T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DIRECTX TABANLI ÜÇ BOYUTLU OYUN MOTORU TASARIMI

YÜKSEK LİSANS TEZİ

Bilg.Müh. Kayhan AYAR

Enstitü Anabilim Dalı : Bilgisayar ve Bilişim Müh.

Tez Danışmanı : Doç. Dr. Cemil ÖZ

Ocak 2010

T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DIRECTX TABANLI ÜÇ BOYUTLU OYUN MOTORU TASARIMI

YÜKSEK LİSANS TEZİ

Bilg.Müh. Kayhan AYAR

Enstitü Anabilim Dalı : Bilgisayar ve Bilişim Müh.

Bu tez 08 / 01 /2010 tarihinde aşağıdaki jüri tarafından Oybirliği ile kabul edilmiştir.

Doç.Dr. Cabir VURAL	Doç.Dr. Cemil ÖZ	Doç.Dr. Nejat YUMUŞAK
Jüri Başkanı	**Üye**	**Üye**

TEŞEKKÜR

Tezimin başından bitimine kadar geçen sürede bana sürekli destek olan danışman hocam Doç. Dr. Cemil ÖZ' e ve desteklerini benden esirgemeyen aileme teşekkür ederim.

Kayhan AYAR

ÖZGEÇMİŞ

Kayhan Ayar, 29.03.1982'de Muş'da doğdu. İlk ve orta öğretimini Sakarya da tamamladı. 2000 yılında İstanbul'un Pendik ilçesinde bulunan Rauf Denktaş Lisesi'nden mezun oldu. 2000 yılında başladığı Sakarya Üniversitesi Bilgisayar Mühendisliği Bölümü'nü 2007 yılında bitirdi. 2007 yılında bir dönem ücretli bilgisayar öğretmenliği yaptı. 2008 yılında Sakarya Üniversitesi Bilgisayar ve Bilişim Mühendisliği Anabilim Dalı'nda yüksek lisansa başladı. 2009 Ocak ayında Sakarya Üniversitesi Bilgisayar Mühendisliği bölümünde araştırma görevlisi olarak göreve başladı. Halen aynı görevi sürdürmektedir.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR LİSTESİ

3-B	: Üç boyut
API	: Yazılım programlama arayüzü
COM	: Parçacık nesne modeli
CPU	: Merkezi işlemci ünitesi
CRT	: Cathode Ray Tube
DAC	: Dijitalden analoğa çevirici
DOS	: Disk işletim sistemi
GDI	: Grafik aracı ara yüzü
GPU	: Grafik işlemci ünitesi
GUI	: Grafiksel kullanıcı arayüzü
HAL	: Donanım soyutlama katmanı
HEL	: Donanım emülasyon arayüzü
STL	:Standart şablon kütüphanesi
SDK	: Yazılım geliştirme aracı

ŞEKİLLER LİSTESİ

TABLOLAR LİSTESİ

ÖZET

Anahtar kelimeler: Oyun motoru, DirectX

Bilgisayar oyunları günümüz yazılım dünyasında çok büyük bir yer kaplamaktadır. Her geçen gün kapladıkları bu alan daha da büyümektedir. Bu günün bilgisayar oyunları yüz milyonlarca dolar gelir getirerek Hollywood film piyasası ile yarışmaktadırlar. Fakat bilgisayar oyunlarını geliştirmek yıllar alabilmektedir. Bu da oyun geliştirmeyi oldukça masraflı bir iş haline getirmektedir. Oyun motorları sayesinde oyunlar çok daha kolay ve hızlı geliştirilebilmektedir.

Bu tez çalışmasında DirectX kütüphanelerini kullanarak üç boyutlu bir oyun motoru geliştirilmiştir. Motor için sırası öncelikle yardımcı kütüphaneler geliştirilmiştir. Ardından DirectX' i kapsayacak olan ara yüzler tasarlanmıştır. Sonraki aşamalarda sırası ile ışık, kaplama, zemin, kamera, model ve sahne yönetim mekanizmaları geliştirilmiştir. Programcı bu mekanizmaları kullanarak çok daha kolay bir şekilde oyun geliştirebilecektir. Diğer motorlardan farklı olarak birden fazla model formatını okuyup, sahne içerisinde hareket ettirilebilmektedir. Zemin oluşturma mekanizması sayesinde de sanal dünyalar hızlı bir şekilde tasarlanabilmektedir.

DEVELOPING A 3D GAME ENGINE BASED ON DIRECTX

SUMMARY

Key Words: Game Engine, DirectX

Today, computer games have a big place in the software world and every passing day this place is getting bigger. With their massive income, today's computer games compete with Hollywood. But making a computer game may take years. For that reason developing a game can be expansive. In that moment game engines enter the scene. With game engines creating a game became much more easy.

In this thesis a 3D game engine based on DirectX libraries was developed. First, utulity library was created. After that a layer between the engine and DirectX was built. In the next stages, model,texture, light, terrain, camera and scene manager were developed. Through using the libraries of the game engines, programming a game is become much more easier. With the editor inside the engine, designers can easily create their game world. As a result , a job may take years can be done in months.

BÖLÜM 1. GİRİŞ

Bilgisayar oyun programcılığı yazılım dünyasının doğuşundan beri sürekli olarak yenilenmektedir. Gelişen teknoloji sayesinde artık oyuncular gerçek rakiplerle karşılaşmakta ve oyunlardan çok daha fazla zevk almaktadırlar[1]. Oyun programlamanın temelinde bilgisayar grafikleri yatmaktadır.

Bilgisayar mühendisliğinin önemli alanlarından birisi olan bilgisayar grafikleri, 1945 yılında ilk elektronik bilgisayarlardan birisi olan ENIAC' ın geliştirilmesi ile ortaya çıkmıştır[2]. İlerleyen yıllarda bilgisayar grafikleri hızla ilerleyen ve genişleyen bir alan olmuştur. Grafik dünyasında gerçekleşen gelişmeler bilgisayar oyunlarının doğmasına da yol açmıştır. 1952 yılında Cambridge Üniversitesinde doktorasını yapmakta olan A.S. Douglas' ın yazdığı "tic-tac-toe" yazılımı geliştirilen ilk oyun olmuştur[3].

90'lı yılların başlarında üç boyutlu oyunlar geliştirilmeye başlanmıştır. 1991 yılında ID Software firması "Wolfenstein 3D" adında üç boyutlu bir oyun geliştirmiştir. "Wolfenstein 3D", oldukça basit bir kaplama ve projeksiyon tekniği kullanmasına rağmen oyun piyasasındaki beklentileri oldukça yükseltmiştir[4].

Teknolojinin gelişmesi ile beraber 3-B matematik işlemler yapabilen ekran kartları kişisel bilgisayarlar içinde üretilmeye başlanmıştır. Bu gelişme daha gerçekçi oyunların geliştirilmesinin yolunu da açmıştır[4].

Ekran kartlarının çeşitliliği önceki yılların en büyük problemlerinden birisini oluşturmaktaydı. Oyunların içerikleri genişledikçe bu problem daha da büyümüştür. Tam bu sırada yazılım programlama ara yüzleri (API) devreye girmiştir. Ara yüzler sayesinde programcılar ekran kartlarından soyutlanmıştır. İletişim ara yüzler ile yapılırken, ekran kartı sürücüleri ara yüzlerden gelen komutları kartın anlayabileceği

dile çevirmektedir. Günümüzün en popüler yazılım programlama ara yüzleri DirectX ve OpenGL' dir[5].

OpenGL fonksiyonel bir kütüphanedir. Windows ve Linux dahil olmak üzere her platform üzerinde işlev görebilmektedir. DirectX ise Microsoft tarafından Windows işletim sistemi üzerinde çalışması için tasarlanmıştır[6].

Yıllar ilerledikçe oyuncular daha gerçekçi oyunlar istemektedirler. Donanım firmaları bu isteğe daha güçlü araçlar üreterek cevap vermekteyken, yazılım firmaları da yeni mekanizmalar ve yeni mekanizmalar geliştirmişlerdir. Oyun motoru bu mekanizmalar için bir tutkal görevi görmektedir. 3-B oyun motoru, bilgisayar oyunlarının geliştirilmesinde kullanılan çekirdek bir teknolojidir. Kısaca 3-B oyun motoru, oyunlar içerisindeki 3-B modellerin geometrik verilerini alıp ekran üzeride gösterilmesini sağlamaktadır. Bu işleme genellikle sunum adı verilir[7].

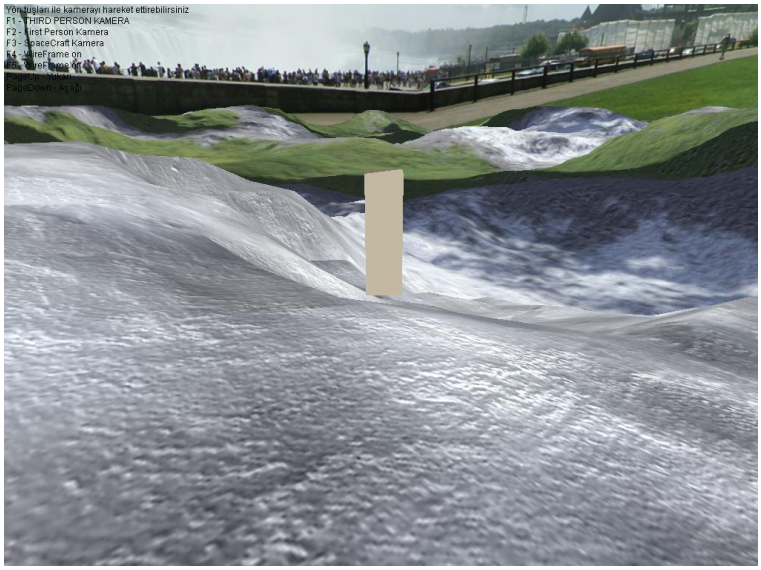
Günümüzde birbirinden farklı özelliklere sahip birçok oyun motoru bulunmaktadır. Unreal, Source ve CryEngine oyun motorlarının en önde gelenlerindendir[8]. Fakat bu oyun motorları açık koda sahip olmadıklarından dolayı üzerlerinde detaylı inceleme yapılamamaktadır. Bu durum göz önünde bulundurularak tezde geliştirilen oyun motoru, açık kaynaklı olan Ogre3D, Doom3 Engine, Panda3D, Wild Magic ve ZFX Engine oyun motorları incelenerek geliştirilmiştir.

Ogre3D oyun motoru daha çok grafiksel özellikleri ile dikkat çekmekte iken gömülü bir fizik motoru bulunmamaktadır[9]. “Doom3 Engine” büyük bir kısmı açık olan bir oyun motorudur. Motor içerisinde yeni bir hafıza yönetim mekanizması kullanılmıştır. Bu sayede en düşük 1.6 kat hız artışı sağlandığı tespit edilmiştir. Panda3D, Disney stüdyoları tarafından sanal gerçeklik uygulamalarında kullanılması için geliştirilmiştir. Ogre3D' de olduğu gibi daha çok grafik motoru üzerine yoğunlaşmıştır [10]. Wild Magic, David H. EBERLY [11] tarafından eğitim amaçlı geliştirilmiştir. ZFX oyun motoru da eğitim amaçlı olarak, Stefan ZERBST ile Oliver DUVEL [12] tarafından geliştirilmiştir. ZFX, Ogre3D ve Wild Magic oyun motorlarında en dikkat çekici özellik platformdan bağımsız olmalarıdır.

Grafik dünyası sadece oyunlar üzerinde kullanılmamaktadır. Örneğin sanal gerçeklik uygulamalarında bilgisayar grafikler çok büyük bir önem taşımaktadır. Oyun motorları 3-B grafik uygulamalarının büyük bir çoğunluğunda kullanılabilmektedir[13]. Her geçen gün araştırmacılar daha güçlü ve daha kolay kullanılabilen oyun motorları geliştirmek üzere çalışmaktadırlar. Tez konusunun seçilmesinde bu faktörlerin büyük etkisi olmuştur.

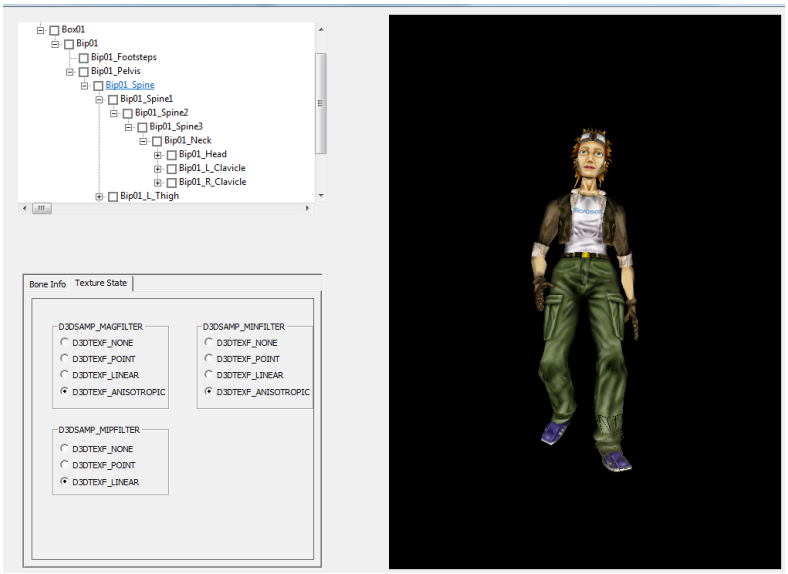
Oyun motorlarının büyük bir çoğunluğu açık kaynak kodlu değildir. Açık kaynak olanlar ise karmaşık yapıları sebebiyle kullanımlarında zorluk yaşanmaktadır[8]. Bu tezde geliştirilecek olan oyun motorunun temel hedefi programcılara daha kolay kullanabilecekleri bir ara yüz sunmaktır. Ayrıca geliştirilen oyun motoru yukarıda anlatılan motorlardan farklı olarak birden fazla model formatı ile çalışabilmektedir. Bu sayede programcılar farklı formattaki modellerini motor içerisinde rahatlıkla kullanabileceklerdir. İçerisinde barındırdığı model editörü sayesinde modeller oyun dünyasına eklenmeden önce incelenebilmektedir.

Geliştirilecek olan motor DirectX kütüphanelerini kullanacaktır. Bunun temel sebebi DirectX' in günümüzde en çok kullanılan grafik kütüphanesi olmasıdır. Ayrıca DirectX, rakibi OpenGL' e göre çok daha geniş dokümantasyona sahiptir.

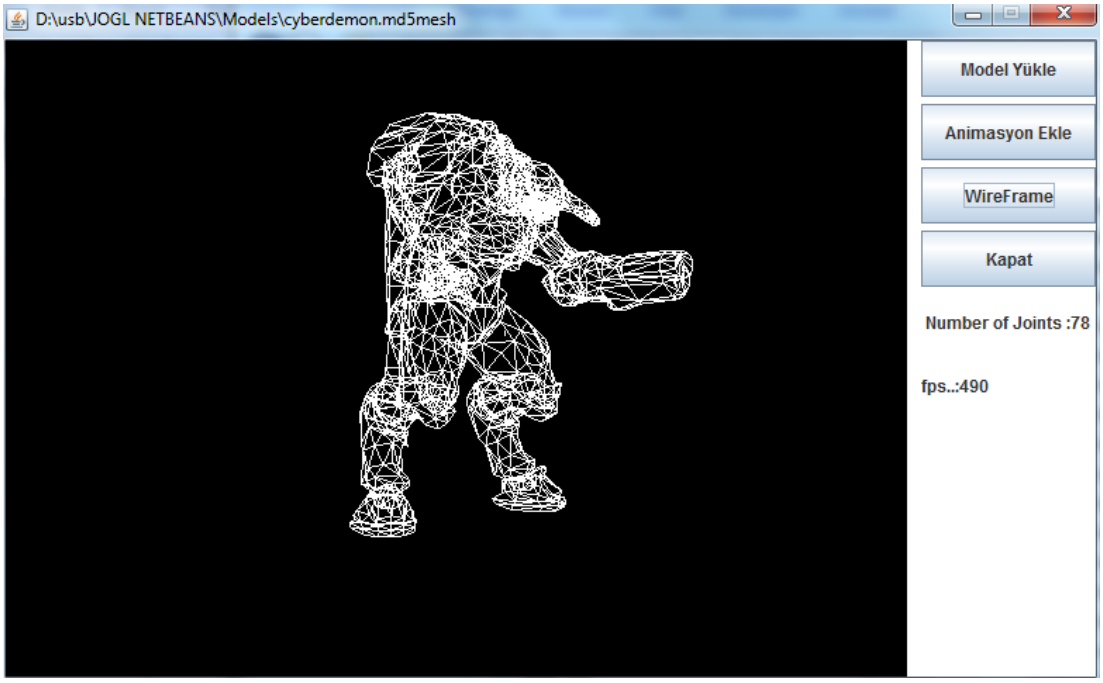


Şekil 1.1. Oyun motorundan bir görüntü

Şekil 1.1' de oyun motoru kullanılarak geliştirilen bir sahne gözükmektedir. Şekil 1.2' de ise modellerin ve sahip oldukları animasyonların incelenmesini sağlayan editör program gösterilmektedir.



Şekil 1.2. Model editöründen bir görüntü



Şekil 1.3. MD5 model formatı için geliştirilen editör program

Şekil 1.3' de MD5 formatındaki modelleri okuyup animasyonlarını uygulayabilen editör program gösterilmektedir.

Tezin ikinci bölümünde öncelikle motorun bileşenleri açıklanmıştır. Bu sayede motor bir bütün olarak incelenebilecektir.

Üçüncü bölümde motor ile beraber kullanılmak üzere geliştirilmiş yardımcı kütüphaneler tanıtılmıştır. Bu kütüphaneler içerisinde 3-B matematik hesaplamalarında kullanılacak olan yapılar ve fonksiyonların yanı sıra temel veri yapıları da bulunmaktadır.

Dördüncü bölümde temel DirectX mekanizmaları ve oyun motoru içerisindeki kullanımlarına değinilmiştir.

Beşinci bölümde ise ışık ve kaplama mekanizması da açıklanmıştır. Ayrıca oyun motorunun kullandığı ışık ve kaplama yapılarından örnekler verilmiştir.

Altıncı bölümde motor için geliştirilen model ve animasyon mekanizması incelenmiştir. Bu mekanizma içerisinde modelleri incelemeye yarayan editör programın geliştirilme aşamaları da açıklanmıştır.

Yedinci bölümde bu tez çalışması ile elde edilen oyun motoru kullanarak geliştirilen uygulamalara yer verilmiştir.

Son bölümde geliştirilen oyun motoru hakkında elde edilen sonuçlara ve eksiklere değinilmiştir

ÖZGEÇMİŞ

Kayhan Ayar, 29.03.1982'de Muş'da doğdu. İlk ve orta öğretimini Sakarya da tamamladı. 2000 yılında İstanbul'un Pendik ilçesinde bulunan Rauf Denktaş Lisesi'nden mezun oldu. 2000 yılında başladığı Sakarya Üniversitesi Bilgisayar Mühendisliği Bölümü'nü 2007 yılında bitirdi. 2007 yılında bir dönem ücretli bilgisayar öğretmenliği yaptı. 2008 yılında Sakarya Üniversitesi Bilgisayar ve Bilişim Mühendisliği Anabilim Dalı'nda yüksek lisansa başladı. 2009 Ocak ayında Sakarya Üniversitesi Bilgisayar Mühendisliği bölümünde araştırma görevlisi olarak göreve başladı. Halen aynı görevi sürdürmektedir.

BÖLÜM 2. OYUN MOTORUNUN BİLEŞENLERİ

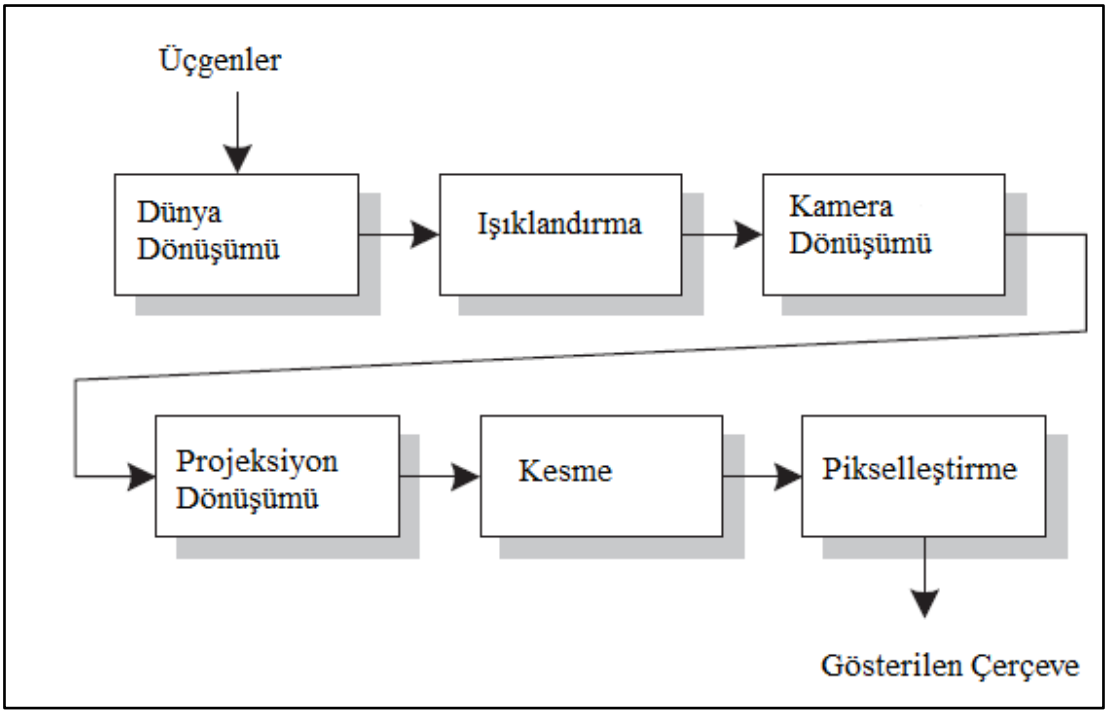
Oyun motoru içerisinde oyunun farklı alanları ile ilgilenen mekanizmalar bulunmaktadır. Bunların en önemlileri aşağıda sıralanmıştır.

- Yardımcı Kütüphaneler
- DirectX
- Işık ve Kaplama Mekanizması
- Model ve Animasyon Mekanizması
- Zemin Oluşturucu
- Kamera Mekanizması

2.1. Üç Boyutlu Boru Hattı

3-B uygulamalarında çerçeve adı verilen resimler oluşturulmaktadır. Bu çerçeveler daha sonra ekranda görüntülenecek olan resimlere dönüşmektedir. Her bir çerçevede üçgenlerden oluşan modeller bulunmaktadır. Oyun motorları için bir çerçeve belirli sayıdaki üçgenlerden ibarettir. Oyun motorlarının temel amacı bu üçgen listesini birleştirerek bir çerçeve oluşturmaktır.

3-B dünyadaki bir üçgen için, üç adet koordinat, bir veya daha fazla normal ve yüzeyinin nasıl doldurulacağı hakkında (kaplama,saydamlık v.b.) bilgi bulundurmaktadır. Üçgenler hakkındaki bu bilgilerin hafızadan alınarak bir çerçeve oluşturana kadar gerçekleştirilen işlemlere 3-B boru hattı adı verilmektedir. Şekil 2.1' de temel bir boru hattı gösterilmektedir[14].



Şekil 2.1. 3-B boru hattı[14]

Boru hattının ilk aşamasında nesne uzayında bulunan üçgenler, bütün oyun sahnesini temsil eden dünya uzayına dönüştürülmektedir. Bu dönüşüm işlemi bir vektör ile dünya matrisinin çarpılmasından ibarettir[15].

