KOCAELİ ÜNİVERSİTESİ * FEN BİLİMLERİ ENSTİTÜSÜ

ÇÖZÜNÜRLÜĞÜ AYARLANABİLİR 3-BOYUTLU NESNE MODELLEMESİ

YÜKSEK LİSANS TEZİ

Elektronik ve Hab. Müh. Özgür ULUÇAY

Anabilim Dalı: Elektronik ve Haberleşme

Danışman: Doç. Dr. Sarp ERTÜRK

Temmuz 2004

KOCAELİ ÜNİVERSİTESİ * FEN BİLİMLERİ ENSTİTÜSÜ

ÇÖZÜNÜRLÜĞÜ AYARLANABİLİR 3-BOYUTLU NESNE MODELLEMESİ

YÜKSEK LİSANS TEZİ

Elektronik ve Haberleşme Müh. Özgür ULUÇAY

Tezin Enstitüye Verildiği Tarih : 6 Temmuz 2004

Tezin Savunulduğu Tarih : 16 Temmuz 2004

TEZ DANIŞMANI	ÜYE	ÜYE
Ç. Dr. Sarp ERTÜRK	Yrd. Doç. Dr. Cabir VURAL	Yrd. Doç. Dr. Mehmet YAKUT
(.....)	(.....)	(.....)

TEMMUZ 2004

ÇÖZÜNÜRLÜĞÜ AYARLANABİLİR 3-BOYUTLU NESNE MODELLEMESİ

Özgür ULUÇAY

Anahtar Kelimeler: 3–Boyutlu Modelleme, Poligon Temelli Ağ, Çok–Çözünürlüklü Model, Ağ Basitleştirme, Artırım, OpenGL.

Özet: Bu tezde, çözünürlüğün ağ basitleştirme yöntemleri kullanılarak ayarlanabildiği üç boyutlu nesne modelleme uygulaması gerçekleştirilmektedir. 3-Boyutlu nesne modellemesi için yaygın olarak kullanılan poligon modeller temel alınarak nesnelerin üç-boyutlu modellerinin gösterimi için poligon yapısından faydalanılmaktadır. Poligonların birleşiminden oluşan ağ örgüsü daha sonra dolgulu, renkli veya doku kaplamalı olarak gösterilebilmektedir. Kullanıcının 3-boyutlu modelle etkileşimini sağlamak üzere döndürme, ışıklandırma, yakınlaştırma ve uzaklaştırma işlemlerine olanak tanınmaktadır. Nesnenin 3-Boyutlu modellinin sahip olduğu poligon ve köşe sayılarının miktarı, çözünürlüğe ve veri miktarına doğru orantılı biçimde etki etmekte, poligon sayısı artınca çözünürlük ile beraber veri miktarı da artmaktadır. Bu çalışmada, çözünürlüğü ve dolayısıyla veri miktarı ayarlanabilir 3-Boyutlu modelleme sayesinde erişim biçimine göre bilgi çözünürlüğünün ayarlanmasına olanak sağlanması amacıyla, yeni bir ağ basitleştirme ve artırım algoritması geliştirilmiştir.

3-DIMENSIONAL (3-D) OBJECT MODELING TECHNIQUE BASED RESOLUTION ADJUSTMENT

Özgür ULUÇAY

Keywords: 3-D Modeling, Polygonal Meshes, Multiresolution Models, Mesh Simplification, Refinement, OpenGL.

Abstract: A 3-Dimensional (3-D) object modeling technique with mesh simplification based resolution adjustment is performed in this thesis. Polygonal models, which are widely utilized for the modeling of 3-D objects, are taken as basis, making use of the polygonal structure and vertex coordinates for the display of 3-D models. The mesh structure obtained by the collection of polygons can be displayed in filled, colored, or texture mapped fashion. Rotation, lighting and zooming functions are supported to enable user interaction with the 3-D model. The amount of polygons and vertices of a model is proportional to the resolution as well as data quantity, resolution and data increase with the number of polygons. In this thesis, we developed a new simplification and refinement algorithm to utilize resolution, and hence data amount, adjustable 3-D modeling so that data resolution can be changed according to access constraints.

ÖNSÖZ VE TEŞEKKÜR

Bilgisayar grafikleri ve ilişkili alanlardaki pek çok uygulama karmaşık poligonsal yüzeye sahip modellerin otomatik olarak basitleştirilmesinden faydalanmaktadır. Uygulamalar, sınırlı kullanılabilir donanım kapasiteleri olmasına rağmen, sıklıkla çok yoğun örneklenmiş yüzey veya modellerle karşı karşıya kalmaktadır. Orijinal modellerin yüksek kaliteli yaklaşıklarını hızlı biçimde üretebilen etkili bir algoritma, veri karmaşıklığını yönetmek için değerlendirilebilir bir araç olarak kullanılabilir.

Bu çalışmada, çözünürlüğü ve dolayısıyla veri miktarı ayarlanabilir 3-Boyutlu modelleme sayesinde erişim biçimine göre bilgi çözünürlüğünün ayarlanmasına olanak sağlanması amacıyla, yeni bir ağ basitleştirme ve artırım algoritması geliştirilmiştir. Geliştirilen ağ basitleştirme tekniği sayesinde modelin çözünürlüğü önemli bölgelerinin modelin diğer bölgelerine nazaran basitleştirme işleminden daha az etkilenmesi ve modelde farklı çözünürlüklü bölgeler elde edilmesi amacı ile geliştirmiş olduğumuz algoritma bu hedefleri temel alarak gerçekleştirilmiştir.

Yapılan çalışmanın, ülkemizde 3-Boyutlu modelleme, canlandırma ve bilgisayar grafikleri alanlarındaki uygulamaların yaygınlaşmasına katkısı olmasını dilerim.

Beni bu güne kadar yetiştiren, büyüten ve hiçbir zaman desteklerini esirgemeyen aileme ve bana bu konuda çalışma olanağı veren ve bu çalışma boyunca bana vermiş olduğu fikir, yorum ve önerileri ile destekleyen sayın Doç. Dr. Sarp ERTÜRK'e minnettarlığımı sunarım.

Bu çalışma 2004K120720 sayı ve "E-devlet için bilgisayar destekli görsel dokümantasyon, arşiv ve yönetim sistemi gerçekleştirilmesi" isimli DPT projesi kapsamında desteklenmiştir.

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

TABLOLAR DİZİNİ

1. GİRİŞ

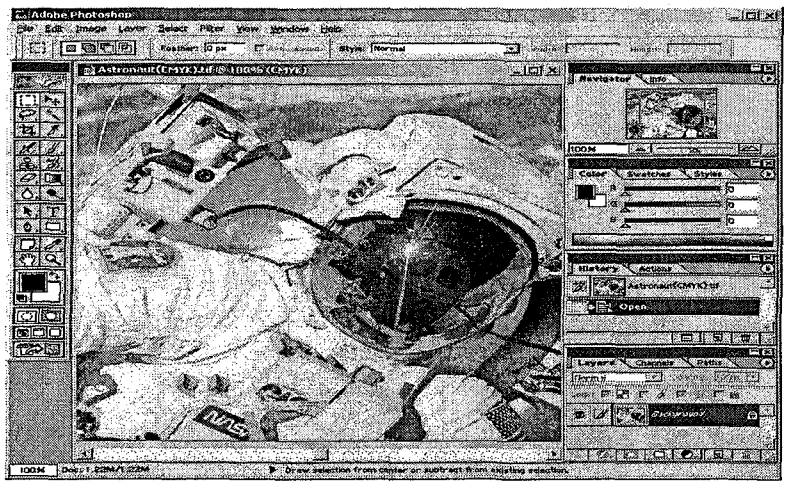
Bilgisayar grafikleri, bilgisayarların ortaya çıkışının hemen ardından yazılı çıktı çizicilerinde (line plotter) ve katot ışınlı tüp ekranlarında (CRT: Cathode Ray Tube) verilerin görüntülenmesi ile başlamıştır. Bu, nesnelerin modellerinin ve imgelerinin oluşturulması, saklanması ve düzenlenmesini içerecek şekilde genişlemiştir. Bu modeller, fiziksel, matematiksel, mühendislik, mimarisel ve hatta kavramsal yapıları içeren farklı alanlardan gelmektedir. Bilgisayar grafikleri günümüzde yaygın şekilde etkileşimli olarak karşımıza çıkmaktadır. Kullanıcılar, klavye, fare ve dokunmaya duyarlı ekranlar gibi giriş cihazlarını kullanarak nesnelerin ve görüntülenen imgelerinin içeriklerini, yapılarını ve görünüşlerini etkileşimli olarak kontrol etmektedir. Giriş cihazları ile ekran arasındaki bu yakın ilişki nedeniyle bilgisayar grafikleri çalışması bu tip giriş cihazlarını konu olarak almaktadır [1].

1980'lerin başlarına dek bilgisayar grafiklerinin küçük, belirli bir alanda kalmasının nedeni o zamana kadar donanımın çok pahalı olması ve kolay kullanılabilir, maliyeti elverişli grafik tabanlı uygulama programlarının çok az olmasıdır. Daha sonraları, Xerox Star gibi taramalı grafik ekranı yerleşik kişisel bilgisayarlar ve daha sonra seri üretilen ve daha ucuz Apple Macintosh ve IBM PC'ler ve bunların kopyaları kullanıcı-bilgisayar etkileşimi için bit haritası (bitmap) kullanımlarıyla yaygınlaşmıştır. Bir bit haritası, ekran üzerindeki noktaların (Picture elements: "pixel" veya "pel" olarak adlandırılır) dikdörtgensel dizisinin birler ve sıfırlarla gösterimidir. Grafik tabanlı kullanıcı ara yüzleri, milyonlarca yeni kullanıcıya, elektronik çizelgeler, kelime işlemciler ve çizim programları gibi düşük maliyetli uygulama programlarını basit kontrol imkânı sağlayarak sunmaktadır [1]. Masaüstü kavramı ekran uzayını organize etmek için popüler bir benzetme olmuştur. Pencere yöneticisinin (window manager) anlamı, pencere adı verilen, her biri uygulama üzerinde koşan sanal grafik terminalleri rolünü üstlenen dikdörtgensel ekran alanlarını oluşturabilir, konumlandırabilir ve yeniden boyutlandırabilir olmasıdır. Bu, kullanıcılara çoklu aktiviteler arasında tipik olarak bir fare ile istenen pencere seçilerek geçiş yapılmasına izin verir. Dağınık bir masa üzerindeki kâğıt parçalarına

benzer şekilde pencereler keyfi biçimde ekranda örtüştürülebilir. Ayrıca bu masa üstü benzetmesinin bir parçası veri dosyaları ve uygulama programları olmaktadır. Diğer bir benzetme ise gerçek-yaşam karşılıklarının bilgisayar-işlem eşdeğerleri gerçekleştiren dosya kabinleri, posta kutuları, yazıcılar, çöp kutuları gibi ortak ofis objelerini simgeleyen simgelerin görüntülenmesidir. Nesnelerin "seçme ve tıklama" ile doğrudan işlenmesi eski işletim sistemleri ve bilgisayar programlarında kullanılan gizli komutların girilmesinin yerini almıştır. Böylece, kullanıcılar karşılık gelen programları veya nesneleri etkinleştirmek için simgelerini seçebilmekte veya aşağı-çekilebilen veya açılan ekran menüler üzerindeki butonları seçerek seçimlerini yapabilmektedir. Günümüzde, hemen hemen tüm etkileşimli uygulama programları ve hatta metin (örn. kelime işlemciler) veya sayısal veri (örn. elektronik çizelge) işleyen programlar dahi kullanıcı ara yüzlerinde uygulamaya özgü nesneleri işlemek ve görüntülemek için kapsamlı biçimde grafikler kullanmaktadır. Taramalı ekranlar (bit haritaları kullanan ekranlar) üzerinden grafiksel etkileşim tamamen abc sayısal terminaller üzerinden abc sayısal metinsel etkileşimin yerini almıştır [1].

Bilgisayar kullanmayan insanlar dahi günlük çalışmalarında televizyon reklâmlarında ve sinematik özel etkilemeler (effect) şeklinde bilgisayar grafikleri ile karşılaşmaktadır. Bilgisayar grafikleri tüm bilgisayar kullanıcı ara yüzlerinin tümleşik parçası olarak 2 boyutlu (2-D), 3 boyutlu (3-D) ve daha yüksek boyutlu nesneleri canlandırmak için vazgeçilmezdir: eğitim, bilim, sanat, tıp, endüstri, iş, hükümet, ticaret, askeri, reklâmcılık ve eğlence gibi farklı alanların tümü bilgisayar grafiklerine ihtiyaç duymaktadırlar. Uygulamaların listesi çok fazla olmakla beraber ticari ürün olarak grafik yetenekleri artan bilgisayar sayesinde bu liste daha hızlı biçimde artacak ve genişleyecektir. Şimdi bu alanlara kısaca değinelim.

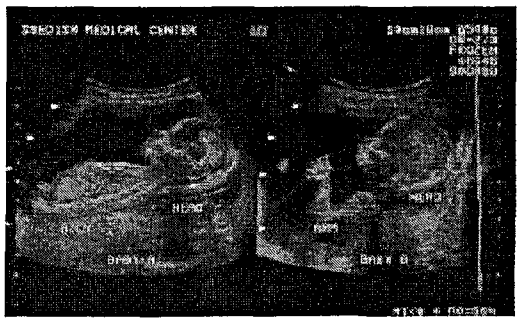
Kullanıcı ara yüzleri: Kişisel bilgisayar ve iş istasyonları üzerinde koşan çoğu program, çoklu eşzamanlı aktiviteleri yönetmek ve kullanıcılara seçme ve tıklama özellikleri sayesinde ekran üzerindeki nesneleri, menü öğelerini ve simgeleri seçmelerine izin vermek için masaüstü pencere sistemine dayanan grafiksel kullanıcı ara yüzü (GUI) içerir. Şekil 1.1'de de bir örneği gösterilen kullanıcı ara yüzleri, bilgisayar grafiklerini kullanır.



Şekil 1.1. Kullanıcı arayüzü.

- İş, Bilim ve Teknolojide Çizim (Etkileşimli): Günümüzün en çok kullanılan ikinci yaygın kullanım muhtemelen matematiksel, fiziksel ve ekonomik fonksiyonların 2-D ve 3-D grafiklerinin oluşturulmasıdır: histogramlar, çubuk ve çember grafikler (bar and pie charts), görev-zamanlama grafikleri, stok ve ürün grafikleri ve benzerleri gibi.

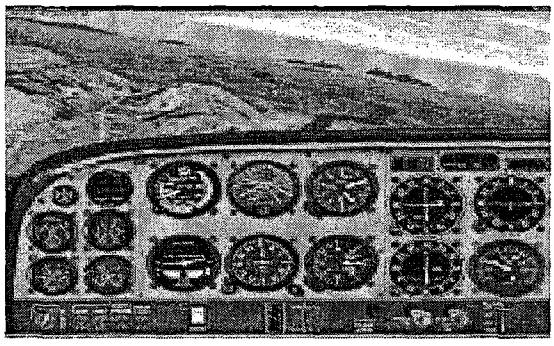
- Tıpta Görüntüleme: Şekil 1.2'de bir örneği verilen bilgisayar grafikleri cerrahinin planlanması ve işletilmesi aşamalarında yaygın biçimde kullanılmaktadır.



Şekil 1.2. Tıp alanında bilimsel görüntüleme.

- Bilimsel Canlandırma ve Eğlence İçin Simülasyon ve Canlandırma: Bilgisayarla üretilen canlandırma (animated) filmleri ve gerçek ve simüle edilmiş nesnelerin zaman-değişim davranışlarının görüntülenmesi bilimsel ve mühendislik canlandırmalarında artan boyutlarda yaygınlaşmaktadır. Şekil

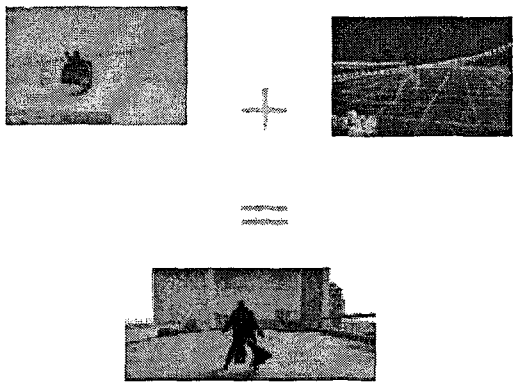
1.3'te bilgisayar grafiklerinin kullanıldığı bir simülasyon örneği verilmektedir.



Şekil 1.3. Simülasyon örneği.

- Sanat ve Ticaret: Müze, dağıtım terminalleri, süper marketler, oteller gibi kamusal alanlardaki ve özel evlerdeki kişisel bilgisayarlar ve teleteks ve videoteks terminalleri kullanıcıları yönlendirmek, seçim yapmalarını sağlamak üzere bilgisayar grafiklerinden faydalanmaktadır.

- Bilgisayar Destekli Tasarım (CAD): Bilgisayar destekli tasarımda binaya benzer yapılar, otomobil gövdeleri, uçak ve gemi omurgaları, çok yüksek ölçekli tümleşik devreler (VLSI chips), optik sistemler ve telefon ve bilgisayar ağlarını içeren mekanik, elektriksel, elektromekanik ve elektronik cihazları tasarlamak için etkileşimli bilgisayar grafikleri kullanılmaktadır. Şekil 1.4'te bilgisayar destekli tasarımda kullanılan bilgisayar grafiklerinin kullanıldığı bir örnek görülmektedir.



Şekil 1.4. Sanat ve tasarım.

- Haritacılık: Ölçülen veriler ile coğrafik ve diğer doğal olayların (phenomena) sağlıklı ve sistemli gösterimlerini üretmek için bilgisayar grafikleri kullanılmaktadır. Coğrafik haritalar, yükseklikleri gösteren (relief) haritalar, ısı haritaları, tesviye eğrili (contour) haritalar ve nüfus-yoğunluk haritaları örnek olarak gösterilebilir.

- Çoklu ortam sistemleri: Bilgisayar grafikleri bu alanda tamamen kritik bir rol üstlenmektedir.

Bilgisayar grafik sistemleri, bilgisayar destekli tasarım, nesne tanımlama, nesne takibi ve modele dayalı video kodlama gibi birçok değişik uygulama nesnelerin en önemli özelliklerinden birisi olan nesne şeklinden faydalanmaktadır. Nesne şekillerinin tanımlanması ve temsil edilmesi işlemi genelde 'modelleme' olarak adlandırılmaktadır. Nesne şekillerinin tanımlanması için birçok değişik yöntem vardır ve her yöntemin kendine özgü üstün ve eksik yönleri mevcuttur.

Son yıllarda özellikle eğlence sektörünün önderliğinde 3-Boyutlu (3-D) nesne modelleme yöntem ve uygulamaları önemli gelişmeler göstermiştir. Bununla beraber nesnenin gerçek şekliyle birebir örtüşen modellerin elde edilmesi hala zor ve karmaşık bir işlem olup, çoğu durumda doğru bir modelin elde edilebilmesi için özel tarayıcı donanım gerektirmektedir. Özellikle nesnenin şekli karmaşıklaştıkça, 3-Boyutlu modelin oluşturulması zorlaşmakta ve model verisi ciddi oranda artmaktadır. 3-Boyutlu model veri miktarının yüksek olması, depolanması ve işlenmesi gereken veriyi arttırmakta ve modellerin animasyonunu da benzer şekilde karmaşıklaştırmaktadır.

Değişik modelleme yöntem ve uygulamaları var olmakla beraber en yaygın kullanılan yöntem poligon yapı kullanan modellemedir. Poligon temelli ağ yapısı (polygonal meshes) şeklinde tanımlanan 3-B modeller yaygın kullanılmakta ve işleme açısından basit bir şekil tasviri imkanı sağlamaktadır. Özellikle etkileşimli üç boyutlu grafik uygulamaları için kolay kullanım sağlayan poligon modelleri, modelleme kesinliğinin değişik oranlarda ayarlanabilmesi ve değişik biçimleri kolayca tanımlayabilme kabiliyeti sayesinde yaygın kabul görmektedir. Geleneksel olarak, bu tip modeller, bir detay seviyesi tanımlayan sabit poligon kümelerinden oluşmuştur. Fakat bu tek tip detay seviyesi bu model tipi kullanıldığı zaman çoğu kez

farklı içerikler için kötü sonuçlar verebilmektedir ve çoğu zaman farklı detay seviyelerinde modellere ihtiyaç duyulmaktadır.

Poligon modelleri ağ yapısı içerisinde, köşeler, kenarlar, yüzler, ve bunlar arasındaki topolojik ilişkilerin bir kümesini içerir. Bu ilişkileri baz alarak, bir veri yapısı, her elemanın nasıl saklanacağını, ve ihtiyaç duyulduğunda komşularına nasıl referans edileceğini belirtir. Poligon yapıları temel olarak yüz-temelli (face-based) ve kenar-temelli (edge-based) olmak üzere iki farklı veri yapısını desteklemektedir. Yüz-temelli poligon yapıları, her yüzün köşe noktalarına ve komşu yüzlere olan işaretçisi ile saklama yapar. Kenar-temelli poligon yapıları ise, köşeler ve komşu kenarlar olmak üzere her kenar için işaretçileri saklamaktadır.

Bu çalışmanın odak noktası, 3-Boyutlu modellerin çözünürlüğünün ayarlanması amacıyla, geliştirmiş olduğumuz otomatik basitleştirme ve artırım algoritmalarıdır. Basitleştirme ve artırım algoritmaları sayesinde yüksek seviyede detay içeren, yoğun biçimde örneklenmiş poligonsal yüzey modellerinden, çok daha az sayıda poligon içeren, orijinaline sadık model yaklaşıklıklarının elde edilmesi ve orijinal modelin, ek verilerle oluşturulabilmesi sağlanmıştır.

Önerilen çözünürlüğü ve dolayısıyla veri miktarı ayarlanabilir 3-Boyutlu modelleme sayesinde erişim biçimine göre bilgi çözünürlüğünün ayarlanmasına olanak sağlanmaktadır. Bu amaçla Visual C++ programı ve OpenGL (Open Graphics Library) grafik kütüphaneleri kullanılarak esneklik, verimlilik, ve kullanım kolaylığı olarak belirlenen üç önemli tasarım hedefi ile gerçekleştirilen etkileşimli 3-Boyutlu görüntüleme sistemi kullanılmaktadır.

1.1 Üç Boyutlu Modelleme

1.1.1 Model nedir?

Bir model, somut bir varlığın bütününe nazaran bazı özelliklerinin temsili veya özet varlığıdır. Bir varlığın modelini çıkarmaktaki amaç, insanların bu varlığın yapısı ve davranışını anlamasını ve gözünde canlandırmasını, modele olan giriş ve çıkışların etkilerini tahmin etmelerini ve "deneyleme" için elverişli bir araç sağlamaktır. Nicel

modeller fiziksel ve sosyal bilimlerde ve mühendislikte ortak olarak eşitlik sistemleriyle ifade edilmekte ve modelleyici bağımsız değişkenler, katsayılar ve üs değerlerini değiştirerek tecrübe etmektedir. Çoğu kez, modeller, modellenmiş varlığın gerçek yapısını ve davranışını basitleştirerek, modelin daha kolay biçimde gözde canlandırılmasını veya bu modeller için eşitlik sistemleri kullanılarak modelin sayısal olarak daha kolay işlenmesini sağlamaktadır [1].

Grafikler, modeli oluşturmak ve düzenlemek, parametre değerlerini ortaya çıkarmak, yapısını ve davranışını canlandırmak üzere kullanılabilir. Model ile grafiksel anlamı, onu oluşturma ve gözünde canlandırma için belirgin biçimde farklıdır; nüfus modelleri gibi modeller herhangi bir içsel grafiksel görünüşe ihtiyaç duymazlar. Bilgisayar grafiklerinin kullanıldığı en temel model tipleri aşağıda ifade edilmiştir [1].

- Örgütsel modeller, kütüphane sınıflandırma planları ve biyolojik sınıflandırmalar gibi kurumsal bürokrasileri ve sınıflandırmaları gösteren sıradüzenlerdir.

- Nicel modeller, ekonometri, mali, sosyolojik, demografik, iklimsel, kimyasal, fiziksel ve matematiksel sistemleri açıklayan eşitliklerdir. Bunlar daha çok grafikler ve istatistiksel çizimlerdir.

- Geometrik modeller, iyi-tanımlanmış geometri ile açıklanan bileşen derlemeleri ve çoğu zaman mühendislik ve mimari yapılar, moleküller ve diğer kimyasal yapılar, coğrafik yapılar ve araçları içeren bu bileşenler arasındaki ara bağlantılarıdır. Bu tür modeller, genellikle blok diyagramlar veya sözde-gerçekçi "sentetik fotoğraflar" ile tanımlanır.

1.1.2 Geometrik modeller

Geometrik veya grafiksel modeller içsel geometrik özellikleri ile bileşenleri ifade ederler. Bir geometrik model aşağıdaki bileşenlerle ifade edilebilir[1].

- Bileşenlerin biçimi ve uzamsal yerleşimi (örn. varlığın geometrisi), ve renk gibi bileşenlerin görünümünü etkileyen diğer öznitelikler.

- Bileşenlerin bağlantısallığı (örn. varlığın yapısı veya topolojisi): bağlantısallık bilgisi özet biçiminde belirtilebilir (ağların komşuluk matrisi içerisinde veya sıradüzen için bir ağaç yapısı içerisinde olduğu söylenebilir), veya kendi içsel geometrisine sahip olabilir (örneğin tümleşik bir devre içerisindeki kanal boyutları).

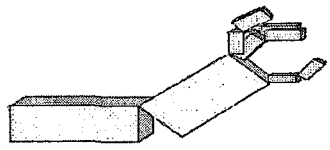
- Elektriksel karakteristikler veya tanımlayıcı metin gibi uygulamaya-özgü bileşenlerle ilişkili veri değerleri ve özellikleri.

Ayrık devre modelleri için lineer devre analizleri, mekanik yapılar için sonlu-eleman analizleri, moleküler modeller için enerji azaltma gibi model ile ilişkilendirilmiş işleme algoritmaları olabilir. Model içerisine açıkça ne yüklendiği ile analiz etme ve göstermeden önce ne işlenmesi gerektiği arasında "klasik uzay-zaman" ödünleşimi vardır. Örneğin bir bilgisayar ağı modeli, bağlantı hatlarını açık biçimde yüklemelidir veya her yeni görüntüleme isteğinde basit bir grafik-yerleşim planı algoritması ile bağlantısallık matrisinden bunları hesaplamalıdır. Analiz ve görüntülemeye izin vermek için model ile birlikte yeterli miktarda bilginin saklanması gerekir, fakat hatasız format ve kodlama tekniklerinin seçimi uygulamaya ve uzay-zaman ödünleşimine bağlıdır [1].

