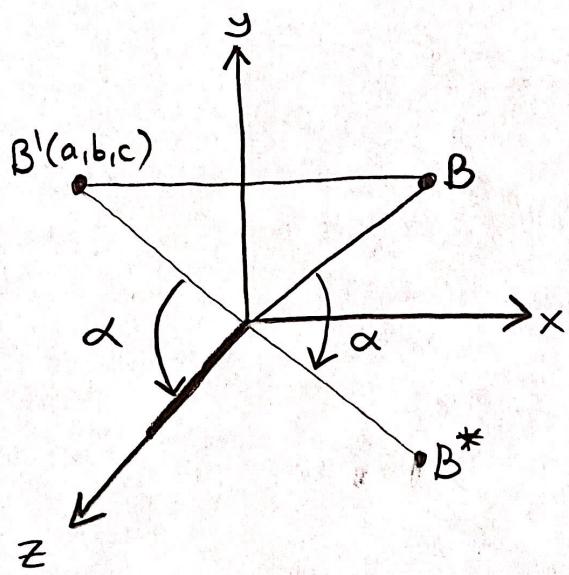
AB doğrusunun z eksenine izdüşümünü alalım. Daha sonra cisim (p noktası) istenilen açı kadar (30°) z eksen etrafında döndürülür.

AB nin z üzerine izdüşümü 2 adımda gerçekleşir:

- AB doğrusunu x eksen etrafında α kadar döndürme (negatif yönde) sonucunda yz düzlemeceye izdüşürülmelii.

- α açısı kadar negatif yönde dönen, elde edilen, dooprugu yekeni etrafında öyle bir β açısı kadar döndürülüm ki dooprular \neq ekesi ile快讯sin.

2. Adım:



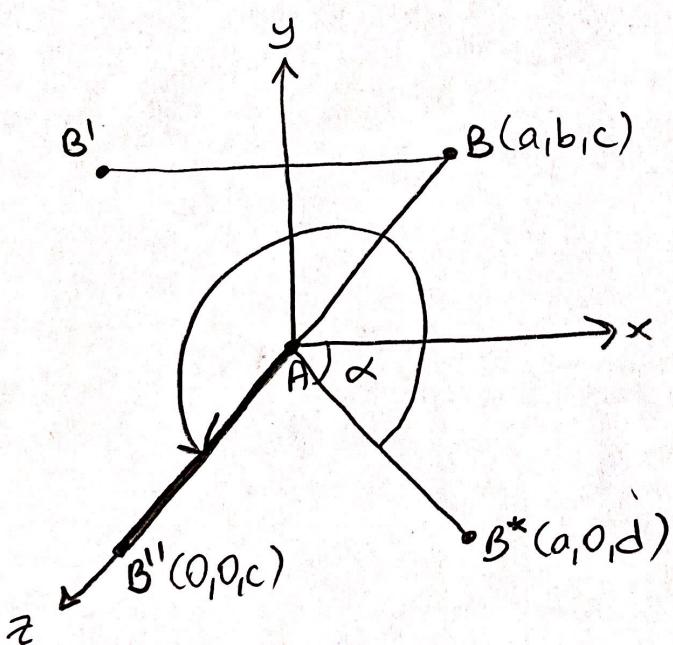
B noktası α açısı kadar x eksene göre negatif yönde döndürülür. Böylece B noktası yz düzleminde izdüşülmüş olur.

$$T_2 =$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{d} & \frac{b}{d} & 0 \\ 0 & -\frac{b}{d} & \frac{c}{d} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(-) yönde döndürdiğimiz için $-$ oldu.

3. Adım: B^* noktası y ekseni etrafında β açısı kadar döndürülerek dooprular z ekseni ile快讯sin sağlanır. Böylece AB doopruları AB'' olur. (yz düzlemindeki B' noktası z ekseni üzerinde gelir(B''))



$$T_3 =$$

$$\begin{pmatrix} \frac{d}{l} & 0 & -\frac{a}{d} & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{a}{d} & 0 & \frac{d}{l} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

4. Adım: AB doğrusu z eksenile çakışır. z ye göre
 30° döndürme yapılır.

$$T_4 = \begin{pmatrix} \cos 30 - \sin 30 & 0 & 0 \\ \sin 30 & \cos 30 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

5. Adım: y eksenine göre + yönde β açısı kadar döndürme

6. Adım: x eksenine göre + yönde α açısı kadar döndürme

7. Adım: AB doğrusu (eksen) eski yerine tasınır.

Not:
$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

izdüşümler bu sırada
perspektif satırı