Işıklandırma aşamasında üçgenin yüzey normali kullanılarak bir veya daha fazla kaynaktan aldığı ışık miktarı hesaplanıp yüzeyin rengi tespit edilmektedir. Bu işlem sonucunda yüzey bir renge sahip olacaktır.

Kamera dönüşüm aşamasında, dünya uzayında bulunan üçgenler kamera uzayına dönüştürülürler. Dünya dönüşümünde olduğu gibi bu işlem bir vektör ile matrisin çarpılmasından ibarettir.

Projeksiyon dönüşümünde 3-B kamera uzayında bulunan üçgenler 2-B görüntü uzayına haritalanmaktadır. Diğer iki dönüşüm metodunda olduğu gibi bu dönüşümde de bir vektör ile matris çarpılacaktır.

Kesme aşamasında izleyici tarafından görünmeyecek olan üçgenler boru hattından atılmaktadır. Örnek olarak üçgenlerin görünmeyen arka yüzleri ile kameranın görüş alanında olmayan üçgenler boru hattından atılacaktır.

Pikselleştirme evresinde her bir üçgen piksellere dönüştürülüp piksellerin alacağı renkler belirlenir. Bu evre içerisinde kaplama giydirme ve saydamlık karışımı işlemleri de gerçekleştirilmektedir.

2.2. Yardımcı Kütüphaneler

3-B oyunların geliştirilmesinde matematik ve temel veri yapıları büyük önem taşımaktadır. Oyun programlarının kendilerine has beklentileri bulunmaktadır. Örneğin görüntünün insan gözü ile uyum sağlayabilmesi için saniyede en az yirmi dört çerçeve oluşturulması gerekmektedir. Bu sebepten dolayı oyun motorları içerisinde hızlı performans verecek matematik kütüphanesi ve veri yapıları tasarlanması gerekmektedir.

2.2.1. Matematik kütüphanesi

Oyun programları matematiksel işlemlerin bazılarını sistem işlemcisine yaptırırken bazılarını da ekran kartının işlemcisine yaptırır. Ekran kartı işlemcisinin yapacağı matematiksel işlemler sabittir. Bu işlemler için programcı sadece gerekli değerleri verir ve işlemleri DirectX ekran kartına yaptırır. Fakat programcı yazdığı uygulamaya göre kendi matematiksel işlemlerini yapması gerekebilir. Bu tip matematiksel işlemler sistem işlemcisinde yaptırılacaktır.

Ekran kartının yapabileceği işlemler 3-B boru hattında gösterilmektedir. Bunun dışında kalan bütün matematiksel işlemler merkezi işlemci tarafından yapılması gerekmektedir.

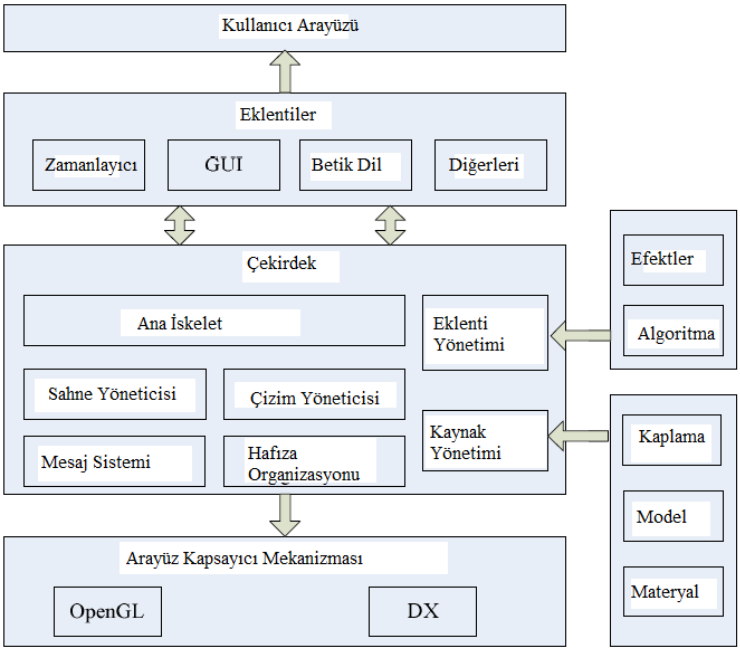
2.2.2. Temel veri yapıları

Günümüz oyunları çok büyük boyutlardaki verileri işlemektedir. Oyunların hızlı çalışması gerektiği düşünüldüğünde verinin saklanma biçimi çok büyük önem kazanmaktadır. Özellikle dinamik dizi , bağlı liste ve ağaç yapıları oyun programlamada çok sık kullanılmaktadır. Bu sebepten dolayı veri yapıları geliştirilirken güvenlikten çok hız ön plana çıkmaktadır[16].

2.3. DirectX' in Motordaki Yeri

DirectX kütüphanelerinin motor tarafından direkt olarak kullanılmasının dezavantajları bulunmaktadır. Bunlardan ilki oyun motorunun DirectX kütüphanesine bağımlı olmasıdır. İkinci olarak, DirectX kütüphanesinin kullanılması zor ve karmaşık yapısı sebebiyle programcıların hata oranı artmaktadır.

Oyun motorlarının ekran kartı ile iletişime geçmesi için bir iletişim katmanı oluşturulmaktadır. Bu katman sayesinde programcı DirectX ve OpenGL gibi kütüphanelerden arındırılmış olacaktır[17].

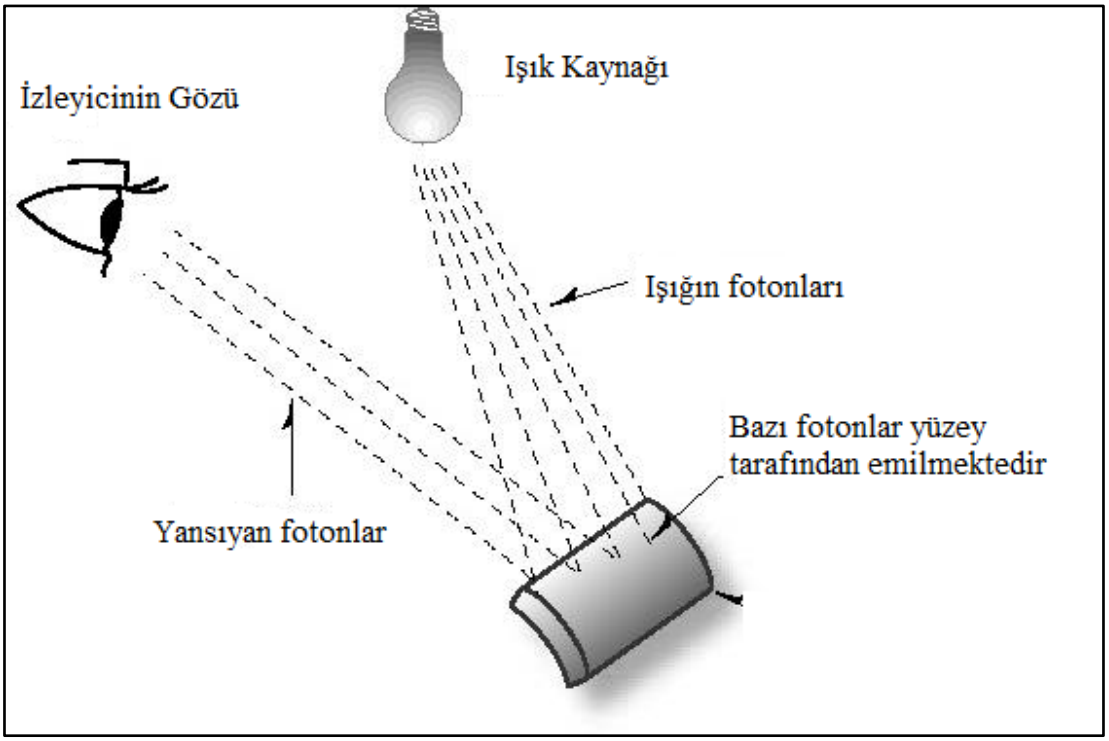


Şekil 2.2. Gingko grafik motorunun mimarisi

Şekil 2.2' de Gingko grafik motorunun mimarisi gösterilmektedir. Ara yüz kapsayıcı mekanizması ile motor, platformdan bağımsız olarak çalışabilmektedir.

2.4. Işık ve Kaplama Mekanizması

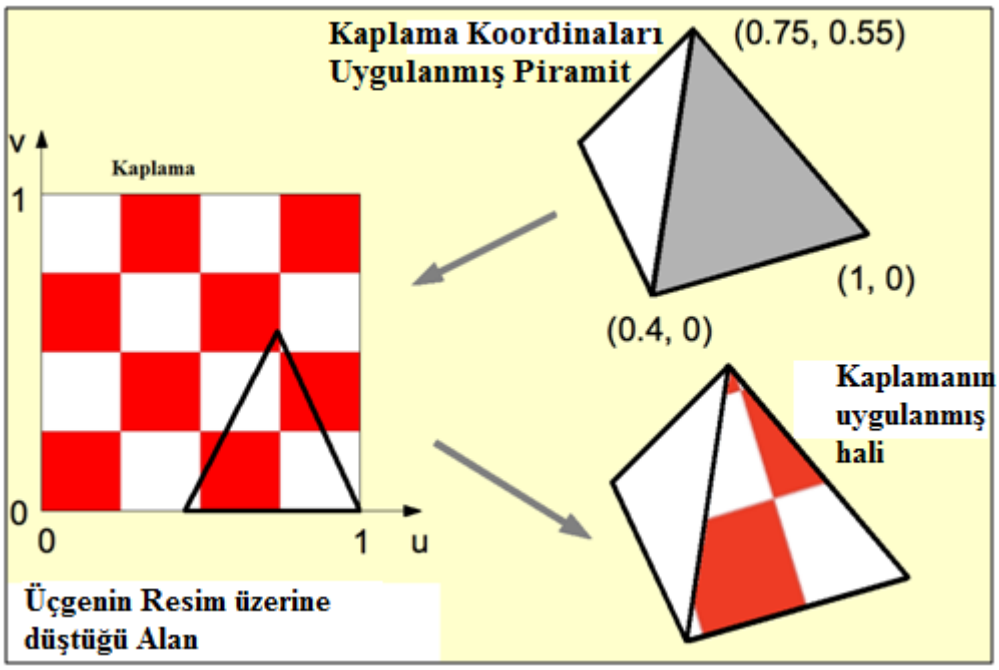
Oyunların gerçekçi görünmesi için gerçek dünyadaki ışık mekanizmasını taklit etmeleri gerekmektedir. Şekil 2.3' de gerçek dünyadaki görme işlemi gösterilmektedir. Şekildeki materyal, üzerine çarpan fotonların bazılarını emerken bazılarını yansıtmaktadır. Yansıyan fotonlar şeklin rengi hakkında bilgi taşımaktadır[18].



Şekil 2.3. Görme olayı

Oyun motorlarının görevlerinden biriside gerçekçi bir ışık mekanizması geliştirmektir. Işık hesaplamaları oldukça çok işlem gerektirdiğinden dolayı DirectX aynı anda sadece sekiz ışığın kullanılmasına izin vermektedir. Oyun motorunun sahne içerisinde sekizden fazla ışığın bulunmamasını sağlayacak şekilde organize olması gerekir.

Kaplama yüzeylerin üzerine gerçek resimlerin giydirilme işlemidir. Şekil 2.4' de buna bir örnek gösterilmektedir. Oyunların hafızada kapladıkları alanın büyük bir kısmını kaplamalar oluşturmaktadır. Bu sebeple geliştirilen oyun motorlarının kaplamaları en uygun şekilde kontrol edecek bir kaplama yönetim mekanizmasına ihtiyacı vardır. Bu mekanizmanın kaplamayı diskten yüklemek ve şekiller üzerine giydirmek gibi yeteneklere sahip olması gerekir. Hepsinden önemlisi bir kaplamanın hafızaya birden fazla yüklenmesinin kesinlikle önüne geçilmesi gerekmektedir.



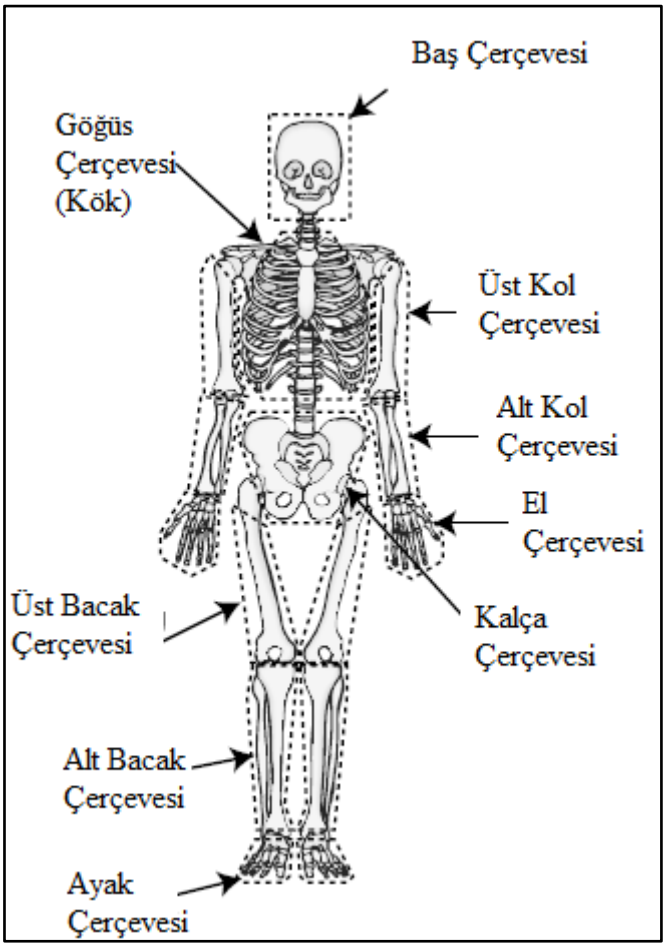
Şekil 2.4. Kaplamanın piramit üzerine uygulanması[19]

2.5. Model ve Animasyon Mekanizması

Oyunlar bir birinden farklı modellerden oluşmaktadır. Bunların bazıları hareket halindeyken bazıları oyun boyunca sabit kalmaktadır. Bütün bu modeller ise üçgenlerden oluşmaktadır.

Modeller çizim programlarında tasarlandıktan sonra belirli formatlar kullanılarak dosyalar içerisinde saklanmaktadır. Oyun motorunun görevi bu dosyalardaki modelleri kayıt formatlarına göre okuyup çizdirebilmek ve eğer hareket edebiliyorsa hareket ettirmektir.

Hareketsiz modeller sadece üçgenlerden oluşmaktayken hareket eden cisimler belirli bir hiyerarşiye sahiptir. Tasarımcı modeli çizdiği gibi hareket etmesini de sağlayabilmektedir. Bu hareketler modelin saklandığı dosya içerisine belirli bir format ile kaydedilmektedir. Örneğin bir insan modelinin yürümesi sağlanabilmektedir. Bunun için model belirli hiyerarşik yapılara bölünmektedir[20]. Şekil 2.5' de bir insan modelinin iskelet hiyerarşisi gösterilmektedir.

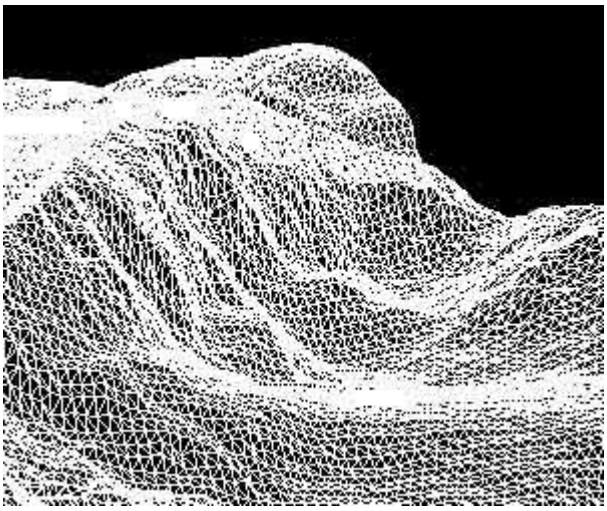


Şekil 2.5. Bir insan modelinin iskelet hiyerarşisi

Bu hiyerarşi sadece çizim için kullanılmamaktadır. Çarpışma testlerinde modelin hangi bölümünün çarpmaya uğradığı tespit edilebilmektedir. Örneğin oyun içerisinde fırlatılan bir taşın modelin hangi uzvuna denk geldiği bu hiyerarşi sayesinde keşfedilebilmektedir.

2.6. Zemin Oluşturucu

Oyunlar tümüyle üçgenlerden oluşan zeminler üzerinde geçmektedir. Geniş görüş alanına sahip oyunlarda zeminin kapladığı alan büyümektedir. Büyüyen alanla beraber çizilmesi ve içi doldurulması gereken üçgen sayısı da artmaktadır. Oyunda yavaşlamaya sebep olmaması için zemin mekanizmasının tasarımına çok dikkat edilmesi gerekmektedir. Oyun motorları içerisinde çeşitli dosya formatları ile saklanan zeminlerin okunup çizilmesini sağlayan bir zemin mekanizması bulunmaktadır.



Şekil 2.6. Zemin oluşturucu[21]

Şekil 2.6' da RAW formatı ile saklanmış bir zemininin okunup çizdirilmesi ile ortaya çıkan sonuç gösterilmektedir.

2.7. Kamera Mekanizması

3-B dünyanın kullanıcı tarafından görünebilmesi için sahne içerisinde en az bir tane sanal kamera yer alması gerekmektedir. Oyunun türüne göre kamera modeli de değişmektedir. Birincil kişi vurucu (FPS) türü oyunlarda kamera insanların görüş yeteneğini taklit etmektedir. Şekil 2.6 'da bu tür bir kameradan elde edilen görüntü gösterilmektedir. Rol yapma oyunu (RPG) türündeki oyunlarda kamera karakterin üzerinde belirli bir açı ile konumlanmaktadır. Bu kamera modeline üçüncü kişi

kamera modeli de denmektedir. Şekil 2.7' de üçüncü kişi kamera modeli gösterilmektedir.

Şekil 2.7. Birincil kişi kamera modeli [22]

Şekil 2.8. Üçüncü kişi kamera modeli [23]

BÖLÜM 3. KULLANILAN YARDIMCI KÜTÜPHANELER

Oyun motorlarının temel görevi dosyalara kaydedilmiş nesneleri okuduktan sonra onları ekrana çizdirmektedir. 3-B sahnelerin oluşturulabilmesi için modellerin birçok matematiksel işlemden geçmesi gerekmektedir. Ayrıca dosyalardan alınan bütün bu verilerin uygun bir şekilde hafızada saklanabilmesi için çeşitli veri yapılarına ihtiyaç duyulmaktadır. Motorun tasarımını ve kullanımını kolaylaştırmak için bütün bu işlemleri gerçekleştirebilecek yardımcı kütüphanelerin tasarlanması şarttır.

Bu bölümde amaç geliştirilen oyun motorundaki yardımcı kütüphaneleri yüzeysel olarak incelemektir. Oyun motorunda temel olarak ikiye ayrılmaktadır. Bunlar;
- Matematik kütüphanesi
- Temel veri yapılarıdır

3.1. Matematik Kütüphanesi

Çoğu oyun motorunun çekirdek yapısını düşük seviyeli matematiksel işlemler oluşturmaktadır. Matematik daha çok ondalıklı sayılar üzerinde gerçekleşmektedir [24]. Oyun oynayan kullanıcılar görüntünün akıcı olmasını beklemektedirler. Bu akıcılığın sağlanabilmesi için de motorun oldukça hızlı işlem yaptırması gerekir. Günümüz oyunları altı ile yedi giga bayt kadar yer kapladıkları düşünüldüğünde verinin nasıl işlendiği de çok büyük önem kazanmaktadır.

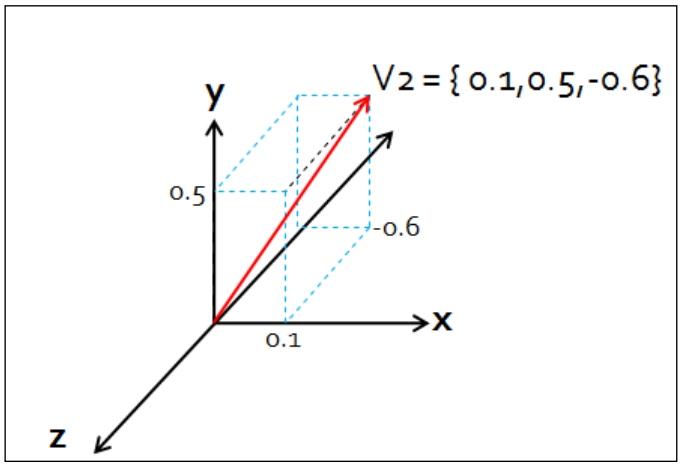
Oyunun üç boyutlu bir analitik dünyada oynanmasından dolayı doğrusal cebir denklemleri de önem kazanmaktadır. Matematik kütüphanesinde bulunan en önemli yapılar aşağıda sıralanmıştır.
- Vektör sınıfı
- Matris Sınıfı
- Dördey Sınıfı

- Yardımcı global fonksiyonlar

3.1.1. Vektörler

Vektör, bir yönü ve boyu dışında bir başlangıç noktası olmayan doğru parçasıdır. Vektörün yönü uzay içerisinde nereyi işaret ettiğini belirtmektedir. Vektörler grafiksel nesnelerin yönlerini belirtmede kullanılmaktadır.

Şekil 3.1' de üç boyutlu bir vektör gösterilmektedir. Vektörün üç bileşeni bulunmaktadır ve bunlar sırası ile x=0.1 ve y = 0.5 ve z= -0.6' dır. Vektör V={0.1,0.5,-0.6} şeklinde gösterilmektedir.



Şekil 3.1. 3-B Koordinat sisteminde bir vektörün gösterimi

Vektörler grafik programcılığında nesnelerin yönlerini belirtmekte kullanıldığı gibi noktaların koordinat bilgileri içinde kullanılmaktadır. Oyun karakterlerinin bakış yönü bir vektör ile temsil edilebilir. Üç boyutlu bir vektörün üç adet elemanı vardır. Bunlar x,y ve z eksenlerini temsil etmektedir. Vektörün V = [4 5 9] olduğu düşünülürse bu vektörün elemanlarının gösterimi $V_x = 4$, $V_y = 5$, $V_z = 9$ şeklinde olacaktır.

3.1.2. Vektör sınıfı

Geliştirilen oyun motorunun en çok kullanılacak olan sınıfı vektör sınıfıdır. Vektör sınıfı tasarlanırken saflığa çok dikkat edilmiştir. Milyonlarca noktayı vektör sınıfı temsil edeceğinden hafızada olabildiğince az yer kaplaması gerekmektedir.

```
class KXVector3
{
public:
        float x,y,z;

        KXVector3(){};
        KXVector3(float _x,float _y,float _z);
        KXVector3(const KXVector3& Vec);
        operator FLOAT* ();
        operator CONST FLOAT* () const;
        KXVector3&  operator     =     (const KXVector3& right);
        KXVector3&  operator     +=    (const KXVector3& right);
        KXVector3&  operator     -=    (const KXVector3& right);
        KXVector3&  operator     *=    (FLOAT Scalar);
        KXVector3&  operator     /=    (FLOAT Scalar);

        bool        operator     ==    (const KXVector3& right);
        bool        operator     !=    (const KXVector3& right);

        KXVector3   operator     +     ()const;
        KXVector3   operator     -     ()const;
        KXVector3   operator     +     (const KXVector3& right)const;
        KXVector3   operator     -     (const KXVector3& right)const;
        KXVector3   operator     *     (FLOAT Scalar) const;
        KXVector3   operator     /     (FLOAT Scalar) const;
};
```

3.1.2.1. Vektör sınıfının yapıcı fonksiyonları

Nesneye yönelik programlamanın önemli parçalarından birisi yapıcı fonksiyonlardır. Programcı yapıcı fonksiyonlar sayesinde, nesneyi oluşturulduğu anda onu kullanıma hazır hale getirebilmektedir. Hazır hale getirmek, çoğu zaman nesneye ait değişkenlerin belirli değerlere atanmasından ibarettir. Vektör sınıfının üç değişkeni bulunmaktadır. Ayrıca üç tane aşırı yüklenmiş yapıcı fonksiyonu bulunmaktadır.