1.1.3 Geometrik modellerde sıra-düzen

Geometrik modeller genellikle aşağıdan-yukarıya yapılandırma sonucu sıra-düzensel bir yapıya sahiptirler. Bileşenler yüksek-seviyeli varlıkları oluşturmak için yapı blokları olarak kullanılır, ve sırasıyla yüksek-seviyeli varlıklara ulaşılıncaya dek yapı blokları vazifesini görürler. Nesne sıra-düzenleri çok yaygın bir kullanıma sahiptir, çünkü her biri tek parça (monolithic) olan az sayıda bütünlükten oluşurlar; bir kez bir bütünlüğü parçalar kümesine ayrıştırdığımızda, en az iki seviyeli bir sıra-düzen elde etmiş oluruz. Yaygın olmayan biçimde, bir üst seviyeli nesne bir alt seviyeli

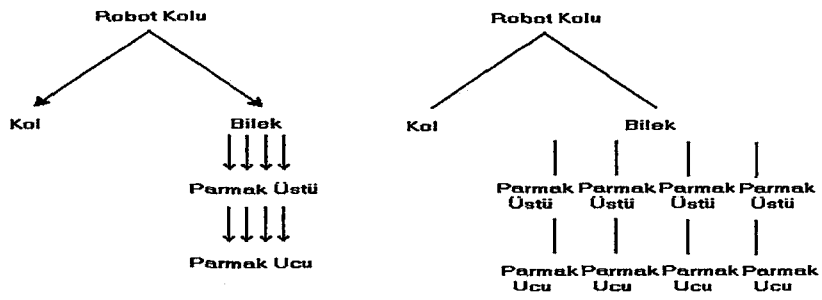
nesneyi yalnızca bir kez içerir; nesneleri bir düğüm olarak ve nesneler arası kapsama ilişkilerini kenarlar olacak biçimde, sıra-düzen, bir ağaç şeklinde sembolize edilebilir. Yaygın olarak kullanımda, bir üst seviyeli nesne bir alt seviyeli objeyi pek çok kez içerebilir ve sıra-düzen bir yönlendirilmiş çevrimsiz grafik (DAG: Directed Acyclic Graph) ile sembolize edilir [1].



Şekil 1.5. Robot kolu ve parmakları'nın perspektif görüntüsü.

Nesne sıra-düzenlerine bir örnek olarak Şekil 1.5'de "Robot Kolu " nesnesinin perspektif bir görüntüsü gösterilmiştir. Şekil 1.6(a)'da robot kolu yapısı DAG olarak ifade edilmektedir. DAG'dan ağaca dönüşüm yaparken DAG'da çoklu kapsanan objeleri ağaçta çoklamak gerekir. Şekil 1.6(b)'de robot kolu yapısı ağaç olarak ifade edilmiştir. Kural gereği sola doğru olan oklar yedek olarak gösterilir, çünkü düğümler arası sıralı ilişkiler, ağaç içerisinde düğümlerin bağıl pozisyonlarıyla gösterilmektedir. Eğer A düğümü B düğümünün üstünde gösterilmişse A, B' yi kapsar denir.

Örnekteki robot kolu, taban olarak bir kolu içermektedir. Kol, dikey eksende hareket edebilen bilek olarak tabir edilebilecek bir parça içermektedir. 4 adet aynı özellikte parmak üstü parçaları bileğe eklenmiş ve parmak ucu olarak ifade edilebilecek 4 adet parçanın her biri de parmak üstü parçalarına eklenmiştir. Parmak üstü ve parmak ucu parçaları dikey eksende hareket edebilmektedir.



(a) Yönlendirilmiş Çevrimsiz Grafik(DAG) (b) Ağaç

Şekil 1.6. Robot kolu parçalarının sıra-düzeni.

Sıra-düzen içerisindeki nesne, geometrik temel öğeler içermesine rağmen düşük-seviye alt nesneleri de kapsamaktadır, DAG ve ağaç, robot kolu yapısını gösterirken yalnızca alt nesnelere olan referansları göstermektedirler. Bu gösterim, yüksek-seviye yordam dilindeki bir programın çağrı yapısını göstermek için yaygın biçimde kullanılan yordam–sıra düzen çizeneğine paralellik gösterir. Karmaşık bir nesnenin nasıl bir sıra-düzende yapılandırılacağına karar vermek tasarımcıya bağlıdır.

Bilgisayar ağları veya kimyasal işletmeler gibi pek çok sistemi ağ çizenekleri ile gösterebilir, bu nesneleri çizenek içerisinde çok kez gösterebilir ve birbirleri ile olan bağlantılarını da keyfi biçimde yapılandırabiliriz. Ayrıca bu tür ağları çevrimler içermesi durumunda dahi grafik olarak modelleyebiliriz, fakat ağın alt-ağları çok kez gösterilmek istendiğinde nesne-içi sıra-düzen özelliklerini sergileyebilmelidir [1].

Karmaşık nesnelerin ve bunların modellerinin yapılandırma görevini basitleştirmek için, yaygın biçimde temel yapı blokları olarak uygulamaya-özgü atomik bileşenler kullanılır. 2-Boyutta, bu bileşenler genellikle, standart sembolik şekillerin (semboller veya şablonlar olarak da adlandırılır) plastik veya bilgisayar-çizim şablonları kullanılarak çizilir. Çizim programlarında bu şekiller sırasıyla çizgiler, üçgenler, dikdörtgenler, poligonlar, elips yaylar gibi temel geometrik şekillerden oluşmaktadır. 3-Boyutta, silindirler, paralelyüzler, küreler, piramitler ve dolanım yüzeyleri gibi şekiller, temel yapı blokları olarak kullanılır. Bu 3-boyutlu şekiller, 3-B poligonlar gibi düşük-seviye geometrik temel öğe terimleri ile tanımlanabilir. Bu durumda, pürüzsüz eğrisel şekiller, beraberinde çözünürlük kaybı getirecek poligonsal eşlerine yaklaşıklaştırılmalıdır. Alternatif olarak, gelişmiş modelleme sistemleri, çözünürlük kaybına neden olmaksızın, bağımsız biçimler veya hacimler, parametrik çokterimli yüzeyler gibi şekiller, silindire benzer katılar, küreler ve koniler gibi doğrudan kendileri tamamen temel öğe olan ve analitik olarak ifade edilen öğeler ile ilgilenir. Bu 2-B veya 3-B bileşenler, kendi modelleme koordinat sistemlerinde geometrik temel öğeler ve düşük-seviye nesneler olarak tanımlanır ve geometrik verilerinin yanında ilişkili uygulama verilerini de içerirler. Bir nesne, böylece, bir (bileşik) biçim ve kendisinin bütün verilerini temsil eder [1].

2. EĞRİ VE YÜZEYLERİN İFADE EDİLMESİ

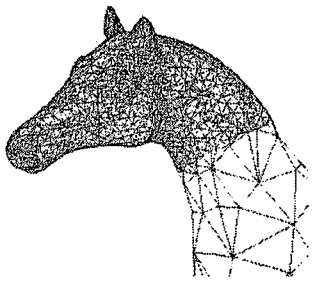
Pek çok bilgisayar grafik uygulamasında, düzgün eğri ve yüzeyler üretilmek zorundadır. Çoğu gerçek-dünya nesneleri doğal olarak pürüzsüz olup bilgisayar grafikleri, gerçek dünyayı modellemeyi gerektirir. Bilgisayar destekli tasarım (CAD), yüksek-kalite karakter yazıyüzleri, veri çizimleri ve sanatçı eksizlerinin tümü düzgün eğriler ve yüzeyler içermektedir. Animasyon dizisindeki bir kamera veya nesne yolu hemen hemen daima pürüzsüzdür; benzer şekilde keskinlik veya renk uzayına doğru olan yol da pürüzsüz olmak zorundadır [1]. Eğri ve yüzeyleri ifade etme ihtiyacı iki durum için doğmaktadır;

- Mevcut olan nesneleri (otomobil, yüz, dağ...) modelleme.
- Mevcut olmayan nesneleri modelleme.

İlk durum için, nesnenin matematiksel açıklaması elde edilemeyebilir. Elbette birisi nesne üzerindeki sonsuz sayıdaki nokta koordinatlarını model olarak alabilir, fakat sonlu yedekleme kapasitesine sahip olan bilgisayarlar ile bu imkânsızdır. Daha sık olarak, biz yalnızca nesneyi matematiksel olarak daha kolay açıklayabilen düzlemler, küreler ve diğer şekillerin bir kümesine yaklaşıklaştırabiliriz. Model üzerindeki bu noktalar, gerçek nesne üzerinde karşılık gelen noktalara yakın olmayı gerektirir [1].

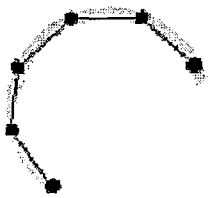
İkinci olarak, modellenecek gerçek bir nesne olmadığı durumda ise, kullanıcı nesneyi modelleme sürecinde meydana getirir; böylece nesne, kendi gösterimi ile tamamen uyuşmaktadır, çünkü nesnenin kendisi sadece bir gösterimdir. Nesneyi oluşturmak için, kullanıcı nesneyi yontabilir, matematiksel olarak açıklayabilir veya bazı programlar ile yaklaşık tanımını çıkarabilir. Örneğin bilgisayar-destekli tasarımda (CAD) bilgisayar gösterimi, özet biçiminde tasarlanmış nesneyi daha sonra fiziksel olarak gerçekleştirmek için kullanılır [1].

Bu bölümde yüzey modellemenin genel alanı incelenecektir. Bu alan biraz geniştir ve sadece 3-B yüzeylerin bizim için en-önemli gösterimi sunulacaktır: poligon temelli ağ yüzeyleri (polygon mesh surfaces).



Şekil 2.1. Poligonlarla gösterilen 3-B bir nesne.

Bir poligon temelli ağ, birbirine poligonlar ile bağlanmış düzlemsel yüzeylerin bir kümesidir. Açık kutular, kabinler ve bina çıkışları kolaylıkla ve doğal olarak poligon ağlarıyla gösterilebilir. Poligon temelli ağlar, eğrisel yüzeylerde de daha az kolaylıkla kullanılabilir, fakat Şekil 2.1'de olduğu gibi gösterim yalnızca bir yaklaşıktır. Şekil 2.2 eğrisel bir şeklin bir kesitini ve onun poligon ağı temsilini göstermektedir. Gösterimdeki açık biçimde görülen hataları azaltmak ve daha iyi bir doğrusal yaklaşıklık oluşturmak için daha fazla poligon kullanılabilir, fakat bu da hafıza gereksinimini ve gösterimi işleyecek olan algoritmaların işleme zamanını artırır. Ayrıca, eğer imge büyütülürse düz kenarlar yeniden açık biçimde görülür. Bu probleme genel bir ifade ile geometrik örtüşme (aliasing) denilmektedir [1].



Şekil 2.2. Eğrisel bir nesnenin kesiti ve poligonsal gösterimi.

2.1. Poligon Temelli Ağlar

Bir poligon temelli ağ, her kenar en çok iki poligon tarafından paylaşılacak şekilde kenarlar (edges), köşe noktaları (vertex), ve poligon (polygon) bağlantılarının bir

derlemesidir. Bir kenar, iki köşe noktasını bağlar ve bir poligon, kapalı bir kenar zinciridir. Bir kenar, iki komşu poligon tarafından paylaşılabilir ve bir köşe noktası en az iki kenar tarafından paylaşılır. Bir poligon ağı, her birinin birbirine nazaran avantaj ve dezavantajları bulunan farklı yollarla gösterilebilir. Uygulama programlayıcısının görevi en uygun gösterimi seçmektir. Bir uygulama içerisinde çeşitli gösterimler kullanılabilir: biri harici yedekleme için, diğeri dahili kullanım için, ve bir diğeri de kullanıcının etkileşimi ile oluşturacağı ağ [1].

Farklı gösterimleri değerlendirmek için iki temel ölçüt olan "uzay" ve "zaman" kullanılabilir. Bir poligon temelli ağ üzerindeki tipik işlemler; bir köşe noktasını kapsayan tüm kenarları bulmak, bir kenarı veya bir köşe noktasını paylaşan poligonları bulmak, bir kenar ile bağlanmış köşe noktalarını bulmak, bir poligonun kenarlarını bulmak, ağı görüntülemek, gösterimdeki hataları saptamak (örneğin atlanan bir kenar, bir köşe noktası veya bir poligon) olarak sıralanabilir. Genelde, poligonlar, kenarlar ve köşe noktaları arasındaki ilişkiler açıkça ifade edilmeli, işlemler hızlı olmalı ve gösterim ihtiyacı için fazlaca alan ayrılmalıdır. Woo [2] bir poligon temelli ağ veri yapısı üzerinde dokuz temel erişim işlemi ve dokuz temel güncelleme işleminin zaman karmaşıklığını analiz edebilmektedir.

Bu bölümün diğer aşamalarında, poligon temelli ağ yapıları ile ilişkili çeşitli kavramlar üzerinde durulacaktır: poligon temelli ağların gösterimi, verilen gösterimin doğruluğunu saptama, bir poligonun bulunduğu düzlemin katsayılarını hesaplama gibi çeşitli kavramlar üzerinde durulacaktır.

2.1.1. Poligon temelli ağların gösterimi

Bu kısımda, farklı üç poligon temelli gösterimi inceleyeceğiz: açık gösterim, köşe noktaları listelerine işaretçiler gösterimi ve kenar listelerine işaretçiler gösterimi.

Açık gösterimde, her poligon köşe nokta koordinatları ile gösterilir [1]:

$$P = ((x_1,y_1,z_1),(x_2,y_2,z_2),\ldots,(x_n,y_n,z_n)).$$

Köşe noktaları, poligon üzerinde dolanırken karşılık geldikleri sıra ile yüklenir. Kenarlar, listedeki ardışık gelen köşe noktaları arasında ve son köşe noktası ile ilk köşe noktası arasında bulunmaktadır. Tek bir poligon için, bu gösterim hafıza-verimlidir; fakat poligon temelli bir ağ için, paylaşılan köşe noktaları kopyalandığı için bir hayli hafıza kaybına yol açmaktadır. Hatta daha da kötüsü paylaşılan köşe noktaları ve kenarlar için açık bir ifade bulunmamaktadır. Örneğin, bir köşe noktasını ve ona bağlı olan tüm kenarları etkileşimli olarak sürüklemek istediğimizde, bu köşeye bağlı olan tüm poligonları bulma zorunluluğu doğar. Bu işlem, bir poligondaki koordinat üçlüsünü bütün diğer poligon üçlüleri ile karşılaştırmayı gerektirir. Bunu yapmanın en iyi yolu bütün N koordinat üçlülerini sıralayarak yapılabilir ve en iyi durumda "Nlog2N" işlem sürer ve hatta işlemsel yuvarlamalardan kaynaklanan hatalar nedeniyle aynı köşe değeri her poligon için az da olsa farklı koordinat değerine sahip olabilme tehlikesini doğurur. Sonuç olarak doğru bir yakalama belki de hiç bir zaman yapılamayabilir.

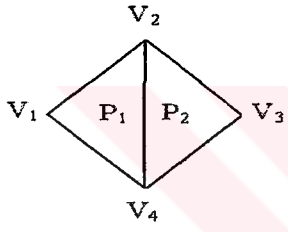
Açık gösterim ile ağ yapısı dolgulu poligonlar veya poligon ana çizgileri ile görüntülenebilirken, her köşe noktasını dönüştürme ve her poligonun bir kenarını kırpma işlemini gerektirir. Eğer kenarlar çizdirilirse, her paylaşılan kenar iki kez çizilecektir; bu kalemli yazıcılarda, film kaydedicilerinde ve vektör görüntüleyicilerinde, üstüne yazma nedeniyle problemlere yol açacaktır. Bir problem de taramalı görüntüleyicilerde kenarların ters doğrultularda çizilmesiyle yaşanabilir. Bu durumda ekstra pikseller kullanılmak zorunda olabilir.

Bir köşe noktaları listesine işaretçiler ile tanımlanan poligonlar, poligon ağı içerisinde V köşe noktaları listesinde "V = ((x1,y1,z1),...,(xn,yn,zn))" her köşe noktasını yalnız bir kez içerir. Bir poligon, köşe noktaları listesine işaret eden işaretçilerin veya indislerin bir listesi ile tanımlanır. Bir poligon, köşe noktaları listesindeki 3, 5, 7 ve 10 köşe noktalarından oluşuyorsa, "P = (3, 5, 7, 10)" olarak ifade edilir.

Bir örneği Şekil 2.3'te verilen bu gösterimin açık poligon gösterimine göre çeşitli avantajları vardır. Her köşe noktası yalnızca bir kez yüklendiğinden dikkate değer bir oranda hafızadan tasarruf sağlanır. Ayrıca, köşe noktasının koordinatları rahatlıkla değiştirilebilir. Diğer taraftan, bir kenarı paylaşan poligonları bulmak hala zor bir

işlem olmakta ve tüm poligon ana çizgileri çizdirilirken paylaşılan poligon kenarları iki kez çizilmektedir. Bu iki problem aşağıdaki yöntemle tüm kenarları açık biçimde göstererek giderilebilecektir.

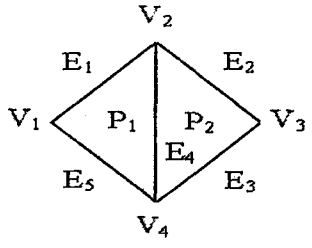
Bir kenar listesine işaret eden işaretçilerle poligonları tanımlarken yine bir V köşe noktaları listesine sahip olunmakta, fakat poligonları gösterirken kullanılan köşe noktaları listesine işaretçilerle işaret etmenin aksine her kenarın yalnızca bir kez belirtildiği bir kenar listesine işaret eden işaretçiler kullanılmaktadır. Böylece, P poligonu "P = (E1,E2,...,En)" ile, bir kenar ise "E = (V1,V2,P1,P2)" şeklinde gösterilmektedir. Bir kenar yalnızca bir poligona ait olduğunda P1 veya P2'den birisi boş (null) olacaktır. Şekil 2.4 bu tip bir gösterim için bir örnek göstermektedir.



$V = (V_1,V_2,V_3,V_4) = ((x_1,y_1,z_1),...,(x_4,y_4,z_4))$
$P_1 = (1, 2, 4)$
$P_2 = (4, 2, 3)$

Şekil 2.3. Köşe noktaları listesine işaret eden indislerle tanımlanmış poligon temelli ağ.

Poligon ana çizgileri tüm poligonları göstermenin aksine tüm kenarları göstermektedir; bu şekilde artık kırpma (redundant clipping), dönüşüm (transformation), ve tarama konuşmasından (scan conversation) kaçınılmaktadır. Dolgulu poligonlar da kolayca görüntülenmektedir. 3-B peteğimsi bir metal levha yapısının ifade edildiği bazı durumlarda, bazı kenarlar üç poligon tarafından paylaşılmaktadır. Bu durumlarda kenar ifadeleri keyfi sayıda poligon içerecek şekilde genişletilebilir: "E = (V1,V2,P1,P2,...,Pn)".



$V = (V_1,V_2,V_3,V_4) = ((x_1,y_1,z_1),...,(x_4,y_4,z_4))$
$E_1 = (V_1,V_2,P_1,\lambda)$
$E_2 = (V_2,V_3,P_2,\lambda)$
$E_3 = (V_3,V_4,P_2,\lambda)$
$E_4 = (V_4,V_2,P_1,P_2)$
$E_5 = (V_4,V_1,P_1,\lambda)$
$P_1 = (E_1,E_4,E_5)$
$P_2 = (E_2,E_3,E_4)$

Şekil 2.4. Her poligon için kenar listeleri tanımlanmış poligon temelli ağ. (λ, boşu temsil eder.)

Bu üç gösterimin hiç birinde (açık, köşe noktaları listelerine işaretçiler ve kenar listelerine işaretçiler), bir köşe noktasını hangi kenarların içerdiğini bulmak kolay olmamaktadır. Tüm kenarlar incelenmek zorundadır. Elbette, bu tür bilgileri belirleyebilmek için ek bilgiler açık biçimde eklenebilir. Örneğin, Baumgart [3] her poligonun bitişik olan iki kenarına işaret eden işaretçiler içerecek biçimde kenar ifadelerini genişletmiştir. Halbuki köşe noktaları ifadeleri, o köşe noktalarını içeren kenarlara işaret eden işaretçileri içermektedir. Böylece daha fazla poligon ve köşe noktaları bilgileri kullanılabilir olmaktadır [1].

2.1.2. Poligon temelli ağ gösteriminin tutarlılığı

Poligon temelli ağlar genellikle operatörler tarafından sayısallaştırılan çizimlere benzer şekilde etkileşimli olarak üretilir. Böylece, tüm poligonlar kapalı olduğundan, tüm kenarlar en az bir kez ve uygulama ile belirlenen bir sayıdan daha fazla kullanılmadığından ve her köşe noktasının en az iki kenar tarafından referans edildiğinden emin olmak için uygun bir yol bulunmuş olur. Bazı uygulamalarda, ağ yapısının tamamen bağlanmış olmasını (her köşe noktasına diğer köşe noktalarından kenarlar üzerinden geçerek ulaşılabildiği), topolojik olarak düzlemsel olmasını (köşe noktaları üzerindeki ikili (binary) ilişkilerin düzlemsel grafiklerle temsil edilen kenarlarla tanımlandığı), veya boşluk içermeyecek türden (yalnızca tek bir sınırın olduğu – bağlanmış kenar dizilerinin her birinin tek bir poligon tarafından kullanılması beklenir.

İncelenen üç gösterimden en fazla bilgi içermesinden dolayı açık-kenar şeması, tutarlılığı kontrol etmenin en kolay yoludur. Örneğin, tüm kenarların, en az bir ve belirli bir maksimumdan fazla olmayan poligonların bir parçasını temsil etmesi gerekliliğini kontrol etmek için Şekil 2.5' teki program parçası kullanılabilir.

Bu yöntemin (procedure) komple bir tutarlılık kontrolünde hiç bir anlamı yoktur. Örneğin, bir kenarın aynı poligonda birden fazla kullanıldığını algılamamaktadır. Benzer bir yöntem ise her köşe noktasının en az bir poligonun parçası olduğundan emin olmak için kullanılabilir; aynı poligonun köşe noktası ile bağlı en az iki kenar olup olmadığını kontrol edebiliriz. Ayrıca, bir kenarın sıfır uzunlukta olmasına izin

verilmediği sürece, iki köşe noktasının aynı değere sahip olması durumunda bir hata olduğu bulunabilir.

```
Begin
      For   "kenar kümesindeki her kenar Ej"  Do
            "sayacj := 0 "

      For   "poligon kümesindeki her poligon Pi"  Do
            For   "Pi poligonunun her Ej kenarı"  Do
            "sayacj := sayacj + 1"

      For   "kenar kümesindeki her Ej kenarı"  Do
            Begin
                 If   "sayacj = 0"  Then
                 Hata()
                If   "sayacj  >  maksimum"  Then
                 Hata()
            End
End
```

Şekil 2.5. Açık poligon gösteriminde tüm kenarların "1 ile maksimum" kez arasında kullanıldığını kontrol eden program.

Poligonlar arasındaki "bir kenarı paylaşma" ilişkisi bir ikili (binary) eşitlik ilişkisidir ve eşitlik sınıfları içerisindeki ağ parçaları "bağlanmış bileşenler" olarak adlandırılır [1].

2.1.3. Düzlem eşitlikleri

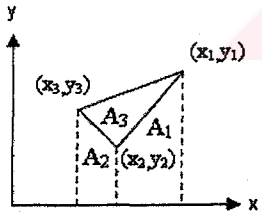
Poligonlar veya poligon temelli ağlar ile çalışırken poligonun üzerinde bulunduğu düzlem eşitliğini iyi biçimde bilmemiz gerekmektedir. Bazı durumlarda, elbette, poligonu tanımlamak için kullanılan etkileşimli tasarım yöntemleri üzerinden düzlem eşitliği açık biçimde bilinmektedir. Eğer bilinmiyorsa düzlemi elde etmek için üç köşe noktasının koordinatlarını kullanabiliriz. Düzlem denklemini ifade edersek;

$$Ax + By + Cz + D = 0. \qquad (2.1)$$

A, B, C katsayıları düzleme olan normali tanımlar, [A B C]. Düzlem üzerinde verilen P1, P2 ve P3 noktalarından düzlem normali, P1P2 × P1P3 (veya P2P3 ×

P2P1, vb.) vektör çapraz-çarpımı olarak hesaplanabilir. Eğer çapraz-çarpım sıfırsa, verilen üç nokta aynı doğrultudadır ve bir düzlem ifade etmez. Eğer varsa diğer noktalar bu noktalar yerine kullanılabilir. Sıfırdan farklı çapraz - çarpım için, D değeri [A B C] normali ve üç noktadan herhangi biri kullanılarak (2.1) eşitliğinden çıkarılabilir.

Eğer üç köşeden daha fazla köşe varsa, sayısal nedenlerden dolayı veya poligonların üretilme yöntemlerinden dolayı bunlar düzlemsel olmayabilirler. Düzlemin A, B, C katsayılarına ulaşmanın diğer bir yolu ise A, B, C katsayılarının, sırasıyla poligonun (x,y), (x,z), (y,z) düzlemlerine izdüşümlerinin işaretlemiş olduğu alanlar ile doğru orantılı olmalarıyla açıklanabilir. Örneğin, eğer poligon (x,y) düzlemine paralel ise A = B = 0 olması beklenir. Poligonun, (y,z) ve (x,z) düzlemlerine olan iz düşümlerinin oluşturduğu alanlar sıfır olmaktadır. Bu teknikte iz düşüm alanları, tüm köşe noktalarının koordinatlarının bir fonksiyonu olduğundan A, B ve C katsayıları elde edilebilmektedir. Örneğin, poligonun (x + y) düzlemine iz düşülen C alanı (aynı zamanda C katsayısı) Şekil 2.6'da koordinat eksen takımı üzerinde gösterilmekte ve (2.2) eşitliğinde yalnızca A1 ve A2 alanlarının A3 yamuk alanından çıkarılmasıyla oluşan alan olarak saptanmaktadır.



$$C = \frac{1}{2}(y_1 + y_2)(x_2 - x_1) + \frac{1}{2}(y_2 + y_3)(x_3 - x_2) + \frac{1}{2}(y_3 + y_1)(x_1 - x_3)$$

Şekil 2.6. (2.2) eşitliğini kullanarak C alanının hesaplanması.

$$C = -A_1 - A_2 + (A_1 + A_2 + A_3) = A_3 \tag{2.2}$$

C katsayısının belirlenmesi, genel bir ifade ile (2.3) eşitliği ile verilmektedir.

$$c = \frac{1}{2}\sum_{i=1}^{n}(y_i + y_{i\oplus 1})(x_{i\oplus 1} - x_i) \tag{2.3}$$

Denklem içerisindeki $\oplus$ operatörü $n \oplus 1 = 1$ eşitliği dışında normal toplama olarak işlem yapmaktadır. Denklem (2.3)'de poligonların peşpeşe gelen kenarları ile oluşan tüm trapezoidlerin alanlarının toplamı elde edilmektedir. Eğer $x_i \oplus 1 < x_i$ ise toplama negatif bir etki katar. Ayrıca toplamın işareti de oldukça yararlıdır. Eğer köşe noktaları saat yönünde sıralanmışlarsa (düzleme iz düşümleri de aynı şekilde) işaret pozitif, aksi halde ise işaret negatif olacaktır. Tüm köşe noktalarını kullanarak düzlem denklemini elde ettikten sonra her noktanın düzleme olan dikey mesafelerini hesaplayarak poligonun nasıl bir düzlem olduğunu kestirebiliriz. Köşe noktası (x, y, z) için dikey mesafe aşağıdaki formül ile hesaplanabilir.

$$d = \frac{Ax + By + Cz + D}{\sqrt{A^2 + B^2 + C^2}} \tag{2.4}$$

Noktanın, düzlemin hangi tarafında konumlandığına bağlı olarak bu mesafe pozitif veya negatif olabilir. Eğer köşe noktası düzlem üzerinde konumlanmış ise d mesafesi "0" olarak bulunacaktır. Elbette, noktanın düzlemin hangi tarafında olduğu d'nin işaretine bağlıdır, bu yüzden paydadaki karekök ifadesi bu anlamda önemini yitirmektedir [1]. Düzlem eşitliği tek bir değere karşılık gelmemektedir; sıfırdan farklı herhangi bir "k" çarpanı düzlemi değiştirmeden eşitliği değiştirmektedir. Bu durumda düzlem katsayılarını normalize edilmiş normal vektörü ile saklamak uygun olacaktır; bu k değerini aşağıdaki şekilde seçerek yapılabilir.