İlk yapıcı fonksiyon parametresizdir ve gövdesi boştur. Tanımlanmasının tek sebebi programcıların derleyici hatası ile karşılaşmasını engellemektir. Sınıfın değişkenlerine bir değer atanması her bir vektör için otomatik olarak aynı işlemin yapılması demektir. Oyunlar içinde milyonlarca vektör bulunacağı düşünülürse bu atama işlemleri de milyonlarca defa yapılması anlamına gelmektedir. Geliştirilen oyun motorunda atama işlemi sadece programcı istediği zaman yapılmaktadır.

Diğer yapıcı fonksiyonlar programcının nesneyi oluştururken değerini de atamasını sağlamaktadır. Bütün fonksiyonların hepsi satır içi fonksiyonlardır. Çok sık kullanılan fonksiyonlar genellikle satır içi yapılmaktadır. Bu fonksiyonların çok daha hızlı çağrılmasına sebep olmaktadır.

3.1.2.2. Vektör sınıfı operatör fonksiyonları

Vektör sınıfı birçok işlemde kullanılacaktır. Programcıların vektör sınıfını matematiksel işlemlerde rahatça kullanabilmesi için bütün işlem operatörleri aşırı yüklenmiştir. Bu sayede programcılar çarpma işlemi yapmak için ayrıca bir fonksiyon çağırmaları gerekmeyecektir. Örneğin vec1 ve vec2 adında iki vektör olduğu düşünülsün. Bu iki vektörün toplamı vec3 vektörüne atanmak istensin. Yazılması gereken kod vec3= vec1+vec2 şeklinde olacaktır. Burada ilk önce vec1 sınıfının + operatör sınıfı devreye girecektir ve parametre olarak vec2 vektörünü alacaktır. Fonksiyonun dönüş değeri vec3 nesnesinin = operatör fonksiyonuna parametre olarak girecektir. Programcı tarafından hiç bir fonksiyon çağrılmamasına rağmen arka planda iki fonksiyon çağrılmaktadır.

3.1.3. Matrisler

Matematiksel olarak matris, sütün ve satırlara bölünmüş dikdörtgen bir alanda sayıların dizilmesi olarak tanımlanabilir. Matrisler oyun dünyası için çok önemli araçlardır. Sahne içerisindeki noktaların dönüşüm geçirmesinde matrisler kullanılmaktadır. Aşağıda bir matris örneği gösterilmektedir. Matris üç sütun ve üç satıra sahiptir. Satır numarası i ve sütun numarası j olan matrisin elemanları m_{ij} şeklinde gösterilir[25].

$$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}$$

3.1.3.1. Birim matris

Birim matris çarpımda etkisiz eleman olarak düşünülebilir. Bir matris birim matrisi ile çarpılırsa hiç bir değişikliğe uğramaz. Oyun programlamada matrisler, üzerinde işlem yapılmadan önce birim matrisine dönüştürülürler. Bu bir değişkene başlangıçta sıfır değeri atanması gibidir. Aşağıda 4x4 boyutlarında bir birim matrisi gösterilmektedir.

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

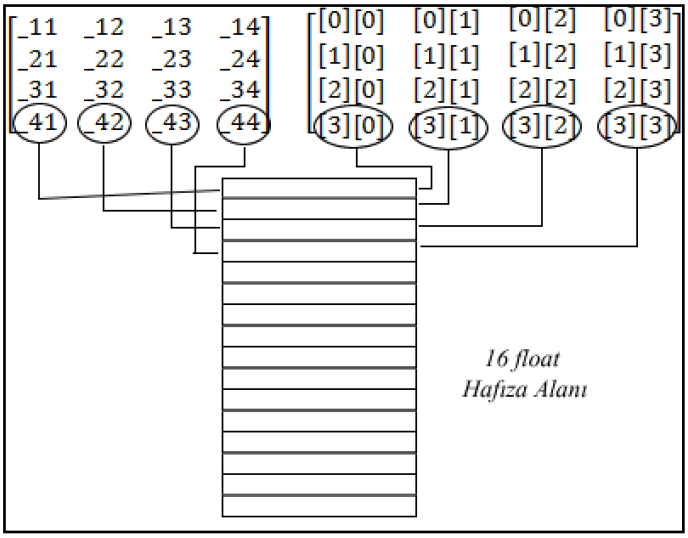
Yukarıdaki bir kare matristir. Kare matrislerin sütun ve satır sayısı birbirine eşittir. Buna göre birim matris, satır ve sütun numaraları bir birine eşit olan elamanların 1, diğer elemanların da 0 olduğu matrislerdir. Oyun motorlarında birim matris genellikle global ve sabit olarak kullanılmaktadır. Bu sayede programcının yeni bir birim matris oluşturmasına gerek olmayacaktır. Ayrıca sabit olduğundan dolayı hiç bir değişikliğe de uğramayacaktır.

3.1.4. Matris sınıfı

```
class KXMatrix : public KXMATRIXELEMS{
public:
        KXMatrix(){};
        KXMatrix(const KXMATRIXELEMS* mtx);
        KXMatrix        operator[]      (int index);
        KXMatrix&       operator=       (const KXMatrix& mtx)       const;
        KXMatrix        operator-       ()                          const;
        KXMatrix        operator*       ( const float a )           const;
        KXVector3       operator*       ( const KXVector3 &vec )    const;
        KXMatrix        operator*       ( const KXMATRIX &a )       const;
        KXMatrix        operator+       ( const KXMATRIX &a )       const;
        KXMatrix        operator-       ( const KXMATRIX &a )       const;
        KXMatrix &      operator*=      ( const float a );
        KXMatrix &      operator*=      ( const KXMATRIX &a );
        KXMatrix &      operator+=      ( const KXMATRIX &a );
        KXMatrix &      operator-=      ( const KXMATRIX &a );
        bool            operator==( const idMat3 &a )               const;
        bool            operator!=( const idMat3 &a )               const;
        void            Identity( void );
        idMat3          Transpose( void ) const;
};
typedef struct _KXMATRIXELEMS{
        union{
                struct{
                        float _11,_12,_13,_14;
                        float _21,_22,_23,_24;
                        float _31,_32,_33,_34;
                        float _41,_42,_43,_44;
                };float m[4][4];
};}KXMATRIXELEMS;
```

Yukarıda matris sınıfı KXMATRIXELEMS yapısından kalıtım almaktadır. Buradaki amaç matrisin değişkenleri ile fonksiyonlarını birbirinden ayırmak ve DirectX fonksiyonlarına uyumlu hale getirmektir. DirectX fonksiyonlarına girilecek matrisin, hafızada birbirini takip eden on altı adet ondalık değişken olması gerekmektedir.

KXMATRIXELEMS yapısında birlik yöntemi kullanılmaktadır. Bu yöntemin amacı iki veya daha fazla değişkenin aynı adres alanını işaret etmesini sağlamaktır. Yukarıda iki değişken iç içe tanımlanmıştır. Bunlar ilki on altı elemana sahip olan bir yapıdır. İkinci olarakda yine on atlı elemana sahip olan m adındaki ondalık dizidir. _11 değişkeni ile m dizisindeki [0][0] elamanı aynı adres alanına karşılık gelmektedir. Bu sayede programcı ister m dizisini kullansın isterse "_" işareti ile başlayan değişkenleri kullansın aynı adres alanlarına erişecektir. Şekil 3.2' de birlik yöntemi ile oluşturulmuş yapının içeriği gösterilmektedir.



Şekil 3.2. Matris sınıfı elemanlarının hafızadaki durumu

3.1.4.1 Matris sınıfı fonksiyonları

Matris sınıfı fonksiyonları vektör sınıfı fonksiyonları ile aynı işlevi görmektedir. Farklı olarak iki "Idendity" ve "Transpose" adında iki fonksiyon bulunmaktadır. Bu fonksiyonlardan ilki matrisi bir birim matrise çevirmektedir. İkincisi ise matrisin

satırları ile sütunlarını yer değiştirmektedir. Matris sınıfı fonksiyonları da vektör sınıfındakiler gibi satır içi fonksiyonlardır.

3.1.5. Dördey

Dördeyler dört boyutlu bir sayı sistemidir. Son yıllarda bilgisayar grafiklerinde sıklıkla kullanılmaktadır. Dördeylerin gösterimi aşağıdaki gibidir.

$$q = w + xi + yj + zk$$

Dördeylerin i,j,k elemanları sanal kısımlarını temsil etmektedir. Kamera hesaplamalarında ve şekillerin animasyon hesaplamalarında döndürme açısı olarak kullanılmaktadırlar[26].

3.1.6. Dördey sınıfı

Aşağıda dördey sınıfı gösterilmektedir. Dördeyler iskelet animasyon modellerinde döndürme matrisi yerine kullanılmaktadır. On altı elemanlı bir matris yerine dört elemanlı bir yapının kullanılması animasyonun daha az hafıza alanı kullanmasına sebep olmaktadır.

```
class KXQuaternion {
public:
        float w, x, y, z;
        void identity() { w = 1.0f; x = y = z = 0.0f; }
        void setToRotateAboutX(float theta);
        void setToRotateAboutY(float theta);
        void setToRotateAboutZ(float theta);
        void setToRotateAboutAxis(const KXVector3 &axis, float theta);
        void normalize();
        float getRotationAngle() const;
        operator FLOAT* ();
        operator CONST FLOAT* () const;
        KXQuaternion&      operator      =      (const KXQuaternion& right);
```

```
        KXQuaternion&     operator     +=     (const KXQuaternion& right);
        KXQuaternion&     operator     -=     (const KXQuaternion& right);
        KXQuaternion&     operator     *=     (FLOAT Scalar);
        KXQuaternion&     operator     /=     (FLOAT Scalar);
        bool          operator     ==     (const KXQuaternion& right);
        bool          operator     !=     (const KXQuaternion& right);
        KXQuaternion      operator     +      ()const;
        KXQuaternion      operator     -      ()const;
        KXQuaternion      operator+    (const KXQuaternion& right)const;
        KXQuaternion      operator-    (const KXQuaternion& right)const;
        KXQuaternion      operator     *      (FLOAT Scalar) const;
        KXQuaternion      operator     /      (FLOAT Scalar) const;
};
```

3.1.6.1. Dördey sınıfı fonksiyonları

Diğer sınıflardan farklı olarak dördeyler içerisinde kendilerini döndürmeye yarayacak fonksiyonlar bulunmaktadır. Dördeyler, vektör veya matris ile direkt iletişime geçemeyeceği için farklı fonksiyonlara ihtiyaç duymaktadırlar. İlk üç döndürme fonksiyonu ("set" ile başlayanlar) dördeyi x,y, ve z ekseninde, parametrede girilen açı kadar döndürmektedir. Bir sonraki fonksiyon ise dördeyin bir vektör etrafında belirli bir açı kadar döndürülmesi sağlamaktadır.

3.1.7. Yardımcı fonksiyonlar

Vektör ve matrisler için kullanılan bazı temel fonksiyonlar bulunmaktadır. Bunların en önde gelenleri aşağıda sıralanmıştır

- Vektör boyu hesaplama
- Vektörler arası çapraz çarpım
- Vektörler arası iç çarpım
- Vektörler arası mesafenin hesaplanması

Fonksiyonların prototipleri aşağıda verilmiştir. İlk fonksiyon vektörlerin boyunu hesaplamak için kullanılmaktadır. İkinci fonksiyon vektörler arasında çapraz çarpım

yapmaktadır. Çapraz çarpım sonucunda, çarpılan iki vektöre dik olan başka bir vektör elde edilmektedir. Üçüncü fonksiyon iki vektör arasındaki mesafeyi bulmaktadır. Son fonksiyon ise iki vektör arasındaki açıyı bulmak için kullanılmaktadır.

```
inline float  KXVec3Lenth(const KXVector3& vec);

inline KXVector3 KXVec3CrossProduct(const KXVector3& Vec1,
                                    const KXVector3& Vec2);

inline float KXVec3Distance(const KXVector3& Vec1,
                            const KXVector3& Vec2);

inline float KXVec3DotProduct(const KXVector3& Vec1,
                              const KXVector3& Vec2);
```

3.2. Temel Veri Yapıları

Günümüz oyunları yüksek grafik kaliteleri, efektleri ve geniş görüş alanları sebebiyle işlemci ile ekran kartı arasında oldukça büyük bir veri akışına yol açmaktadır. Bu sebepten dolayı programcılar oyunun kalitesini azaltmadan veri akışını azaltmanın yollarını aramaktadır.

Profesyonel oyun programcıları, standart kütüphanelerde tanımlanmış veri yapılarını kullanmamaktadır. Bunun temel sebebi standart kütüphanedeki veri yapılarının her türlü uygulamada istenen başarıyı gösterecek şekilde tasarlanmış olmasıdır. Fakat oyunların beklentileri çok farklıdır. Bu yüzden oyun programcıları geliştirecekleri oyun için uygun olan yeni veri yapıları geliştirmektedirler. Hatta bazı durumlarda hafıza yönetim mekanizmasını tümüyle değiştirebilmektedirler. Doom3 oyun motoru üzerinde yapılan incelemelerde derleyicinin hafıza yönetim mekanizması yerine, kendi hafıza yönetim mekanizması kullanıldığı görülmüştür. Firmanın motor üzerinde yaptığı testler de yeni hafıza yönetim mekanizmasının eskine göre daha hızlı olduğu tespit edilmiştir[27].

Bu tez çalışmasında da temel veri yapıları, geliştirilecek olan motora uygun olacak şekilde yeniden tasarlanmıştır. Bu veri yapıları aşağıda sıralanmıştır

- Dinamik Dizi
- Bağlı Liste
- Ağaç

3.2.1. Dinamik dizi tasarımı

Programlama dünyasında iki temel tip dizi bulunmaktadır. Bunlar statik ve dinamik dizilerdir. Dinamik dizinin anlaşılabilmesi için öncelikle statik dizi kavramının detaylı olarak açıklanması gerekmektedir. Sonuçta statik dizilerin dezavantajları dinamik dizinin geliştirilmesine sebep oluştur.

Boyutları değiştirilemeyen dizilere statik dizi adı verilir. Bu diziler en kolay oluşturulup ve kontrol edilen dizilerdir. C++ dilinde statik bir dizi tanımlamak oldukça kolaydır.

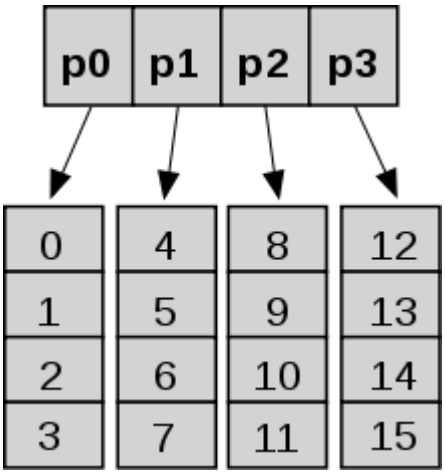
Statik diziler hafızanın yığın alanına yerleştirilir. Bu sebepten dolayı statik dizileri oluşturmak oldukça hızlı gerçekleşen bir işlemdir. Fakat yığın alanına yerleştirilen dizilerin boyutları kesinlikle değiştirilemez. Dizilerin ismi aslında hafızadaki alanlarının başlangıç adresini tutmaktadır.

Dinamik diziler statik dizilerden çok daha karmaşıktır. Dinamik diziler statik diziler kadar oluşturulamazlar. Ayrıca kontrolleri de daha zordur. Fakat statik diziler göre daha avantajlıdırlar. Dinamik diziler istenildiği zaman boyut değiştirebilmektedirler. Örneğin on elemanı olan bir dinamik dizinin eleman sayısı on beşe çıkartılabilir. Veya on olan eleman sayısı beş elemana düşürülebilir. Bütün bu özellikler bir dezavantajı da beraberinde getirmektedir. Dinamik dizi kimi durumlarda statik diziden daha yavaş olmaktadır. Karmaşıklık arttıkça gerekli işlem sayısı da artmaktadır. İşlem sayısı arttıkça da hız azalmaktadır.

Bu tezde geliştirilecek olan dinamik dizinin mümkün olan en hızlı şekilde çalışması gerekmektedir. Oyun programları içerisinde çok büyük boyutlardaki dinamik diziler

sık kullanılacağından hız önemlidir. C++ STL kütüphanesi içerisinde bir dinamik dizi sınıfı hazırda bulunmaktadır. Fakat bu dizi bütün uygulama alanları düşünülerek tasarlanmıştır. Yani bu dizi bütün uygulamalarda optimum hız verecek şekilde tasarlanmıştır. Fakat oyun programlarının kendine has özellikleri ve beklentileri bulunmaktadır. Bu sebepten dolayı dinamik dizi yapısının sil baştan yapılması gerekmektedir.

C++ dilinde oluşturulan programlar yığın ve küme adında iki hafıza alanına sahiptirler. Küme, yığına göre daha dağınık bir yapıdadır. Yığın de yeni değişkenler sıra ile yerleştirilmekteyken kümede neresi boş bulunursa oraya yerleştirilecektir. Boş alanların bulunması için de bir mekanizmaya ihtiyaç duyulacaktır. Bu sebepten dolayı yığın, kümeye göre daha hızlı çalışır[28]. Şekil 3.3' de dinamik bir dizinin hafızaya nasıl yerleştirildiği gösterilmektedir. p0-p3 dizinin satırlarını gösteren işaretçileri temsil etmektedir.



Şekil 3.3. Dinamik Dizilerin Hafızadaki Görünümü[29]

3.2.1.1. C++' da şablon mekanizması

Programlama içerisinde bir fonksiyon birden fazla veri tipi için kullanılabilmektedir. Programcı aynı işlemi farklı veri yapılarında gerçekleştirebilmek için birden fazla fonksiyon tanımlaması gerekecektir. Şablon yapısı bu problem için geliştirilmiştir. Şablonlar daha tanımlanmamış olan veriler için kullanılan fonksiyon veya sınıf

olabilmektedir. Şablonlar kullanılırken veri tipi parametre olarak girilmektedir. Derleme esnasında şablon olarak tanımlanmış olan bu sınıf ve fonksiyonların kullanıldıkları veri yapılarına göre kopyaları oluşturulmaktadır[30].

3.2.1.2. Dinamik dizi sınıfı

Tezde kullanılacak olan dinamik dizi sınıfı yukarıda verilmiştir. Sınıfın ilk elemanı m_pArray dizinin hafızdaki adresini tutacaktır. İkinci değişken m_iAllocSize dizi için açılmış toplam hafıza alanını tutmaktadır. Üçüncü değişken m_iSize, dizi içerisinde bulunan toplam eleman sayısını tutmaktadır. Programcı diziden büyümesini isterse, dizi en az m_iGrowSize kadar büyüyebilecektir.

```
template<typename T>
class KXArray
{
private:
        T               *m_pArray;
        int             m_iAllocSize;
        int             m_iSize;
        int             m_iGrowSize;
public:
        inline                  KXArray(int Size);
        inline                  KXArray();
        inline                  ~KXArray();
        inline                  operator T*()const;
        inline  T&              operator [](int index);
        inline  const T&        operator [](int index)const;
        inline  KXArray<T>&     operator=(const KXArray<T> &RList);
        inline  void            Grow(int Size=0);
        inline  int             GetAllocSize()const;
        inline  int             GetSize()const;
        inline  int             GetGrowSize()const;
        inline  int             Append(const T& Item);
```

```
        inline   int            Append(const KXArray<T> &List);
        inline   int            Append(const T *pItems,int ElemCount);
        inline   int            Insert(const T& Item,int index);
        inline   int            Insert(const T *pItems,int index,int Count);
        inline   int            Insert(const KXArray<T> &List,int index);
        inline   int            Remove(const T& Item,bool bAll=false);
        inline   int            Remove(int index );
        inline   int            Remove();
        inline   int            FindItem(const T& Item)const;
        inline   int            FindItemIndexes(const T&Item,
                                                int *pIndexes,
                                                bool bJustCount=false)const;
        inline   int            IncrementSize(int Size);
        inline   int            Clear();
        inline   int            FreeMemory();
};
```

Sınıfın en önemli fonksiyon diziyi genişletmeye yarayan "Grow" fonksiyonudur. Fonksiyonun temel görevi diziyi genişletmektir. Fonksiyon öncelikle dizide kalan boş alanı kontrol edilmektedir. Eğer yeterince alan yoksa dizi parametre olarak girilen değer kadar büyüyecektir.

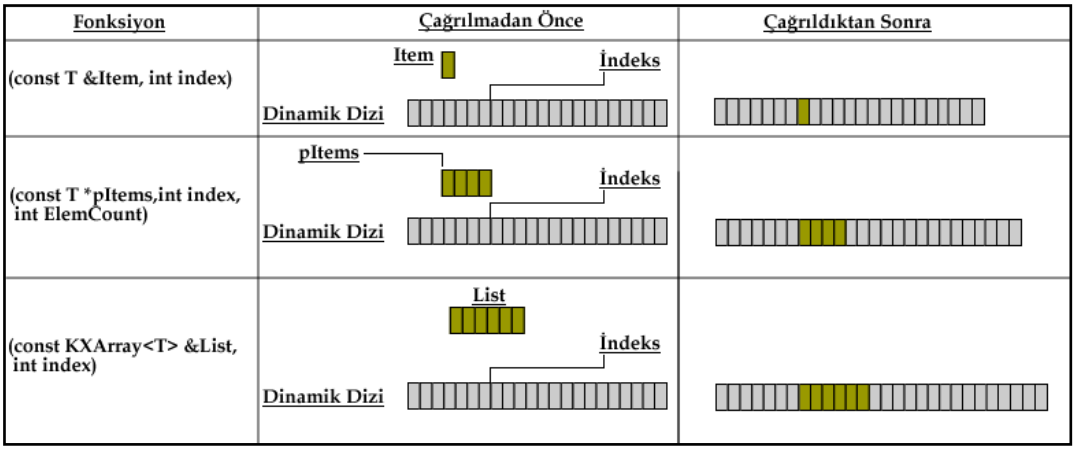
Oyun programcıları çok sayıda nokta bilgisini dinamik diziler içerisine yerleştirmektedirler. Diziye eklenecek nokta sayısının bilinmediği durumlarda programcı optimum bir büyüme oranı seçerek işlem gücünden tasarruf edebilir.

```
template<typename T>
inline void KXArray<T>::Grow(int Size)
{
        if(Size<20)
        {
                Size = GetGrowSize();
        }
```

```
        T *pTemp = new T[Size+GetAllocSize()];
        if(!pTemp)  return;
        memcpy(pTemp,m_pArray,sizeof(T)*GetAllocSize());
        delete [] m_pArray;
        m_pArray        = pTemp;
        m_iAllocSize+= Size;
}

template<typename T>
inline int KXArray<T>::Insert(const T &Item, int index){
        int Gap = GetSize()-index;
        if(GetSize()==GetAllocSize())         Grow();
        if((index>=GetSize())||(index>=GetAllocSize()))
                return -1;
        memcpy(m_pArray+index+1,m_pArray+index,sizeof(T)*Gap);
        m_pArray[4] = Item;
        m_iSize++;
        return GetSize();
}
```

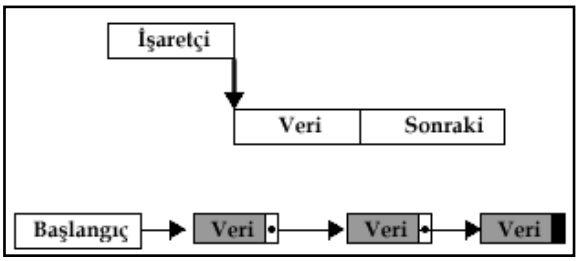
"Insert" fonksiyonu dizi içerisinde herhangi bir noktaya eleman eklemek için kullanılmaktadır. Aynı isim de üç adet fonksiyon bulunmaktadır. İlki parametre olarak bir adet eleman almaktadır. İkincisinde aynı elemanlardan oluşan bir dizi parametre olarak alınmaktadır. Üçüncüsünde ise parametre olarak bir başka dinamik dizi alınmaktadır. Fonksiyonların amacı parametre olarak girilen verileri dinamik dizinin istenilen kısmına yerleştirmektedir. Şekil 3.4' de fonksiyonun hafızadaki gerçekleştirimi gösterilmektedir.



Şekil 3.4. "Insert" fonksiyonlarının hafızada gerçekleştirilmesi

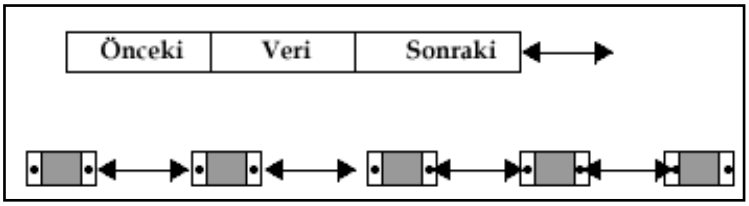
3.2.2. Bağlı liste

Bağlı liste, bir birine işaretçilerle bağlanmış adları düğüm olan veri elemanlarından oluşmaktadır. Her bir düğüm iki temel parçadan oluşmaktadır. Parçalardan birisi verinin ta kendisidir. Diğeri ise bir sonraki düğümü gösteren bir işaretçidir[31]. Şekil 3.5' de bağlı liste grafiksel olarak gösterilmiştir.



Şekil 3.5. Bağlı listenin grafiksel gösterimi

Şekilde gösterilen liste, tek bağlı liste olarak adlandırılmaktadır. Düğümler sadece kendilerinden sonra gelen düğümleri göstermektedir. Bu tezde kullanılacak olan bağlı listede ise düğümler arasında iki adet bağ bulunacaktır. Şekil 3.6' da ikili bağlı listenin düğüm yapısı gösterilmektedir.



Şekil 3.6. İkili bağlı listenin grafiksel gösterimi

3.2.2.1. İkili bağlı liste sınıfı

Bağlı bir liste oluşturmak için düğümler yeterli olabilir. Programcı eğer isterse bir tane ana düğüm yaparak listeyi oluşturabilir. Fakat kontrolün daha iyi sağlanabilmesi için bu tezde bir liste sınıfı geliştirilmiştir. Bu sınıfın amacı ikili liste düğümlerini yönetmektir. Bütün üye fonksiyonları bu amaca uygun olacak şekilde tasarlanmıştır. Üç adet değişkeni bulunmaktadır. Bunlar listenin baş ve kuyruk düğümlerinin adreslerini tutan "m_pHead","m_pTail" ile listedeki düğüm sayısını tutan "m_piCount" dur. Bağlı liste sınıfı aşağıdaki gibidir.

```
template <typename T>
class KXLinkList{
public:
        inline                          KXLinkList();
        inline                          ~KXLinkList();
        inline  int                     Append(const T& Item);
        inline  int                     Prepend(const T& Item);
        inline  void                    RemoveHead();
        inline  void                    RemoveTail();
        inline  KXListIterator<T>       Find(const T&Item);
        inline  KXListNode<T>*          GetTail()               const;
        inline  KXListNode<T>*          GetHead()               const;
        inline  KXListIterator<T>       GetIterator();          const;
        inline  int                     GetCount()              const;
        inline  int             Insert(const KXListIterator<T>& Itr,const T& Item);
        inline  int             Remove(KXListIterator<T>& Itr);
```

```
private:
        inline  KXListNode<T>*          FindFirst(const T& Item);
        int                             m_iCount;
        KXListNode<T>*                  m_pHead;
        KXListNode<T>*                  m_pTail;
};
```

3.2.2.2. "Append" fonksiyonu

Bu fonksiyonun temel görevi yeni bir düğüm oluşturup bu düğümü listenin sonuna eklemektir. Dinamik diziler ile karşılaştırıldığında bağlı listelere eleman eklemek çok daha hızlı gerçekleşmektedir. Dinamik dizide eleman eklemek için bütün elemanları kopyalamak gerekirken. Bağlı listede tek bir eleman oluşturup son düğüme bunu göstermek yeterli olacaktır.

3.2.2.3. "Prepend" fonksiyonu

Bu fonksiyonun görevi "Append" fonksiyonu ile neredeyse aynıdır. Tek fark yeni elemanın listenin sonu yerine başına eklenecek olmasıdır. Fonksiyonun gövdesi aşağıda gösterilmektedir

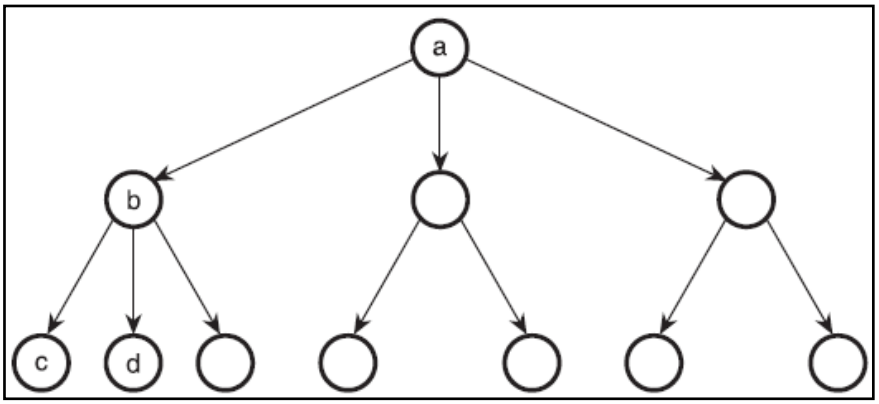
```
template<typename T>
inline int KXLinkList<T>::Prepend(const T &Item){
        if(!m_pHead)  {
                KXListNode<T> *pNode = new KXListNode<T>(Item);
                m_pHead = pNode;
                m_pTail = pNode;
        }else{  m_pHead->InsertBefore(Item);
                m_pHead = m_pHead->GetPrev();
        }m_iCount++;
        return m_iCount;
}
```

3.2.2.4. "RemoveHead" fonksiyonu

Bu fonksiyonun görevi listenin başında bulunan elemanı silmektedir. Silme işlemi başlamadan önce ikinci sırada bulunan eleman işlem bittikten sonra birinci sıraya alınmalıdır.

3.2.3. Ağaç

Ağaç veri yapısının ne olduğunu anlayabilmek için gerçek hayattaki ağaçlara bakmak yeterli olacaktır. Ağaçlar bir köke ve kökünde kendisine bağlı dalları bulunmaktadır. Bu dallar başka dallara ev sahipliği yapmaktadır. Şekil 3.7' de ağaç veri yapısı grafiksel olarak gösterilmektedir.



Şekil 3.7. Ağaç veri yapısının grafiksel olarak gösterimi

Şekildeki ağaç yapısında a kök düğüm olarak adlandırılır. Kök düğümlerinin bir ebeveyni yoktur. Kök düğümünün ilk çocuğu b düğümüdür. Görüldüğü gibi b düğümünün de çocukları bulunmaktadır. Ağaç veri yapısında kök düğüm hariç her düğümün bir ebeveyni bulunmaktadır. Ayrıca her bir düğümün iki komşu düğümü ve belirli sayıda çocuğu bulunabilir.

3.2.3.1. Ağaç sınıfı

```
template<typename T>
class KXTree{
public:
        inline KXTree(const T& Data);
        inline ~KXTree();
        inline  void            Destroy();
        inline  int             AddChild(KXTree<T> *pChild);
        inline  int             AddChild(const KXTree<T>& Child);
        inline  void            SetParent(KXTree<T> *pParent);
        inline  void            SetNextSibling(KXTree<T> *pSibling);
        inline  void            SetPrevSibling(KXTree<T> *pSibling);
        inline  void            SetData(const T& Data);
        inline  T               GetData()       const;
        inline  KXTree<T>*      GetParent() const;
        inline  KXTree<T>*      GetNextSibling() const;
        inline  KXTree<T>*      GetPrevSibling() const;
private:
        T                                       m_Data;
        KXTree<T>*                              m_pParent;
        KXTree<T>*                              m_pNextSibling;
        KXTree<T>*                              m_pPrevSibling;
        KXLinkList<KXTree<T>* >         m_Children;
};
```

BÖLÜM 4. DIRECTX KATMANI

DirectX, Windows platformundaki grafik uygulamaları için geliştirilmiş programlama ara yüzlerinden en popüler olanıdır. DirectX ekran kartı ile programcı arasında bir iletişim mekanizması oluşturmaktadır. DirectX, oyun programcılarına tek bir kütüphane sistemi ile bütün donanımlarda çalışacak oyunlar geliştirebilmelerine imkân sağlamaktadır [32] .

DirectX' in en büyük rakibi OpenGL kütüphaneleridir. OpenGL, DirectX' in aksine her platformda çalışabilmektedir. Zamanla popülerliğini kaybeden OpenGL günümüzde çoğunlukla farklı platformlarda çalışması gereken uygulamalar için kullanılmaktadır.

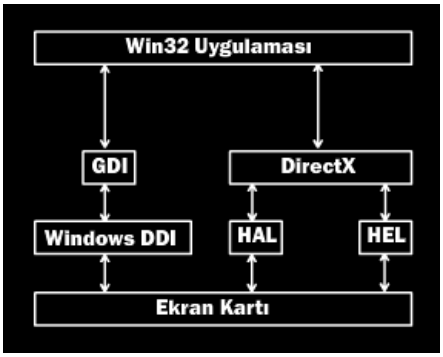
4.1. DirectX' e Neden İhtiyaç Duyuldu

Windows işletim sistemi geliştirilmeden önce programcılar oyunlarını DOS işletim sistemi üzerinde geliştirmekteydiler. Bu işletim sistemi tek iplikli çalışmanın yanı sıra grafiksel kullanıcı ara yüzüne de (GUI) sahip değildi. DOS işletim sistemi programcı ile donanımlar arasında direkt iletişime izin vermekteydi. Bunun avantajları olmasının yanı sıra dezavantajları da bulunmaktadır. Programcı, ekran kartı ile direkt iletişime geçtiği için donanımın bütün imkânlarını kendi isteğine uygun olarak kullanabilmekteydi. Fakat programcının her donanım için kendi sürücüsünü yazması gerekiyordu. Çünkü DOS donanımlar için bir iletişim mekanizmasına sahip değildi[33].

Windows 3.1 işletim sisteminin çıkarılması da sorunu çözemedi. Çünkü bu işletim sistemi de DOS üzerinde çalışmaktaydı. Ayrıca programcıların donanıma direkt erişimine izin vermemekteydi. Windows 3.1 işletim sistemi için ancak çok basit

yapılara sahip platform oyunları geliştirilmekteydi. Hız gerektiren aksiyon oyunları DOS üzerinde yapılmaya devam edildi.

Bu sebeplerden dolayı Microsoft DirectX' i geliştirdi. DirectX programcı ile donanım arasında arabulucu bir katman görevi görmektedir. Programcı çizimlerini DirectX kütüphanelerini kullanarak yaparken donanım sürücüsü kendisine gelen bu DirectX komutlarını kartın anlayabileceği dile çevirmektedir. Yıllar ilerledikçe DirectX' in yeni sürümleri çıkmaya devam etmiştir. Şu an DirectX 11 sürümü kullanılmaktadır[34]. Şekil 4.1' de DirectX 'in Windows hiyerarşisindeki yeri gösterilmektedir.



Şekil 4.1. DirectX' in Windows mimarisindeki yeri

DirectX çıkmadan önce Windows platformunda ekran kartı ile konuşabilmek için grafik aracı ara yüzü (GDI) adı verilen bir mekanizma kullanılmaktaydı. Bu ara yüz beklenilenden çok daha yavaş iş görmekteydi. Bu sebepten dolayı oyun programcılar DOS işletim sisteminde program yazmaya devam ettiler. DirectX bu problemi gidermek için geliştirildi. DirectX' in ekran kartı ile konuşması için izlediği yol GDI' ın izlediği yoldan daha farklıdır.

4.2. DirectX Mimarisi

DirectX "Component Object Model" (COM) adı verilen bir mimari kullanılarak geliştirilmiştir. COM, DirectX' i dilden ve sürümden bağımsızlaştıran bir teknolojidir. COM, Microsoft tarafından geliştirilmiş bir nesne yönelik programlama

standardıdır. COM objeleri bir ara yüz olarak düşünülebilir. Örneğin bir C++ sınıfı bir COM objesi olarak düşünülebilir[35].

COM nesneleri "new" operatörü kullanılarak oluşturulmazlar. Bu nesnelerin oluşturulması için Microsoft özel fonksiyonlar geliştirilmiştir. Programcılar COM nesnelerini oluşturabilmek için bu fonksiyonları kullanmak zorundadır. COM nesnelerine direkt erişim söz konusu değildir. Oluşturucu fonksiyonlar ile COM nesnesi oluşturduktan sonra fonksiyon oluşan nesnenin başlangıç adresini gösteren bir ara yüz işaretçisi döndürür.

Üretilen her kart için aygıt sürücüsü (Device Driver) adı verilen küçük yazılım paketleri geliştirilir. Bu yazılımın görevi aldığı DirectX direktiflerini kartın anlayabileceği kodlara çevirmektir. Bu sayede programcı, üzerindeki büyük bir yükten kurtulmuş olur. Ekran kartları için üretilen sürücüler içerisinde donanım soyutlama katmanı (HAL) adı verilen bir yapı bulunmaktadır. HAL ekran kartı ile direkt iletişim kurabilmektedir. Bu da onu oldukça hızlı yapmaktadır. HAL' ın asıl görevi aldığı DirectX komutlarını kartın anlayabileceği komutlara çevirmektir (Tabi ekran kartının desteklediği komutlarsa). Programcı, ekran kartının desteklemediği bir özelliği kullandığında donanım emülasyon katmanı (HEL) devreye girer. HEL ekran kartının desteklemediği işlemleri merkezi işlemciye yaptırır. HAL' dan daha yavaştır fakat program çalışmaya devam edecektir. DirectX sayesinde kullanıcı ekran kartının faydalarından yararlanırken arka planda gerçekleşen birçok ağır yükten de arındırılmaktadır.

4.3. Windows Programlama

Çizim yapabilmek için bir pencereye ihtiyaç duyulur. Pencere oluşturmak istendiğinde devreye Windows programlama girer. Direkt olarak işletim sistemi ile konuşulup ondan bir pencere oluşturması istenir. Ama önce Windows işletim sisteminde bir program çalıştırıldığında neler olduğunun bilinmesi şarttır.

Kullanıcı bir uygulamaya çift tıkladığında işletim sistemi seçilen dosyanın uzantısını kontrol eder. Windows'un bir uzantı veri tabanı bulunmaktadır. Bu listede hangi

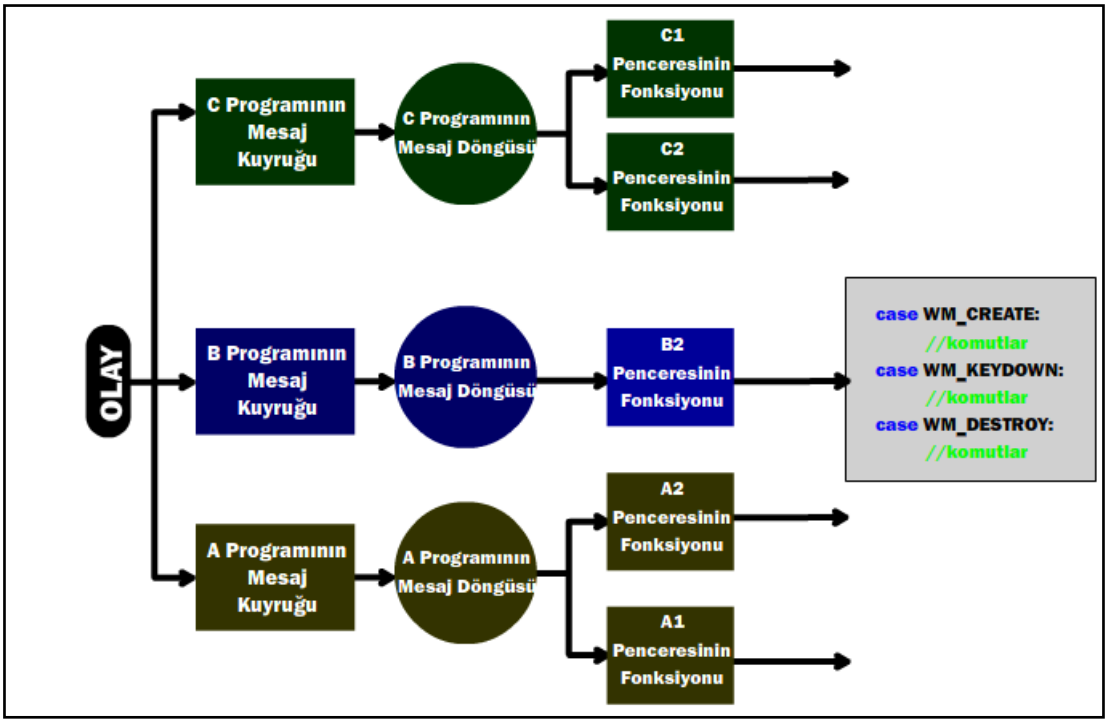
uzantılı dosyaların hangi program tarafından çalıştırılacağı belirtilmiştir. Buna göre uzantısı kontrol edilen dosyanın bu liste ile karşılaştırması yapılır. Örneğin uzantısı ".doc" ise Word uygulaması çalıştırılır ve parametre olarak da seçilen dosya gönderilir[36].

Uzantısı ".exe" olan dosyalar ise Windows işletim sistemi tarafından çalıştırılabilecek uygulamalardır. Windows bu dosyalar çalıştırıldığında ilk olarak dosyanın içeriğini hafızaya yerleştirir ve bu uygulamanın kullanabilmesi için bir proses alanı oluşturur. Dosyanın içeriği hafızaya yüklendikten sonra, Windows programı çalıştıracak fonksiyonu çağırır. Konsol uygulamalarında bu fonksiyon "main" iken Windows uygulamalarında "WinMain" fonksiyonu çağrılmaktadır[37].

4.3.1. Mesaj döngüsü

Windows olay yürütmeli bir programlama modeline sahiptir. Konsol programlamada program "main" fonksiyonu ile başlar ve satır satır sıra ile işlemekte ancak döngü veya fonksiyonlarda atlama yapmaktaydı. Windows' da durum farklıdır. Program durur ve bir olayın meydana gelmesini beklemektedir. Bu olay farenin hareketi veya klavyede bir butonun basılması olabilir. Buna benzer bir olay olduğunda Windows uygun mesajı, olayı tetikleyen uygulamanın mesaj kuyruğuna yerleştirir. Windows da birden fazla programın çalışabileceği göz önünde bulundurulmalıdır. Olayı tetikleyen hangi uygulama ise mesaj onun kuyruğuna yerleştirilecektir.

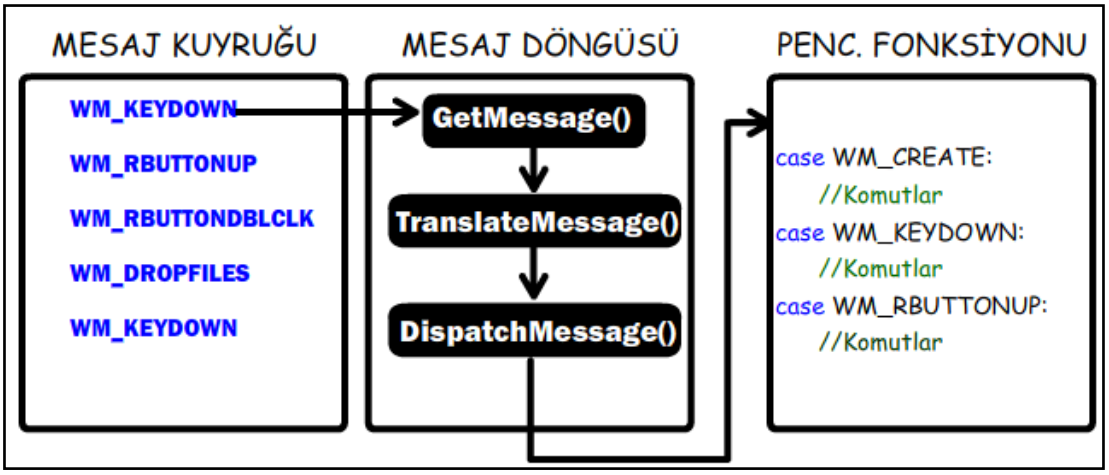
Şekil 4.2' de mesajların izlediği yol gösterilmektedir. Buna göre işletim sistemi bir olay tespit ettiğinde öncelikle olaya uygun bir mesaj üretir. Ardından bu mesajı ait olduğu uygulamanın mesaj kuyruğuna ekler. Programcıda bu mesajları okuyup uygun pencereye yönlendirmekle görevlidir. Programcı sonsuz bir döngü içerisinde mesajları sürekli okuması gerekmektedir.



Şekil 4.2. Windows işletim sisteminde mesajların izlediği yol

Aşağıda basit bir mesaj döngüsü gösterilmektedir. Programcı "GetMessage" fonksiyonu ile sıradaki mesajı "message" değişkenin içine yerleştirir. Ardından okunan mesaj gerekli dönüşüm işlemlerinin yapılması için "TranslateMessage" fonksiyonuna gönderilir. Son olarak mesaj hangi pencereye ait ise o pencerenin mesaj işleme fonksiyonu çağrılır. Bu işlem "DispatchMessage" fonksiyonu aracılığı ile gerçekleştirilir. Şekil 4.3' de mesaj döngüsünün grafiksel gösterimi yer almaktadır.

```
while(GetMessage(&message,NULL,NULL,NULL))
{
        TranslateMessage(&message);
        DispatchMessage(&message);
}
```



Şekil 4.3. Mesaj döngüsünün grafiksel gösterimi

4.3.2. Pencere fonksiyonu

Her bir pencerenin kendisine gelen mesajları işlemesi için bir fonksiyonu bulunmaktadır. Pencere fonksiyonunun prototipi aşağıdaki gibidir.

```
LRESULT CALLBACK WndProc( HWND hWnd,UINT message,
                          WPARAM wParam,
                          LPARAM lParam)
```

Fonksiyonun ismi kullanıcı tarafından seçilebilir. Windows, bu fonksiyonun sadece başlangıç adresini istemektedir. Programcı bu adrese fonksiyonun ismi ile ulaşabilmektedir. Fonksiyonun ilk parametresi fonksiyonun ait olduğu pencerenin kimlik değeridir. İkinci parametre fonksiyonun işleyeceği mesajdır. Bu parametreler "DispatchMessage" fonksiyoundan gelen değerlerdir. Üçüncü ve dördüncü parametredeki değerler ise mesajla birlikte gelen ek bilgilerdir. Örneğin mesaj fare butonun tıklanması iken üçüncü ve dördüncü parametrelerde de hangi koordinatlarda tıklandığını belirtebilir.