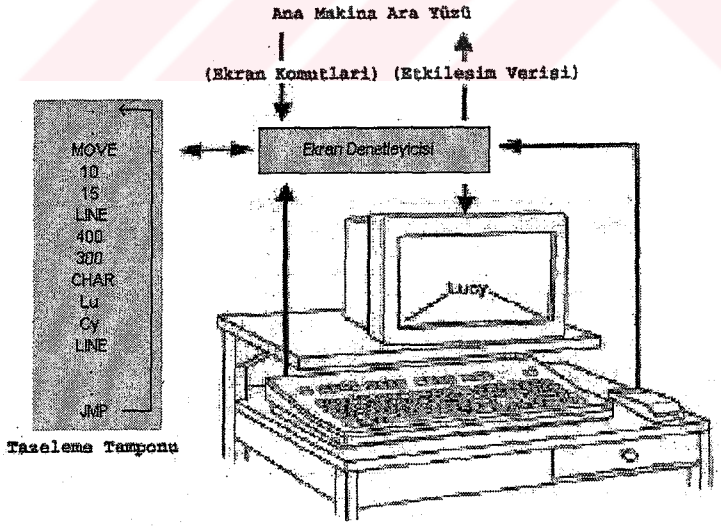
$$k = \frac{1}{\sqrt{A^2 + B^2 + C^2}} \tag{2.5}$$

(2.5) ifadesi normalin uzunluğuna karşılık gelmektedir. Böylece, paydanın "1" yapılması durumunda (2.4) denklemi daha kolay biçimde hesaplanabilmektedir.

3. BİLGİSAYAR GRAFİKLERİNDE DONANIM

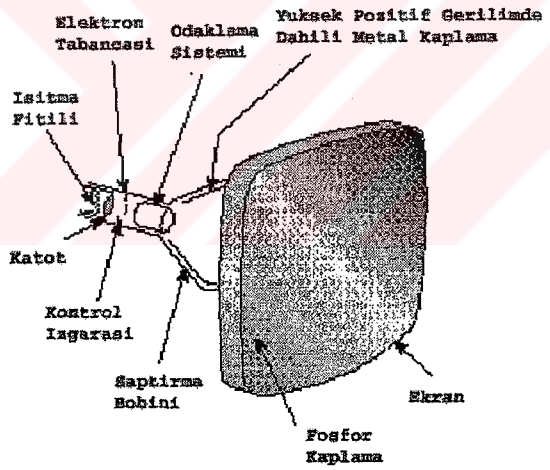
3.1. Çıkış Teknolojisi

Vektör, darbe ve çizgi çizicileri olarak adlandırılan görüntüleme cihazları altmışların ortalarında geliştirilmiş ve seksenlerin ortalarına kadar yaygın biçimde kullanılmışlardır. "Vektör" terimi burada "çizgi" ile eş anlamlı olarak kullanılmıştır; "darbe" ise kısa çizgi yerine kullanılmış ve karakterler bu darbelerin ardışık dizilerinden oluşmaktadır. Tipik bir vektör sistemi, merkezi işlem birimine (CPU) bir giriş-çıkış (I/O) çevre birimi olarak bağlanmış bir ekran işlemcisi, ekran tampon hafızası (display buffer memory), bir tane de CRT'den (cathode ray tube) oluşmaktadır. Tampon, bilgisayarda üretilen "görüntüleme listesi"ni veya "görüntüleme programı"nı saklamaktadır; (x, y) ve (x, y, z) uç nokta koordinatları ile nokta ve çizgi çizme komutlarını ve aynı şekilde karakter çizme komutlarını saklamaktadır. Şekil 3.1'de tipik bir vektör görüntüleme mimarisi gösterilmiştir [1].



Şekil 3.1. Vektör görüntüleme mimarisi.

Noktaları, çizgileri ve karakterleri çizmek için gereken komutlar ekran işlemcisi tarafından yorumlanır. Ekran işlemcisi vektör üretecine sayısal nokta koordinatlarını göndererek sayısal koordinat değerlerinin CRT'nin fosfor tabakası üzerine yazılan elektron ışın demetinin yerini değiştiren huzme-saptırma devreleri için analog gerilim değerlerine dönüştürülmesini sağlar. Vektör sisteminin temeli demetin uç noktadan uç noktaya keyfi düzendeki görüntüleme komutlarının işletilmesiyle saptırılmasıdır; bu teknik rasgele tarama (random scan) olarak adlandırılır. (Lazer gösterilerinde de lazer ışın demeti rasgele taramalı olarak saptırılır.) Fosforun ışık çıktısının onlarca veya en fazla yüzlerce mikro saniyede bozulduğu göz önüne alındığında titreşim (flicker) etkilerinden kaçınmak için ekran işlemcisi görüntüleme listesi üzerinden fosforu en az saniyede 30 kez (30 Hz) yenilemelidir; bu nedenle görüntüleme listesini tutan tampon genellikle yenileme tamponu (refresh buffer) olarak adlandırılır. Şekil 3.1'e dikkat ederseniz "jmp" komutu çevrimsel bir yenileme sağlamak için görüntüleme listesinin başına geri dönülmesini sağlar [1].



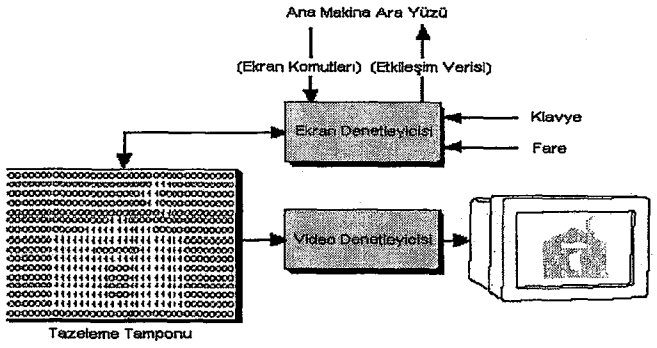
Şekil 3.2. Katotlu ışın tüpü (CRT).

Altmışlarda, en az 30 Hz yenileme hızına sahip tampon hafızalar ve işlemciler oldukça pahalı idi ve sadece birkaç bin satır göze çarpan titreşim etkilerinden kaçınılarak gösterilebiliyordu. Altmışların sonlarında, doğrudan-gösteren bellek tüpü (DVST: direct-view storage tube) tüm titreşim etkilerini yok ederek tampon ve yenileme işlemlerinin üstesinden gelmiştir. Bu, etkileşimli grafiği mümkün kılan hayati bir adımdı. DVST imgeyi, gömülü olan fosfor tabakasındaki yedekleme ağı

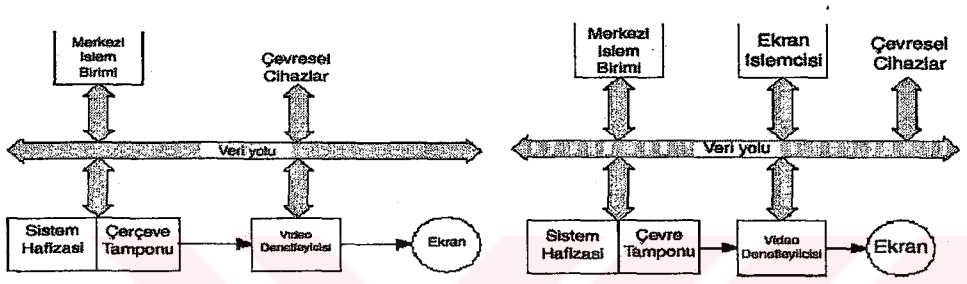
üzerine göreceli olarak yavaş-hareket eden elektron demeti ile imgeyi bir kez yazarak saklar. Bu küçük, kendine yeten DVST terminali tipik yenileme sisteminden daha ucuz olup zaman-paylaşımlı sistemlere düşük hızlı (300-1200 baud) telefon ara yüzü olarak kullanmak için idealdir. DVST terminalleri etkileşimli grafikler için pek çok kullanıcı ve programcı tarafından incelenmiştir [1].

Altmışların sonlarında donanımdaki diğer önemli gelişme ise mini bilgisayarlara ekran eklemek olmuştur; bu düzenleme ile merkezi zaman-paylaşımlı bilgisayar tazelenen ekran cihazlarına, özellikle kullanıcı-etkileşimli kullanıma ve imgeyi ekran üzerinde güncelleme isteklerine olan yoğun talebi rahatlatmıştır. Mini bilgisayarlar uygulama programlarını iyi biçimde koşturabilmekte ve büyük analiz programlarını koşturabilmek için daha büyük merkezi ana bilgisayara bağlanabilmektedir. Mini bilgisayar ve DVST düzenlemelerinin her ikisi de binlerce grafik sisteminin kurulmasına vesile olmuştur. Bu noktada, donanım olarak ekran işlemcisinin kendisi, grafik yazılımının zaman harcayan işleri olan pek çok yordamın görevini devralmış ve daha da karmaşık bir yapıya dönüşmüştür. Gelişmiş bu tür cihazlar, 1968 yılında geometrik dönüşümler için ekran tazeleme donanımlarının icat edilmesiyle, ölçekleyebilme, döndürebilme, noktaları ve çizgileri ekran üzerinde gerçek zamanlı olarak dönüştürebilme, 2-B ve 3-B kırpma sağlayabilme, paralel ve perspektif izdüşüm üretebilme özelliklerine kavuşmuşlardır [1].

Yetmişlerin başlarında televizyon teknolojisini temel alarak gelişen ucuz taramalı grafikler, bulunduğu alana diğer teknolojilere nazaran daha fazla katkı sağlamıştır. Taramalı ekranlar (raster displays) Şekil 3.3'te görüldüğü gibi görüntü öğelerini (çizgiler, karakterler vb.) piksel olarak tabir edilen bileşenler biçiminde tazeleme tamponuna saklamaktadır. Bazı taramalı ekranlarda, vektör ekranlara benzer şekilde çıkış komutları dizisini alan ve yorumlayan donanım ekran denetleyicisi bulunmaktadır (şekilde görüldüğü gibi); basit biçimde, kişisel bilgisayarlar gibi en yaygın kullanılan sistemlerde ekran denetleyicisi yalnızca grafik kütüphanesi paketinin bir yazılım bileşeni olarak mevcuttur. Tazeleme tamponu ise sadece geçek imgeyi ekran üzerinde oluşturan imge görüntüleme alt sistemi (video denetleyicisi olarak da adlandırılır) tarafından okunan CPU hafızasının bir bölümüdür. Şekil 3.4(a)'da taramalı ekran basit sistemi ve (b)'de ise ekran denetleyicisi bulunan taramalı ekran sistemi gösterilmektedir [1].



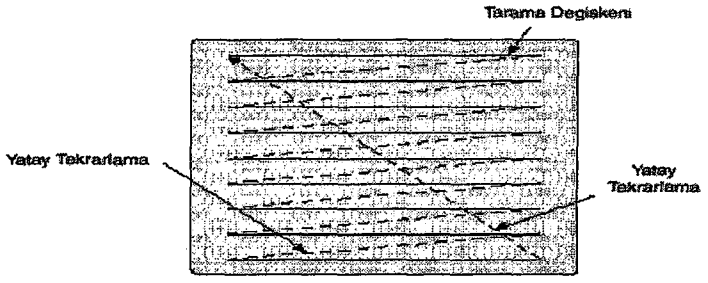
Şekil 3.3. Taramalı ekran mimarisi.



(a) Taramalı Ekran Basit Sistemi (b) Tümleşik Ekran Denetleyicili Sistem

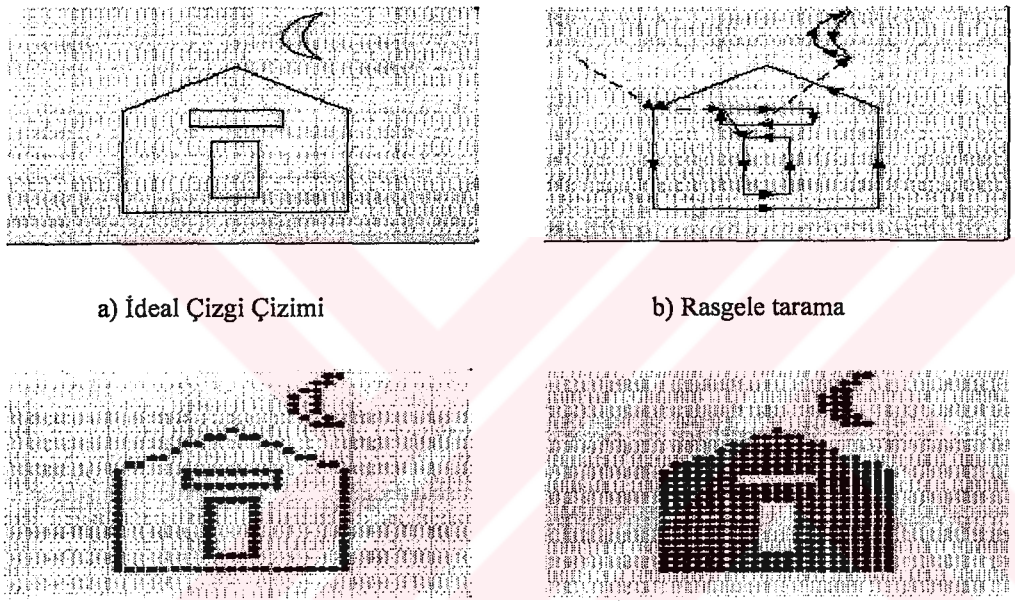
Şekil 3.4. Taramalı ekran sistemi.

Taramalı ekran üzerindeki imgenin bütünü, her biri ayrı bir piksel satırı olan yatay tarama çizgilerinin bir kümesini ifade eden tarama örüntüsü (raster) tarafından biçimlendirilir; tarama örüntüsü böylece tüm ekran alanını gösterecek biçimde bir piksel matrisi olarak saklanmaktadır. Tüm imge, video denetleyicisi tarafından yukarıdan aşağıya ve tekrar başa dönecek şekilde "zamanda bir" tarama çizgisi sıralı olarak taranır (Şekil 3.5'te gösterilen biçimde) [1].



Şekil 3.5. Çizgi tarama (Raster scan).

Her pikselde ışının yoğunluğu piksel yoğunluğunu yansıtmak için ayarlanır; renk sistemlerinde üç ışın demeti kontrol edilmektedir- her piksel değerinin üç kontrol bileşenini tanımlamak üzere kırmızı, yeşil ve mavi temel rekleri için kontrol yapılmaktadır. Şekil 3.6(a) da bir evin basit bir 2-B çizgisel çiziminin görüntülenmesi ile rasgele ve çizgisel tarama arasındaki fark gösterilmektedir. Şekil 3.6(b) de ok uçları ile gösterilen vektör yayları ışının rasgele saptırılmasını göstermektedir. Şekil 3.6(c) de kareler, poligonlar, yaylarla ifade edilen dolgusuz ev gösterilmekte olup (d) kısmında ise dolgulu gösterimi sunulmuştur.



a) İdeal Çizgi Çizimi

b) Rasgele tarama

c) Dış Hat Öğeleri ile Çizgi Tarama

d) Dolgu Öğeleri İle Çizgi Tarama

Şekil 3.6. Çizgi tarama ve rasgele tarama.

3.2. Giriş Teknolojisi

Giriş teknolojisi yıllar boyunca fazlasıyla gelişmiştir. Vektör sistemlerinin kırılgan ışıklı kalemi her yerde bulunabilen fareler(ilk olarak ofis otomasyon öncülerinden Doug Engelbart tarafından altmışların ortalarında geliştirilmiştir), veri tabletleri ve ekran üzerine monte edilmiş dokunmaya duyarlı, saydam paneller ile yer değiştirmiştir [1].

Daha fonksiyonlu giriş cihazları sadece (x,y) koordinatlarının konumlarını belirtmez, ayrıca üç boyutlu giriş değerlerinin kullanımını da mümkün kılar. Ses haberleşmesi de ayrıca elle dokunmadan giriş yapılmasına (hands-free input), basit komutların doğal çıktılarına ve geri beslemeye izin verdiği sürece ilgi çekici bir potansiyele sahiptir. Kullanıcı, standart giriş cihazlarıyla komut girerek, ekran üzerindeki çizime yeni bilgiler ekleyerek veya mevcut çizim üzerinde seçim yaparak işlemleri ve resim bileşenlerini belirleyebilir. Bu etkileşimler herhangi bir bilgi birikimine gereksinim duymaz ve yalnızca klavye kullanımıyla gerçekleştirilebilir. Kullanıcı seçimlerini menü butonlarını veya simgeleri seçerek, form içerisine birkaç karakter girerek veya içerisindeki özellikleri kontrol ederek, ekran üzerindeki imlecin hareketiyle boyanmış ve gri, renkli veya çeşitli desenlerle biçimlendirilmiş boyalı alanları veya poligonlarla sınırlanmış kapalı alanları doldurarak basitçe yapabilmektedir [1].

4. MODELLEME

Muhteşem bilgisayar grafikleri harika görüntü ve geometrik modeller ile başlar. Geometrik modeller ile istenilen herhangi bir gerçek dünya nesnesi bilgisayar grafiklerine dönüştürülebilir ve böylece istenen deneyler bilgisayar ortamında gerçekleştirilebilir. Bu bölümde 3-Boyutlu modelleme kuralları, üç-boyutlu uzayda kullanılan matematiksel dönüşümler ve modelleme özellikleri incelenecektir. Son olarak OpenGL ile modelleme yaparken dikkat edilmesi gereken modelleme problemlerine değinilecektir.

4.1. 3-Boyutlu Modelleme

3-Boyutlu bilgisayar grafikleri bir şeyi canlandırmak veya öğretmek için pek çok farklı uygulama çeşitlerinde kullanılabilir. Bu mimarisel tasarım, endüstriyel ilk örnekleme (prototyping) , eğlence gibi konuları içeren çok geniş bir alan olmakla birlikte 3-Boyutlu grafikleri kullanmak istediğiniz alanlar için bir sınır bulunmamaktadır. Ekran üzerinde istenen çıktının görüntülenebilmesi birkaç adımda gerçekleştirilebilir: Nesne oluşturma veya kamera ile görüntü yakalama, 3-Boyutlu dönüşümler, aydınlatma gibi çevresel düzenlemeler yapılarak ekran üzerinde görüntülenmesi istenen görüntü elde edilir [4].

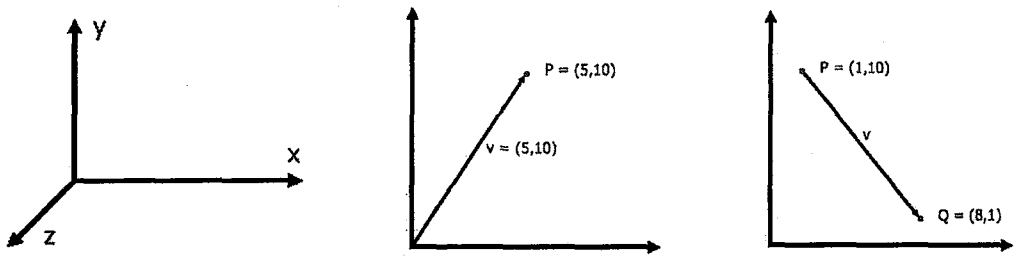
4.1.1. 3-Boyutlu uzayda nesneleri oluşturma

Nesneyi oluşturmadan önce 3-Boyutlu nesnelerle ilişkili 3-Boyutlu koordinat sistemi hakkında iyi düzeyde bilgi sahibi olmak gerekir. Bu 3-Boyutlu uzayda orijin noktası üç eksenin kesiştiği noktada bulunmaktadır. Bu üç eksen genellikle X, Y ve Z eksenleri olarak adlandırılmaktadır. Esas koordinat sistemi dünya koordinat sistemi olarak bilinmektedir fakat farklı nesneleri ekran üzerinde tanımlamak istediğimizde

bu koordinat sistemini yerel koordinat sistemi olarak da ifade edebiliriz. Koordinat sistemleri farklı biçimlerde çizilmektedir. Şekil 4.1(a)'da yaygın biçimde kullanılan 3-Boyutlu koordinat sistemi gösterilmektedir.

Bir nokta, her eksen doğrultusunda verilecek değerlerle tanımlanabilir, örneğin (2, 0, -4) noktası "x" ekseninde pozitif doğrultuda iki birim ve "z" ekseninde negatif doğrultuda 4 birim uzaklıktaki bir noktayı ifade etmektedir. Nokta uzayda bir konuma sahiptir. Bir noktadan diğer bir noktaya en kısa yol vektör olarak tanımlanmaktadır. Vektörler doğrultuya ve uzunluğa sahip olmakla beraber bir konuma sahip değildirler. Noktalar ve vektörler benzer biçimde gösterilmekte fakat farklı şeyleri ifade etmektedir. Yalnızca bir nokta ifade edilmişse o noktanın orijine olan uzaklığı bir vektörü ifade edebilir, fakat bir vektör genel bir ifade ile iki nokta arasındaki fark bulunarak belirlenir. Şekil 4.1(b)'de bir P(5,10) noktası ve bu P noktasının orijine olan yolunu ifade eden v(5,10) vektörü gösterilmektedir. Şekil 4.1(c)'de P(1,10) ve Q(8,1) olarak iki adet nokta ve bu noktalar arasındaki v vektörü gösterilmektedir. Anlatılanlardan anlaşılacağı gibi buradaki "v" vektörü "v = Q – P" olarak belirlenir ve sonuçta "v = (7, -9)" bulunur. Vektörlerin bu bölümde incelenmesinin nedeni uzayda nesnelerin nerede konumlanacağının, nesnelerin boyutları ve yönelimlerinin, nesneler arasındaki uzaklıkların ne olduğunun, yansımanın nasıl çalıştığının, fiziğin nasıl işlediğinin, nesnelerin üzerine ışığın nasıl düştüğünün bilinmesi gerekliliğindendir [4].

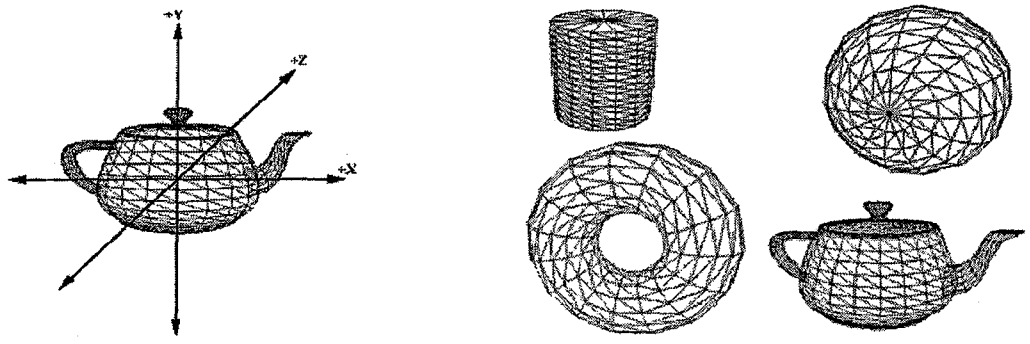
Noktalar 3-boyutlu uzayda kendi başlarına çok fazla bir değer ifade etmemektedirler. Noktaları, görünür yüzeylerde köşe noktaları (vertice) olarak kullanmak faydalı olacaktır. Bu tür yüzeyler ile herhangi düzlemsel poligonlar şeklinde ve 2-Boyutlu imge üzerinde oldukları durumlarda ise daha çok, çok parçaya bölünmüş üçgensel poligonlar şeklinde karşılaşmaktayız. Bu kullanım sayesinde basit bir algoritma ile tüm imgeyi hesaplayabilmekteyiz. Bu durumda, üçgensel yüzler daima düzlemsel olup, hiçbir durumda dış veya içbükey olarak karşımıza çıkmayacaktır [4].



(a) Koordinat Sistemi (b) Nokta ve Vektör (c) İki Nokta ve Vektör

Şekil 4.1. Yerel koordinat sistemi, 3-boyutlu uzayda bir nokta ve vektörün gösterilmesi.

3-Boyutlu uzaydaki düzlemsel veya eğrisel karmaşık nesneleri ekran üzerinde gösterebilmek yalnızca poligon temelli yüzlerin kullanılmasıyla mümkün olabilir. Nesnelerin çoklu-detay seviyeleri veya diğer bir ifade ile çözünürlük değerleri gösterimde kullanılan üçgensel yüzlerin sayısına bağlıdır. Her üçgensel yüzün yüzeye dik olan normal vektörleri mevcuttur ve bu normal vektörü ekrandan dışarı doğru olduğunda üçgensel yüz ekranda görünür olmaktadır. O halde nesneler görüntülemek istendiklerinde yüzey normallerinin hesaplanması faydalı olacaktır. Model normallerinin bulunmasına ileriki aşamada değinilecektir. Şekil 4.2'de 3-Boyutlu uzayda nesnelerin poligonsal yüzler ile nasıl görüntülendiği gösterilmektedir. Şekil 4.2(a)'da koordinat sistemi üzerinde 3-Boyutlu çaydanlık modeli ve (b) kısmında ise dört adet farklı poligonsal model görülmektedir.



(a) (b)

Şekil 4.2. 3-boyutlu uzayda nesnelerin 3-boyutlu modellerinin görüntülenmesi.

4.1.2. Kamera ile görüntü yakalama

Ekran üzerinde görünmesi gerekenin ne olduğunu belirlemek gerekmektedir. Bunu yapabilmek için üç-boyutlu koordinat sistemine uyacak biçimde kameranın doğrultusunu ve pozisyonunu belirlemek gerekir. Daha sonra kameraya doğrultu vektörü etrafında dönme etkisi (rotation) verebiliriz. Kamera sabit (fixed) veya uçan (flyby) olmak üzere iki farklı temel tipten biri olabilir ve bazı diğer özelliklerinin ayarlanması gerekmektedir. Bu bölümde bu özelliklere değinilmeyecektir.

4.1.3. Aydınlatma

İmge üzerindeki algılanan gerçekliği artıracak diğer bir özellik de nesneleri aydınlatmaktır. 3-Boyutlu uzayda çeşitli ışık kaynakları kullanılarak, bu kaynakları nesnelerin herhangi bir tarafına konumlandırarak ve çeşitli ışıklılık ve materyal özellikleri eklenerek nesneler aydınlatılmaktadır. Bu çeşitli ışık kaynakları, aydınlatma ve materyal özellikleri bu bölümün ileriki bir aşamasında, programlanması ise OpenGL konusu altında incelenecektir.

4.2. 3-Boyutlu Dönüşümler

3-Boyutlu grafik programlama alanındaki temel dönüşümler 3-Boyutlu uzaydaki noktaların ötelenmesi, dönmesi ve ölçeklenmesi olarak gösterilir. Bu nedenle bu işlemler standart formatlarıyla gösterilir ve kolayca programlanabilir. Dönüşümler ($b = M \bullet a$) biçimindeki genel formu almaktadırlar. Böylece yalnızca dönmeler ve ölçeklemeler bu şekilde olmayacak, ayrıca dönüşümler de bu doğrusal formda olacaktır. Matematiksel gösterimde afin uzayının (R3) iz düşüm uzayının P(R4) içerisine gömülü olduğunu varsayarak, herhangi bir afin dönüşümünü matris çarpımları ile ifade edebilmekteyiz. Şekil 4.3'te 3 boyutlu dönüşümlerde kullanılan matris ve vektör yapıları gösterilmektedir. 3-Boyutlu uzay için şekildeki "a" ve "b" vektörlerinin 3 elemanı, "M" matrisinin "3x3" formunda olması gerekirken "a" ve

"b" vektörlerinin 4, "M" matrisinin "4x4" formunda olmasının nedeni işlemleri kolaylaştıracak olan "Xw" homojen koordinatının eklenmesidir [4].

$$a = \begin{pmatrix} a_x \\ a_y \\ a_z \\ a_w \end{pmatrix} \qquad b = \begin{pmatrix} b_x \\ b_y \\ b_z \\ b_w \end{pmatrix} \qquad M = \begin{pmatrix} t_{xx} & t_{xy} & t_{xz} & t_{xw} \\ t_{yx} & t_{yy} & t_{yz} & t_{yw} \\ t_{zx} & t_{zy} & t_{zz} & t_{zw} \\ t_{wx} & t_{wy} & t_{wz} & t_{ww} \end{pmatrix}$$

Şekil 4.3. 3-boyutlu dönüşümlerde kullanılacak matris ve vektörler.

Tüm dönüşümler köşe noktaları (vertex) seviyesinde gerçekleşmekte, bu yüzden bir nesne değiştirilmek istendiğinde nesnenin tüm köşe noktalarının dönüştürülmesi gerekir. Köşe noktalarında yapılan değişiklikler sonucunda üçgensel yüzler ve üçgensel yüzlerin değişimi sonucu nesnenin değiştirilmesi mümkün olmaktadır.

Aklımızın bir köşesinde tutmamız gereken bir nokta ise, ölçekleme (scaling) ve döndürme (rotation) dönüşümleri esnasında nesne kendi merkez noktasından dünya koordinat sistemi orijin noktasına kaydırılmalı, ölçekleme ve döndürme dönüşümleri uygulanmalı ve sonuçta eski pozisyonuna yeniden getirilmelidir. Bunun yerine yerel koordinat sistemi kullanılabilir ki nesne dönüşümleri bulundukları pozisyonda bu sistemde gerçekleştirilebilir fakat ölçekleme ve döndürme işlemlerinin yalnızca kısıtlı bir kısmı gerçekleştirilebilir ya da 3-D bir CAD sistemi (OpenGL olabilir) bu işlemi bizim için yapabilir. Aşağıdaki tüm dönüşümler 4x4'lük matrislerle gösterilecek ve eğer bir köşe noktası bir seri dönüşüme tabi tutulacaksa ilk olarak bu dönüşüm matrisleri çarpılacaktır ve sonuç matrisi köşe noktasına uygulanacaktır.

Yalnızca üç boyutla çalışmamıza rağmen dört boyutu da içerecek bir matris kullanmamızın nedeni ölçekleme ve döndürme etkilerinin çarpımsal, dönüşümün toplamsal bir etkisinin bulunmasıdır. Matrislere bir fazla boyut eklenmesi ile tüm dönüşümler çarpımlar ile gerçekleştirilebilir olmaktadır. Dördüncü koordinat işlevsiz olmakla beraber sonuç koordinat sistemi "homojen koordinat sistemi" olarak adlandırılmaktadır [4].

4.2.1. Yer değiştirme dönüşümü

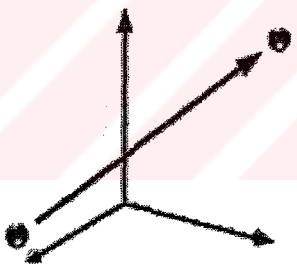
Yer değiştirme-dönüşümü, köşe noktasının dönüşüm vektörü ile her eksen doğrultusunda belirli bir mesafe yer değiştirmesini sağlar. Bu dönüşüm aşağıdaki biçimde ifade edilebilir:

$$(x', y', z') = (x, y, z) * (dx, dy, dz)$$

Ya da aşağıdaki çarpım matrisi formunda gösterilebilir.

$$b = T(t)a = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} a_x \\ a_y \\ a_z \\ 1 \end{pmatrix} = \begin{pmatrix} a_x + t_x \\ a_y + t_y \\ a_z + t_z \\ 1 \end{pmatrix}$$

Bu dönüşümün, nesnenin tüm köşe noktalarına uygulanması ile nesne, Şekil 4.4'te görüldüğü gibi hareket etmektedir.

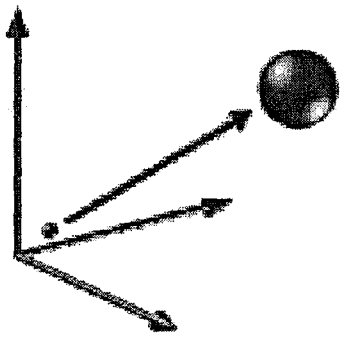


Şekil 4.4. Yer değiştirme dönüşümü.

4.2.2. Ölçekleme

Bir köşe noktasını ölçeklemek için koordinatların her birisini ölçekleme faktörü ile çarparız. X, Y ve Z ölçekleme faktörleri, tekdüze (uniform) ölçekleme elde etmek için aynı, tekdüze olmayan (non-uniform) ölçekleme elde etmek içinse farklı olabilir. Daha önce de bahsedildiği üzere tüm nesnenin ölçeklenmesi durumunda, her köşe noktası dünya koordinat sistemi orijinine göre ölçeklenmektedir. Bunun anlamı, eğer

nesne dünya koordinat sistemi orijinine yerleştirilmemiş ise ölçeklendiği anda şekilde görüldüğü gibi konum değiştirecektir.



Şekil 4.5. Ölçekleme dönüşümünün etkisi.

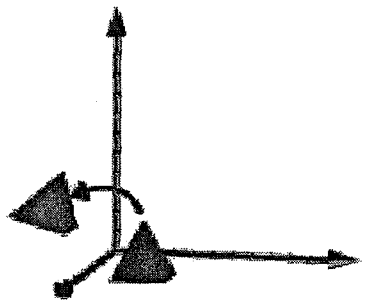
Bu durumdan kaçınmak için nesne ilk olarak dünya koordinat sistemi orijinine yerleştirilmeli (nesne merkezi orijine kaydırılmalı), ölçeklenmeli ve son olarak gerçek pozisyonuna yeniden konumlandırılmalıdır. Şekil 4.5'te ölçekleme dönüşümünün etkisi gösterilmektedir. Ölçeklemede kullanılan dönüşüm matrisi aşağıdaki biçimdedir:

$$b = Sa = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} a_x \\ a_y \\ a_z \\ 1 \end{pmatrix} = \begin{pmatrix} s_x \cdot a_x \\ s_y \cdot a_y \\ s_z \cdot a_z \\ 1 \end{pmatrix}$$

4.2.3. Eksen etrafında döndürme

Eksenler etrafında döndürme işlemi daima eksen etrafında olmaktadır. En basit ifade ile bir eksen dünya eksen takımlarının birisi olabilir. Bir köşe noktasını z ekseni etrafında saat yönünün tersi yönünde α açısı kadar döndürmek (Şekil 4.6) için noktanın koordinatlarına aşağıdaki ifadeler uygulanır.

$$x' = x\cos\alpha - y\sin\alpha$$
$$y' = x\sin\alpha + y\cos\alpha$$
$$z' = z$$



Şekil 4.6. Eksen etrafında döndürme etkisi.

Bu tür bir döndürme işlemi Şekil 4.6'da gösterilene benzer şekilde olmaktadır. Diğer iki eksen etrafında döndürme işlemi ise benzer bir yolla yapılmaktadır. Bunu yapabilmek için tek gereken yukarıdaki eşitliklerdeki harfleri değiştirmek olmaktadır. Sırasıyla x, y ve z eksenlerinde α kadar döndürme faktörlerini $R(e_x, \alpha)$, $R(e_y, \alpha)$, $R(e_z, \alpha)$ olarak gösterelim:

$$R(e_x,\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad R(e_y,\alpha) = \begin{pmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R(e_z,\alpha) = \begin{pmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Şekil 4.7. Sırasıyla x, y, ve z eksenlerinde α açısı kadar dönüş sağlayan matrisler.

Herhangi bir doğrultudaki döndürme işlemleri koordinat eksenlerindeki döndürme faktörlerinin birleşimden oluşur.

$$R(p, \alpha) = R(e_x, \alpha) \bullet R(e_y, \alpha) \bullet R(e_z, \alpha).$$

4.2.4. Bir Nokta Etrafında Döndürme

Yukarıda tanımlanan döndürme matrislerinin tamamı orijin etrafında döndürme işleminin gerçekleştirilmesiyle ilişkilidir. Fakat genellikle dönüşün sağlanması gereken nokta orijinden farklı bir "u" noktasıdır. Böyle bir dönüşün sağlanması için dönüş işleminin yapılacağı noktanın orijin olarak alınması gerekir ve bunun için de bazı yer değiştirme dönüşümü ve döndürme işlemlerinin bir arada kullanılması gerekir. Bunun anlamı koordinat sisteminin değiştirilmesi gerektiğidir [4].

$$b = T(u) \cdot R(p,\alpha) \cdot T(u)^{-1} \cdot a$$

Grafiksel olarak anlatılmak istenirse; ilk olarak "a" noktasının "u" vektörüyle yeri değiştirilmeli, ardından "α" kadar döndürme uygulanmalı ve son olarak nokta eski pozisyonuna geri getirilmelidir.

Grafik programlamada, programcı bu türde birleştirilmiş matrislere tekrar ve tekrar ihtiyaç duyabilir ve bu şekilde 3-D nesneler ait oldukları yerde görüntülenebiliyor. OpenGL, dönüşüm matrislerinin oluşturulması ve işleyişi ile ilgili çalışmaların birçoğunu geliştiricinin yapmasına gerek bırakmadan nesnelerin görüntülenmesini sağlamaktadır.

Dönüşümler ile ilgili incelemeler yapıldıktan sonra son olarak ifade etmemiz gereken bir nokta da 3-Boyutlu koordinatlar ile ilgilidir. (aw = 0) koordinatının sıfır olması durumunda nokta, sonsuza uzanır. Bu durum Şekil 4.8'deki 3-Boyutlu koordinatın homojen koordinatına bölünmesi sonucunda karşımıza çıkmaktadır.

$$a_3 = \begin{pmatrix} a_x / a_w \\ a_y / a_w \\ a_z / a_w \end{pmatrix}$$

Şekil 4.8. Uzaydaki bir noktanın homojen koordinata bölünmesi.

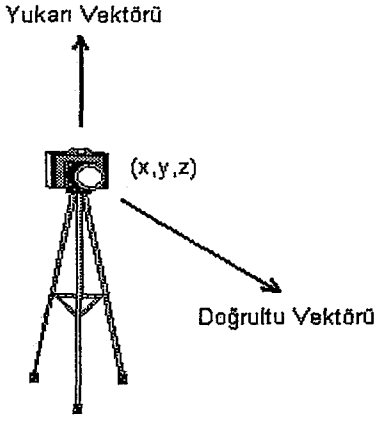
4.3. İz Düşüm

İz düşüm ile ekran üzerinde ne tür bir nesnenin, nasıl görüntüleneceğini tanımlarız. Nesneyi hangi pozisyondan ve hangi doğrultuda görüntülememiz gerektiğini de belirlememiz gerekir.

Bunu açıklamak için kamera analojisini kullanmamız gerekir. Kamera pozisyonu, basit bir üç-boyutlu koordinat ve doğrultuyu veren iki vektör ile ayarlanmaktadır. İlk vektör doğrultu vektörüdür ve kamera merceklerinin doğrultusunu göstermektedir. Ayrıca kameranın dönüşünü, doğrultu vektörü etrafında tanımlamak gerekir. Bunu gerçekleştirmek için ise Şekil 4.9'da görüldüğü gibi kameraya bir yukarı vektörü (up-vector) tanımlamak gerekir.

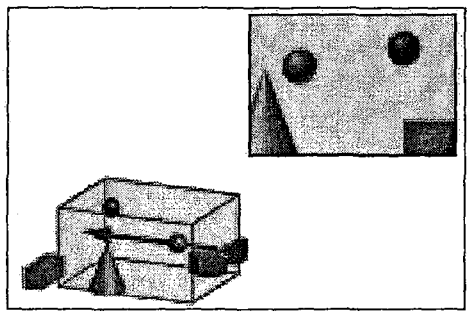
Peki, şimdi kamera ile ne tür bir görüntü elde edilecek. Bu sorunun cevabı ne tür bir iz düşüm kullanıldığı ile ilişkilidir. Seçilebilecek en temel iki adet iz düşüm yöntemi mevcuttur: Paralel ve perspektif.

Paralel iz düşüm ile nesneler ekrana kameranın doğrultu vektörü ile paralel hatlar üzerinden haritalanır. Bunun sonucunda oluşan imge ile nesneler arasındaki oranlar desteklenmektedir. Bu tür bir iz düşüm, oranların ve açıların doğruluğunun önemli olduğu mühendislik ve mimarisel çizimler için uygundur. Fakat bu tür bir iz düşüm ile çok da gerçekçi bir görüntü sağlanamaz. Çünkü gerçek yaşamda izleyici aynı nesneyi yakın bir bölgede daha büyük, uzak bir bölgede ise daha küçük görmektedir.



Şekil 4.9. Pozisyon ve iki vektör.

Paralel iz düşümü anlayabilmek için, kameranın önüne bir paralelyüz konulduğunu ve bu kutu içerisindeki nesnelerin ekrana yansıtıldığını gözümüzde canlandıralım. Bu canlandırma Şekil 4.10'un sol kısmında ve sonuçta elde edilen görüntü de sağ kısmında görülmektedir.

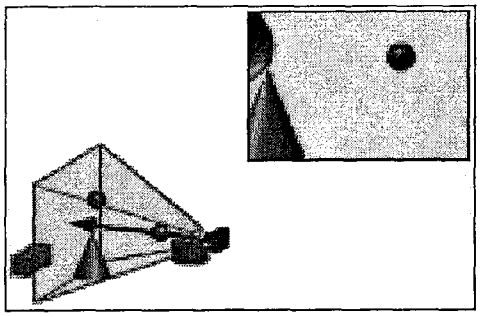


Şekil 4.10. Paralel izdüşüm.

Şekil 4.10'da görüldüğü gibi uzaktaki ve yakındaki nesneler aynı boyutta görüntülenmekte ve paralel hatlar, kameradan uzaklaştıkça paralelliklerini korumaktadır.

Diğer bir iz düşüm yöntemi ise perspektif iz düşümdür ve bu da kameranın veya gözün çalışması ile aynı türdendir. Nesneleri, paralel çizgiler üzerinden yansıtmanın yerine kameradan bir piramit şekline benzer biçimde yayılan çizgiler üzerinden yansıtır. Bunun sonucunda Şekil 4.11'de görüldüğü gibi görüntüleme kutusu, koni (frustum) adı verilen kesme (cut-off) bir piramit olmaktadır.

Bu iz düşüm ile kameraya yakın olan nesnelerin büyük, kameradan uzaklaştıkça görüntülenen nesnelerin daha küçük boyutlarda görüntülendiği görülmektedir. Koninin kameraya en yakın kısmındaki düzleme ön-kırpma (front-clipping) düzlemi, en uzak tarafına ise arka-kırpma (back-clipping) düzlemi denmektedir. Bu düzlemlere olan mesafeler, hangi nesnelerin ekrana çizdirileceğinin ayarlanmasını sağlar. Diğer bir özellik ise kamera merceklerinin açılarının ayarlanabilir olmasıdır. Bu özellik ile uzaklaştırma ve yakınlaştırma işlemlerinin yapılması sağlanır.



Şekil 4.11. Perspektif izdüşüm.

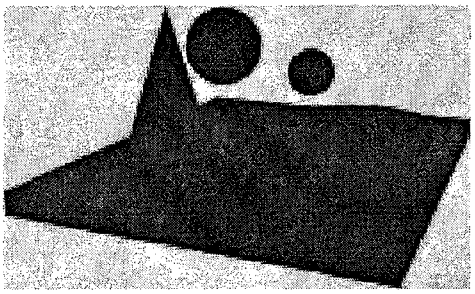
4.4. Aydınlatma

Bu bölümde üç-boyutlu bilgisayar-grafiklerinde kullanılan en yaygın dört ışık kaynağını inceleyeceğiz. Elbette bu kaynakların çok-fazla değişik şekli ve karışımı mevcut olmasına rağmen tezin amacı için dört temel ışık kaynağını incelememiz yeterli olacaktır.

Işık kaynaklarının en basit olanı çevresel (ambient) ışık kaynağıdır. Çevresel ışık kaynağı her noktayı aynı yoğunlukta aydınlatır. Çevresel ışık kaynakları ile ışık kaynağı belirli bir bölgeye konumlanmadığından ışık ışınları tek bir noktadan gelmeyecektir.

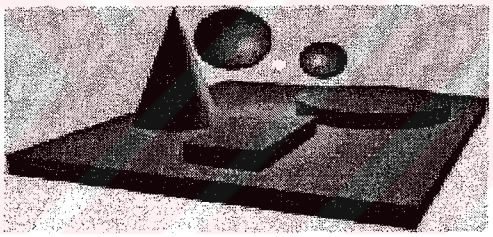
Gerçek yaşamda hiç birinin konumları ve doğrultuları belirli olmayan çok sayıda ışık kaynağı etrafı aydınlatmaktadır. Bu türdeki arka plan ışıkları için çevresel ışık kaynağı kullanımı uygun olacaktır. Elbette birden fazla çevresel ışık kaynağı kullanılmasının bir anlamı olmayacak ve kullanılan ışık kaynaklarının tamamı yalnızca bir çevresel ışık kaynağı olarak davranacaktır.

Yalnızca çevresel türden bir ışık kaynağına sahip olunmasının sonucunda ekranda Şekil 4.12'de canlandırıldığı gibi bir aydınlatma sağlanacaktır. Görüldüğü üzere tüm nesnelerin etrafı aynı gölgeliğe sahip olmakta, nesneleri çevreleyen dış hatlar bir bütün olarak gösterilmekte ve izleyiciyi yanıltmaktadır. Bu nedenle çevresel ışık kaynağı ancak diğer ışık kaynakları ile birlikte kullanıldığında başarılı sonuçlar üretmektedir.



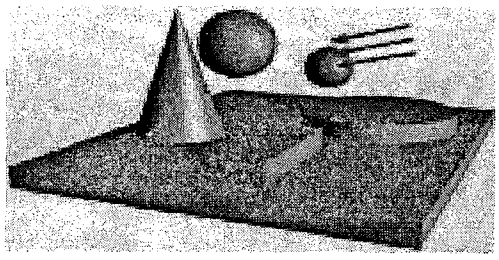
Şekil 4.12. Çevresel ışık kaynağı ile elde edilen görüntü.

Bir diğer ışık kaynağı ise noktasal ışık (point) kaynağıdır. Noktasal ışık kaynakları 3-Boyutlu uzayda bir konuma sahip olmakta ve ışık tüm doğrultulara eşit olarak yayılmaktadır. Şekil 4.13'de nesneler arasına konumlanmış yalnızca bir adet noktasal ışık kaynağı eklenmesi ile imge derinliğinin nasıl etkili biçimde arttığı görülmektedir. Bu imgede nesneler arasındaki yüzeyler birbirinden rahatlıkla ayırt edilebilmektedir.



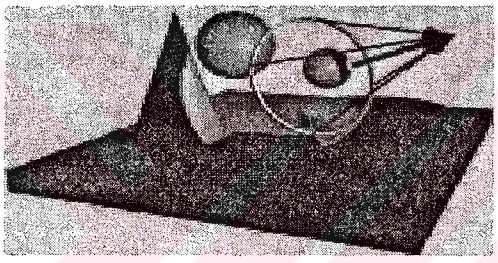
Şekil 4.13. Noktasal ışık kaynağı ile elde edilen görüntü.

Üçüncü tip ışık kaynağı yönsel (directional) veya paralel (parallel) ışık kaynağıdır. Bu kaynak ile ışık belirli bir doğrultudan daima paralel olacak ışınlar halinde gelmekte ve bu nedenle ışığın sonsuz uzaklıktaki bir bölgeden geldiği düşünülmektedir. Bu türde ışık kaynaklarının, kaynağın çok uzak olması gereken durumlarda kullanılması uygun olacaktır ve bu tür bir ışık kaynağı olarak güneş en iyi örnek olacaktır. Şekil 4.14'te yönsel ışık kaynağı kullanılarak elde edilen görüntü sunulmaktadır.



Şekil 4.14. Yönsel ışık kaynağı ile elde edilen görüntü.

İnceleyeceğimiz son tip ışık kaynağı ise ışıldaktır (spotlight). Işık kaynakları arasında en karmaşık olan ışık kaynağı ışıldaktır. Işıldağı yalnızca tek bir doğrultuda ışık yayan noktasal ve böylelikle koniye benzer bir ışık kaynağı olarak görebiliriz. Işıldak bu koni içerisindeki tüm nesneleri aydınlatır (Şekil 4.15).



Şekil 4.15. Işıldak (Spot Light) ile elde edilen görüntü.

Işıldağın ve noktasal ışık kaynağının azalma (fall-off) özelliği olması nedeniyle kaynağa farklı mesafede bulunan bölgelerdeki ışık yoğunluğu farklılaşmaktadır. Normalde iki farklı mesafe tanımı yapmaktayız. Bir tanesi azalmanın başlangıç noktası ve bir diğeri de azalmanın son bulduğu bitiş noktadır. Başlangıç noktasına kadar olan bölgede maksimum ışık yoğunluğu görülmekte, bitiş noktasından sonra ise ışık yoğunluğunun sıfır olduğu görülmektedir. Başlangıç ve bitiş noktası arasındaki bölgede ise doğrusal veya daha farklı bir fonksiyona bağlı olarak azalan ışık yoğunluğu görülmektedir.

Tüm ışık kaynaklarına uygulanabilecek iki özellik mevcuttur. Bunların birisi renk ve bir diğeri ise ışığın gücüdür. Işığın gücü genellikle ışığın değeri olarak anılır ve yoğunluğu ifade etmektedir.

4.5. Modellemede Dikkat Edilmesi Gereken Hususlar

Bu kısımda OpenGL konusuna bir giriş yapılarak modelleme kavramlarının OpenGL ile kullanımı sırasında çıkabileck bazı problemler, dikkat edilmesi gereken hususlar hakkında bilgi verilecektir. Bu konuların OpenGL konusu altında incelenmemesinin tek nedeni anlatılacak olan kavramların OpenGL'e özgü olmayıp genel olarak modelleme ile ilgili olmasıdır.

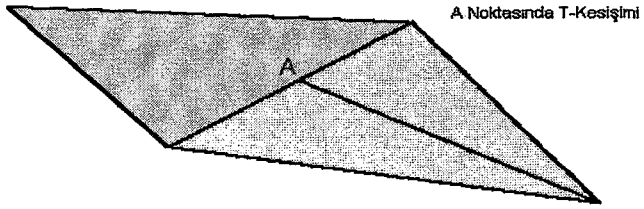
OpenGL bir modelleme aracı değil, bir çeviricidir. Modelleme işlerini destekleyen ve yardım eden OpenGL Utility Library (GLU) gibi bazı yararlı kütüphaneler vardır, fakat bütün pratik amaçlarda modelleme uygulayıcı sorumluluğundadır. Modellemede dikkat edilmesi gereken kurallara özen göstermek önemlidir, çünkü görüntü kalitesi doğrudan modelleme görüntüsü ile ilişkilidir. Örnek olarak parçalarına ayrılan (tessellate) model geometrileri sonucunda, model köşelerinde kötü gölge görüntüler (siluet) oluşmaktadır [5].

Diğer, manüel olarak oluşturulan model geometrilerinde ise görüntü, model ve OpenGL sıralama düzenine göre sonuçlanır. Örnek olarak, köşe noktalarındaki aydınlatma denkleminin değerlendirmesi sonucu ile belirlenen renk ara değerlemesi (interpolation) eğer geometri yeteri kadar örneklendirilmezse tatmin edilen değerin çok daha altında bir speküler parlaklık ile sonuçlanır [5]. Aşağıda OpenGL programcılarının aşina olması gereken ve modellerde dikkate alması gereken kısa bir liste bulunmaktadır:

a) Üçgenler, üçgen şeritleri ve üçgen yelpazeleri kullanımını ele alınsın. Bilgisayar grafik öğelerinden olan poligon ve dörtgenler OpenGL'de görüntülenmeden önce genellikle üçgenlere ayrıştırılır. OpenGL, bu ayrışımın nasıl olması gerektiği üzerine bir kontrol işlevi gerçekleştirmez. Bu yüzden daha tahmin edilebilir sonuçlar elde etmek için uygulama, şekillerin parçalarına ayrılması (tessellation) doğrudan yapmalıdır. Ayrıca eğer aynı model ekran üzerine birden fazla çizilecekse uygulama ayrımının yapılması daha verimli olur (örnek olarak her çerçeve (frame) için birçok örnek, çoklu geçiş (multipass) algoritmasının bir parçası olarak veya birden çok çerçeve

için olmak üzere ayrı ayrı uygulamalar). GLU kütüphanesinin ikinci sürümü (sürüm 1.1) çok iyi genel bir poligon ayrıştırıcı (tessellator) içerir [5].

b) T-kesişimlerinden (T-köşeleri olarak da adlandırılır) kaçınma: T-kesişimleri bir veya birden fazla üçgenin bir kısım kenarlarının, diğer bir üçgenlerle paylaşma girişimi durumunda ortaya çıkar. Şekil 4.16'da bu durum gösterilmektedir. Bu durumda geometri mükemmel bir şekilde hizalanmış olsa bile, dönüşüm sonrasında geometrinin kusursuz bir uyum göstermesi garanti edilememektedir. Her iki kenar da ortak noktaları paylaşmıyorsa üçgen çiziminde sınırlı-kesinlik (finite-precision) algoritmaları kullanıldığından, çizimlerde kenarlar her zaman için kusursuz şekilde hizalanmayabilir. Bu problem tipik olarak animasyonlarda model hareket ettiğinde kendini gösterir ve poligon kenarlarında çatlaklar (boşluklar) görünür ve kaybolur. Bu problemden sakınmak için, paylaşılan kenarların ortak köşe pozisyonlarını paylaşması gerekir ki kenar denklemlerinin aynı olması sağlansın. Ayrı gözüken modellerin bir kenarı paylaşmaları durumunda bu zorunluluğun sağlanması gerektiği dikkate alınmalıdır. Örneğin bir uygulama bir odanın iç yüzeyindeki duvarları ve tavanı birbirinden bağımsız olarak modelleyebilir. Fakat bunlar kesiştiklerinde ortak kenarları paylaşmaktadırlar. Oda değişik görüş açılarında sunulduğunda çatlak oluşumunu önlemek için paylaşılan kenar boyunca her üçgen için duvar ve tavanın aynı köşe koordinatlarını kullanması gerekir. Bu genellikle kenar eklemeyi ve kenarların ortak sınırlarının olmasını ve nesnelerin bir bütün olarak görünmesini sağlamak amacıyla yeni üçgenler yaratmayı gerektirir [5].