Tezde geliştirilen oyun motorunda Windows'un bu karmaşık pencere mekanizması kolay kullanılabilen sınıflar içerisine gömülmüştür. Motoru kullanan programcı için pencere oluşturmak çok kolay bir işlem haline gelmiştir. Aşağıda motor sınıfları kullanılarak basit bir pencere oluşturulmaktadır.

```
int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR
                                                    lpCmdLine,int nShowCmd)
{
        CGameApp g_App;
        g_App.InitInstance(hInstance,lpCmdLine,nShowCmd);
        g_App.BeginGame();
}
```

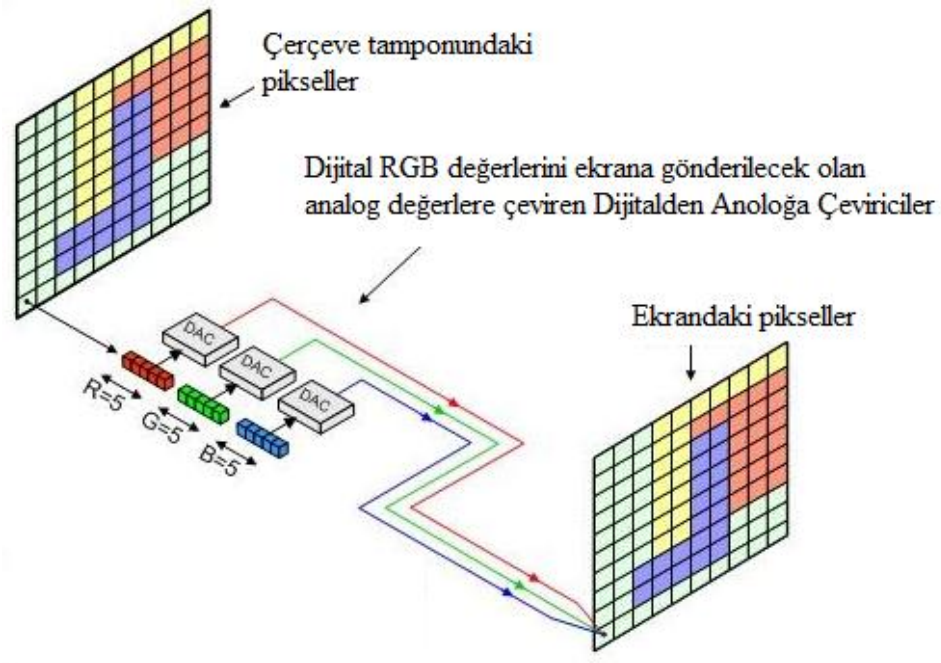
4.4. DirectX ile Programlama

DirectX her programlama dili ile beraber kullanılabilmektedir. Bu tezde C++ dili kullanılacaktır. DirectX, COM nesnelerinden oluşmaktadır. Bu nesneleri kullanabilmek için en uygun dilin C++ olduğunu söylenebilir. C++ düşük seviyeli dillerin avantajlarına sahip olduğu gibi yüksek seviyeli dillerinin de artılarına sahiptir. Bu sayede C++ oyun programcılarının vazgeçilmez tercihi olmuştur.

4.4.1.Çerçeve tamponu

Ekrandaki görüntülenecek resim için hafızada ayrılan alana çerçeve tamponu veya video tamponu denmektedir. Ekrana çıkan bütün görüntü bu hafıza alanına haritalanmıştır[38]. Şekil 4.4' de görüldüğü gibi çerçeve tamponundaki veriler aynen ekrana aktarılmaktadır.

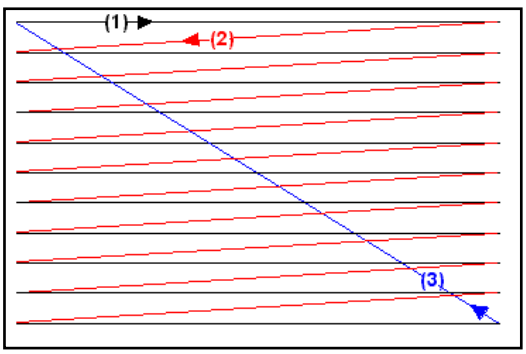
Çerçeve tamponu, ekrandaki görüntünün birebir karşılığı olduğundan çözünürlüğe göre tamponun boyutuda değişecektir. Örneğin 800x600 çözünürlüğe sahip bir monitör için 32 bitlik bir görüntü elde edilsin. Bunun için çerçeve tamponunda 800x600 = 480000 piksel olmalıdır. Her bir piksel için dört byte(32 bit) olduğuna göre toplamda 480000x4 = 1920000 baytlık hafıza alanı kaplamaktadır. Programcı çizimlerini bu tampon üzerinde gerçekleştirir. Daha sonradan bu tampon aynen monitöre iletilir.



Şekil 4.4. Çerçeve tamponu ile ekrandaki görüntü arasındaki ilişki [39]

4.4.2. Tazeleme oranı

Mönitörin bir saniye de çizebileceği görüntü sayısına tazeleme oranı adı verilir. Tazeleme oranı herz (1/saniye) ile temsil edilir. Örneğin tazeleme oranı 60 Hz olan bir monitör saniyede 60 adet resim çizdirebilir. Monitör ekran kartı içersinde çerçeve tamponu adı verilen hafıza alanıdaki verileri kullarak çizim yapmaktadır. 60 herzlik tazeleme oranına sahip bir monitör, bilgisayardan saniyede 60 görüntü alabilmektedir.



Şekil 4.5. CRT Monitörlerde Çizim

Şekil 4.5'de CRT monitörlerde piksellerin doldurulması gösterilmektedir. Pikseller sol üst köşeden doldurulmaya başlanmakta ve sağa doğru devam etmektedir. Bir satır bittiğinde ikinci satırın başından aynı işlem tekrar edilmektedir. Bu işlem bütün satırlar bitene kadar devam eder. Bütün satırlar bittiğinde tekrardan sol üst köşeye geri dönülür

Çerçeve tamponundaki bir görüntü ekrana çizilirken tampona başka bir resim kopyalanabilir. Bu tip durumda erkanın belli bir kısmında eski resim verken diğer kısmında da yeni resim bulunacaktır. Bu duruma yırtılma adı verilir. Şekil 4.6. da yırtılmaya örnek verilmiştir. Şekilde iki tane yırtılma noktası söz konusudur. Monitör birinci resmi çizerken çerçeve tampuna ikinci bir resim yüklenmiştir. Bu sebepten dolayı monitör kaldığı hafıza noktasından çizime devam edecektir. Fakat artık o hafıza noktası ikinci resme ait olduğundan İkinci resim çizilecektir. Bu durum şekilde iki defa gerçekleşmiştir.

Şekil 4.6. Görüntü Yırtılması [40]

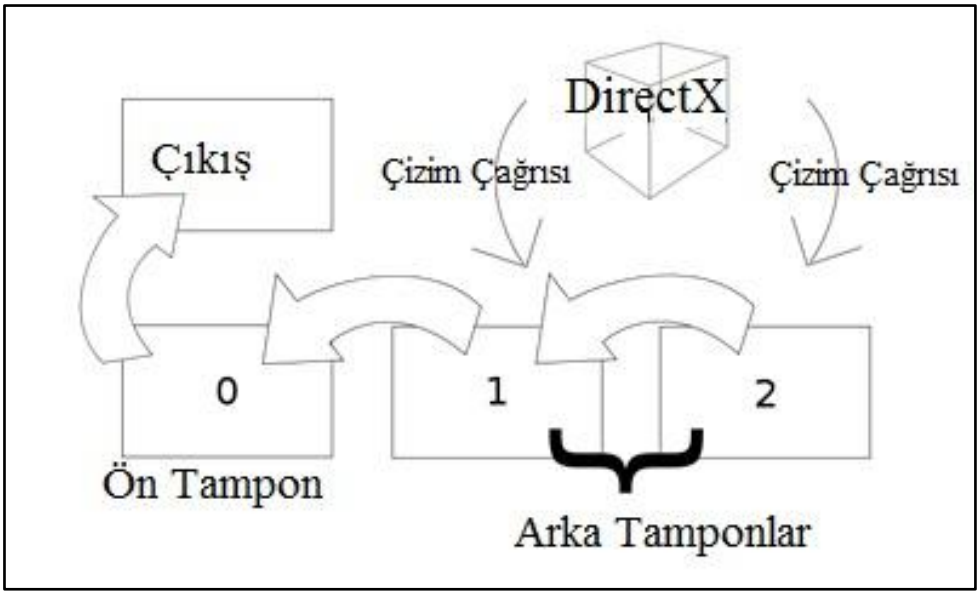
Bu etkiden kurtulmak için çeşitli yöntemler geliştirilmiştir. Monitör soldan sağa çizim işlemini bitirdikten sonra sağ alt köşeye ulaşmış olacaktır. Bir sonraki görüntünün çizilmesi için bu noktadan sol üst köşeye gitmesi gerekmektedir. Bu iki nokta arasındaki geçiş süresine dikey dönüş süresi adı verilmektedir. Programcı çerçeve tamponuna yeni resmi bu zaman aralığında yüklerse yırtılma olmasını engelleyebilmektedir. Diğer bir yöntem ise değişim zinciridir.

4.4.3. Değişim zinciri

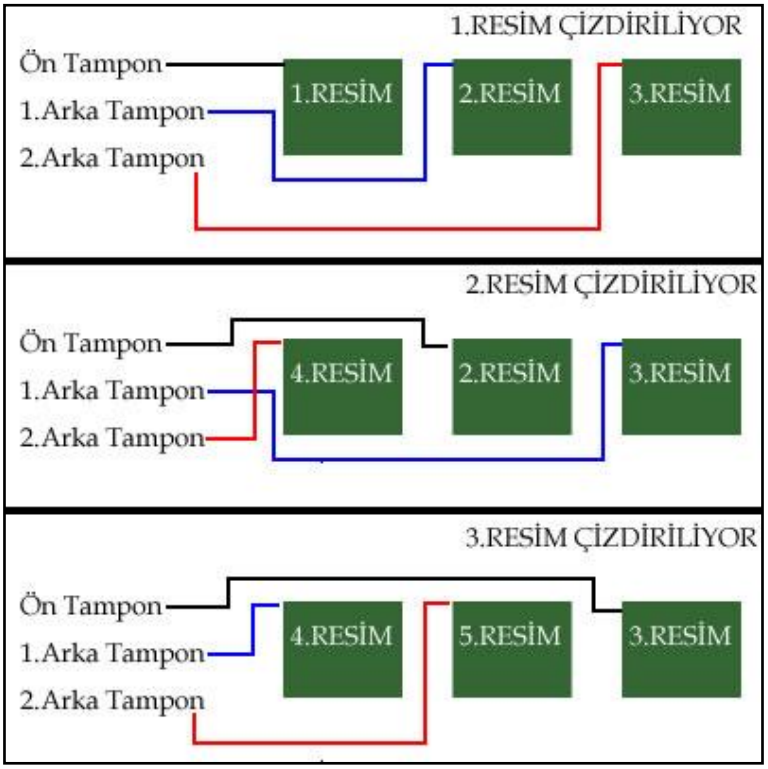
Günümüz oyunları için dikey dönüş süresi çok uzun olabilmektedir. Bu sebepten dolayı birden fazla tampon kullanılmaktadır. Monitör bir tamponu kullanıp çizim yaparken arka planda bir sonraki görüntü de başka bir tamponda oluşturulmaktadır. Zamanı geldiğinde bu tamponları birbirine kopyalamak yeterli olacaktır. Tamponların bu şekilde yer değiştirmesine değişim zinciri adı verilmektedir. Monitöre direkt olarak aktrılan tampona ön tampon denirken çizimin yapıldığı tampona da arka tampon denmektedir. Bütün çizimler arka tampona yapılmaktadır. Çizim işlemi bittikten sonra bir komut ile bu tampondaki bütün veriler ön tampona kopyalanmaktadır. Daha sonra monitör ön tampondaki verileri okuyup çizdirmektedir[41].

Günümüz oyunları üç tamponla dahi çalışabilmektedir. Şekil 4.7’ de üç tamponu bulunan bir değişim zinciri görülmektedir. Çizimler 2 numaralı tampona yapılmaktadır. Çizim tamamlandığında bu tampondakiler 1 numaralı tampona kopyalanmaktadır. Monitör 0 numaralı tampondaki resmi çizdikten sonra da 1 numaralı tampondaki veriler 0 numaralı tampona kopyalanır.

Zamanla tampon kopyalama işlemininde gerektiği kadar hızlı olmadığı ortaya çıkmıştır. Bu sebepten dolayı tampon kopyalamak yerine tamponların başlangıç adreslerini değiştirmek çok daha hızlı sonuç verecektir.



Şekil 4.7. Değişim zinciri [42]



Şekil 4.8. Üçlü tampon

Şekil 4.8' de üç tamponun kullanımı gösterilmektedir. Bu sistemde üç adet işaretçi tutulmaktadır. Bunlar sırası ile “Ön Tampon” ,”1.Arka Tampon” ve “2. Arka Tampon” adlarını almışlardır. İlk resim çizilirken bu işaretçiler şekildeki tamponları göstermektedir. İkinci resim çizdirilirken işaretçiler tampon değiştirmektedirler.

Buna göre “2.Arka Tampon” işaretçisinin gösterdiği alana 4. resim çizdirilir. “Ön Tampon” işaretçisinin gösterdiği alan ekrana çizdirilmektedir. Bu değişim sayesinde ekrana görüntü çıkarken arka planda iki resimde hazır bulunacaktır.

4.4.4. Direct3D nesnesi

Direct3D objesi ekran kartının özelliklerini araştırma ve belirlemede kullanılan fonksiyonları içerisinde barındırır. DirectX uygulamalarının ilk olarak bu objeyi oluşturmaları gerekmektedir. Uygulama bittiğinde de Direct3D objesi serbest bırakılmalıdır. DirectX uygulamarı için bir tane Direct3D objesi yeterli olacaktır. Direct3D objesi IDirect3D9 tipinde bir COM arayüzünden ibarettir ve aşağıdaki fonksiyon ile oluşturulmaktadır[43].

```
IDirect3D9* WINAPI Direct3DCreate9(UINT SDKVersion);
```

Fonksiyon parametre olarak sistemde kurulu olan DirectX'in yazılım geliştirme aracının(SDK) sürümü girilmelidir. Bu parametreye D3D_SDK_VERSION değeri girilmesi yeterli olacaktır. Bir hata oluşmadığı sürece dönüş değeri IDirect3D9 türünde bir işaretçi olacaktır. Aşağıdaki kod bloğunda bir Direct3D objesi oluşturulmaktadır.

```
IDirect3D9* pD3D = NULL;
if((pD3D=Direct3DCreate9(D3D_SDK_VERSION))==NULL)    return E_FAIL;
```

DirectX objesi kullanabilmek ve oluşturabilmek için öncelikle gerekli kütüphanenin eklenmesi gerekmektedir. Kullanılacak olan kütüphane “d3d9.h” dir. Ayrıca statik kütüphanede eklenmelidir. Kullanılacak olan statik kütüphane “d3d9.lib” dir. Aşağıdaki kod bloğunda sırası ile bu işlemler yapılmaktadır.

```
#include<d3d9.h>
#pragma comment(lib,"d3d9.lib")
```

Yukarıdaki kodun ilk satırında d3d9.h kütüphanesi eklenirken diğerinde de statik kütüphane eklenmektedir.

4.4.5. Direct3D aracı

Direct3D objesinin temel amacı ekran kartına bütün çizimleri yaptıracak olan Direct3D aracını oluşturmaktır. Direct3D aracı IDirect3DDevice9 tipinde bir COM arayüzüdür. Bu araç aslında grafik kartının ta kendisidir. Bütün çizim işlemleri bu araç aracılığı ile ekran kartına yaptırılır[44].

İki tip Direct3D aracı mevcuttur. Bunlardan ilki HAL aracıdır. Bu araçlar bütün işlemleri için ekran kartını kullanırlar. Eğer komutları ekran kartı desteklemiyorsa program devam etmeyecektir. Diğer Direct3D aracı ise referans aracıdır. Ekran kartının desteklemediği komutlar işlemciye yaptırmaktadır. Programcı ekran kartının komutu destekleyip desteklemediğini kontrol edip sonuca göre HAL veya Referans aracı oluşturabilir[45]. Direct3D aracı IDirect3D COM arayüzünün "CreateDevice" motodu ile oluşturulur. Metodun protipi aşağıdaki gibidir.

```
HRESULT IDirect3D9::CreateDevice(
        UINT Adapter, D3DDEVTYPE DeviceType,
        HWND hFocusWindow, DWORD BehaviorFlags,
        D3DPRESENT_PARAMETERS *pPresentationParameters,
        IDirect3DDevice9** ppReturnedDeviceInterface);
```

Fonksiyonun ilk parametresine, aracın oluşturulacağı ekran kartının kimlik numarasını girilmelidir. Çoğu sistemde tek bir ekran kartı oluduğundan bu değer sıfır olarak kullanılabilir. Bu tezde Direct3D aracı oluşturulurken ilk parametre olarak D3DADAPTER_DEFAULT değeri girilecektir. Bu değer sistemdeki varsayılan ekran kartı için bir araç oluşturulmasını sağlayacaktır.

İkinci parametre oluşturulacak olan aracın donanım desteklimi yoksa emülasyon yazılımının mı kullanılacağına karar vermektedir. Bu parametreye D3DDEVICETYPE türünde bir değer beklenmektedir. Bu sıralı yapı içerisinde üç

adet değer bulunmaktadır. Bunlardan ilki (D3DDEVTYPE_HAL), aracın donanım destekli oluşturulacağını belirtmede kullanılır. Bu durumda Direct3D aracı bütün işlemlerini ekran kartına yaptıracaktır. İşlemler çok hızlı gerçekleşecektir. Fakat ekran kartının desteklemediği komutlar programın çökmesine sebep olabilir. Programcı ekran kartına gönderdiği komutların desteklenip desteklenmediğini kontrol etmelidir. İkinci değer ise (D3DDEVTYPE_REF), aracın yazılımsal bir emülatör kullanacağını belirtir. Yani araç bütün hesaplamaları merkezi işlemciye yaptıracaktır. Bu yöntem diğerine göre daha yavaştır. Fakat programın devamı her halükarda sağlanmaktadır. Son değer ise (D3DDEVTYPE_SW), üçüncü kişiler yazılım emületörlerinin kullanılmasını sağlamaktadır. Bu parametre günümüzde pek kullanılmamaktadır. Bu tezde çoğunlukla ilk değer yani D3DDEVTYPE_HAL kullanılacaktır.

```
typedef enum _D3DDEVTYPE
{
        D3DDEVTYPE_HAL = 1,
        D3DDEVTYPE_REF = 2,
        D3DDEVTYPE_SW = 3,
        D3DDEVTYPE_FORCE_DWORD = 0xffffffff
} D3DDEVTYPE;
```

Fonksiyonun üçüncü parametresi aracın hangi pencereye çizim yapacağını belirlemede kullanılır. Bu parametreye çoğunlukla uygulama ana penceresinin kimlik numarası girilecektir.

4.4.6. Derinlik tamponu

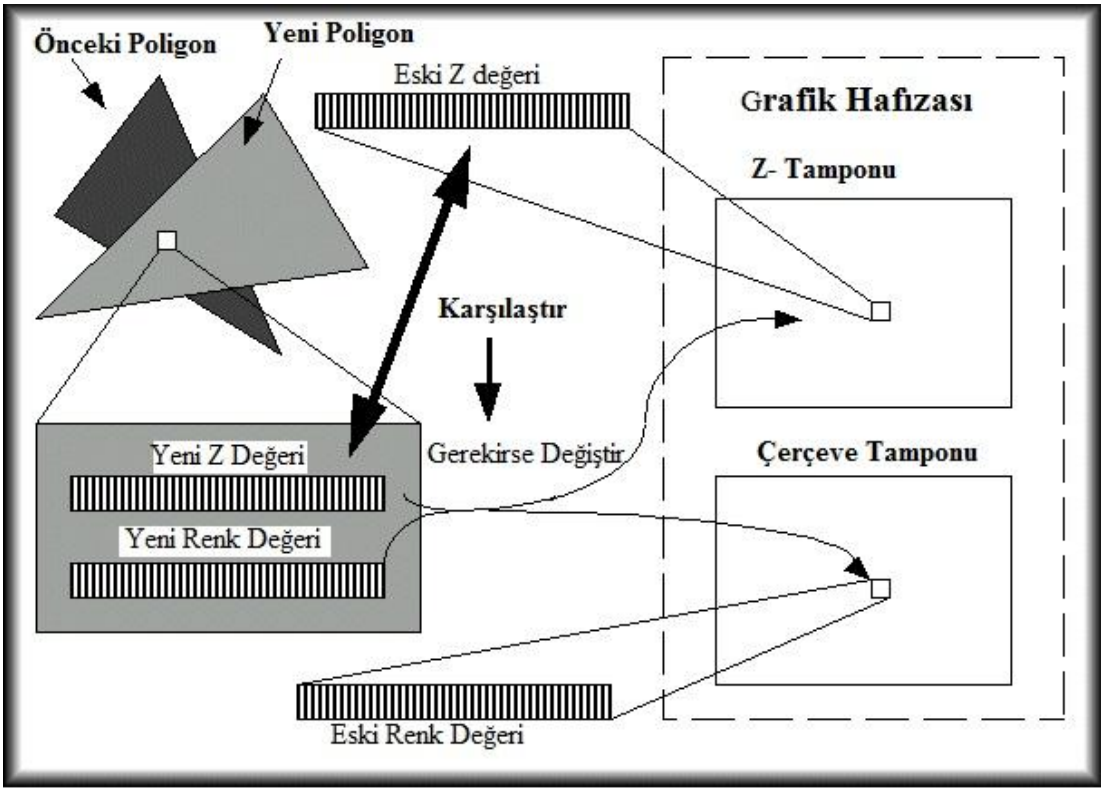
3-B oyun programlamanın en zor bölümlerinden birisi, kameraya daha yakın olan poligonların daha uzak olanların üzerine çizilmesini sağlamaktır. Örneğin kameraya bir birim uzaklıkta olan model, kameraya iki birim uzaklıkta olan modelin üzerine çizilmelidir. Ressam algoritmasına göre çizilecek olan poligonlar kameraya olan uzaklıklarına göre önden arkaya sıralanmaktadırlar[46]. Buna göre kameraya en uzak olan poligonlar ilk önce çizilirken kameraya en yakın olan poligonlar son olarak

çizilmektedir. Bu yöntem ressamların çizim yaparkenki tekniklerine benzemektedir. Ressamlar öncelikle arka tarafta kalan objeleri çizerler ardından önde görünecek olan objeleri çizerler.

Ressam algoritması küçük sahneler için iyi sonuç vermekteyse de günümüzün büyük sahneleri için uygun değildir. Sahnenin görünür olan kısmında on binlerce poligon olduğu düşünülürse hepsinin sıralanması oldukça zaman isteyen bir işlem olacaktır. Bazı durumlar poligonların çoğunun üzerine yakında bulunan poligonlar çizilebilir. Bu yüzden hiç görünmeyecek olan poligonlar gereksiz yere sıralama işlemine dahil edilmiş olacaktır. Bu problemin çözümü için daha küçük alanlarla uğraşmak gerekir. Sonunda bütün bu poligonlar piksel piksel çizilecektir. Bu durumdaki piksellerin hangisi daha yakın hangisi daha uzak kontrol edilip yakın olan kalırken uzak olan dışlanacaktır. Oyun dünyasında bu işlemi kolaylaştırmak için Z-tamponu kullanılmaktadır.