Şekil 4.16. T – kesişimi.

c) T-kesişim probleminin görüntüye bağlı parçalara ayırma (tessellation) işlemi için bazı neticeleri mevcuttur. Uç bir perspektifte bir nesneyi çizdiğimizi düşünelim. Öyle ki nesnenin küçük bir kısmı ekranın büyük bir kısmına haritalanmış ve elbette nesnenin büyük bir kısmı (nesne koordinatlarında) ekranın küçük bir kısmına haritalanmış olsun. Nesnenin çeviri (rendering) zamanını minimize etmek için, uygulamalar nesneyi ekranda kapladığı alana bağlı olarak değişik açılarla parçalarına ayrılır (tessellation). Bu, ekranda daha az piksel kaplayan üçgenlerin çiziminde, zamanın boşuna harcanmamasını sağlar. Bu, eğer nesnenin görüntüsü değişiyorsa doğru uygulama (implementation) için zor bir mekanizmadır, çünkü çerçeveden çerçeveye ayrışımdaki değişimler fark edilir hareket olguları (artifact) ile sonuçlanır. Genellikle izlenmesi gereken en iyi yol ya daha az parçalara ayırma yaptırarak (undertessellate) bu hareket olgularına göz yummak ya da gereğinden fazla parçalara ayırma yaptırarak (overtessellate) düşük performansı kabullenmektir. GLU NURBS Kütüphanesi, görünüme bağımlı ayrışımı gerçekleştiren ve örnekleme metodu ve ayrışım için tolerans üzerinde sağlam bir kontrol sağlayan bir paket örneğidir [5].

d) T-kesişimi ile ilişkin diğer bir problem de dikkatsizce belirlenen yüzey sınırları nedeniyle oluşur. Eğer bir yüzey kapalı olarak tasarlanmışsa, yüzey belirlemesinin başladığı ve bittiği yerde aynı köşe koordinatlarının paylaşılması gerekir. Buna basit bir örnek olarak köşe koordinatlarının $[0:2\Pi]$ aralığının bölümlenmesi ile çizilen bir küre çizimi verilebilir. 0 noktasındaki köşe değeri ile 2Π noktasındaki köşe değerinin aynı olması gerekir. OpenGL tanımlamaları bu konuda çok katıdır ve "glMapGrid" rutininin bile değerlendirilen yüzeylerin bir arada düzgün biçimde durmasını garanti etmek için kesin olarak sınırlarda değerlendirme yapması gerekir [5].

e) Göz önünde bulundurulması gereken diğer bir konu da köşe noktaları ile belirtilen niteliklerin, özellikle köşe (veya yüzey) normalleri ve doku (texture) koordinatlarının, kalitesidir. Bir nesnenin normalleri hesaplanırken, düz kenarların ortak normallerinin olması, diğer yandan keskin kenarların ortak noktalarda ayrı normallerinin olması gerekir. Örnek olarak, bir küp altı adet dörtgenden oluşur ve her köşe üç poligon tarafından paylaşılır fakat her köşede köşeyi paylaşan üç ayrı poligon için değişik normallerin kullanılması gerekir. Diğer yandan küre her köşenin ortak normalleri paylaştığı birçok poligondan oluşur. Niteliklerin düzgün biçimde belirlenememesi veya belirlerken hata yapılması sonucunda doğal olmayan aydınlatma etkileri veya ortam tasarımının hataları arttırması ile sonuçlanan kabul edilemeyen (doğal olmayan) gölgelendirme sonuçları ortaya çıkmaktadır [5].

f) Son bir öneri ise poligonların yönelimi (orientation) ile ilgili olacaktır. Dışarıdan görüntülendiğinde yüzeydeki her poligonun aynı yönde (saat yönü veya tersi) olmasını sağlamak gerekir. Bu tutarlılığın sağlanmasının amacını ifade etmek için en az iki tane sebep verilebilir. İlk olarak OpenGL dışbükey yüzeyler için gizli yüzeylerin elemesinde (hidden surface elimination) etkili bir form olarak yüzey seçim (culling) metodu kullanılabilir. İkinci olarak çeşitli algoritmalar bir yüzeyin sadece ön yüzü veya arka yüzünü çizmede seçici çizim yetisini ortadan kaldırabilir [5].

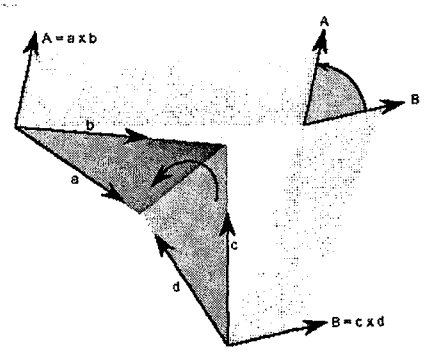
4.5.1. Ayrışım ve Parçalama (Decomposition & Tessellation)

Şekli parçalarına ayırma (tessellation), küre gibi karışık yüzeylerin üçgen veya dörtyüzlü gibi daha ilkel şekillere parçalanma sürecidir. Birçok OpenGL uygulaması üçgen şeritleri ve üçgen yelpazeleri kullanmak üzere ayarlanmıştır. Üçgenler düzlemsel oldukları, kolay taranabildikleri (rasterizing) ve her zaman için belirsiz olmayan ara değerlendirme yapılabildikleri (interpolated) için tercih edilirler [5].

Bir uygulama üçgenler oluşturmak için optimize edildiğinde, daha karmaşık öğelerden olan dördüz şeritleri (strips), dördüzler ve poligonlar işlemin başlangıç safhasında üçgenlere ayrıştırılırlar [5].

Eğer bir uygulama bu ayrışmayı bu öğe ile her karşılaşıldığı yerde değil de veri tabanı oluşturulurken veya gerçeklemenin başlangıç zamanında yapıyorsa, bu ayrışmaya öncelik verdiği için performans kazancı sağlanır. Uygulama kontrolü altında yapılan bu ayrışmanın ikinci bir avantajı da ayrışmanın düzenli aralıklarla ve OpenGL gerçeklemesinden bağımsız olarak yapılabilmesidir. OpenGL kendi ayrışma algoritmasını belirtmediğinden değişik gerçeklemeler verilen dörtyüzlüyü değişik köşegenler boyunca ayrıştırabilir. İki OpenGL gerçeklemesi ile çizilen bir çizim farklı biçimde gölgelendirilebilir ve farklı gölge görüntü (silhouette) köşelerine sahip bir görüntü ile sonuçlanabilir [5].

Dörtyüzlüler yönlendirmede maksimum fark olacak şekilde yaratılan iki üçgeni yaratan köşegeni bularak ayrıştırılabilirler. Bu köşegeni bulmanın iyi bir yolu karşılıklı köşelerin normalleri arasındaki açıyı bularak iç çarpım (dot product) yapmak ve daha sonra aralarındaki açı en büyük olan ikiliyi (en küçük iç çarpımı) seçmektir (Şekil 4.17'de görüldüğü gibi). Bir köşenin normalleri ise başlangıçları bu köşe olan iki vektörün çapraz çarpımları yapılarak bulunur. Alternatif bir ayrıştırma metodu ise dörtyüzlüleri boyut olarak birbirine en yakın olan üçgenlere bölmektir [5].



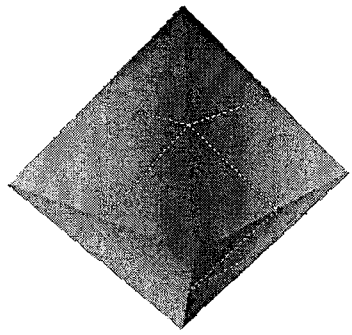
Şekil 4.17. Dörtyüzlü ayrışımı.

Küre ve silindir gibi basit yüzeylerin parçalarına ayrılması kolaydır. Küre için GLU kütüphanesi, birçok basit enlem-boylam ayrışımını kullanır. Algoritmayı yerine getirmek kolay olsa da bir dezavantajı vardır ki parçalama sonucu oluşan üçgenler değişik boyutta üretilirler. Bu oldukça değişik boyutlar özellikle nesne parlak ve döner ise fark edilir olgulara (artifact) neden olur [5].

Daha iyi bir algoritma daha tutarlı boyutlara sahip üçgenler oluşturur. Sekiz yüzlü ve yirmi yüzlü ayrışımlar iyi sonuçlar verir ve gerçeklemesi güç değildir. Sekiz yüzlü ayrışım, bir küreyi, köşeleri birim kürede olan sekiz yüzlüye yakınlaştırır. Sekiz yüzlünün yüzeyleri üçgen olduğundan bunlar kolayca dört üçgene ayrılabilir. Şekil 4.17'de bu ayrışım görülebilir.

Her üçgen, her kenarın ortasında yeni birer nokta yaratılarak ve üç adet yeni kenar eklenerek ayrılabilir. Bu noktalar başlangıca olan uzaklıklarına bölünerek (normalize edilerek) birim küreye oranlanır. Bu süreç istenildiği kadar tekrarlanabilir ve her üçgen tekrar tekrar bölünerek yaratılabilir.

Aynı algoritma yirmi yüzlüyü temel nesne olarak kullanarak, tekrar tekrar yirmi yüzü de bölerek de uygulanabilir. Her iki durumda da algoritmalar kodlanabilir ve böylece bağımsız üçgenler yerine üçgen şeritleri oluşturulur ve tarama (rendering) performansı maksimize edilmiş olur. Üçgen kenarlarını iki eş parçaya bölmek de zorunlu değildir, üçgenler farklı miktarlarda, örneğin üç veya daha farklı bir sayıda parçalara ayrılarak, daha arzu edilir tek biçimli üçgen boyutları oluşturulabilir. Şekil 4.18'de üçgensel parçalarına bölümlenen bir sekiz yüzlü gösterilmektedir.



Şekil 4.18. Üçgenlere ayrılan sekiz yüzlü.

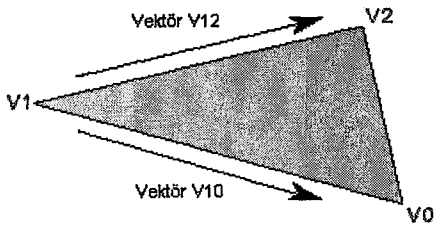
4.5.2. Model Normallerinin Oluşturulması

Herhangi bir poligon modelinin normalleri daha önceden hesaplanmamış ise, yüzler üzerinde gölgelendirme oluşturmak için poligon normallerini belirlemek, düzgün gölgelendirme için doğru köşe normallerini belirlemekten daha kolaydır. İki köşe vektörünün çapraz çarpımının (cross product) bulunması ve birim uzunlukta vektör elde etmek için bunu takiben elde edilen sonucun normalize edilmesi ile yüzey normali oluşturulur. Köşe normalini doğru olarak hesaplamak için, bu normali paylaşan bütün yüzeyleri ve bütün bu yüzeylerin bu normale katılıp katılmadığını dikkate almak gerekir [5].

Bir üçgenin yüzey normalini hesaplamak için, bir köşe noktası seçilir, bu köşeden diğer köşelere olan vektörler hesaplanır ve daha sonra bu iki vektörün çapraz çarpımı yapılır. Şekil 4.19'da yüzeyin normal vektörlerinin bulunması için kullanılan kenar vektörlerinin hesaplanması, şekil 4.20'de ise bu kenar vektörleri gösterilmektedir.

```
x10 = x1 - x0;
y10 = y1 - y0;
z10 = z1 - z0;
x12 = x1 - x2;
y12 = y1 - y2;
z12 = z1 - z2;
```

Şekil 4.19. Kenar vektörlerinin elde edilmesi.



Şekil 4.20. Kenar vektörleri.

Aşağıdaki şekil 4.21'de vektör çapraz çarpımının nasıl yapıldığı, şekil 4.22'de ise birim uzunlukta yüzey normali elde etmek için yapılan vektör çapraz çarpımlarının normalize edilme işlemlerinin nasıl yapıldığı gösterilmektedir.

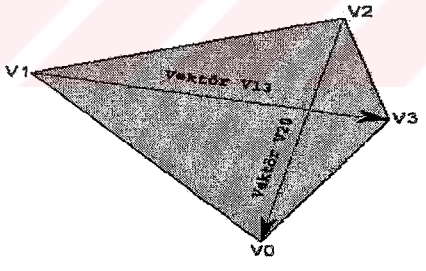
```
cpx = (z10 * y12) - (y10 * z12);
cpy = (x10 * z12) - (z10 * x12);
cpz = (y10 * x12) - (x10 * y12);
```

Şekil 4.21. Çapraz çarpımın gerçekleştirilmesi.

```
r = sqrt(cpx * cpx + cpy * cpy + cpz * cpz);
nx = cpx / r;
ny = cpy / r;
nz = cpz / r;
```

Şekil 4.22. Normalize model normalleri.

Üçten fazla köşesi olan bir poligonun yüzey normalini hesaplamak biraz daha zordur. Böyle poligonlar genelde tam düzlemsel değildir, bu yüzden seçtiğiniz köşe noktalarına göre farklı sonuçlar elde edebilirsiniz. Şekil 4.23'de gösterildiği gibi eğer poligon dört yüzlü ise karşılıklı köşelerin vektörlerinin çapraz çarpımını yapmak iyi bir metottur. Şekil 4.24'teki kod, dört yüzlü için çapraz çarpım hesaplar:



Şekil 4.23. Dörtgen bir poligon ve köşe vektörleri.

```
        x20 = x2 - x0;
        y20 = y2 - y0;
        z20 = z2 - z0;
        x13 = x1 - x3;
        y13 = y1 - y3;
        z13 = z1 - z3;

cpx = (z20 * y13) - (y20 * z13);
cpy = (x20 * z13) - (z20 * x13);
cpz = (y20 * x13) - (x20 * y13);
```

Şekil 4.24. Dörtgen poligon için köşe vektörlerinin ve çapraz çarpımın yapılması.

5. OPENGL – AÇIK GRAFİK KÜTÜPHANELERİ

OpenGL (Open Graphics Library), grafik donanımları için geliştirilen bir yazılım ara yüzüdür. Bu ara yüz, nesneleri ve etkileşimli 3-Boyutlu uygulamaları üretmek için ihtiyaç duyulan işlemlerde kullanılan 120 ayrı komuttan oluşmaktadır.

OpenGL, oluşturulan grafikler başka bir bilgisayar üzerinden de görüntülenebilecek şekilde tasarlanmıştır. Birbirine sayısal veri taşıma yeteneğine sahip kablolarla bağlı bir bilgisayar ağı ortamında, OpenGL komutlarının ve programın çalıştığı bilgisayar istemci (client) ve bu komutları alıp çizimi gerçekleştiren bilgisayara da sunucu (server) adı verilir. OpenGL komutlarının iletilme biçimi - protokolü her zaman aynı biçimdedir ve bu sayede OpenGL programları, istemci ve sunucu bilgisayarları farklı biçimlerde olsalar bile network üzerinde çalışabilirler. Eğer bir OpenGL programı network üzerinde çalışmıyorsa, bu durumda sadece bir bilgisayar var demektir ve bu bilgisayarın hem istemci hem de sunucu olduğu varsayılır [6].

OpenGL üç boyutlu nesnelerin tanımlanmasında yüksek seviyeli komutların kullanılmasını gerektirmez. OpenGL ile istenilen model, noktalar, çizgiler ve poligonlar ile oluşturulmaktadır.

5.1. OpenGL Komut Dizisi

OpenGL komutları, "gl" ön eki ve baş harfleri büyük olarak yazılan komutu oluşturan diğer kelimelerin birleşiminden oluşmaktadır (örneğin "glClearColor" gibi). Benzer olarak OpenGL'de sabitler "GL" ön eki ile başlar ve bütün harfler büyük harf olacak şekilde kullanılır [6]. Ayrıca ayrı sözcükler de alt çizgi birleştirilir (örneğin "GL_COLOR_BUFFER_BIT" gibi). Tablo 5.1'de komut ön ekleri ve değer veri tipleri verilmektedir.

Ön Ek	Veri Tipi	Karşılık Gelen C Dili Tipi	OpenGL Tipi Tanımı
B	8-bit integer	signed char	GLbyte
S	16-bit integer	Short	GLshort
İ	32-bit integer	Long	GLint, GLsizei
F	32-bit floating-point	Float	GLfloat, GLclampf
D	64-bit floating-point	Double	GLdouble, GLclampd
Ub	8-bit unsigned integer	unsigned char	GLubyte, GLboolean
Us	16-bit unsigned integer	unsigned short	GLushort
Ui	32-bit unsigned integer	unsigned long	GLuint, GLenum, GLbitfield

Tablo 5.1. Komut önekleri ve değer veri tipleri.

Aşağıdaki iki komut birbirine eşit olmasına rağmen birinci komut tepe noktasını 32 bitlik tamsayılarla ifade ederken ikinci komut bu noktayı 32 bitlik kayan (floating point) sayılar ile belirtmektedir [6].

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

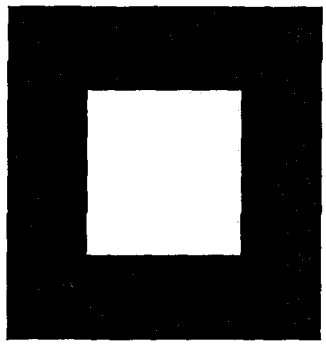
Şekil 5.1. Köşe noktalarını tanımlayan komut (ilki tamsayı, ikincisi kayan sayı versiyonu).

Bazı komutların ise vektör ile vektör olmayan adı altında iki versiyonu vardır. Vektör versiyonunda komutun sonuna "v" harfi eklenir [6]. Şekil 5.2'de ikisi de renk bileşenlerini tanımlayan komutlardan ilki vektör olmayan, ikincisi ise vektör olan türden versiyonları ile gösterimi sunulmaktadır.

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

Şekil 5.2. "glcolor" komutunun vektör ve vektör olmayan versiyonları.

OpenGL komutlarını daha yakından inceleyebilmek için aşağıdaki siyah zemin üzerine çizilmiş dikdörtgeni inceleyelim.



Şekil 5.3. Siyah zemin üzerinde beyaz dikdörtgen.

Şekil 5.4'teki kodun ilk iki satırındaki "main()" ve "OpenAWindowPlease()" komutları ekranda bir pencere açmak kullanılan rutin komutlardır. Bu komutları takip eden iki satırdaki komutlar ise ekranın ayarlanacağı rengi seçme ve ekranı silme komutlarıdır. "glClearColor()" komutu ekranın hangi renge ayarlanacağı hususunda gerekli bilgiyi içerirken, "glClear" bu bilgiyi eyleme dönüştüren komuttur. Benzer şekilde, "glColor3f()" komutu da çizilen nesnelerin hangi renkte olacağını belirler, bu örnekte renk beyaz olarak seçilmiştir.

Programda kullanılan bir sonraki OpenGL komutu olan "glOrtho()", çizilen şeklin ekranda nasıl haritalanacağı yani OpenGL in algılaması gereken koordinat sistemi belirler. "glBegin()" ve "glEnd()" komutlarının içindeki parantezli bölgedeki bilgiler çizilecek nesneyi belirtir. Poligonun köşeleri "glVertex2f()" komutları tarafından belirlenir. Son olarak "glFlush()" komutu çizim komutlarının çalışmasını sağlar [6].

```
#include <whateverYouNeed.h>

main()
{
        OpenAWindowPlease();

        glClearColor(0.0, 0.0, 0.0, 0.0);
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(1.0, 1.0, 1.0);
        glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);

        glBegin(GL_POLYGON);
```

```
            glVertex2f(-0.5, -0.5);
            glVertex2f(-0.5, 0.5);
            glVertex2f(0.5, 0.5);
            glVertex2f(0.5, -0.5);
        glEnd();
        glFlush();

        KeepTheWindowOnTheScreenForAWhile();
}
```

Şekil 5.4. Opengl ile yazılmış kod örneği.

5.2. Durum Makinesi Olarak OpenGL

OpenGL bir durum makinesidir. OpenGL programcının kontrolü altında farklı durumlarda çalışabilmekte, programcı bu durumları değiştirene kadar tüm değişkenler etkisini korumaktadır. Örneğin kullanılan renkler birer durum değişkenidir. O anda kullanılan renk beyaza, kırmızıya ve herhangi renge ayarlanabilir ve bu ayarlar başka bir renge değiştirilene kadar nesneler bu renklerle çizilir. Kullanılan renk OpenGL'in muhafaza ettiği birçok durum değişkeninden sadece birisidir. Diğer durum değişkenleri (ya da modlar), "glEnable()" ve "glDisable()" komutları ile etkin hale getirilebilir [6].

5.3. OpenGL Kütüphaneleri

OpenGL güçlü fakat ilkel yardımcı komutlardan oluşur ve bütün yüksek seviyeli çizimler bu komutlar sayesinde oluşur. Bu nedenle programcılar, programlama işlerinin basite indirgemek amacıyla OpenGL'in üzerine kendi kütüphanelerini yazabilmektedir. Ayrıca, OpenGL programının pencereleme sistemi ile kolayca çalışmasına imkân tanıyacak bazı rutinler de yazılabilir. Aslında birçok kütüphane ve rutinler bazı özelliklerin sağlanması için zaten yazılmış durumdadır. OpenGL yardımcı kütüphanesi (Auxiliary Library – Glaux) bu türden bir kütüphane olarak OpenGL'in etkili biçimde ve kolayca kullanılmasını sağlayacak rutinlere sahiptir.

Aşağıdaki bölümde, OpenGL yardımcı kütüphanesinde kullanılan bazı rutinlerin kısaca açıklamaları verilmiştir.

5.3.1. Pencere yönetimi

Bir pencereyi başlatmak ve açmak için kullanılan rutinler:

- auxInitWindow(), ekranda bir pencere açar. Bu rutin Escape tuşu ile programdan çıkmayı mümkün kılar ve zemin rengini siyah olarak ayarlar.
- auxInitPosition(), auxInitWindow() komutuna pencerenin ekranın neresinde olacağı bilgisini verir.
- auxInitDisplayMode(), auxInitWindow() komutuna RGBA veya renk indeksi penceresi açıp açmamasını bildirir.

5.3.2. Manüel giriş olayları

- auxReshapeFunc() komutu pencere yeniden boyutlandırıldığında, hareket ettirildiğinde ya da meydana çıkarıldığında hangi aksiyonun yapılacağını belirler.
- auxKeyFunc() ve auxMouseFunc() bir klavye tuşuna basıldığında ya da fareye tıklandığında aksiyon oluşmasına izin verir.

5.4. Üç-boyutlu cisimlerin çizimi

Üç boyutlu cisimleri çevre çizgileri ile veya katı gölgeli çizerken öncellikle yüzey normalleri tanımlanmalıdır. Örnek vermek gerekirse bir küre ve bir torus için bazı rutinler Şekil 5.5'te verilmiştir.

```
void auxWireSphere(Gldouble radius);
void auxSolidSphere(Gldouble radius);
void auxWireSphere(Gldouble radius);
void auxWireTorus(GLdouble innerRadius, GLdouble outerRadius);
void auxSolidTorus(GLdouble innerRadius, GLdouble outerRadius);
```

Şekil 5.5. Opengl yardımcı kütüphanesinin bazı küre ve torus rutinleri.

5.5. Geometrik Nesnelerin Çizimi

5.5.1. Çizim öncesi hazırlıklar

Bu bölümde çizim işlemine başlamadan önce, pencerenin nasıl temizlendiği, çizilecek nesnelerin renginin nasıl hazırlandığını ve bunlara benzer gibi yardımcı olaylar incelenecektir. Bu konulardan hiç birisinin geometrik nesnelerle direkt bağlantısı olmamasına rağmen, geometrik nesne çizen her programın bu konularla ilgilenmesi gerekmektedir.

5.5.2. Pencerenin temizlenmesi

Şekil 5.6 ile gösterilen kodda ilk satır, silme işleminin siyaha yapılacağını yani ekranın silindikten sonra zemin renginin siyah olacağını ayarlar ve sonraki satırdaki komut ise bu ayarlanan renge göre ekranı siler. "glClear" komutundaki tek parametre hangi tamponların temizleneceğini belirler. Bu örnekte program sadece renk tamponlarını temizlemektedir. Tipik olarak öncelikle temizleme rengi seçilir ve daha sonra da temizlenmesi gerekli olan renk tamponları temizlenir.

```
glClearColor(0.0, 0.0, 0.0, 0.0);
glClear(GL COLOR BUFFER BIT);
```

Şekil 5.6. OpenGL ekranının temizlenmesi.

Şekil 5.7'deki örnekte "glClearColor()" önceki örnekteki aynısı olup, "glClearDepth()" komutu ise ayarlanacak derinlik tamponundaki her bir pikselin değerini belirler. "glClear()" bu durumda temizlenecek bütün tamponların mantıksal olarak VEYA bilgisini içerir. Tablo 5.2'de OpenGL'de kullanılan tamponlar ile komutları verilmektedir.

```
glClearColor(0.0, 0.0, 0.0, 0.0);
glClearDepth(0.0);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Şekil 5.7. OpenGL renk ve derinlik tamponlarının silinmesi.

Tampon	İsim
Renk tamponu	GL_COLOR_BUFFER_BIT
Derinlik tamponu	GL_DEPTH_BUFFER_BIT
Toplanma tamponu	GL_ACCUM_BUFFER_BIT
Kalıp tamponu	GL_STENCIL_BUFFER_BIT

Tablo 5.2. OpenGL tamponları.

5.5.3. Renk Belirleme

Genel olarak bir OpenGL programcısı öncelikle bir renk belirler ve daha sonra nesnesini çizer. Renk veya renk şeması değişene kadar bütün nesneler aynı renk ile çizilir. Örneğin;

```
set_current_color(red);
draw_object(A);
draw_object(B);
set_current_color(green);
set_current_color(blue);
draw_object(C);
```

Şekil 5.8. Çizilecek nesnenin rengini belirlemek.

Şekil 5.8'deki örnekte A ve B nesneleri kırmızı renkte ve C nesnesi mavi renkte çizilmiştir. Dördüncü satırdaki rengi yeşile çeviren komut ise boşa kullanılmıştır.

Bir renk ayarlaması yaparken "glColor3f()" komutu kullanılabilir. Komut içerisinde sırasıyla kırmızı (red), yeşil (green) ve mavi (blue) renk katsayıları girilerek ayarlanması istenen renk seçilir. Bu komut içerisine 0.0 ve 1.0 değerleri arasında değerler kullanılması ile renk belirlemesi yapmak mümkün olmaktadır. [0.0] değerinin kullanılmasının manası hiçbir bileşenin kullanılmayacağı, [1.0] değerinin kullanılmasının manası ise bütün o bileşenin kullanılacağı manasına gelmektedir. Böylelikle glColor3f(1.0, 0.0, 0.0) komutunda hiçbir yeşil veya mavi bileşen kullanılmazken sistemin çizimde kullanması için kırmızı rengi seçilmiş olur. Bütün sıfırlar siyahı, bütün birler ise beyazı oluşturur denebilir. Şekil 5.9'da 8 renk için seçenekler verilmiştir.

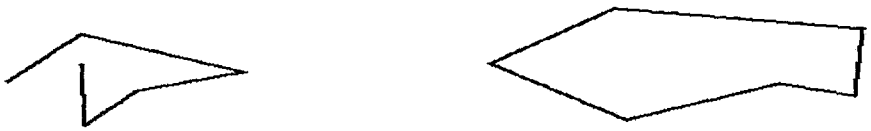
```
glColor3f(0.0, 0.0, 0.0);   siyah
glColor3f(1.0, 0.0, 0.0);   kırmızı
glColor3f(0.0, 1.0, 0.0);   yeşil
glColor3f(1.0, 1.0, 0.0);   sarı
glColor3f(0.0, 0.0, 1.0);   mavi
glColor3f(1.0, 0.0, 1.0);   mor
glColor3f(0.0, 1.0, 1.0);   cyan
glColor3f(1.0, 1.0, 1.0);   beyaz
```

Şekil 5.9. OpenGL'de kullanılan temel renklerin katsayıları.

5.5.4. Nokta, çizgi ve poligon belirleme

- Noktalar: Bir nokta, virgüllü sayılardan oluşan köşe noktalarının bir kümesi ile temsil edilir. Tüm dâhili hesaplamalar köşe noktaları üç boyutlu koordinatlardan oluşuyorsa yapılır. Kullanıcı tarafından iki boyutlu (sadece x ve y koordinatları) köşe noktaları girilmiş ise z koordinatı OpenGL tarafından sıfıra eşitlenerek kullanılır [6].
- Çizgiler: OpenGL'de çizgi teriminin anlamı matematikte her iki doğrultuda sonsuz uzunlukta doğru olarak ifade edilen terimin aksine yalnızca sonlu uzunluktaki çizgi parçası demektir. Birleşmiş, seri halindeki çizgi parçalarını

(ki bunlar kapalı da olabilir) belirtmenin kolay yolları mevcuttur. Şekil 5.10'da iki adet birleşmiş, seri halinde çizgi parçaları gösterilmektedir.



Şekil 5.10. İki adet birleşmiş, seri halinde çizgi parçaları.

- Poligonlar: Poligonlar tipik olarak iç kısımdaki pikseller ile çizilir fakat noktalar kümesi olarak da çizmek mümkündür. OpenGL de poligonların kenarları birbirleri ile kesişemez. Buna matematikte basit poligon adı verilir. İkinci olarak OpenGL poligonları konveks yani dışbükey olmalıdır [6]. Şekil.5.11'de bazı geçerli ve geçerli olmayan poligon çizimleri verilmiştir.

Şekil 5.11. Geçerli olan (sol) ve olmayan (sağ) poligon çizimleri.

- Dörtgenler: Dörtgenler grafik uygulamalarında yaygın olarak kullanıldığından, OpenGL, içi dolu dörtgen çizmek için "glRect*()" komutunu önermektedir [6]. Dörtgeni bir poligon gibi çizebiliriz. OpenGL'de dörtgen çizdirmek için Şekil 5.12'deki kod formatlarından birisi kullanılmalıdır. Bu komutlar köşe noktaları (x1,y1) ve (x2,y2) olan dörtgeni çizer. Dörtgen "z = 0" düzlemi üzerindedir ve x ve y eksenlerine paralel kenarları vardır. Fonksiyondaki vektör kullanılırsa, köşeleri iki işaretçi tarafından verilen her biri bir (x,y) çifti içerir.

```
void glRect{sifd}(TYPEx1, TYPEy1, TYPEx2, TYPEy2);
void glRect{sifd}v(TYPE*v1, TYPE*v2);
```

Şekil 5.12. OpenGL'de dörtgen çizme.

- Eğriler: Herhangi bir eğri çizgi veya yüzey kısa çizgi parçaları ve küçük poligonsal alanlar ile oluşturulabilir. Kısa çizgi parçalarının birleşmesi ile bu yüzey eğri gibi görünür. Net bir eğri görüntüsü için her bir çizgi parçası ya da poligon ekrandaki bir pikselden bile daha küçük olmalıdır. Şekil 5.13'te OpenGL'de eğrileri oluştururken kullanılan yaklaşım gösterilmiştir.

Şekil 5.13. Eğrileri oluştururken kullanılan yaklaşım.

5.5.5. Köşe noktalarını belirleme

OpenGL ile bütün geometrik nesneler köşe noktalarının sıralı bir kümesi olarak tanımlanmıştır. Bir köşe noktasını belirlemek için "glVertex*()" komutu kullanılır. "glVertex*()" komutu "glBegin()" ve "glEnd()" çifti arasında kullanılır [6]. Şekil 5.14'te "glVertex()" komutunun kullanım biçimi ve birkaç örnek sunulmaktadır.

```
void glVertex{234}{sifd}[v](TYPE coords);   // Komutun Kullanımı

glVertex2s(2, 3);                           // 2 Boyutta Kullanım
glVertex3d(0.0, 0.0, 3.1415926535898);      // 3 Boyutta Kullanım

GLdouble dvect[3] = {5.0, 9.0, 1992.0};     // Vektörlü Kullanım
glVertex3dv(dvect);
```

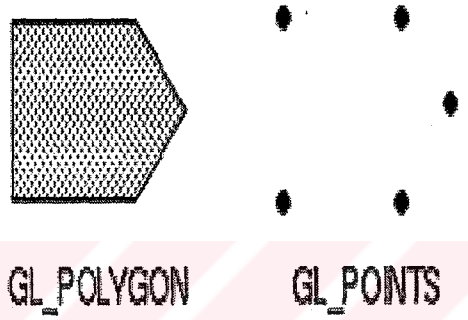
Şekil 5.14. Opengl'de köşe noktası oluşturma.

Birinci ikinci ve üçüncü satırlarda parantez içindeki sayılar köşe noktasının koordinatlarını vermektedir. Son örnekte ise, "dvect" bir dizinin işaretçisi olarak kullanılmıştır.

OpenGL ile noktalar kümesi, çizgi veya poligon oluşturmak için her bir köşe noktası kümesi "glBegin()" ve "glEnd()" parantezleri içinde yer almalıdır. glBegin() den sonra gelen komutlar ne tip geometrik şeklin çizileceğine karar verilmesi sağlar. Şekil 5.15'te çizilmesi istenen nesnenin farklı geometriler ile çizilmesini göstermektedir. glBegin() komutuna girilen sabit, "GL_POLYGON" ise düzgün bir dışbükey beşgen çizilmekte, "GL_POINTS" ise aynı beşgen yalnızca bulunduğu noktalar itibariyle ifade edilmektedir [6].

```
glBegin(GL_POLYGON);
    glVertex2f(0.0, 0.0);
    glVertex2f(0.0, 3.0);
    glVertex2f(3.0, 3.0);
    glVertex2f(4.0, 1.5);
    glVertex2f(3.0, 0.0);
glEnd();

glBegin(GL_POINTS);
    glVertex2f(0.0, 0.0);
    glVertex2f(0.0, 3.0);
    glVertex2f(3.0, 3.0);
    glVertex2f(4.0, 1.5);
    glVertex2f(3.0, 0.0);
glEnd();
```



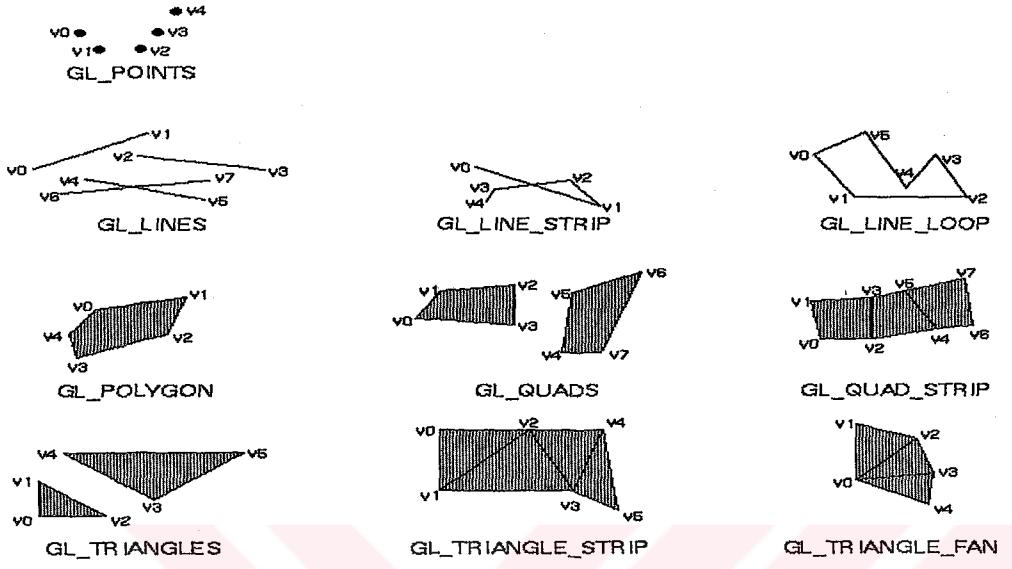
Şekil 5.15. OpenGL'de öğelerin çizilmesi.

Tablo 5.3'te glBegin() komutu içinde kullanılan 10 adet argüman tipi verilmiştir.

Değer	Anlamı
GL_POINTS	Bağımsız noktalar
GL_LINES	Köşe noktaları çiftleri ile belirtilen bağımsız çizgi parçaları
GL_POLYGON	Dışbükey poligon
GL_TRIANGLES	Köşe noktaları üçlüleri ile belirtilen üçgenler
GL_QUADS	Köşe noktaları dörtlüleri ile belirtilen dört-yüzlü poligonlar
GL_LINE_STRIP	Birbirine bağlı çizgi parçaları serisi
GL_LINE_LOOP	İlk ve son köşeleri de birbirine bağlı çizgi parçaları kapalı serisi
GL_TRIANGLE_STRIP	Bağlanmış üçgen serisi
GL_TRIANGLE_FAN	Birbirine bağlı üçgen yelpazeleri
GL_QUAD_STRIP	Bağlanmış dörtgen serisi

Tablo 5.3. Geometrik öğelerin isimleri ve anlamları.

Şekil 5.16'daki gösterimler Tablo 5.3'te verilen öğelerin şekillerle gösterimini sunmaktadır.



Şekil 5.16. OpenGL'de öğelerin gösterimi.

5.5.6. Görüntüleme

Görüntüleme dönüşümü belirlenmeden önce matris, tanımlama (identity) matrisine ayarlanır. Bu olay "glLoadIndetity()" komutu ile sağlanır. Bu komutu takiben "glTranslate()" komutu sayesinde görüntüleme dönüşümü belirlenir. Eğer kamera başka bir yöne doğru döndürülecekse "glRotate()" komutu ile bakış açısı değiştirilebilir.

Eğer gerçekçi resimler oluşturulmaya çalışılıyorsa perspektif iz düşümler kullanılmak istenebilir. Bu durumda "glFrustum()" komutu kullanmak uygundur. Bu komut önceki bölümlerde de anlatıldığı üzere bir bakış biçimi tipidir ve iz düşüm matrisinde değişiklik gerektirir. glFrustum() komutunu kullanmadan önce bazı hazırlıklar yapılmalıdır. İlk olarak matris modunda değişiklik gerçekleştiren "glMatrixMode()" komutu iz düşüm kullanmak üzere "GL_PROJECTION" argümanı ile birlikte kullanılır. Daha sonra "glMatrixMode()" bu kez görüntüleme

argümanı alarak yeniden çağrılır ve elbette bu sefer "GL_MODELVIEW" argümanı ile birlikte kullanılmalıdır [6].

5.6. Genel Amaçlı Dönüşüm Komutları

Şu ana kadar "glMatrixMode()" ve "glLoadIdentity()" komutlarını tanıtıldı. Diğer iki komut ise "glLoadMatrix()" ve "glMultMatrix()" komutlarıdır. Bunlar herhangi bir dönüşüm matrisi belirlenmesine ve daha sonra bu belirlenmiş matrisin mevcut matris ile çarpılmasını sağlar. Dönüşüm matrisinin seçimi Şekil 5.17'de gösterilmiştir.

```
void glMatrixMode(Glenum mode);
```

Şekil 5.17. Dönüşüm matrisinin seçimi.

"Mode" parametresi için "GL_TEXTURE", "GL_PROJECTION", "GL_MODELVIEW" argümanlarından birini kullanarak matris düzenlenebilir.Şekil 5.18'deki kod serisinde dönüşüm işlemlerinin uygulanması gösterilmektedir. Model görüntüleme (modelview) matrisi "I" tanımlama matrisini temsil ederken sırasıyla "N", "NM", ve son olarak "NML" matrislerini içermektedir. Dönüştürülmüş köşe noktası ise "NMLv" sonuç matrisine karşılık gelecektir. Bu yüzden köşe noktası dönüşümüm "N(M(Lv))" olur. "V" matrisi önce "L" matrisi ile, sonra sonuçta ortaya çıkan "Lv" matrisi "M" matrisi ile, en sonunda "MLv" matrisi ise "N" matrisi ile çarpılır.

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(N);              /* N dönüşümü uygula */
glMultMatrixf(M);              /* M dönüşümü uygula */
glMultMatrixf(L);              /* L dönüşümü uygula */

glBegin(GL_POINTS);
glVertex3f(v);                 /* Dönüştürülmüş v tepe noktasını çiz*/
glEnd();
```

Şekil 5.18. Dönüşüm matrisinin kullanıldığı örnek kod.

Nesnenin işlemler sonrasında eksen takımı üzerinde görünmesi isteniyorsa önce rotasyon bunu takiben de yer değiştirme dönüşümü yapılmalıdır. Bu durumda kod Şekil 5.19'daki şekilde oluşur; (R rotasyon, T dönüşüm matrislerini göstersin.)

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(T);
glMultMatrixf(R);
draw the object();
```

Şekil 5.19. Rotasyon ve yer değiştirme dönüşümleri.

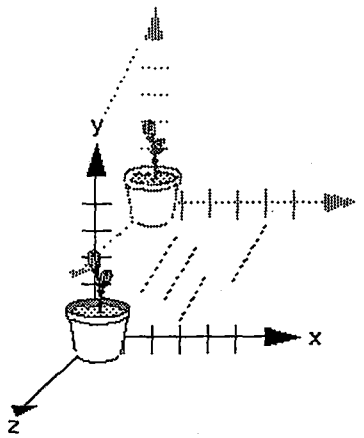
5.6.1. Model Dönüşümleri

- Dönüşüm: Bu fonksiyonu mevcut matrisi yeni bir matrisle çarpar, böylelikle nesne x, y, z koordinat değerleri ile verilen dönüşüme göre hareket edecektir. Dönüşüm fonksiyonunun kullanımı Şekil 5.20'de gösterilmektedir.

```
void glTranslate{fd}(TYPEx, TYPE y, TYPEz);
```

Şekil 5.20. Dönüşüm fonksiyonunun kullanımı.

Bu etki Şekil 5.21 ile görülmektedir.



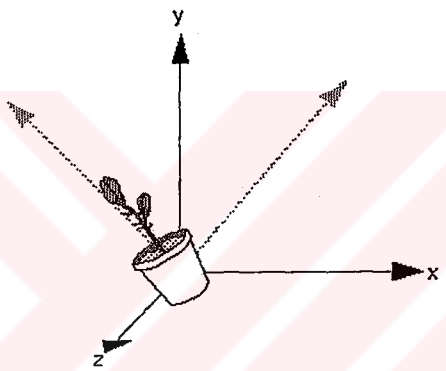
Şekil 5.21. Bir nesnede dönüşüm etkisi.

- Döndürme: Bu fonksiyonu, nesneyi orijinden (x,y,z) noktasına doğru uzanan ışın yönünde saat yönünün tersine döndürür. Fonksiyonun kullanımı Şekil 5.22'deki şekildedir. Döndürme etkisi Şekil 5.23 ile gösterilmektedir.

```
void glRotate{fd}(TYPE angle, TYPE x, TYPE y, TYPE z);
```

Şekil 5.22. Döndürme fonksiyonunun kullanımı.

Fonksiyondaki "angle" parametresi, dönme işleminin derece cinsinden açısını belirler.



Şekil 5.23. Bir nesneyi döndürme etkisi.

5.7. Aydınlatma

5.7.1. Işık Kaynaklarını Oluşturmak

Işık kaynaklarının renk, konum ve doğrultu gibi pek çok özelliği vardır. Bu bölümde bu özelliklerin nasıl kontrol edildiği ve elde edilen ışığın nasıl göründüğünü inceleyeceğiz. Işıkların özelliklerinin tümünü belirlemek için kullanılan komut glLight*() komutudur ve üç parametre alır. Komutun kullanımı Şekil 5.24'de olduğu gibidir.

```
void glLight{if}[v](GLenum light, GLenum pname, TYPEparam);
```

Şekil 5.24. Işık kaynağı oluşturmak.

Light parametresi GL_LIGHT0'dan GL_LIGHT8'e kadar değerler alarak oluşturulacak olan ışığı belirtir. Işığın karakteristiği aşağıdaki tablodaki parametrelerden birini içeren pname parametresi ile belirlenir. Param parametresi seçilen pname karakteristiğinin değerini belirtmektedir.

Parametre	Varsayılan Değer	Anlamı
GL_AMBIENT	(0.0, 0.0, 0.0, 1.0)	Işığın çevresel RGBA yoğunluğu
GL_DIFFUSE	(1.0, 1.0, 1.0, 1.0)	Işığın dağılım RGBA yoğunluğu
GL_SPECULAR	(1.0, 1.0, 1.0, 1.0)	Işığın spekküler RGBA yoğunluğu
GL_POSITION	(0.0, 0.0, 1.0, 0.0)	(x, y, z, w) olarak ışığın konumu
GL_SPOT_DIRECTION	(0.0, 0.0, -1.0)	(x, y, z) olarak ışıldak doğrultusu
GL_SPOT_EXPONENT	0.0	Işıldağın üssü
GL_SPOT_CUTOFF	180.0	Işıldak kesim açısı
GL_CONSTANT_ATTENUATION	1.0	Sabit zayıflama faktörü
GL_LINEAR_ATTENUATION	0.0	Lineer zayıflama faktörü
GL_QUADRATIC_ATTENUATION	0.0	Kuadratik zayıflama faktörü

Tablo 5.4. Işık kaynağı karakteristiği.

Tablo 5.4'te ifade edilen "GL_DIFFUSE" ve "GL_SPECULAR" parametreleri için verilen varsayılan değerler yalnızca "GL_LIGHT0"a uygulanabilir. Diğer ışık kaynakları için bu iki parametrenin de varsayılan değerleri (0.0, 0.0, 0.0, 1.0)'dır.

Şekil 5.25'te "glLight*()" komutunun kullanımına ilişkin bir örnek verilmektedir.

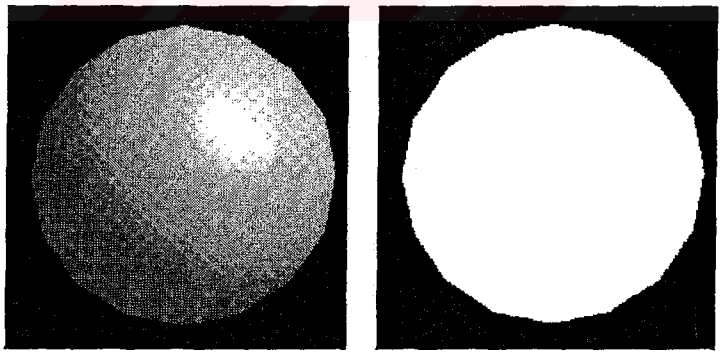
```
GLfloat light_ambient[] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };

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

Şekil 5.25. Işık kaynağı üretme kod örneği.

Görüldüğü gibi, parametre değerleri için diziler tanımlanmış ve glLightfv() komutu çeşitli parametreleri ayarlamak için tekrarlanarak kullanılmıştır. Bu örneğin ilk üç glLightfv() çağrısı parametrelerin varsayılan değerleri kullanıldığı için lüzumsuz yere kullanılmıştır.

Tüm ışık kaynakları da OpenGL'de kullanılan pek çok özellik gibi glEnable() komutu ile etkinleştirilir ve glDisable() komutu ile etkisiz kılınır [6].



Şekil 5.26. Aydınlatılmış ve aydınlatılmamış iki küre.

5.7.2. Işık Rengi

OpenGL herhangi bir ışığa "GL_AMBIENT", "GL_DIFFUSE" ve "GL_SPECULAR" olmak üzere üç farklı ışıkla-ilişkili parametreyi ilişkilendirmenizi

sağlar. "GL_AMBIENT" parametresi, ekrandaki belirli bir ışık kaynağının çevresel ışığının "RGBA" yoğunluğu ile ilgilidir. Tablo 5.4'te görüldüğü gibi "GL_AMBIENT" parametre değeri varsayılan (0.0, 0.0, 0.0, 1.0) olduğu sürece hiçbir çevresel ışık olmayacaktır. Fakat, örneğin bir mavi çevresel ışık isteniyorsa bu parametre değeri Şekil 5.27'deki gibi değiştirilebilir.

```
GLfloat light_ambient[] = { 0.0, 0.0, 1.0, 1.0};
glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
```

Şekil 5.27. Mavi renkli çevresel ışık kaynağı.

GL_DIFFUSE parametresi doğal olarak "ışık rengi" olarak ilişkilendirilecek en muhtemel parametredir ve belirli bir ışık kaynağının dağınık ışığının RGBA rengini tanımlamaktadır. Varsayılan olarak "GL_DIFFUSE", "GL_LIGHT0" için (1.0, 1.0, 1.0, 1.0) değeri ile beyaz ışık üretir. (GL_LIGHT1, ... ,GL_LIGHT7) gibi diğer ışık kaynakları için bu değer (0.0, 0.0, 0.0, 0.0)'dır.

"GL_SPECULAR" parametresi nesne üzerindeki speküler parlaklığın rengini etkiler. Tipik olarak, cam bir bardak gibi gerçek-dünya nesneleri üzerlerindeki ışık rengi olarak (genellikle beyaz) speküler parlaklığa sahiptir. Bu nedenle, gerçekçi bir etki istiyorsak "GL_SPECULAR" parametresi ile "GL_DIFFUSE" parametresini aynı değere ayarlamalıyız. Varsayılan olarak, "GL_SPECULAR" parametresi "GL_LIGHT0" için (1.0, 1.0, 1.0, 1.0) değerinde ve diğer ışık kaynakları için (0.0, 0.0, 0.0, 0.0) değerindedir.

5.7.3. Konumlandırma ve Zayıflama

Daha önce de incelendiği gibi, ışık kaynağının ekrandan çok uzak veya ekrana çok yakın olması belirlenebilmektedir. Işık kaynağının, ekrandan sonsuz uzaklıkta konumlandırıldığı düşünüldüğünde bu tip ışık kaynağının yönlendirilmiş ışık kaynağı (directional) olarak düşünülmesi ve nesne üzerine düşen ışınların paralel olduğunun

düşünülmesi gerekecektir. Gerçek-dünya yönlendirilmiş ışık kaynağına örnek olarak, güneş gösterilebilir.

İkinci tip konumlama olarak yakın bir noktaya konumlama gösterilebilir ve bu tip bir ışık kaynağı konumlandırılmış (positional) ışık kaynağı olarak adlandırılır ve ışığın konumu ekran üzerinde ışığın etkisini ve ışık ışınlarının gelme doğrultularını belirler [6]. Bir masa lambası konumlandırılmış ışık kaynağına örnek olarak gösterilebilir. Şekil 5.28'de ışığın konumlandırılmasının nasıl yapılması gerektiği gösterilmektedir.

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

Şekil 5.28. Mavi renkli çevresel ışık kaynağı.

Görüldüğü gibi, "GL_POSITION" parametresi için (x, y, z, w) değerlerini içeren 4 değerli bir vektör tanımlıyoruz. Son değerin sıfır olması durumunda karşılık gelen ışık kaynağı yönlendirilmiş bir ışık kaynağı olacak ve (x, y, z) değerleri doğrultusunda çalışacaktır. Eğer w parametresi sıfırdan farklı değerlikli ise ışık konumlandırılmış bir ışık kaynağı olacak ve (x, y, z) değerleri ışığın konumunu belirtecektir.

Gerçek-dünya ışıkları için, ışığın yoğunluğu ışığın mesafesi arttıkça azalacaktır. Yönlendirilmiş ışık kaynağı sonsuz uzaklıkta düşünüldüğünden zayıflama bu kaynak için etkisiz kılınacaktır. Bununla beraber istediğiniz takdirde ışık kaynaklarınıza zayıflama etkisi katabilirsiniz. OpenGL, ışık kaynaklarını Şekil 5.29'da ifade edilen zayıflama faktörü katkısı ile çarparak zayıflatır.

$$\text{Zayıflama Faktörü} = \frac{1}{k_c + k_l d + k_q d^2}$$

Şekil 5.29. Opengl zayıflama faktörü.

Bu formülde d, poligonun köşe noktasının ışık konumuna mesafesine; "kc", "GL_CONSTANT_ATTENUATION" parametresine, "kl", "GL_LINEAR_ATTENUATION" parametre değerine ve "kq", "GL_QUADRATIC_ATTENUATION" parametresine karşılık gelmektedir. Varsayılan olarak "kc" 1.0, "kl" ve "kq" 0.0 değerindedir fakat bu parametrelere Şekil 5.30'da olduğu gibi farklı değerler atanabilir.

```
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 2.0);
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 1.0);
glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.5);
```

Şekil 5.30. Zayıflama faktörü katsayılarının değiştirilmesi.

Unutulmaması gereken bir nokta ise çevresel, dağılımsal ve speküler ışık katkılarının tamamı zayıflayan türdendir. Yalnızca emisyon ve global çevresel değerler zayıflamamaktadır.