Z-tamponu için hafızada çerçeve tamponu kadar yer ayrılmaktadır. Çerçeve tamponunda bulunan her bir pikselin kameraya olan uzaklı Z-tamponunda tutulmaktadır. Her bir poligon piksel piksel çizildiğinde Z-tamponunda piksellere karşılık olarak kameraya uzaklıkları tutulmaktadır. Bu pikseller üzerine başka poligonların pikselleri yerleştirilirken öncelikle Z-tamponundaki uzaklık değeri kontrol edilecektir. Eğer yeni çizilen poligonunun piksellerinin uzaklık değerleri, Z-tamponunda bulunan önceki uzaklık değerinden küçük ise yeni pikseller yerlerini alırken Z-Tamponu da buna göre yenilenir.

Z-tamponunda tutulan değer 0 ile 1 arasındadır. Sıfır değeri kameraya en yakın noktayı gösterirken, bir değeri kameraya en uzak noktayı göstermektedir. Z-tamponu kimi sistemlerde 16-bitlik ondalık sayı sistemi ile oluşturulurken kimilerinde de 32-bitlik ondalık sayı sistemi kullanılmaktadır. Direct3D aracı oluşturulurken beraberinde Z-tamponu da oluşturulabilmektedir.



Şekil 4.9. Z-tamponunda değer değişimi[47]

Şekil 4.9' da Z-tamponunun çerçeve tamponu ile beraber nasıl çalıştığı gösterilmiştir. Şekilde çerçeve tamponu içerisindeki eski bir poligon üzerine yeni bir poligon çizilmiştir. Z-tamponunda eski poligonun piksellerine ait uzaklık değerleri bulunmaktadır. İlk olarak yeni poligonun piksellerinin kameraya olan uzaklığı ile daha önceden çizilmiş olan piksellerin uzaklıkları karşılaştırılmaktadır. Eğer yeni z değeri öncekine göre daha küçükse yeni piksel daha önde olacak demektir. Buna göre çerçeve tamponuna yeni poligondan alınan renk değeri girilirken, Z-tamponundaki yerine de bu pikselin kameraya olan uzaklığı yazılacaktır. Z-tamponu ile çerçeve tamponu dikey ve yatay olarak aynı sayıda hücreye sahiptir.

4.4.7. Çizim fonksiyonu

Bilgisayar oyunlarında çizim işlemi sürekli gerçekleştirilmektedir. Kullanıcı klavye veya fareyi bıraktığında dahi çizim devam etmektedir. Çizim sahne içerisinde bir kameradan çekilen fotoğraflar gibi düşünülebilir. Kameranın sürekli olarak sahneden fotoğraflar çekmesi gerekir. İnsan gözüde aynı şekilde davranmaktadır. Sürekli

olarak fotoğraf çekip işlemesi için beyne yollamaktadır. İnsan gözü saniyede atmış fotoğraf çekme yeteneğine sahiptir. Çizim daha öncede bahsedildiği gibi arka tampon üzerine yapılır. Resimin ekran da görünebilmesi için arka tampon ile ön tampon arasında bir değişim gerçekleştirilmesi gerekir. Bunun için "Present" fonksiyonu kullanılır. Fakat öncelikle ön tampon temizlenmelidir. "Clear" fonksiyonu bu işlem için kullanılmaktadır[48]. Fonksiyonun prototipi aşağıdaki gibidir:

```
HRESULT     IDirect3DDevice9 ::Clear(
        DWORD Count,
        const D3DRECT *pRects,
        DWORD Flags,
         D3DCOLOR Color,
        float Z,
        DWORD Stencil
);
```

Temizleme işlemi yapıldıktan sonra tamponlar arası değişim gerçekleştirilebilir. Değişimi gerçekleştiren "Present" fonksiyonudur. Bu fonksiyon "Clear" fonksiyonu gibi "IDirect3DDevice9" nesnesine ait bir metotdur. Metodun prototipi aşağıdaki gibidir[49]:

```
HRESULT IDirect3DDevice9 ::Present(
        CONST RECT *pSourceRect,
        CONST RECT *pDestRect,
        HWND hDestWindowOverride,
        CONST RGNDATA *pDirtyRegion
);
```

Aşağıda temel bir çizim fonksiyonu gösterilmektedir.

```
void dx_ciz()
{
        if(pDevice!=NULL)
        {
```

```
                pDevice->Clear(0, NULL, D3DCLEAR_TARGET,
                                        D3DCOLOR_XRGB(0, 0, 0), 1.0f, 0);
                pDevice->Present(NULL,NULL,NULL,NULL);
        }
}
```

Fonksiyonda öncelikle Direct3D aracı kontrol edilmektedir. Eğer araç oluşturulmuş ise çizim fonksiyonları çağrılmaktadır. Program sonlanacağı zaman DirectX nesnelerinin de serbest bırakılması gerekir. DirectX, COM nesnelerini kullandığı için delete parametresi ile silme işlemi yapılmaz. Bunun yerine her bir COM nesnesinin sahip olduğu "Release" fonksiyonu kullanılır. Aşağıda program sonlandığında devreye girecek olan temizleme fonksiyonu verilmiştir.

```
void dx_temizle()
{
        if(pDevice!=NULL)
                pDevice->Release();
        if(pD3D!=NULL)
                pD3D->Release();
}
```

4.4.8. Mesaj yakalama fonksiyonundaki değişim

GetMessage fonksiyonu öncelikle kuyruktaki mesajları okumaktadır. Eğer mesaj bulamazsa yeni bir mesaj gelene kadar programı beklemeye almaktadır. Oyun uygulamalarının her koşulda çalışmaya devam etmesi gerekir. Bunun için "PeekMessage" fonksiyonu kullanılmaktadır[50]. Bu fonksiyon öncelikle kuyrukta mesaj olup olmadığını kontrol eder. Eğer mesaj yoksa uygulamanın devam etmesine izin verilir. Oyun motorunda kullanılan mesaj döngüsü aşağıda gösterilmektedir.

```
int CGameApp::BeginGame()
{
        MSG msg;
```

```
        m_Timer.Start();
        while(1)
        {
                if(PeekMessage(&msg,NULL,0,0,PM_REMOVE))
                {
                        if(msg.message==WM_QUIT)
                                break;
                        TranslateMessage(&msg);
                        DispatchMessage(&msg);
                }
                else
                {
                        FrameAdvanced();
                }
        }
        return 0;
}
```

BÖLÜM 5. IŞIK VE KAPLAMA KATMANI

Işık ve kaplama günümüz oyunları için vazgeçilmez yapılardır. Gerçekçi sahnelerin elde edilebilmesi için gerçekçi ışık ve kaplama mekanizması gerekmektedir. Ayrıca bu iki mekanizma oyunların en çok işlem gücü gerektiren parçasını oluşturmaktadır. Bu sebepten dolayı oyun motorları ışık ve kaplama mekanizmalarının tasarımına ayrı bir önem vermektedirler.

Bu bölümde amaç ışık ve kaplama yapıları hakkında temel bilgileri vermek ve geliştirilen oyun motorlarında da bu iki yapının nasıl gerçekleştiğini göstermektir.

5.1. Işık

Günümüz oyunlarının geliştirilmesinde ışık çok önemli bir yer almaktadır. Oyunlardaki nesne ve karakterlerin gerçekçi görünebilmesi için üzerlerine düşen ışığın gerçeğe yakın olacak şekilde tasarlanması gerekir. Oyun dünyasında dört çeşit ışık bulunmaktadır. Bunlar aşağıda sıralanmıştır.

-Ambiyans Işık
-Dağınık Işık
-Yansıma Işık
-Salıcı Işık

5.1.1. Ambiyans ışık

Gerçek dünyada ışık yoluna çıkan objelere çarpıp yön değiştirmektedir. Buna göre ışık kaynağından direkt olarak yararlanamayacak olan yüzeylerde ışık alabilmektedir. Örneğin ışıkların tavanda olduğu bir oda düşünülsün. Ahşap bir masanın altı, ışık kaynağından direkt olarak yararlanamayacağından karanlık olması gerekir. Fakat gerçekte durum böyle değildir. Işık kaynağından çıkan ışınlar duvarlara çarptıktan

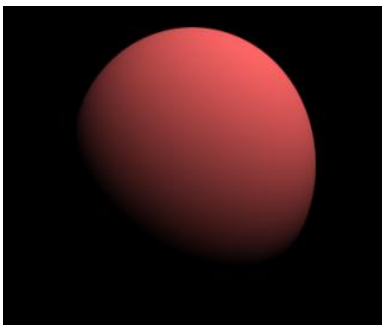
sonra masanın altına ulaşabilmektedir. Bundan dolayı masanın altı bir miktar da olsa aydınlanacaktır. Bilgisayar grafiklerinde ışığın izlediği bu yolun gerçek zamanlı olarak takip edilmesi çok zordur. Bu sebepten dolayı ambiyans ışık modeli kullanılmaktadır.

Ambiyans ışık üç boyutlu sahne içerisindeki bütün yüzeylere aynı miktarda uygulanmaktadır. Ambiyans ışığın bir yönü ve kaynağı bulunmaz. Şekil 5.5' de üç boyutlu bir sahne üzerinde sadece ambiyans ışığı kullanılmaktadır.

Şekil 5.1.Ambiyans ışık. [52]

5.1.2. Dağınık ışık

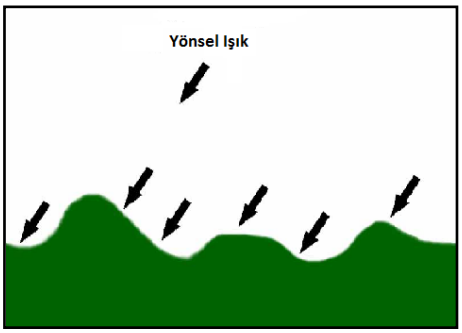
Belirli bir yönü ve kaynağı olan ışıklardır. Gerçek ışık kaynaklarının bir taklidi olarak düşünülebilir. Şekli 5.6' de bu ışık kaynağının şekil üzerindeki etkisi gösterilmektedir. Üç farklı dağınık ışık bulunmaktadır



Şekil 5.2.Dağınık ışığın şekil üzerindeki etkisi

5.1.2.1. Yönsel ışık

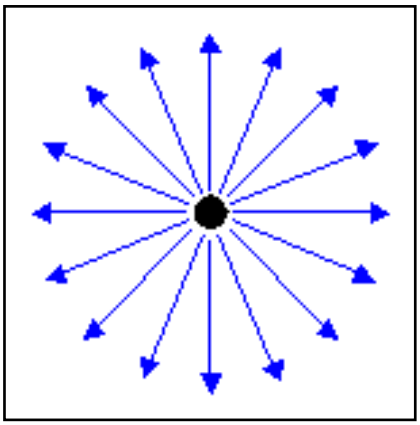
Bu ışık kaynağının belirli bir çıkış noktası bulunmamaktadır. Fakat belirtilen yönde paralel ışınlar yollamaktadır. Yönsel ışık, güneş gibi düşünülebilir. Şekil 5.8' de yönsel ışığın yüzeyler üzerine olan etkisi gösterilmektedir.



Şekil 5.3. Yönsel ışık [53]

5.1.2.2 Noktasal ışık

Noktasal ışık evlerde kullanılan lambalar gibi düşünülebilir. Belirli bir noktadan bütün yönlere ışık yaymaktadırlar. Şekil 5.3' de noktasal kaynağının nasıl ışık yaydığı gösterilmektedir. Şeklin hemen ardından noktasal ışık kaynağını oluşturan fonksiyon gösterilmektedir.

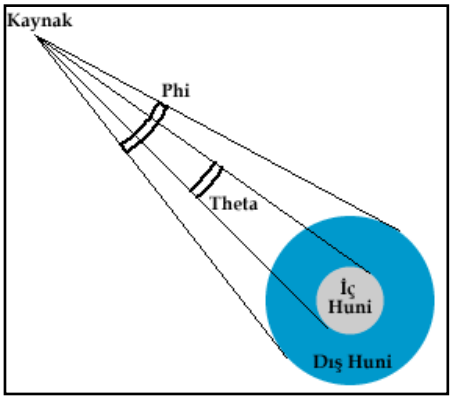


Şekil 5.4. Noktasal Işık

5.1.2.3 Fener ışığı

Fener ışık kaynağı diğerlerine göre geliştirilmesi en karmaşık olan kaynaktır. Gerçek dünyadaki fener ışığını taklit etmektedir. Şekil 5.8' de fener ışığın örneği verilmiştir.

Şekil 5.5. Fener ışığı

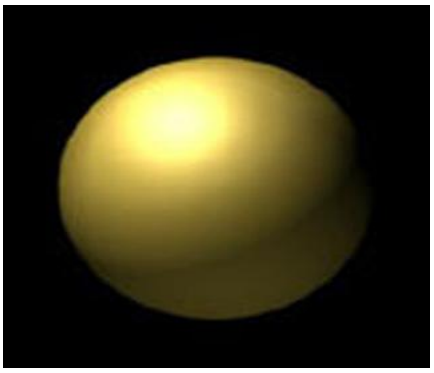


Şekil 5.6. Fener ışığının parametreleri

Fener ışığında iki önemli parametre devreye girmektedir. Bunlar şekil 5.9' da gösterilen iki açıdır. Bu açılar oluşacak ışığın iç ve dış hunilerini belirleyecektir[54].

5.1.3. Yansıma ışık

Gerçek dünyada parlak nesneler üzerine ışık çarptığında cisim üzerinde parlama olmaktadır. Bilgisayar dünyasında bunun karşılığı yansıma ışıklardır. Şekil 5.7' de buna bir örnek gösterilmektedir.



Şekil 5.7.Yansıma ışığının şekil üzerindeki etkisi

5.1.4. Salıcı ışıklar

Bir ışığın sahne üzerindeki etkisinin hesaplanması oldukça uzun işlemler gerektirmektedir. Bu sebepten dolayı DirectX aynı anda sekiz ışığın hesabına imkan vermektedir. Gerçek dünyada sokak lambaları gibi yüzlerce ışık kaynağı çok küçük bir alanda bulunabilmektedir. Bu sebepten dolayı yeri değişmeyen ışıkların aydınlatma hesapları daha önceden yapılır. Tasarımcılar sahneyi oluşturduklarında sabit ışık kaynaklarının etkisini sahneye uygulatmaktadırlar. Böylece programcı bu etkiyi hesaplattırmaktan kurtulmuş olur. Yeri değişmeyen ışınların şekiller üzerindeki etkisini temsil etmektedir.

Şekil 5.7' de salıcı ışıkların sahne üzerindeki etkisi gösterilmektedir. Çembere alınmış lambalar hiç yer değiştirmeyecekleri için sahne üzerinde yapacakları etki daha önceden hesaplanabilmektedir.

Şekil 5.8.Salıcı ışığın sahne üzerinde etkisi

5.2. DirectX Işık Sistemi

DirectX ışık sistemi gerçek dünya ışık modelinin basitleştirilmiş bir halini kullanmaktadır. Buna göre ışığın çarptığı yüzeyin materyali ve ışığın çarpış açısı kullanılarak yüzeyin alacağı renk hesaplanmaktadır. Gerçek dünyada ışık yüzeylere

çarptıktan sonra yoluna devam etmekte ve başka yüzeylere de çarpmaktadır. Bu tür bir hesaplamanın gerçek zamanlı olarak gerçekleştirilmesi mümkün görünmemektedir. Bu sebepten dolayı gerçek zamanlı ışık hesaplamaları için daha basit bir sistem kullanmaktadır . DirectX' de ışık sisteminin aktif edilebilmesi için prototipi aşağıda verilmiş olan fonksiyon kullanılmaktadır[55].

```
HRESULT          IDirect3DDevice9::SetRenderState (
        D3DRENDERSTATETYPE          State,
        DWORD          Value );
Direct3DAraci->SetRenderState (D3DRS_LIGHTING, TRUE);
```

5.2.1. DirectX' de ışık yapısı

DirectX' de kullanılacak olan temel ışık yapısının prototipi aşağıda verilmiştir. Direkt ışıklar bu yapı kullanılarak oluşturulmalıdır. Oluşturulacak ışığın türüne göre bu yapı farklı şekillerde doldurulacaktır[56].

```
typedef struct _D3DLIGHT9 {
        D3DLIGHTTYPE Type;
        D3DCOLORVALUE Diffuse;
        D3DCOLORVALUE Specular;
        D3DCOLORVALUE Ambient;
        D3DVECTOR Position;
        D3DVECTOR Direction;
        float Range;
        float Falloff;
        float Attenuation0;float Attenuation1;float Attenuation2;float Theta;float Phi;
} D3DLIGHT9;
```

5.2.2. DirectX' de materyal

Gerçek dünyada cisimler maddesel yapılarına göre ışığı yansıtmaktadırlar. Örneğin kırmızı görünen bir elma yapısı itibari ile kendisine gelen ışığın kırmızı rengini

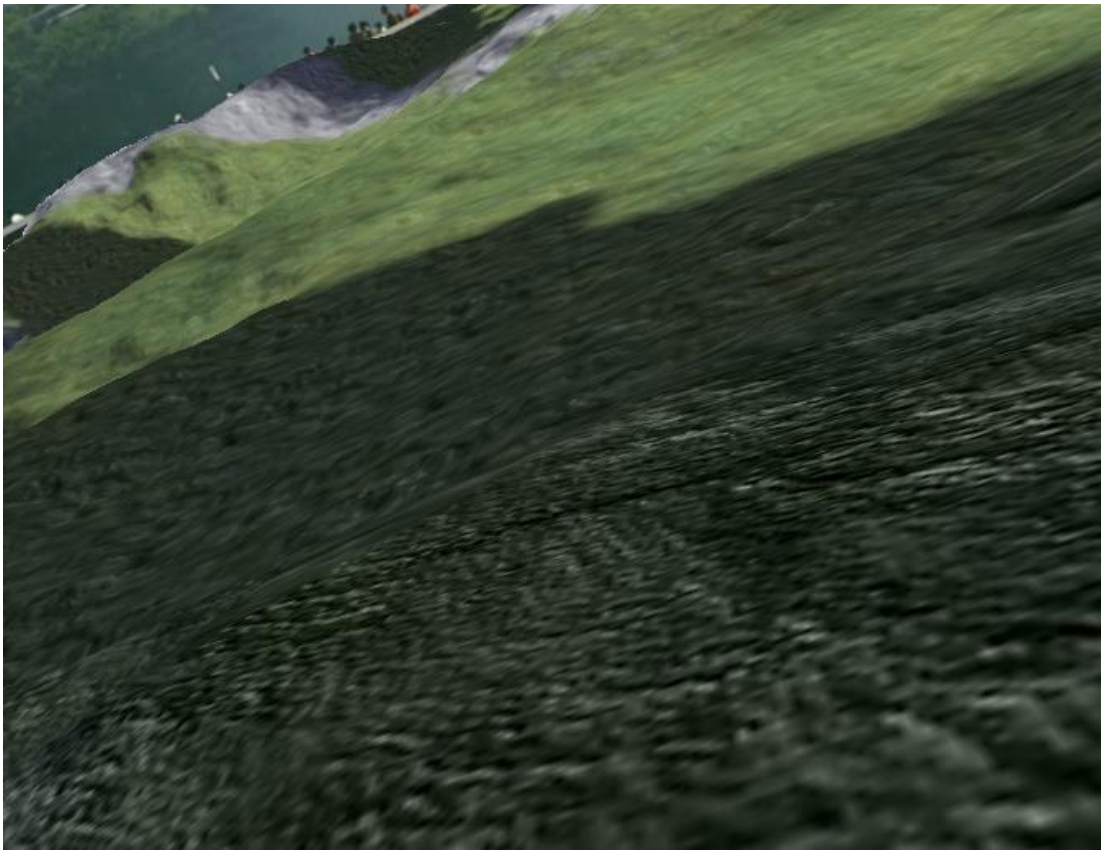
yansıtırken diğerlerini emmektedir. Bu sebepten dolayı elma göze kırmızı görünmektedir. Aynı şekilde bilgisayar grafiklerinde de yüzeylerin materyalleri mevcuttur. Materyal yüzeylerin yansıtacağı rengi belirtmektedir. DirectX yüzeylerin materyallerini aşağıdaki yapıyı kullanarak belirtmektedir[57].

```
typedef struct _D3DMATERIAL9 {
        D3DCOLORVALUE Diffuse,
        D3DCOLORVALUE Ambient,
        D3DCOLORVALUE Specular,
        D3DCOLORVALUE Emissive;
        float Power;
} D3DMATERIAL9;
IDirect3DDevice9::SetMaterial(CONST D3DMATERIAL9* pMaterial)
```

Materyalin ilk dört değişkeninin ismi görevini açıklamaktadır. Her bir ışık türüne materyalin nasıl karşılık vereceğini belirtmektedir. Materyali şekle uygulamak için çizimden önce "SetMaterial" fonksiyonu kullanılmaktadır.

5.3. Oyun Motorundaki Işık Mekanizması

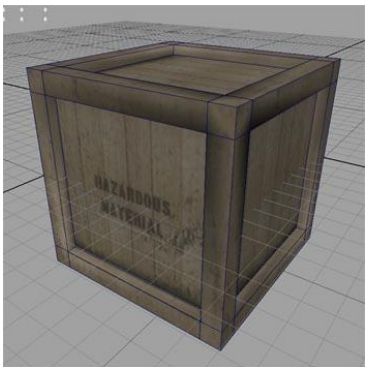
Oyun motorundaki ışık mekanizması ile programcı sahneye hızlı bir şekilde farklı ışık kaynaklarını kullanabilmektedir. Motor aynı anda sekiz ışığın aktif olmasına izin vermemektedir. Motor bunu ışıklar tarafından etkilenen şekilleri gruplandırarak gerçekleştirmektedir. Nesneler kendilerini etkileyen ışık sayısına göre gruplandırılmaktadır. Şekil 5,8'de motordan alınan bir sahne gösterilmektedir.



Şekil 5.9. Oyun motorunun ışık sistemi

5.4. Kaplama

Kaplama üçgenler üzerine gerçek resimlerin geçirilmesi işlemidir. Bu işlem sayesinde şekillerin detayı ve gerçekçiliği artmaktadır. Örneğin normal bir küp şekildeki gibi görünebilir.



Şekil 5.10. Kaplama uygulanmış bir kutu

Kaplama 3B uygulamalarda çok büyük bir öneme sahiptir. Kaplamalar aslında iki boyutlu resimlerden ibarettir. Bu resimler yüzeyler üzerine giydirilmektedir. Şekilde de görüldüğü gibi üzerine kaplama uygulanmış yüzeyler bir dörtgenden çok gerçek bir kutuyu andırmaktadır.

Kaplamalar genellikle eşit yükseklik ve genişliğe sahip olan resimlerden oluşurlar. Bunun temel sebebi hızdır. Bilgisayar ikinin üssü sayılar üzerinde çok daha hızlı işlem yapabilmektedir. Kaplama boyutlarına örnek vermek gerekirse 32x32 , 256x256, 1024x1024 veya 4098x4098 olabilmektedir.

5.4.1. Kaplamaların hafızadaki yeri

Kaplamaların kullanılabilmesi için öncelikle hafızaya yerleştirilmesi gerekir. Günümüz oyunları yüksek çözünürlüklü kaplamalar kullandığı düşünülürse bu kaplamaları hafızada nasıl saklanacağını bilmek ve kontrol etmek çok önemlidir.