5.7.4. Çoklu Işık Kaynakları

Bahsedildiği gibi en az sekiz ışık kaynağına sahip olunabilir. Elbette OpenGL uygulamalarına bağlı olarak daha fazla ışık kaynağı kullanılabilir. Işık kaynaklarının artmasıyla her köşe noktasının alması gereken ışığın OpenGL tarafından hesaplanması gerekecek ve bu da performansı etkileyecektir.

Sekiz ışık kaynağını belirten sabitler "GL_LIGHT0", "GL_LIGHT1", ... , "GL_LIGHT7"'dir. Önceki ifadelerde parametrelerin tümü "GL_LIGHT0" ışık kaynağı için ayarlanmıştı. Eğer ilave ışık kaynağı konulacaksa parametrelerini ayarlamak gerekecektir ve hatırlanmalı ki Tablo 5.4'teki varsayılan değerlerin tümü "GL_LIGHT0" için olduğundan diğer kaynakların varsayılanları bu değerlerden farklıdır. Şekil 5.31'de konumlandırılmış, zayıflayan ve beyaz renkli bir ışık kaynağı oluşturulması ve etkinleştirilmesi için hazırlanmış kod örneği gösterilmektedir.

```
GLfloat light1_ambient[] = { 0.2, 0.2, 0.2, 1.0 };
GLfloat light1_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light1_specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light1_position[] = { -2.0, 2.0, 1.0, 1.0 };

glLightfv(GL_LIGHT1, GL_AMBIENT, light1_ambient);
glLightfv(GL_LIGHT1, GL_DIFFUSE, light1_diffuse);
glLightfv(GL_LIGHT1, GL_SPECULAR, light1_specular);
glLightfv(GL_LIGHT1, GL_POSITION, light1_position);
glLightf(GL_LIGHT1, GL_CONSTANT_ATTENUATION, 1.5);
glLightf(GL_LIGHT1, GL_LINEAR_ATTENUATION, 0.5);
glLightf(GL_LIGHT1, GL_QUADRATIC_ATTENUATION, 0.2);
glEnable(GL_LIGHT1);
```

Şekil 5.31. Işık kaynağı oluşturma ve etkinleştirme kod örneği.

6. ÜÇ-BOYUTLU MODELLERİN ETKİLEŞİMLİ GÖRÜNTÜLENMESİ

6.1. 3-Boyutlu Model Görüntüleme Sistemi

3-Boyutlu model görüntüleme sistemi tasarlanırken esneklik ön planda tutulmuştur. Ele alınabilecek modellerin sınırlanmasının aksine, sadece üçgen değil, keyfi poligon yüzlerini (facet) işleyebilen bir yapı kullanılmış, köşelere, kenarlara ve poligonlara rasgele erişim sağlanmasına olanak tanınmıştır. Bununla beraber kolay kullanılabilir bir ara yüz ve basit etkileşim ön planda tutulmuştur. Görüntüleme sistemi tasarımında Microsoft Visual C++ 6.0 programlama dili temel olarak kullanılmış ve gerekli olan grafik fonksiyonları için önceki bölümde anlatılan OpenGL kütüphaneleri kullanılmıştır. Sistemin grafiksel kullanıcı ara yüzünü oluşturmak için mui (Micro User Interface) kütüphanesinin fonksiyonlarından yararlanılmıştır. Bu bölümde sisteme modellerin tanıtılması, modellere gerçek-dünya özelliklerinin verilmesi, kullanıcı kontrolü sayesinde modellerin etkileşimli olarak nasıl görüntülendiği konuları incelenecektir.

OpenGL grafik kütüphanelerinden OpenGL32.lib, Glu32.lib, Glaux.lib kütüphaneleri modeli desteklemek için gereksinim duyulan renklendirme, ışıklandırma, döndürme, dolgu kaplama gibi pek çok işlemi destekleyerek kolaylık sağlamakta ve modelin üç boyutlu uzaydaki konumunu, doğrultusunu, kamera ve ekrandaki pencere arasındaki dönüşümlerin yapılmasına olanak tanımaktadır [7].

6.1.1. Model veri yapısı

Üç-boyutlu uzayda model yüzeyindeki her bir noktanın (x,y,z) koordinatından oluşan köşe noktası (vertex) ile poligonları oluşturan noktaların dizini kullanılarak modellerin sisteme tanıtılması mümkündür. Görüntüleme sistemi giriş olarak model verisini, bir nesne (obj) dosyası olarak almaktadır. Bu nesne dosyası, modelin kaç adet poligondan oluştuğunu, poligonları ifade etmek için köşe noktalarına işaretçileri,

modelin kaç adet köşe noktası içerdiğini ve köşe noktalarının her satırda sırasıyla x, y ve z koordinatlarını içermektedir. Örnek bir nesne dosyasının içeriği Şekil 6.1'de özet biçimde verilmektedir.

```
# 7664 vertices
v 17.836 9.836 292.032
v 17.952 11.974 290.053
v 18.039 9.413 288.028
...
v 162.898 -37.326 26.048
```

```
# 15324 polygons
f 4150 4136 4226
f 4226 4372 4150
f 4285 4150. 4372
...
f 2080 1936 2644
```

Şekil 6.1. Sağ el model (rthand) dosyasının köşe noktaları ve poligonları içeren ifadeleri.

Görüntüleme sistemi tasarlanırken bu model verisi bir fonksiyon aracılığıyla satır satır okunarak poligonlar ve köşe noktaları iki adet çok boyutlu diziye yüklenmektedir. Bu işlem için C programında dosya işleme komutları kullanılmalıdır. İlk olarak dosya açılacak, bir fonksiyon yardımıyla satırlar yukarıdan aşağı kadar okunacak ve her satır okunuşunda okunan değerler matris içerisinde karşılık geldikleri bölgelere yazdırılacaktır. Tüm dizileri ve diğer değişkenleri bir çatı altında toplamak için bir sektör (sector) tanımlanmıştır. Bu sektör; poligonları, köşe noktalarını, poligon sayısını, köşe nokta sayısını, yüzey normallerini, merkez koordinatlarını ve daha pek çok model ile ilişkili değişken ve dizileri içerecek şekilde tasarlanmıştır (Şekil 6.2).

```
typedef struct tagVERTICE
{       float t, u, v;          } VERTICE;

typedef struct tagPOLYGON
{       VERTICE* vertice;
        int x, y, z;            } POLYGON;

typedef struct tagSECTOR
{       int numvertices;
        int numpolygons;
        POLYGON* polygon;
        VERTICE* vertice;       } SECTOR;
SECTOR sector1;
```

Şekil 6.2. Yapıların tanımlanması.

Köşe noktaları (vertice) yapısı, bileşenleri t, u, ve v olan ve 3-Boyutlu uzaydaki her noktanın üç koordinatını belirten değerlerden oluşmaktadır. Modeli oluşturan

poligonlar ise üçgensel bir formda oldukları için her üç köşe noktasının birleşimi bir poligon ifade etmektedir. Her satırda poligonun hangi üç köşe noktasının birleşimi ile oluşturulması gerektiği belirtilmektedir. İşte bu nedenle poligon (polygon) yapısı ise poligonu oluşturan x, y ve z'inci köşe noktalarını içerecektir. Bu iki çok boyutlu dizi kullanılarak istenilen her an için model görüntülenebilmektedir. Yalnızca bu değerler ile model ham bir haldedir ve bu haliyle ancak anahat çizgileriyle görüntülendiğinde bir anlam ifade edebilir. Eğer model, hiçbir ilave özellik girilmeksizin dolgu kaplı olarak görüntülenmek isteniyorsa, modelin ne ifade ettiği anlaşılamayacaktır. Modelin dolgulu olarak, anlaşılır biçimde görüntülenmesi isteniyorsa model en azından bir ışık kaynağı ile aydınlatılmak zorundadır.

6.1.2. Model görüntüleme özellikleri

Modelleri görüntüleyebilmek için bir fonksiyon yazmak gerekir. Bu fonksiyona görüntüleme fonksiyonu (Display function) adını verilebilir. Bu fonksiyonun içerisinde ilk olarak ekranın arka planı silinmeli (silme işlemi herhangi bir renge ayarlama olabilir), model görüntüleme (modelview) matrisi sıfırlanmalı, model eğer orijinden farklı bir yere çizilmek isteniyorsa "glTranslate" komutu ile görüntüleme matrisi değiştirilmelidir. Çizim, daha önceki bölümde de bahsedildiği gibi, "glBegin" komutuyla başlar ve "glEnd" komutu ile sona erer. Çizimde nesnenin değişik özellikler ile görüntülenmesi amaçlanmış ise tüm bu özellikler çizimin başlama aşamasına kadar tanımlanmış olmalıdır. Bütün bu özellikler girildikten sonra "glBegin" komutuyla çizdirilecek olan nesnenin tipi girilmelidir. Çizdirilecek olan öğe (primitive) nokta, çizgi, kare, poligon vb. olabilir ki bu noktada yeni bir seçim yapılmalıdır. Eğer nesne üçgensel poligonlardan oluşuyorsa, "glBegin" komutu içerisinde "GL_POLYGON" veya "GL_TRIANGLES" seçimi yapılmalıdır. Bu seçimin ardından poligonun köşe noktaları için alt alta "glVertex" komutu ile giriş yapılacak ve "glEnd" komutu ile çizim sonlandırılacaktır [6]. Nesne belirli sayıdaki poligon ve köşe noktalarından oluşuyorsa bir for döngüsü ile nesnenin poligonları, önceden hazırlanmış olan çok boyutlu bir diziden çekilerek çizim kolaylıkla yaptırılabilmektedir. Şekil 6.3'te çizime geçilmeden önce yapılması gereken

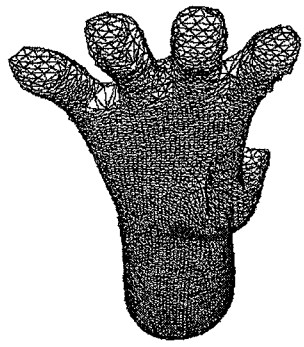
hazırlıklar ve model çiziminin sahip olması istenen özellikler görüntüleme fonksiyonunun içerisinde ifade edilmektedir.

```
void displayFunc()
{
        glClear(GL_COLOR_BUFFER_BIT |GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        glTranslatef(xmerkez, ymerkez, zmerkez);
        glTranslatef(xkontrol, ykontrol, zkontrol);
        glPolygonMode(GL_FRONT, GL_LINE);
        glCullFace(GL_BACK);
        glColor3f(0.75,0.75,0.75);
        glRotatef(xdonus,1.0f,0.0f,0.0f);
        glRotatef(ydonus,0.0f,1.0f,0.0f);
        glRotatef(zdonus,0.0f,0.0f,1.0f);
        ...
}
```

Şekil 6.3. Görüntüleme fonksiyonunda gerekli hazırlıkların yapılması.

Şekil 6.3'te kullanılan programda görüldüğü gibi modelin çizimine başlanmadan önce ekranın temizlenmesi, görüntüleme matrisinin sıfırlanması, çizim yerinin değiştirilmesi, döndürme işleminin etkinleştirilmesi ve modele renk verilmesi gibi işlemler gerçekleştirilmiş ve çizime hazır hale gelinmiştir. Model verileri daha önceden çok boyutlu dizilere yüklenmiş olan bir modelin çizimi için görüntüleme fonksiyonu içerisinde bir "for" döngüsü yardımıyla çizim işlemi gerçekleştirilmekte ve görüntüleme fonksiyonu sonlandırılmaktadır.

Şekil 6.4'te 7664 köşe noktası, 15324 poligondan oluşan sağ el (rthand) modelinin ham halinin yukarıda anlatılan programlama kavramları kullanılarak anahat çizgileri ile görüntülenmesi sonucu elde edilen görüntüsü gösterilmektedir.

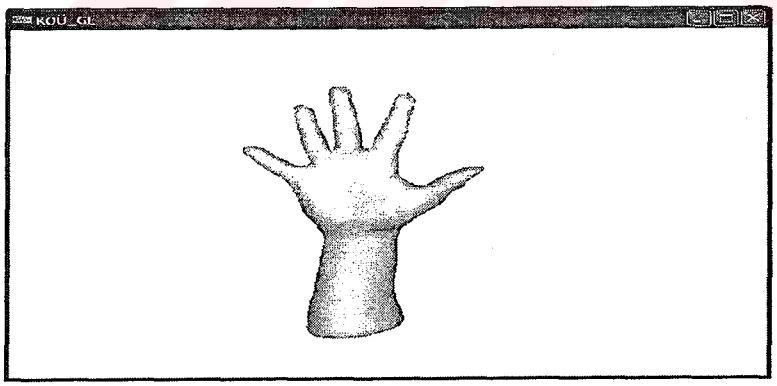


Şekil 6.4. Sağ el (Rthand) modelinin ana hat çizgileriyle elde edilen görüntüsü.

Model görüntüleme penceresinin oluşturulması için iki farklı kullanılabilir yöntem mevcuttur. Bunların birincisi Windows'un kendi özelliklerini kullanan bir C programı ile bir pencere oluşturarak ve bu pencerenin özelliklerini belirleyerek yapılabilir. Bu işlemi daha kolay yapabileceğimiz diğer bir yol ise"glut" kütüphanesinin fonksiyonlarını kullanmaktır. Bu yol ile işlemler çok basit komutlarla bilgisayara yaptırılmaktadır. Bu komutlar C programında "main()" fonksiyonunun altında yapılabilir. Şekil 6.5'de "glut" kütüphanesi [8] kullanılarak bir görüntüleme penceresi oluşturulması işlemi ve Şekil 6.6'da ise oluşturulan görüntüleme penceresi gösterilmektedir.

```
int main( int argc, char ** argv )
{
        glutInit( &argc, argv );
        glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH );
        glutInitWindowSize( g_width, g_height );
        glutInitWindowPosition( 100, 100 );
        KOU = glutCreateWindow( "KOÜ_GL" );
        ...
}
```

Şekil 6.5. Ekran Üzerinde Görüntüleme Penceresinin Oluşturulması



Şekil 6.6. Görüntüleme penceresi.

Pencere oluşturulduktan sonra pencerenin fare ile köşelerinden çekilerek boyutunun değiştirilebilmesi amacıyla bir fonksiyon daha tanımlamamız gerekecektir. Ayrıca modelin ekrana dik eksende yaklaştırılması ile model boyutunun büyütülmesi,

uzaklaştırılması ise model boyutunun küçültülmesi için OpenGL 'in ilgili perspektif (perspective view/projection) bakış biçimi kullanılmalıdır [6]. Bu iki özelliği aynı fonksiyonda gerçekleştirebiliriz. Şekil 6.7'de bu iki özelliğin basit biçimde gerçekleştirilmesini gösterilmektedir.

```
void reshapeFunc(GLsizei width, GLsizei height)
{
     g_width = width; g_height = height;
     glViewport(0,0,width,height);
     glMatrixMode(GL_PROJECTION);
     glLoadIdentity();
     gluPerspective(45.0f,(GLfloat)width/(GLfloat)height,1.0f,300.0f);
     glMatrixMode(GL_MODELVIEW);
     glLoadIdentity();
}
```

Şekil 6.7. Görüntüleme penceresinin yeniden boyutlandırılması ve bakış biçiminin seçimi.

Görüntüleme sisteminde OpenGL komutları kullanılarak, ham modele renk verilmesi, ışıklandırma ve keyfi materyal özellikleri eklenebilmektedir. Uzayın herhangi değişik bölgelerine konumlandırılan noktasal ve/veya çevresel ışık kaynakları kullanılarak, modellerin görüntülerinde değişiklikler yapılabilmektedir. Örneğin ışık kaynakları ve model poligonları çeşitli özellikler ile zenginleştirilerek modellerin görüntülerinde yansıtma özelliklerine bağlı olarak farklı renkli, farklı gölgelikli görüntülerin elde edilmesi sağlanmaktadır. Materyal özelliklerini belirleyen, "GL_AMBIENT", "GL_DIFFUSE", "GL_SPECULAR" ve "GL_SHININESS" olarak belirtilen dört farklı parametre mevcuttur. İlk üçü, materyali belirleyen parametreler ve sonuncusu ise parlaklık derecesini belirleyen değerdir. Bu parametreler, ışık kaynağından çıkan ışığın modelden yansıması ve modelin bu materyal özeliği ile ekranda gösterilmesi esnasında yumuşak ve düzgün bir görüntü elde etmek için farklı değerlere ayarlanabilir [5].

Işık kaynağının kullanılabilmesi için model yüzlerinin yüzey normallerinin ifade edilmesi gerekmektedir. Yüzey normalleri ifade edilerek modeli oluşturan her yüzün aynı oranda aydınlatılması engellenmekte ve farklı aydınlıklı bölgeler oluşturulması sağlanmaktadır. Yüzey normallerinin bulunması için Newell yöntemi kullanılmaktadır [5]. Model normallerinin üretilme işlemi bir önceki bölümde açıkça

gösterilmiştir. Model poligonlarının normal ifadeleri programda çizim aşamasında köşe nokta ifadeleri ile birlikte glBegin ve glEnd komutları arasında girilecektir.

6.1.3. Model ile etkileşim

Model ile etkileşimli bir kontrol sağlanması amacı ile klavye veya fare gibi giriş cihazları kullanılabilir. Bunun dışında tasarlanan bir ara yüzdeki butonlar sayesinde etkileşim gerçekleştirilebilir. Etkileşim için kontrolün sağlanacağı işlemler (yakınlaştırma, uzaklaştırma ve döndürme vb.) önceden belirlenmeli ve ardından OpenGL "glut" kütüphanesinin "glutKeyboardFunc()" ve "glutMouseFunc()" fonksiyonları ile çağrılacak olan büyük bir "switch–case" yapısını (if-else veya benzer bir yapı da olabilir) barındıran fonksiyonlar programda düzenlenmelidir.

Döndürme işlemin esnasında, nesnenin kendi etrafında düzgün biçimde dönmesini sağlamak amacıyla modelin merkezinin bulunması ve eksen takımının bu merkez noktasına yerleştirilmesi gerekecektir. Bu yapılmadığı takdirde model, döndürme komutunun işlemesi ile birlikte 3-Boyutlu koordinat uzayının orijini etrafında bir dönme gerçekleştirilecektir. Döndürme işlemin bir kontrol ile gerçekleşmesi istendiğinde klavye, fare veya bir kullanıcı ara yüzü bu işlem için kolaylıkla kullanılabilir. Bu işlemin yapılması için gereken tek şey kontrolü sağlayacak tuşun veya butonun programda bulunduğu yerde "GL_ROTATE" komutunun katsayı olarak aldığı değişkenin değerinin değiştirilmesidir. Şekil 6.8'de örnek bir fare ile kontrol fonksiyonu gösterilmektedir.

```
void MouseFunction( int button, int state, int x, int y )
{
    if( button == GLUT_LEFT_BUTTON )
    {  zkontrol += 10.0f;  }

    else if ( button == GLUT_RIGHT_BUTTON )
    {  zkontrol -= 10.0f;  }

glutPostRedisplay( );
}
```

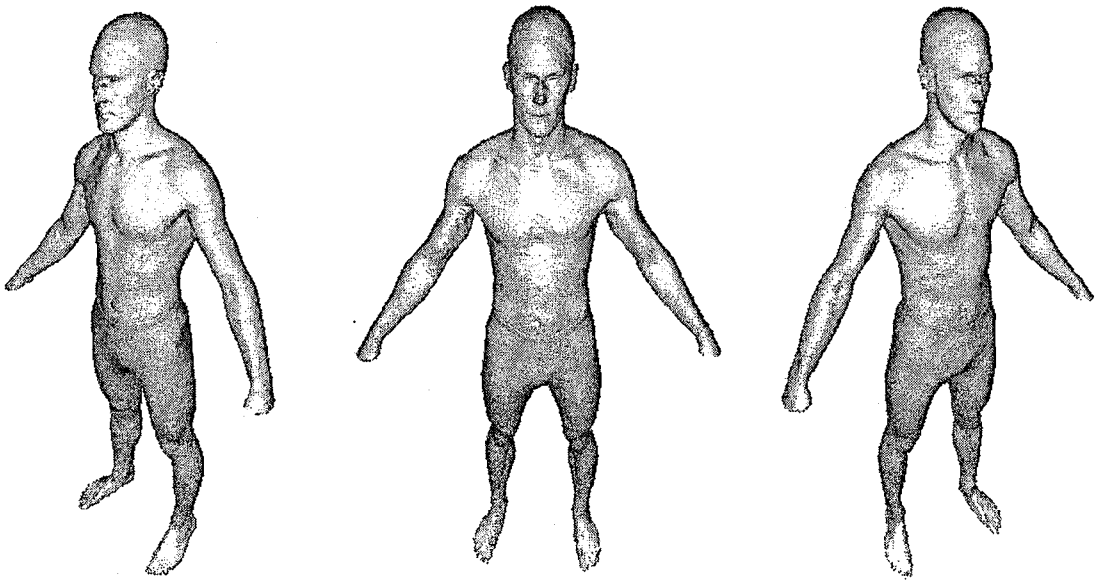
Şekil 6.8. Fare ile etkileşimin sağlanması.

Klavye kontrol fonksiyonu da fare fonksiyonuna benzer şekilde "if-else" veya fazla sayıda tuş kullanılacaksa "switch-case" yapılarından biri seçilerek oluşturulabilir.

Şekil 6.9'da noktasal bir ışık kaynağı (point light), bir çevresel ışık kaynağı (ambient light) ve ışığa renk vermek üzere altın rengi materyal özellikleri kullanılarak farklı açılardan görüntülenen, rengi beyaza yakın ayarlanmış 67038 köşe ve 134076 poligondan oluşan 3-Boyutlu bir tavşan modelinin görüntüleri gösterilmektedir. Bu modelin farklı açılardan görüntüleri klavye ve fare ile modelin 3-B uzayda etkileşimli olarak kontrol edilmesi ile mümkün olmaktadır. Dikkatle incelendiğinde ışığın modele direkt karşıdan verildiği ve arkaya bakan bölgelerin karanlık kaldığı ve farklı gölgelikli bölgeler oluşturulduğu görülmektedir. Işığın açılıp kapanması da ayrıca bir tuş ile kontrol edilmektedir. Ayrıca elde edilen görüntülerde altın rengini materyal özelliği olarak kullanmak için Şekil 6.11 ile ifade edilen değerler kullanılmıştır, diğer herhangi bir materyal özelliği elde etmek için de verilen şekil kullanılabilir.

Şekil 6.9. Tavşan (bunny) modelinin farklı açılardan elde edilen görüntüleri.

Şekil 6.10'da 153140 köşe ve 306276 poligondan oluşan 3-B bir insan vücudu modelinin döndürülmesi sonucu elde edilen görünümler gösterilmektedir. Geliştirilen görüntüleme sisteminde kullanıcı ışıklandırma, döndürme, yaklaştırma ve uzaklaştırma gibi işlemleri etkileşimli olarak kontrol edebilmektedir.



Şekil 6.10. İnsan vücudu (whole body) modelinin farklı açılardan elde edilen görüntüleri.

Material	GL_AMBIENT	GL_DIFFUSE	GL_SPECULAR	GL_SHININESS
Brass	0.329412 0.223529 0.027451 1.0	0.780392 0.568627 0.113725 1.0	0.992157 0.941176 0.807843 1.0	27.8974
Bronze	0.2125 0.1275 0.054 1.0	0.714 0.4284 0.18144 1.0	0.393548 0.271906 0.166721 1.0	25.6
Polished Bronze	0.25 0.148 0.06475 1.0	0.4 0.2368 0.1036 1.0	0.774597 0.458561 0.200621 1.0	76.8
Chrome	0.25 0.25 0.25 1.0	0.4 0.4 0.4 1.0	0.774597 0.774597 0.774597 1.0	76.8
Copper	0.19125 0.0735 0.0225 1.0	0.7038 0.27048 0.0828 1.0	0.256777 0.137622 0.086014 1.0	12.8
Polished Copper	0.2295 0.08825 0.0275 1.0	0.5508 0.2118 0.066 1.0	0.580594 0.223257 0.0695701 1.0	51.2
Gold	0.24725 0.1995 0.0745 1.0	0.75164 0.60648 0.22648 1.0	0.628281 0.555802 0.366065 1.0	51.2
Polished Gold	0.24725 0.2245 0.0645 1.0	0.34615 0.3143 0.0903 1.0	0.797357 0.723991 0.208006 1.0	83.2
Pewter	0.105882 0.058824 0.113725 1.0	0.427451 0.470588 0.541176 1.0	0.333333 0.333333 0.521569 1.0	9.84615

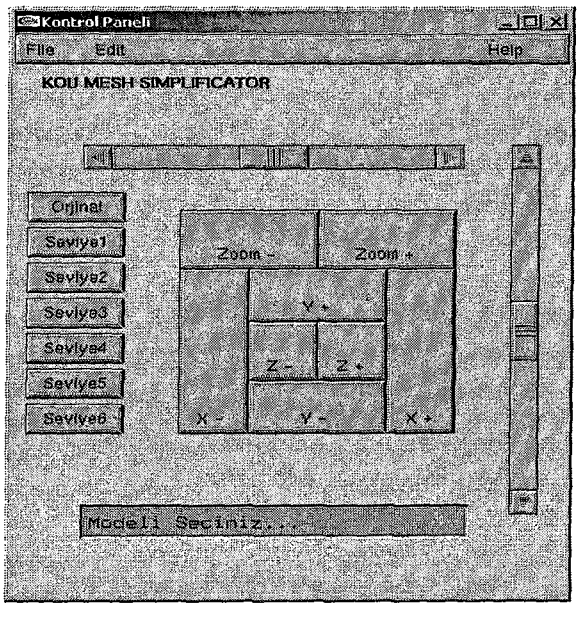
Şekil 6.11. Yaygın biçimde kullanılan materyal parametreleri [5].

6.2. Mikro Kullanıcı Ara yüzü

3-Boyutlu modellerin etkileşimli olarak görüntülenmesinde klavye ve fare gibi giriş cihazlarının kullanılmasının yanı sıra bir kullanıcı ara yüzü üzerinden de etkileşim sağlanabilir. Bu ara yüzü gerçekleştirebilecek çeşitli yöntemler mevcut bulunmaktadır: ilk olarak kullanılan temel programlama dilinin özellikleri kullanılarak yapılabilir. Örneğin kullanılan MS Visual C++ 6.0 programı ile görsel bir ara yüz oluşturulabilir. Diğer bir seçenek ise OpenGL tabanında geliştirilmiş olan yeni bir kütüphanenin fonksiyonlarını kullanmaktır. Bu kütüphane, OpenGL 'in diğer kütüphaneleri gibi açık bir şekilde bulunabilen ve kullanılabilen, Mikro kullanıcı ara yüzü (MUI: Micro User Interface [9])'dür.

Mikro kullanıcı ara yüzünün başlık dosyası ve kütüphane dosyası Windows ile beraber kullanım için sırasıyla "mui.h" ve "mui.lib"dir. MUI, glut kütüphanesinin geri çağrı (callback) fonksiyonlarını kullandığı için glut ile birlikte kullanılmak zorundadır. Glut etkinleştirilmeden, mui çalışmayacaktır. MUI ile menüler, kaydırıcılar, butonlar ve yazı kutuları oluşturulabilir.