Kaplamalar lineer renk bilgilerinden ibarettir. Renkler 32 bitlik doğal sayı değişkenlerinden ibarettir. Buna göre kaplama renk bilgileri hafızaya statik bir dizi gibi yerleştirilecektir. Yani doğrusal olarak sıralanmış baytlardan oluşacaktır. Örneğin 1024x1024 boyutlarında 32 bitlik bir kaplamanın hafızada kaplayacağı alan 1024x1024x4 bayt olacaktır.

Eğer kaplamaların hafızda kapladıkları alan ikinin üzeri bir rakam değilse DirectX devreye girer. Örneğin bir kaplamanın hafızadaki boyutu 1000 bayt olsun. DirectX bu alana 24 bayt daha ekleyip toplam alanı 1024 bayt yapacaktır. Bu sayede işlemci veya ekran kartı işlemcisi kaplamayı çok daha hızlı çekebilecektir. Eklenen baytlar anlamsızdır sadece boşluğu doldurmak amacı ile kullanılır.

5.4.2. Kaplama formatı

Kaplamalar piksellerden ibarettir. 32x32 boyutlarında bir kaplamanın 1024 adet pikseli bulunmaktadır. Bu pikseller 32 bit ile temsil edildiği gibi 16 veya 8 bitle de

temsil edilebilmektedir. DirectX de kullanılan temel piksel formatları tablo 5.1' de gösterilmektedir.

Tablo 5.1. Piksel formatları

D3DFORMAT Üyesi	Açıklama
D3DFMT_A8R8G8B8	Her bir piksel 32 bitlik bir değer ile temsil edilir. Bunların 8'i alfa yani saydamlığı , 8'i kırmızı ,8'i mavi ve 8'i yeşil rengi temsil etmektedir.
D3DFMT_X8R8G8B8	32 bitlik bir değerle temsil edilir. Fakat alfa değeri kullanılmamaktadır. Burada 8 bit boşluk doldurup 32 bite tamamlaman için kullanılmaktadır. JPEG formatı da aynıdır.
D3DFMT_A2B10G10R10	2 bit alfa 10 bit de diğer renkler için kullanılmaktadır.
D3DFMT_A8B8G8R8	İlk format ile tek farklı renklerin hafızaya yerleştirilmesidir.
D3DFMT_X8B8G8R8	İkinci formattan tek farkı renklerin hafızaya yerleştirilmesidir.
D3DFMT_G16R16	Yeşil ve kırmızı renkleri için 16 bit bulunduran bir formattır. Çok kullanılmamaktadır.

5.4.3. Çoklu kaplama

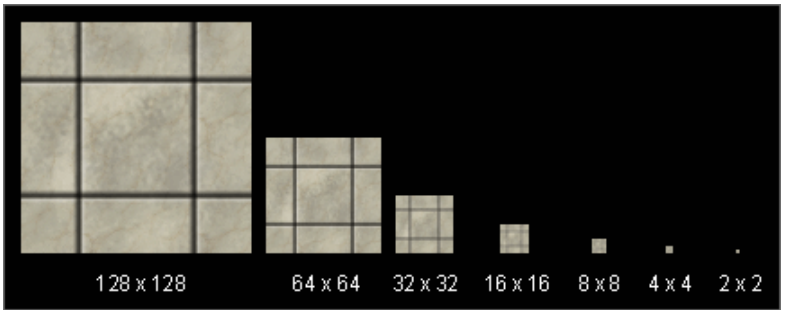
Kameranın kaplamalı yüzeye uzaklığı değiştiğinde kaplamanın görünüşü de değişmektedir. Örneğin 128x128 boyutlarında olan bir kaplama ekranda aynı boyutlarda piksel kaplayacaktır. Fakat kamera hareket ettiğinde kaplamanın ekranda kaplayacağı alan değişecektir. Örneğin kamera yüzeyden uzaklaştığında kaplama ekranda daha az yer kaplayacaktır ve bazı kaplama bilgileri ekarte edilecektir. Tam aksine kamera yaklaştığında normalden daha fazla yer kaplayacaktır. Bu sebepten dolayı da kaplama kendi boyutundan daha fazla yer kaplayacağından bir bozulma söz konusu olacaktır[58].

Oyun programcıları bu sorun için Latince "Multum In Parvo" açılımını taşıyan MIP haritalarını geliştirdi. Türkçe "küçük içinde fazla" anlamını taşımaktadır. Şekil 5.11' de MIP haritalamanın getirdiği avantaj görülmektedir.

Şekil 5.11. MIP haritalama tekniğinin etkisi

Şekilde ilk kısım MIP haritalamanın kullanılmadığı durumu göstermektedir. Dikkat edilecek olursa uzakta kalan kaplamalar monitörde çok daha az piksel kapladığından görüntüde bozulma olmaktadır. MIP haritaları kullanıldığında bozulma azalmaktadır.

MIP haritalama bir kaplamanın farklı boyutlarda kopyalarının kameranın uzaklığına göre kullanılması işlemidir. Örneğin 64x64 boyutlarında olan bir kaplamanın boyutları daha küçük veya büyük olan kopyaları bulunabilir. Şekil 5.11' de bu kopyalar gösterilmektedir. Sisteme bu kopyalar gösterildiğinde kamera uzaklığına göre otomatik olarak kaplamanın uygun hali kullanılacaktır. Örneğin kamera fazla yaklaşırsa 128x128 boyutlarındaki kaplama kullanılırken, kamera uzaklaştığında da 32x32 boyutlardaki kaplama kullanılabilir.



Şekil 5.12. MIP Haritaları

5.4.4. Kaplama filtreleri

Kaplamaların ekranda kapladığı alan kendi boyutlarından farklı olduğunda görüntüde bozulma olmaktadır[59]. MIP haritalama tekniği bunu belli bir oranda çözebilmektedir. Fakat harita sayısını arttırmak kullanılacak hafızayı da arttıracaktır. Bu sebepten dolayı boyutları değişen kaplamalardaki bozulmayı en aza indiren filtreleme teknikleri kullanılmaktadır.

Temel filtreleme metotları nokta, eş yönsüz ve doğrusal filtrelemedir. Doğrusal filtreleme noktasal olana göre daha iyi sonuçlar vermektedir. Şekil 5.12' de oyun motorunda kullanılan noktasal ve doğrusal filtreleme tekniğinin sonuçları gözükmektedir. Eş yönsüz filtreleme kameranın kaplamalara farklı açılarda baktığı durumlarda kullanılmaktadır. Kamera yüzeye dikey olarak bakmadığında kaplamada bozulma olacaktır. Bunu önlemek için eş yönsüz filtreleme kullanılır. Eş yönsüz filtreleme tekniği yüksek işlem gücü istemektedir.

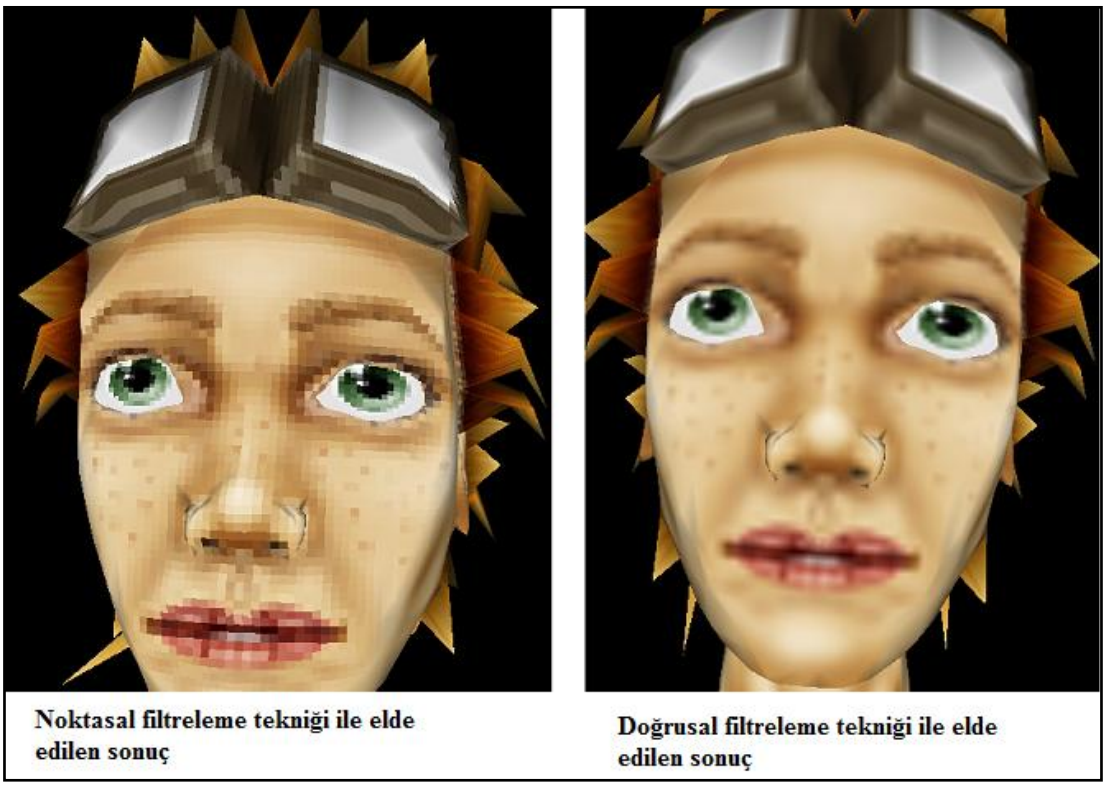
5.5. Oyun Motorundaki Kaplama Mekanizması

Oyun motoru içerisindeki kaplama mekanizmasının temel görevi, gerçekçi sahneler oluştururken hız ve hafızanın sağlıklı bir şekilde kullanılmasını sağlamaktır. Oyunlarda hafızanın büyük bir bölümünü kaplamaların dolduracağı düşünüldüğünde kaplama yönetimi daha büyük bir önem kazanmaktadır.

Kaplama mekanizması yüklenen bütün kaplamaların bir listesini tutmaktadır. Bu sayede aynı kaplamanın hafızaya birden fazla yüklenmesi engellenebilmektedir.

Ayrıca kaplamalar arası geçişi azaltmak için aynı kaplamaya sahip olan yüzeyler gruplandırılmaktadır. Gruplandırma özelliği ile her saniye de gösterilen çerçeve sayısının arttığı gözlemlenmiştir.

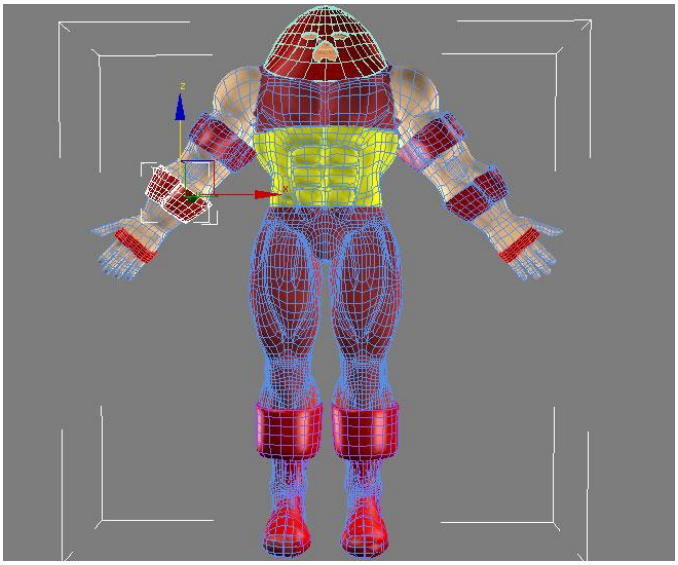
Geliştirilen filtreleme mekanizması sayesinde programcı kaplamalara istediği filtreleri kolaylıkla uygulayabilmektedir. Şekil 5.12' de motordan alınmış filtre örnekleri gösterilmektedir.



Şekil 5.13. Oyun motorundaki filtreleme mekanizması

BÖLÜM 6. MODEL VE ANİMASYON KATMANI

Oyunların temel şekli üçgen olmasına rağmen tek başına üçgenlerin hiç bir anlamı yoktur. Oyunlarda, üçgenler modelleri, modellerde sahneyi oluşturmaktadır. Örneğin bir karakter veya bir ev birden fazla üçgenden oluşmaktadır. Birden fazla üçgenden oluşan modellere zincir adı da verilmektedir. Zincir hakkındaki bilgiler model dosyalarında tutulmaktadır. 3D Studio Max veya Maya gibi programlar ile çizilen modeller belirli formattaki dosyalara kaydedilir. Örneğin 3D Studio Max programı model dosyalarını. max uzantılı dosyalara kaydetmektedir. Fakat bu dosya formatı kamuoyuna açıklanmadığı için içeriği hakkında bir bilgi bulunmamaktadır.



Şekil 6.1. 3D Studio Max programında tasarlanan bir model

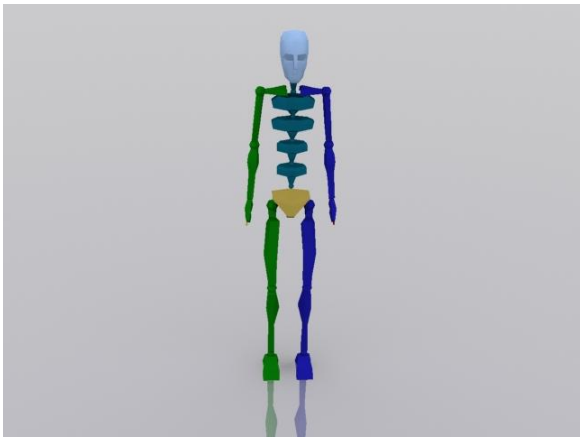
Bazı model formatlarının içeriği kamuoyuna açıklandığından programcılar tarafından rahatlıkla kullanılabilmektedir. Örneğin tezde geliştirilen oyun motorunda 3ds, X ve md5 dosya formatları kullanılmaktadır.

İki tür model bulunmaktadır. Bunlar statik ve dinamik modellerdir. Statik modeller hiç bir hareket kabiliyetine sahip değildirler. Dinamik modeller ise belirli bölgelerini hareket ettirebilmektedirler. Örneğin bir insan modeli dinamik olmak zorundadır.

6.1. Model Hiyerarşisi

Dinamik modellerin hareket ettirebilmesi için hiyerarşik bir yapıya ihtiyaç duyulmaktadır. Modelin harekete geçirilecek olan uzuvları hiyerarşide birer düğümü temsil edecektir. Şekil 6.1'de bir model hiyerarşisi gösterilmektedir. Hiyerarşideki her bir parçaya kemik adı verilmektedir. Her bir kemik başka bir kemiğe bağlıdır[60].

Modellerin çizildikleri ortamda hiyerarşileri de belirlenmektedir. Çizilen modeller bir dosyaya kaydedilirken hiyerarşi bilgileri de ek olarak kaydedilmektedir.



Şekil 6.2. Dinamik bir modelin kemik gösterimi

Geliştirilen oyun motoru X,md5 ve 3ds formatı ile saklanmış olan modellerin hiyerarşilerini yükleyebilmektedir.

6.2. Alt Kümeler

Modellerin içerisindeki aynı özelliklere sahip olan üçgenlerin oluşturduğu yapıya alt küme adı verilmektedir. Model zincirlerinin birden fazla altkümesi bulunabilir.

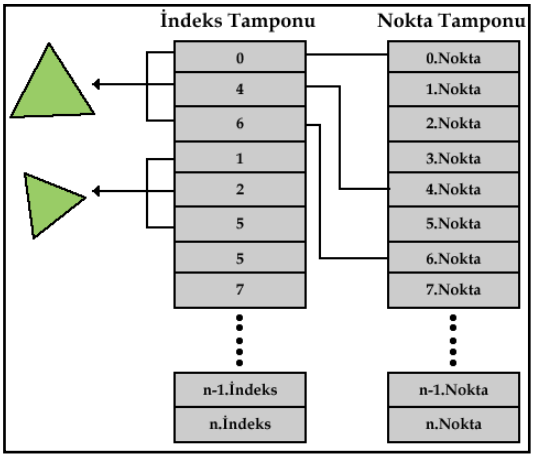
Altküme aynı özellikler (materyal ve kaplama) ile çizilebilen üçgenler grubundan ibarettir. DirectX kaplama ile materyali bir arada kullanıp bu ikisinin oluşturduğu yapıya özellik adını vermiştir.

Modeller başlangıçta belirli alt kümelerden oluşmamaktadır. Programcı, model dosyasını okuyup aynı özelliklere sahip olan üçgenleri gruplandırmalıdır. Geliştirilen oyun motoru bu yükü programcının üzerinden almaktadır. Programcının yapması gereken bir model sınıfı oluşturduktan sonra modeli bir fonksiyon yardımı ile yüklemekten ibarettir. Aşağıdaki kod bloğunda iki farklı modelin motor kullanılarak yüklenişi gösterilmektedir.

```
CActor m_pActor = new CActor();
m_pActor->LoadFromX("tiny.x",m_pD3DDevice );
```

6.3. İndeks Tamponu

Modeller üzerindeki üçgenler aynı noktayı paylaşabilmektedirler. Bu durum hafıza kaybına sebep olmaktadır. Hafıza kaybını minimuma indirmek için indeks tamponu kullanılır. Bu teknik de, öncelikle modeli oluşturan bir birinden farklı bütün noktalar nokta tamponuna yerleştirilir. İndeks tamponunda ise hangi noktaların üçgenleri oluşturacağının bilgisi tutulur.

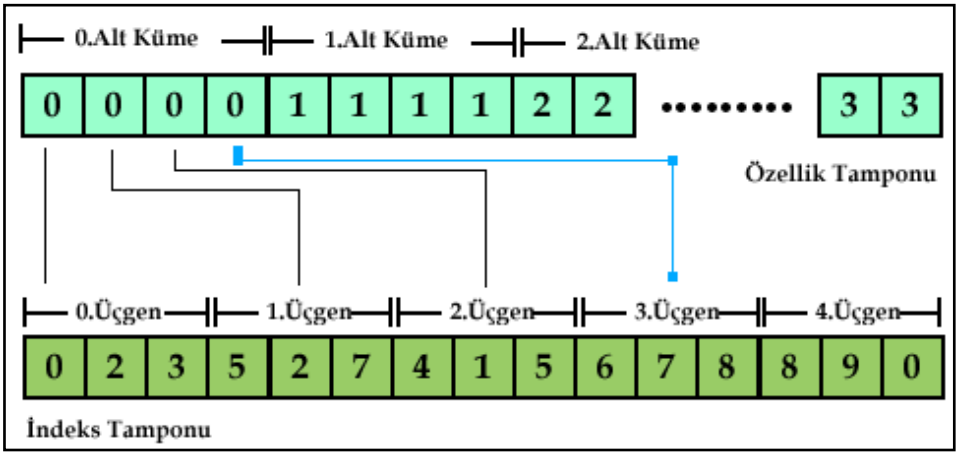


Şekil 6.3. İndeks Tamponu

Şekil 6.2' de indeks tamponunun işlevi gösterilmektedir. Buna göre indeks tamponundaki ilk üç değer kullanılarak nokta tamponundan üç nokta seçilmektedir. Ardından bu noktalar kullanılarak üçgen çizilmektedir. Oyun programlarında noktalar kimi zaman yüz bayta kadar büyüklüğe sahip olabilmektedir. Buna karşın indeks tamponunda iki baytlık sayılar bulunacaktır.

6.4. Özellik Tamponu

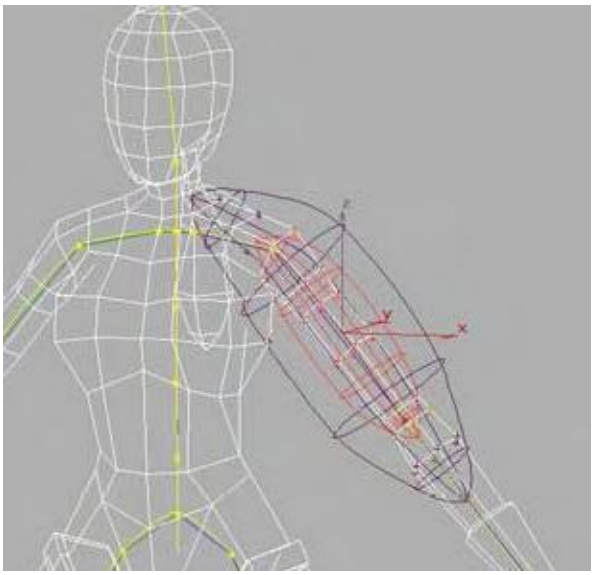
Model zincirinde farklı özelliklere ait üçgenler bulunabilir. Örneğin bir model içerisinde dört farklı özellik bulunduğu varsayılsın. Buna göre her bir üçgenin hangi özelliği kullandığı programcı tarafından bilinmelidir.



Şekil 6.4. Özellik tamponu

6.5. İskelet Animasyonu

İskelet animasyonu 3-B modellerin gerçekçi hareket etmesini sağlayabilen bir animasyon tekniğidir. Bu teknikte model yine hiyerarşik bir yapıdadır. Her bir kemiğin etkilediği noktalar bulunmaktadır. Kimi noktalar aynı anda birden fazla kemik tarafından etkilenebilmektedir. Örneğin bir insan modelinin diz bölümünde bulunan noktalar birden fazla kemik tarafından etkilenecektir. Şekil 6.4' de bir kemiğin etki alanı görünmektedir.



Şekil 6.5. Kemiklerin etki alanı

Kemikler hareket ettiğinde etkileri altında olan noktalarda hareket edecektir. Bu sayede modelin uzuvları hareket etmiş olacaktır. Birden fazla kemik tarafından etkilenen noktalar ise etki katsayısına göre hareket edecektir.

Bu tezde geliştirilen oyun motoru sayesinde programcılar iskelet animasyonunun karmaşık yapısı ile ilgilenmek zorunda kalmayacaklardır. Model yüklendiğinde onunla beraber bütün animasyon mekanizması da hazır hale gelecektir.

6.6. Model Materyali

Model dosyaları içerisinde modelin çizilmesinde kullanılacak olan materyal bilgileri de bulunmaktadır. Programcılar model formatına göre bu bilgileri okuyup materyalleri hazırlamalıdırlar. Oyun motorunun görevlerinden birisi bütün bu işlemlerin programcının üzerinden alınmasıdır.

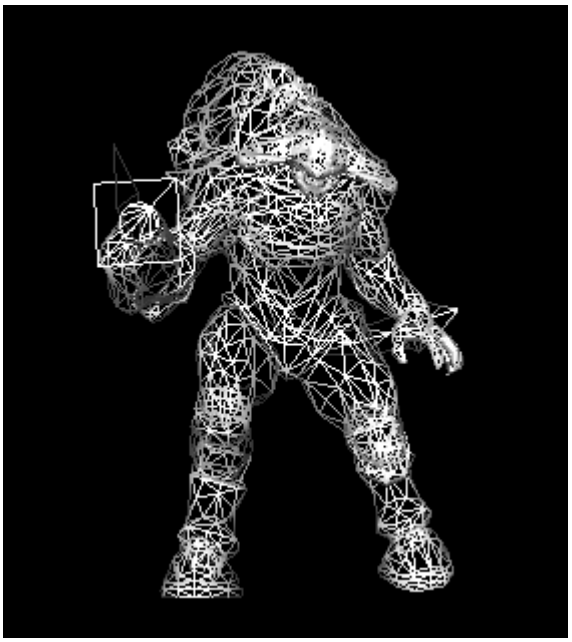
6.7. Oyun Motorunun Model Mekanizması

Geliştirilen oyun motorunda üç farklı formatı okuyup anime ettirebilen bir model mekanizması bulunmaktadır. Bunlar x, md5 ve 3ds model formatlarıdır. Örnek olarak md5 modelini kontrol eden sınıf aşağıdaki gibi tanımlanmıştır

```
class CDoomActor
{
public:
        CDoomActor();
        virtual ~CDoomActor();
public:
        bool    LoadMeshFile(LPCSTR pFileName,LPDIRECT3DDEVICE9
        pDevice);
        bool    LoadAnimFile        (LPCSTR pFileName);
        void    ReadJoints                  ();
        void    ReadMesh                    ();
        float   GetQuatW                    (float x,float y,float z);
        float   GetQuatW                    (D3DXQUATERNION *pQ);
        char     *CopyString                (char *pStr);
        bool    CreateMeshes        ();
        void    Draw                        (double fTimeElapsed);
        char     *GetNextLine                       ();
        void    BuildVertexPosition ();
        void    AdvanceTime                 (double fTimeElapsed);
        void    AddAnimationSet             ();
        void    CreateJointFrames   ();
        void    SetAnimation        (int AnimationIndex);
        int     GetCurrentAnim              (){return m_nActiveAnim;}
private:
        LPDIRECT3DTEXTURE9  m_pTexture;
        D3DXMATRIX          m_mtxWorld;
        FILE                 *m_pFile;
        char                 **m_pStrBuffer;
        int                 m_nCurLine;
        int                 m_nNumLines;
        int                 m_nMaxWeightCount;
        int                 m_nNumAnim;
        int                 m_nNumAnimationJoints;
```

```
        int             m_nActiveAnim;
        bool            m_bIsAnimChanged;
        AnimationSet    *m_pAnimSets;
        IDirect3DDevice9 *m_pD3DDevice;
        LPD3DXMESH      *m_ppD3DMeshes;
        LPCSTR          m_strMeshVersion;
        LPCSTR          m_strAnimVersion;
        LPCSTR          m_strCommandLine;
        UINT            m_nNumJoints;
        UINT            m_nNumMeshes;
        JOINTS          *m_pJoints;
        MESH            *m_pMeshes;
};
```

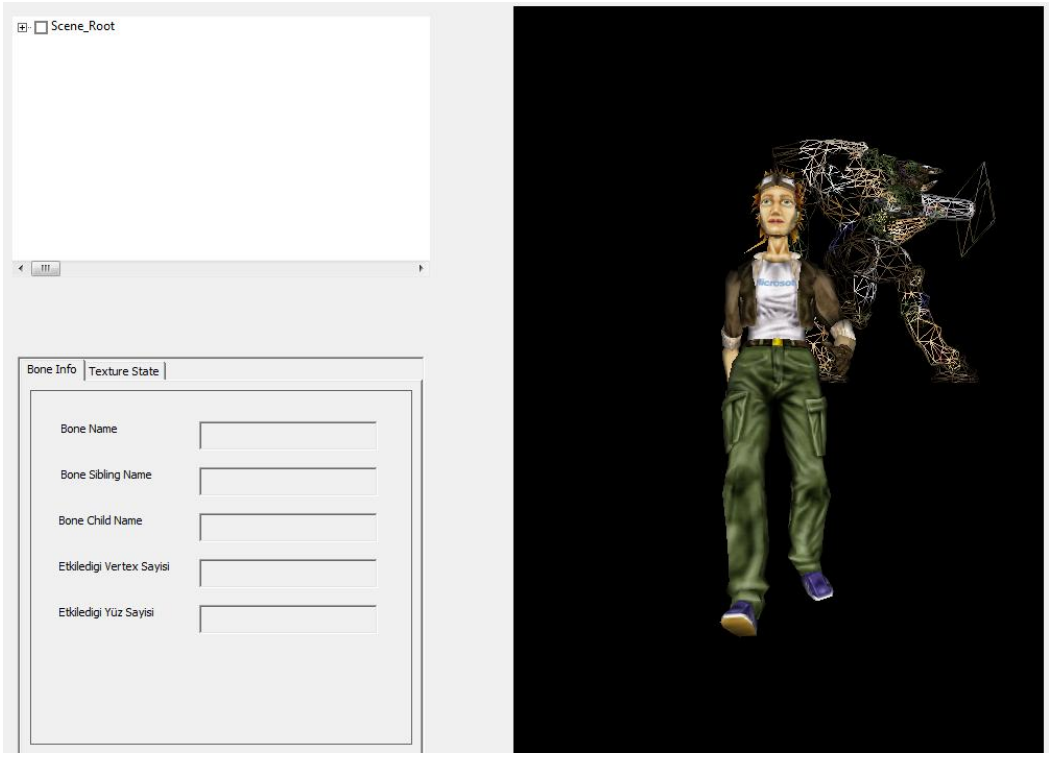
Bu sınıf sayesinde ortaya çıkan sonuç şekil 6.5' de gösterilmektedir.



Şekil 6.6. MD5 formatının çizdirilmesi

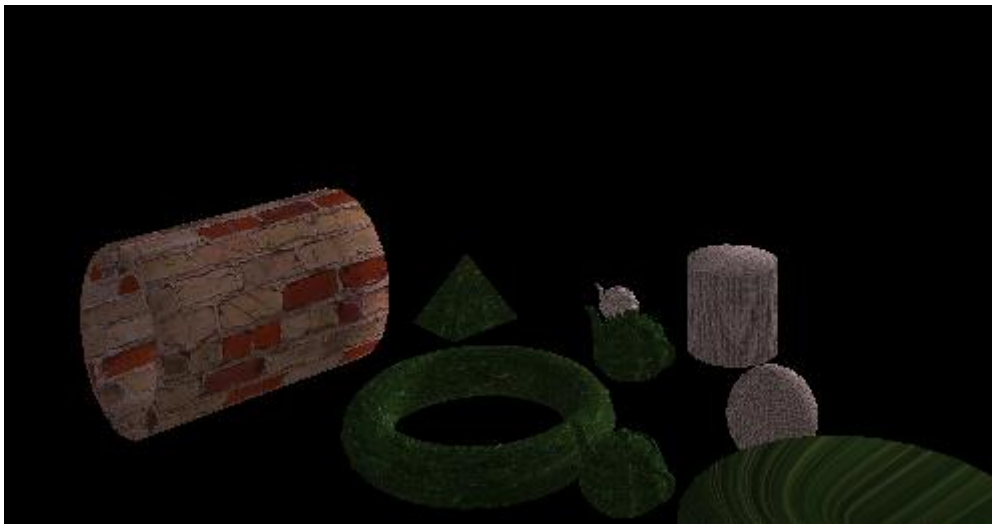
BÖLÜM 7. OYUN MOTORUNUN UYGULANMASI

Günümüz oyun motorlarının en temel özelliği, oyun programlamadaki iş yükünü azaltmasıdır. Bu tezde geliştirilen oyun motorunu diğer motorlardan ayıran temel özellik, farklı model formatları ile çalışabilmesidir. Şekil 6.1' deki model editöründen alınan bir resim bunu kanıtlamaktadır. Ayrıca şekil 6.2' de 3D Studio Max programından çıkartılan 3ds dosya formatının çizimi de gösterilmektedir.



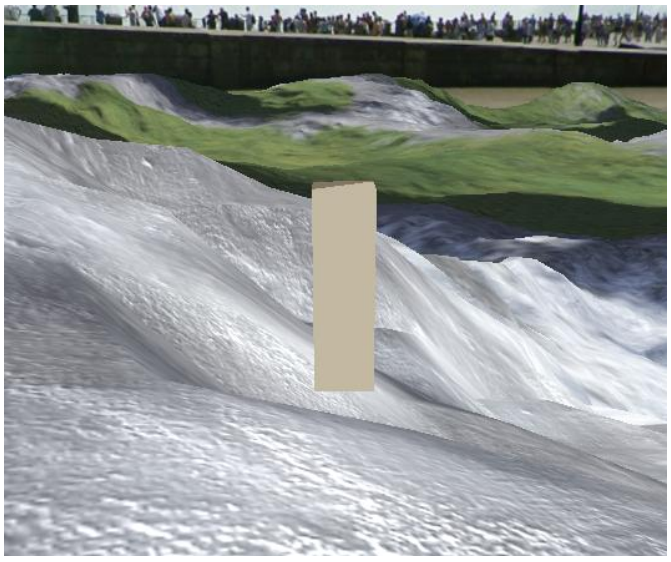
Şekil 7.1. Model editöründe birden fazla modelin incelenmesi

Şeklin ön kısmında X dosya formatından okunmuş model görünürken arka taraftaki ızgara biçiminde çizdirilmiş olan modelse md5 dosya formatından okunmuştur. Modeller için geliştirilmiş sınıflar sayesinde programcı onları kolayca manipüle edebilecektir.



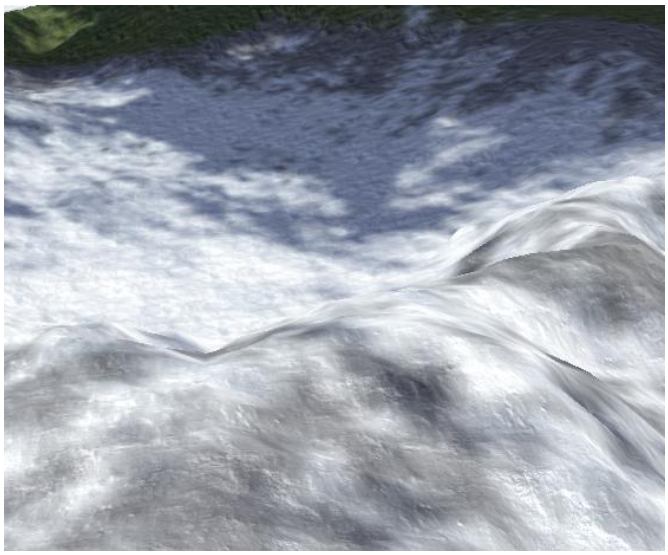
Şekil 7.2. Model editöründe 3ds dosya formatının çizilmesi

Üç adet kamera modeli sayesinde programcı farklı türde oyunları geliştirme imkanına sahiptir. Kamera modelleri arasında geçiş tek bir tuş ile hiç bir yavaşlama yapmadan geçiş yaptığı gözlemlenmiştir. Şekil 6.3,6.4 ve 6.5' de bu kamera modelleri gösterilmektedir.



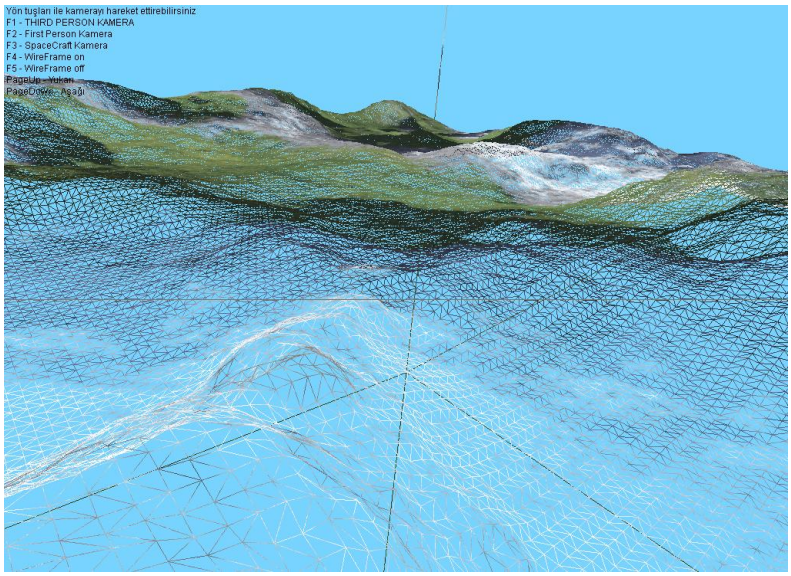
Şekil 7.3. Üçüncü kişi kamera modeli

Şekil 7.4. Birincil kişi kamera biçimi



Şekil 7.5. Uzay kamera modeli

Motorda bulunan zemin geliştirici sayesinde oyun dünyasının temeli hızlı bir şekilde oluşturulmaktadır. Şekil 6,6'de RAW formatında saklanmış bir zeminin motor tarafından işlenmiş hali gösterilmektedir.



Şekil 7.6. Izgara biçiminde çizdirilmiş olan zemin

Kaplama mekanizması sayesinde şekiller daha gerçekçi görünmektedir. Bu sayede kamera hangi açıdan veya mesafeden bakarsa baksın kaplamalar üzerinde büyük bir bozulma olmamaktadır. Şekil 6.7' de modele uygulanan kaplama filtrelerinin etkisi gösterilmektedir.

Şekil 7.7. Kaplamalara filtre uygulanması

KAYNAKLAR

[1] EMERAN, R., MONGOMERY, S., Design and Development of Peer-to-Peer Online Multiplayer Game Using DirectX and C#, TENCON 2004. IEEE Region 10 Conference, Singapore, pp. 278-281 , 2004

[2] EMERAN, R., MONGOMERY, S., Pixel Perfect-Graphics Card Review, Personel Computer World, London, England , pp. 195-213 , 2006

[3] http://inventors.about.com/library/inventors/blcomputer_videogames.htm 10.01.2010

[4] THOMAS, C.S.C., KOK-WHY, N., A Review and Development of 3-D Accelerator Technology for Games, Second International Symposium on Intelligent Information Technology and Security Informatics, Moscow, Rusia , pp. 59-63, 2009

[5] WRIGHT ,R.S., SWEET, M., OpenGL Super Bible, Waite Group Press, pp. 18, Massachusetts,U.S.A. 1997.

[6] WALSH, P., Advanced 3D Game Programming with DirectX 9.0, Wordware Publishing, INC., pp. 54, Texas ,U.S.A., 2003.

[7] THOMAS, C.S.C., KOK-WHY, N., A Practical Implementation of a 3-D Game Engine, International Conference on Computer Graphics, Imaging and Visualization, Beijing, China , pp. 351- 358, 2005

[8] NOH, S.S., HONG, S.D., PARK, J.W., Using a game engine technique to produce 3D Entertainment contents, 16th International Conference on Artificial Reality and Telexistence, Hangzhou, China, pp. 246-251, 2006

[9] JUNKER, G., Pro OGRE 3D Programming, Apress, Inc., pp. 2, U.S.A., 2006.

[10] GOSLIN, M., MINE, M.R., The Panda3D Graphics Engine, Entertainment Computing, U.S.A. , pp. 112-114 , 2006

[11] EBERLY ,D.H., 3D Game Engine Architecture, Elsevier Inc., pp. 1, San Francisco ,U.S.A., 2005.

[12] ZERBST S.,DÜVEL O., 3D Game Engine Programming, Premier Press, Boston USA, pp.3, 2004

[13] WOLFE, A., NOONBURG, D.B., Game Engine Virtual Reality with CaveUT, Entertainment Computing, U.S.A. , pp. 79-82 , 2005

[14] WOLFE, A., NOONBURG, D.B., A Superscalar 3D Graphics Engine, 32nd Annual International Symposium on Microarchitecture, Haifa, Israel, pp. 50-61, 1999

[15] FOLEY, J.D., VAN DAM,A., FEINER ,S.K., HUGHES, J.F.,Computer Graphics: Principles and Practice, Adison - Wesley Publishing Company Inc., USA, pp.201, 1997

[16] LAMOTHE A., PENTON R., Data Structures For Game Programmers, Premier Press, pp. 43, Cincinnati, Ohio , 2003

[17] QIU, H., CHEN, L., An Object-Oriented Graphics Engine, International Conference on Computer Science and Software Engineering, Wuhan, Hubei, pp. 1027-1030, 2008

[18] ASTLE, D., HAWKINS K., Beginning OpenGL Game Programming, Premier Press, pp. 113, U.S.A. , 2004

[19] http://blog.tartiflop.com/2008/11/first-steps-in-away3d-part-3-texture-mapping/ 27.11.2009

[20] ADAMS, J., Programming Role-Playing Games with DirectX 8.0, Premier Press, pp. 241, Cincinnati, Ohio , 2002

[21] http://www.chadvernon.com/blog/tutorials/directx9/terrain-generation-with-a-heightmap/ 10.01.2010

[22] http://yasamkadin.com/v2/genel/18145-first-person-shooter-fps-oyunlarinin-gelisimi.html 10.01.2010

[23] http://www.merlininkazani.com/review_screen.asp?GID=3499&PN=2 10.01.2010

[24] VAN VERTH, J.M., BISHOP, L.M., Essential Mathematics for Games and Interactive Application,Morgan Kaufmann Publishers, pp. 1-2, Burlington, U.S.A. , 2008

[25] KODICEK, D., Mathematics and Physics for Programmers, Charles River Media, Inc, pp. 125, Hingham, Massachusetts , 2005

[26] LEITERMAN, J.C., Vector Game Math Processors, Wordware Publishing Inc., pp. 359-374, Texas, U.S.A. , 2003

[27] ftp://ftp.idsoftware.com/idstuff/doom3/source/win32/D3_1.3_SDK.exe 13.12.2009

[28] SOLTER, N.A., KLEPER S.J., Professional C++,Wiley Publishing,Inc., pp. 18, Inianapolis, USA , 2005

[29] http://en.wikipedia.org/wiki/Hashed_array_tree 13.12.2009

[30] VANDEVOORDE, D., JOSUTTIS N.M., C++ Templates: The Complete Guide, Pearson Education, Inc, pp. 89, Boston, USA , 2003

[31] DAS, V.V. Principles of Data Structures Using C and C++, New Age International Publishers, pp. 88, Daryaganj, New Delhi, 2006

[32] JONES, V. Beginning DirectX 9, Premier Press, pp. 2, Boston, USA, 2004

[33] http://www.gamedev.net/reference/articles/article1775.asp 13.12.2009

[34] http://www.nextdawn.nl/the-history-of-directx 15.12.2009

[35] LUNA, F.D., Introduction to 3D Game Programming with DirectX 9.0, Worldware Publising,Inc. pp. 37, Texas, USA, 2003

[36] PETZOLD, C., Programming Windows, Microsoft Press. pp. 3-71, U.S.A., 1999

[37] MART, J.M., Windows System Programming, Addison Wesley Professional. pp. 15-88, U.S.A., 2005

[38] CANTON, M.P., SANCHEZ J., DirectX 3D Graphics Programming Bible, IDG Books Worldwide, Inc., pp.19, Texas,USA, 2005

[39] http://www.pcb007.com/pages/zone.cgi?a=51646&_pf_=1 15.11.2009

[40] http://upload.wikimedia.org/wikipedia/commons/thumb/0/03/Tearing_(simulated).jpg/797px-Tearing_(simulated).jpg 25.09.2009

[41] LUNA, F.D., Introduction to 3D game programming with DirectX 9.0c: a shader approach, Wordware Publishing, Inc., pp.71, Texas,USA, 2006

[42] http://upload.wikimedia.org/wikipedia/en/thumb/d/df/Swap_chain_depiction.svg/800px-Swap_chain_depiction.svg.png 25.09.2009

[43] THORN, A., DirectX 9 User Interface: Design and Implementation, Wordware Publishing, Inc., pp.29, Texas,USA, 2006

[44] KOVAK, P.J., Inside Direct3D, Microsoft Press, pp.66, USA, 2000

[45] THORN, A., DirectX 9 Graphics, The Definitive Guide to Direct3D, Wordware Publishing, Inc., pp.21 25.09.2009

[46] http://www.cs.unc.edu/~mcmillan/comp136/Lecture21/Painter.html 24.12.2009

[47] http://www.beyond3d.com/images/articles/ZStencil/Z-Buffer.gif 24.10.2009

[48] http://doc.51windows.net/Directx9_SDK/?url=/Directx9_SDK/graphics/reference/d3d/interfaces/idirect3ddevice9/clear.htm 30.10.2009

[49] http://doc.51windows.net/Directx9_SDK/?url=/directx9_sdk/graphics/reference/d3d/interfaces/idirect3ddevice9/Present.htm 30.10.2009

[50] http://msdn.microsoft.com/en-us/library/ms644943(VS.85).aspx 10.11.2009

[51] LENGYEL E., Mathematics for 3D Game Programming, Charles River Media, INC., pp. 163, Hingham Massachusetts, 2004.

[52] https://www.ogre3d.org/forums/viewtopic.php?f=11&t=47927&start=75 26.11.2009

[53] http://www.toymaker.info/Games/html/lighting.html 27.11.2009

[54] WOO, M., NEIDER, J., DAVIS, T., Opengl Programming Guide: The Official Guide to Learning Opengl, Version 1.1, Addison Wesley Longman, Inc., pp. 126-161, U.S.A., 1997

[55] http://msdn.microsoft.com/en-us/library/ee421767(VS.85).aspx 28.07.2009

[56] http://msdn.microsoft.com/en-us/library/ee416501(VS.85).aspx 30.09.2009

[57] http://msdn.microsoft.com/en-us/library/ee416506(VS.85).aspx 30.09.2009

[58] http://www.gamedev.net/reference/articles/article1233.asp 31.09.2009

[59] HARBOUR, J.S., Beginning Game Programming, Thomson Course Technology PTR, pp. 91-111, U.S.A., 2005

[60] LEVER, N., Real-time 3D Character Animation with Visual C++, Focal Press, pp. 124-145, Woburn, U.S.A., 2002

[61] http://www.cmis.brighton.ac.uk/staff/alb14/CI219/Lectures%20&%20Tutorials/Lecture_10_Animation_Biped.html 13.12.2009

BÖLÜM 8. SONUÇ VE ÖNERİLER

Bu çalışmada günümüz bilgisayar oyun programcılarının temel araçlarından birisi olan oyun motoru geliştirilmiştir. Bu oyun motoru sayesinde sanal dünyalar daha hızlı ve daha kolay oluşturulabilmektedir. Kolaylıkla test edilebilen alt yapısı sayesinde programcının hata yapma riskini azaltmaktadır. Ayrıca bu çalışmayla farklı dosya formatındaki modeller ile çalışmak daha kolay bir hale gelmektedir.

Geliştirilen kaplama mekanizması sayesinde hafıza daha verimli yönetilmiştir. Aynı kaplamayı kullanan modeller gruplandırılarak, bu modellerin hepsi için hafızada tek bir kaplama kullanılmıştır. Bu sayede ekran kartı ile sistem hafızası arasındaki veri alışverişi azaltılmıştır.

Çalışmada kullanılan zemin geliştirme mekanizması sayesinde sanal dünyada kullanılacak olan zemin hızlı ve kolay bir şekilde tasarlanabilecektir. Kullanılan çizim tekniği sayesinde aynı noktaların hafızada yer tutması engellenerek bellek daha da verimli kullanılmıştır.

Bu çalışmanın daha verimli hale gelebilmesi için uygun yapay zeka teknikleri içeren bir mekanizma motora ilave edilebilir. Ayrıca akışkanlara ait bir fizik kütüphanesi eklenerek motorun daha gerçekçi sahneler üretmesi sağlanabilir.

ÖZGEÇMİŞ

Kayhan Ayar, 29.03.1982'de Muş'da doğdu. İlk ve orta öğretimini Sakarya da tamamladı. 2000 yılında İstanbul'un Pendik ilçesinde bulunan Rauf Denktaş Lisesi'nden mezun oldu. 2000 yılında başladığı Sakarya Üniversitesi Bilgisayar Mühendisliği Bölümü'nü 2007 yılında bitirdi. 2007 yılında bir dönem ücretli bilgisayar öğretmenliği yaptı. 2008 yılında Sakarya Üniversitesi Bilgisayar ve Bilişim Mühendisliği Anabilim Dalı'nda yüksek lisansa başladı. 2009 Ocak ayında Sakarya Üniversitesi Bilgisayar Mühendisliği bölümünde araştırma görevlisi olarak göreve başladı. Halen aynı görevi sürdürmektedir.