Sonuç olarak menü çubuğu, butonlar, kaydırıcılar, metin kutuları gibi pek çok mui nesnesi kullanılarak hazırlanan bir ara yüz Şekil 6.12'de görüldüğü gibi oluşacaktır.

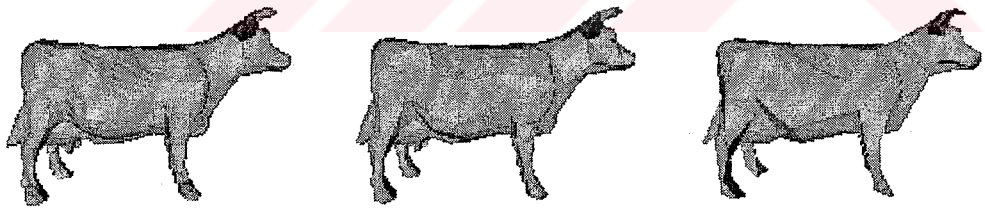


Şekil 6.12. Gerçekleştirilen ara yüz.

7. AĞ BASİTLEŞTİRME KAVRAMLARI

7.1. Giriş

Simülasyon ve gösterim için bilgisayar grafikleri ve ilişkili alanlardaki sayısız uygulama poligonsal yüzey modellerine bağımlıdır. Geleneksel olarak bu tür modeller poligon kümelerine sabitlene gelmiştir ve bu şekilde tek düzeyde detay sağlanmaktadır. Fakat bu tek düzeyli detay, çeşitli içerikler için kullanılacak bu tipte bir model tipine genellikle uymamaktadır. Bu işin merkez odak noktası, ince detaylanmış poligonsal yüzey modellerin aslına uygun yaklaşık modellerle daha az poligon içerecek şekilde otomatik basitleştirilmesidir (Şekil 7.1). Bu bölümde günümüze dek geliştirilen basitleştirme algoritmaları üzerinde durulacak ve basitleştirme sonucunda yüzeyde oluşturulan sıra düzensel yapı incelenecektir. Elde edilen sıra düzen, orijinal modelin geniş oranda yaklaşıklıklarının oluşturulmasını destekleyen yüzey gösterim "çok-çözünürlüklü model" olarak kullanılmaktadır [10].



(a) Orijinal (b) % 83 Daha Az Poligon (c) % 95 Daha Az Poligon

Şekil 7.1. Otomatik basitleştirme ile üretilen poligonsal model yaklaşıklıkları.

7.2. Motivasyon

Teknolojideki gelişmeler, en çok kullanılan yüzey gösterimlerinden biri olan poligon yüzey modellerinin çok büyük veri tabanlarının oluşturulmasını sağlamıştır. Fakat bu

modeller genellikle karmaşık olmakta ve yüzeylerde sıklıkla milyonlarca poligon görülmektedir. Lazer menzilli tarayıcılar, bilgisayar görüntü sistemleri ve tıbbi görüntüleme araçları karmakarışık fiziksel nesnelerin modellerini üretirler. Birçok firma bilgisayar destekli tasarım sistemleri (CAD) kullanarak ürün tasarlarlar ve bu sayede çok karmaşık, yüksek derecede detaylanmış yüzeyler elde edilir. Yüzeyin yeniden oluşturulması ile üretilen modeller ve eşit yüzey (isosurface) çıkarma metotları genellikle noktaların yüzeylere homojen dağıtması ile yoğun biçimde örneklendirilen ağ olarak karşımıza çıkarlar. Özel etkileme gösterimi için kullanılan dağıtılmış sanal ortamlardan, sınırlı eleman metodu alanlarına kadar olan bütün uygulamalar bu tür sistemler ile oluşturulan poligon yüzey modellerine dayanır. Bütün bu uygulamalarda doğruluk ve yüzey modelleme için gereken zaman arasında doğru orantı vardır [10].

Makul işlem zamanı elde etmek için genellikle orijinal modelin basit yaklaşıklaştırmalarının yapılması gerekir. Arşiv veri kümeleri yaratırken düzgün yüzey detayı içeren modeller arzu edilir; bu şekilde daha sonradan modeli kullanacak uygulamalar için yeterli ve doğru veriler garanti edilmiş olur. Fakat birçok uygulama bütün veri kümeleri içinde, olandan çok daha az detaya gereksinim duyar [10].

(a) 1 milyon yüz (b) 20 bin yüz (c) 1000 yüz

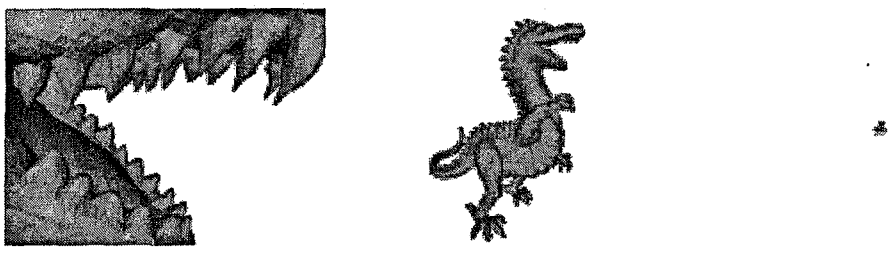
Şekil 7.2. Taranmış "Buda" modeli ile iki yaklaşıklığının gösterimleri.

Yüzey basitleştirme, bireysel uygulamalardaki gereksinimi karşılamada, büyük veri kümelerine yeni biçim vermek ve daha ekonomik yüzey modelleri elde etmek için çok yararlı bir araçtır. Şekil 7.2'deki model ele alınsın. Bir milyondan biraz daha fazla üçgen yüzleri içeren orijinal yüzey, çok sık şekilde yüksek derecede örneklendirilmiştir. Karşılaştırırsak, kısım (b)'deki yaklaşık model %98 daha az üçgen içerir, fakat modelin bütün ana özelliklerine sahiptir. Etkileşimli çeviriyi de içeren birçok uygulama için bu yaklaşık model, uygun bir yer değişimdir. Kısım (c)'deki yaklaşık model ise sadece 1000 yüz içermektedir. Yüzey detayı büyük ölçüde yitirilmiş olsa da genel yapıyı içermektedir. Hacim gibi, yüzeyin kaba özelliklerini ölçmeye çalışan bir uygulama için bu basit modelden makul bir tahmine ulaşılabilir [10].

Detay kontrol derecesi de, gerçek zamanlı çeviri yapan sistemler için çok önemlidir. Esasen verilen herhangi bir sistem için, çerçeve tamponu dolum hızı, dönüşüm ve aydınlatma hızı ve şebeke bant genişliği gibi elde edilebilir donanım kapasitesi sınırlıdır. Fakat görünümü sunmadaki karmaşıklık çok değişken olabilir. Sabit çerçeve hızı elde etmek için, diyelim ki 30 Hz, görünümdeki detay derecesini donanım kapasitesini aşmayacak şekilde tutmalıyız. Bu gereksinim, dar anlamda genellikle bilgisayar oyunları ve dağıtılmış sanal ortamların işletildiği erişilebilir kaynakların sınırlandırılmış olduğu sistemlerde ortaya çıkar. Fakat geniş anlamda, çok güçlü grafik istasyonlarının bile kapasitesini aşan nesne veri tabanlarına sahip, realist benzetim ve bilimsel görüntü üreten sistemlerde de baş gösterebilir [10].

Bir nesnenin detay düzeyini yönetebilmek için yüzeyin işlem zamanında uyarlama yapabilmesini sağlayan çok-çözünürlüklü model gösterimlerine gereksinim duyulur. Etkili olmak için bu çok-çözünürlüklü gösterimlerin, geniş kapsamlı görüntüleme içeriklerini gerçekleştirmek için, geniş kapsamlı detay seviyelerinin yeniden yapılandırılmasının desteklemesi gerekmektedir. Örnek olarak, Şekil 7.3 (b)'de gösterilen yaklaşık olarak yüz bin üçgensel yüzü içeren yüzey modeli ele alınsın. Farz edilsin ki Şekil 7.3 (a)'daki yüzey, detaylı bir şekilde inceleniyor ve ekran bütün yüzeyin küçük bir kısmı tarafından tamamen doldurulmuş durumda olsun. Böyle bir durumda, bu kısım çok iyi şekilde incelenebilir ve görünümde olmayan modelin diğer kısmı çok az üçgenle tanıtılıp önemsenmeyebilir[10]. Şekil 7.3 (c)'deki görüntü ele alındığında, modelin az sayıda noktadan oluştuğu görülmektedir. Bu durumda,

modelde çevrilen piksel sayısı için çok sayıda poligon vardır. Çok-çözünürlüklü modelin, hem bu üç farklı duruma uygun yaklaşık modeller elde etmeyi sağlaması hem de fazla uğraş gerektirmeden detay derecesinin değiştirilmesine olanak vermesi gerekir. Eğer değişim ve daha düşük detay derecesine çevirmek için gereken zaman basitçe daha yüksek detay derecesine çevirmek için gereken zamanı aşarsa çok çözünürlüklü modelden avantaj sağlanamaz [10].



(a) Dişlerin Yakın Gösterimi (b) Normal Görüntü (c) Uzak Çekim Gösterimi

Şekil 7.3. "Ejderha" modelinin üç farklı gösterimi.

7.3. Yüzey Gösterimi

Genel anlamda poligon yüzey modeli basit olarak ifade edilirse üç boyutlu Öklit uzayında (R3) düzlemsel poligonlar kümesidir. Genel prensipten ayrılmadan, modelin sadece üçgensel yüzeylerden oluştuğu düşünülebilir. Çünkü üçgensel olmayan poligonlar işlemden önce üçgensel hale getirilebilir [11, 12]. Model herhangi bir üçgene ait olmayan izole edilmiş köşe ve kenarları da içerebilir. Pratikte, en iyi sonucun elde edilmesi için basitleştirme sürecinde bu öğeler ele alınmalı ve işlem zamanında çevrilmelidir [13-16]. Fakat incelemenin daha da verimli hale getirilmesi adına modellerin sadece üçgenlerden oluştuğunu varsayılır. Birçok algoritma için, izole edilmiş köşe ve kenarların tek etkisi uygulamayı karmaşık hale getirmeleridir. Temel algoritmalar sabit kalmaktadır. Son olarak, eğer iki üçgenin kenarları uzayda kesişiyorsa, model bağlamlığını modelin geometrisi ile tutarlı olduğu varsayılarak, bu üçgenlerin aynı köşeyi paylaştığı söylenebilir [10].

7.4. Basitleştirme ve Çok-Çözünürlüklü Modeller

Geleneksel poligon modelleri sabit köşe ve yüzey setlerinden oluşurlar ve bu yüzden bir nesnenin tek sabit çözünürlüklü gösterimini sağlarlar. Fakat bu tek çözünürlük, modelin kullanılacağı bütün içerikler için uygun olmayabilir. Şekil 7.3'teki üç görüntüyü ele alalım. Düz gölgelendirilmiş her görüntüde bir ejderhanın 108,588 yüz içeren poligon temelli modeli gösterilmektedir. Şekil 7.4'te de yüzey ağının detaylı görüntüsü görülmektedir. Görüntü (b) için bu model makul detay derecesi içermektedir fakat biraz daha basit bir model de aynı şekilde sunulabilirdi. Görüntü (a)'da ise dişler etrafında daha fazla detay arzu edilebilir. Son olarak modelin uzak bir yerden izlenmiş görüntüsü olan (c) de, aynı sonuçlar 100'den daha az üçgenle elde edilebilir [10].

Şekil 7.4. 54,296 köşe noktası ve 108,588 yüzden oluşan poligon temelli "Ejderha" modeli.

Farz edelim ki M poligon modelimiz mevcut ve yaklaşık model olan M_0'ı elde etmek isteniyor. Bu yaklaşık model daha az poligon içerirken M'ye de mümkün olabildiği

kadar benzemelidir. Poligon temelli yüzey basitleştirmesinin amacı otomatik olarak yaklaşık modeller elde etmektir. Kullanıcı denetimi genellikle olanaklı değildir. Doğal olarak basitleştirme ile hedeflenen, elle başa çıkılması zor olan büyük ve karmaşık veri kümeleridir. Genel bir basitleştirme uygulamasının amacı yoğun olarak örneklendirilen modellerin karmaşıklığını düşürmektir. Tarama araçları ve algoritmalar ile elde edilen eşit yüzeyler (örnek olarak "marching cubes [17]") basitleştirme ile yarar sağlarlar. Böyle modeller genellikle homojen olarak parçalanırlar (Bkz. Şekil 7.4). Üçgen yoğunluğu, hem düz hem de hayli kavisli bölgelerde aynıdır. Çoğunlukla üçgensel kavrama ekonomik olduğu için tercih edilir; yerel üçgen yoğunluğu yerel eğime adapte olmalıdır. Üçgen sayısı genellikle %50 veya daha fazla azaltılabilir ve sonuç yaklaşık olarak orijinalle hemen hemen aynı olur [10].

Daha genel olarak belirli bir kullanım için biçim verilen bir yaklaşık model üretmek istenebilir. Örneğin Şekil 7.3 (c)'de gösterildiği gibi gösterim koşularına uygun olarak Şekil 7.4'deki ejderha modelinin yaklaşık modelinin üretilmesi istenebilir. Görüntü işlemede, benzer durum ele alınsın. Farz edilsin ki her inçte 300 nokta olacak çözünürlükte taranmış bir resim var, fakat ekranda her inçte 72 nokta olacak şekilde bu resmin gösterilmesi planlanıyor. Orijinal görüntüyü yeniden örneklendirerek daha az çözünürlükte ve daha küçük gösterimi üretilebilir. Eğer görüntü iyi bir şekilde yeniden örneklendirilirse, her inçte 72 nokta olacak çözünürlükte indirgenen görüntünün neredeyse orijinal görüntüyle ayırt edilemez olması gerekir. Diğer durumlarda tek sabit yaklaşıklıktan daha esnek olunması istenir [10].

Etkileşimli bir durumda kullanıcının Şekil 7.3'ü değişik içeriklerde görüntülediği varsayılsın. Bütün bu değişik gösterim koşullarına uygun olacak tek bir poligon kümesi bulunamaz. Bunun yerine birden fazla değişik yaklaşıklıkların kullanılabilir olması beklenir ve her durum için en iyi seçim yalnız bu şekilde yapılabilir. Sabit çözünürlük modeli yerine çok-çözünürlüklü model tercih edilmektedir. Çok-çözünürlüklü model bir nesnenin yaklaşık modellerini geniş ölçüde kapsayan ve isteğe uygun olarak bunları kullanarak yeniden yapılandıran model gösterimidir. Yeniden yapılandırılan yaklaşık modellerin düşük maliyetli olması gerekir. Çünkü işlem zamanında, genellikle birçok değişik yaklaşık model yapmak zorunda kalınır.

Diğer önemli bir konu da çok-çözünürlüklü model, boyutta az, miktarda sabit faktör artışı kabul edilebilir olsa da en detaylı yaklaşık modelin boyutu ile aşağı yukarı aynı boyuta sahip olmalıdır [10].

7.5. Poligon-Temelli Basitleştirme Metotlarına Genel Bir Bakış

Yüzey basitleştirme ve çok-çözünürlüklü modelleme problemlerine son dönemde artan ölçüde dikkat edilmekte ve özen gösterilmektedir. Basitleştirme, yüz yılı aşkın süredir matematiksel araştırma alanı olan fonksiyon yaklaşımı ile birçok ortak özelliğe sahiptir. Çok-çözünürlüklü yüzey modellerinin temel düşüncesi yeni değildir; Clark [18] 25 yıl önce genel düşünceyi araştırmıştır. Fakat üzerinde çalışılan eğriler ve yükseklik eğrileri gibi basit nesneler dışındaki sonuçlar özellikle son yıllarda elde edilmiştir. Bu bölümde, P. Heckbert ve M. Garland tarafından yazılan kapsamlı bir inceleme [19] üzerine değinilecektir. Değişik basitleştirme algoritmalarının göreceli performans verileri Cignoni [20] inceleme yazısında veya başka diğer kaynaklarda [21, 22] bulunabilir.

Yüzey basitleştirmesindeki iki genel yöntembilim, "yararlı ilaveler" ve "büyük kısmını yok etme" yöntembilimleridir. Yararlı ilave algoritması başlangıçta kaba bir yaklaşım ile başlayan ve her adımda yeni öğeler ekleyerek tekrar eden bir algoritmadır. Aslında yararlı ilave algoritmasının tam tersi olan yok etme algoritması ise orijinal yüzey ile başlayarak her adımda öğeler kaldırarak tekrar eder. Her iki algoritma da çok önemli olan ortak bir özellik içerirler: Her ikisi de başlangıç yüzeyinin dönüşümü yoluyla yaklaşık model bulmaya çalışırlar. Algoritmalar arasındaki büyük fark ise yüzeyde geometrik yaklaşım yapıp yapmamalarıdır. Birçok metot üç kategoride yer alır. Bazıları, özellikle geometrik değişikliği yasaklar [23]. Algoritmaların büyük çoğunluğu kapalı olarak geometriyi basitleştirir. Bir anlamda geometriye göre seçim yaparlar fakat geometriyi yan etki olarak basitleştirirler. Son olarak, bazı algoritmalar açık olarak geometrik basitleştirme yaparak yüzey geometrisini basitleştirmeyi hedeflerler [24, 25].

7.5.1. Yüzeyler

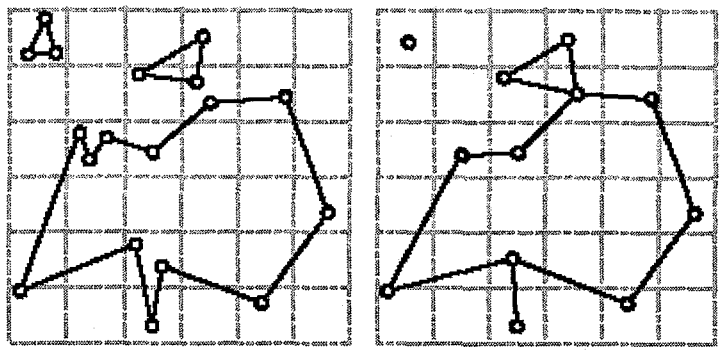
Eğri ve yükseklik eğrilerini basitleştiren başarılı algoritmalar 25 yıl önce geliştirildi [26, 27], fakat genel yüzey basitleştirmesi üzerine olan çalışmalar çok yenidir. Bu algoritmalar, bu çalışma ile yakın ilgili olduğu için bunlardan daha detaylı olarak bahsedilecektir.

7.5.1.1. Elle hazırlama

Çok-çözünürlüklü yüzey modellerine geleneksel yaklaşım, modellerin elle hazırlığıdır. Bir tasarımcının elle birçok detayı yapılandırması gerekir. Elle yapım teknikleri uçuş simülatörü alanında yıllardır [28], ve benzer teknikler oyun yapımcıları tarafından bugün de kullanılmaktadır [29]. Bu işlem özel tasarlanan yüzey editörü ile desteklense de [30] buna rağmen zaman alan ve zor bir iştir. Yüzey basitleştirmesi üzerinde uğraşılmasındaki temel hedef bu işlemi otomatik hale getirmektir.

7.5.1.2. Köşe kümeleme

Köşe kümeleme metotları, [13, 15, 16, 31] köşe setini, küme setlerine böler ve aynı küme içinde olan bütün köşeleri birleştirirler. Genelde çok hızlıdırlar ve üçgen yığınlarını rasgele seçerek çalışırlar. Sadece çeşitli olmayan nesneleri desteklemekle kalmaz bütün birleşim verisine de gereksinim duymadan çalışırlar. Maalesef nispeten düşük kalitede yaklaşık modeller üretirler. En basit kümeleme metodu Rossignac ve Borrel [15] tarafından tanımlanan homojen köşeleri bir araya getirme metodudur. Basit bir homojen kümeleme metodu Şekil 7.5'de gösterilmektedir. Bu algoritma bilgisayar destekli tasarım (CAD) verisinden otomatik olarak çok-çözünürlüklü model yapılandırma için tasarımlanmıştır. Bu nedenle rasgele verilen poligon girdisine izin verilmektedir.



(a) Önce (b) Sonra

Şekil 7.5. İki boyutta tekdüze kümeleme işlemi.

Köşe seti, düzgün bir ızgara üzerindeki sınırlandırılmış kutunun parçalara ayrılmasıyla bölünebilir ve her hücre için yeni gösterim köşesi bir kritere, örneğin kenar uzunluğuna, dayanan basit araştırıcı yaklaşımlar kullanılarak hesaplanır. Bu süreç oldukça etkili bir şekilde uygulanabilir. Algoritma ayrıca orijinal modelin geometrisine büyük değişimler yapmaya meyillidir. Şekil 7.5'e bakarak sol üst köşedeki üçgenin noktaya indirgendiği ve üst kenar boyunca iki ayrı bileşenin birleştiği görülmektedir. Dikkat edilirse, çözünürlüğü düşürülmüş benzer görüntüler gibi algoritma sonucu, ızgara hücrelerin asıl yerlerine oldukça hassas olabilir. Algoritma ayrıca hücre boyutundan daha büyük parçaları basitleştirememektedir. Özellikle iki üçgene hatasız yakınlaştırılabilse bile hücre boyutundan büyük olan üçgenlerden oluşan düzlemsel bir dikdörtgen basitleştirilmez. Homojen küçülmeyi sağlamanın en doğal yolu daha ayrıntılı uzaysal bölünme planları kullanmaktır. Luebke [31, 32] bölünme uzayını daha uyumlu olarak kullanmak için bu algoritmanın genellemesini incelemiştir. Low ve Tan [13] daha esnek bölünme planı önermişlerdir.

Hücreler küp veya küre gibi herhangi basit bir şekilde olabilirler ve en büyük öneme sahip köşe etrafında merkezlenmişlerdir. Birden fazla hücre içinde kalan köşeler en yakın merkeze sahip hücreye tahsis edilirler. Eğer model yüksek derecede örneklendirilmişse ve gerekli basitleştirme oranı çok yüksek değilse kümele getirme metodu iyi çalışmaktadır. Ayrıca yüzey üçgenleri hücre boyutundan küçükse daha iyi çalışacaklardır. Köşeler, hücrelerinin çapından daha fazla hareket edemeyecekleri

için bir araya getirme algoritmaları, M ve M_0 köşelerinde örneklendirilen Hausdorff yakınlaşma hatası üzerinde garantilenmiş sınırlar sağlar. Fakat sağlam basitleştirme elde etmek için gerekli hücre boyutu, hata sınırını yetersiz kılacak düzeyde, oldukça hızlı şekilde artar. Özellikle daha yüksek derecede basitleştirme düzeylerinde elde edilen yaklaşıklık sonuçları hızlı şekilde azalabilir [10].

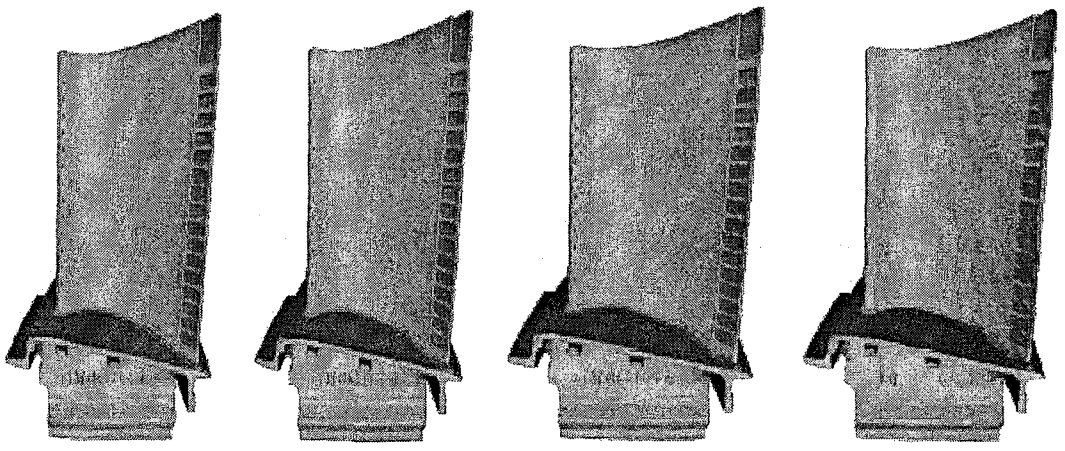
7.5.1.3. Köşe yok etme

En çok kullanılan algoritmalardan biri olan köşe yok etme yinelemeli basitleştirme algoritması orijinal olarak Schroeder [33] tarafından önerilmiştir.

Şekil 7.6. Bir köşe noktası yok ediliyor ve oluşan boşluk yeniden üçgenselleştiriliyor.

Yok etme sürecinin her adımında bir köşe, çıkarılmak için seçilir. Bu köşeye bitişik olan yüzler de modelden çıkarılır ve oluşan delik tekrar üçgenselleştirilir (Bkz. Şekil 7.6). Bu yeniden üçgenselleştirme, yerel yüzeyin düzleme izdüşümünü gerektirdiği için bu algoritmalar genellikle çeşitli yüzeylerle sınırlandırılmıştır. Köşe silinişindeki temel işlem modelin geometrisinin basitleştirilmesi ile uyumsuzluk gösterir. Schroeder [34] bu sınırlamaları, basitleştirme işlemine kesim ve birleştirme operasyonlarını katarak kaldırmayı başarmıştır. Orijinal köşe yok etme algoritması [33] yakınlaştırma hatasının oldukça mantıklı tahminini kullanmıştır.

Daha güncel metotlar [22, 35, 36, 108] sınırlandırılmış Hausdorff hatası[Bkz. 10, Bölüm 2.3.2] gibi daha yanlışsız hata ölçümleri kullanırlar. Bu metotlar orijinal yüzeydeki noktalar arasında ve yaklaşık modelin mutabık çevresinde bağlantı sağlarlar ve bu noktalar arasındaki uzaklık ve birleşen yüzler yakınlaşma hatasını belirler. Schroeder algoritması hem zaman hem de alan yönünden yeteri derecede



(a) Orijinal 1.7 milyon yüz (b) 420 bin yüz (c) 80 bin yüz (d) 8000 yüz

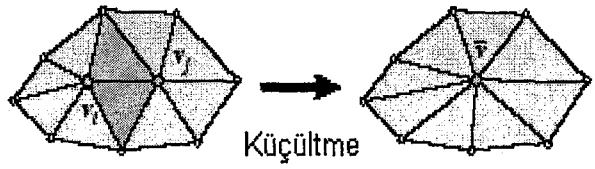
Şekil 7.7. Çok büyük türbin pervanesi modelinin yaklaşıklıkları.

etkilidir fakat düz yüzey [34, şekil 9] sağlamada bazı zorluklara sahiptir. Şekil 7.7'deki türbin pervanesinin gövdesi başlangıçta düzdür fakat basitleştirme sürecinde oldukça kabalaşmaktadır. Diğer köşe yok etme algoritmaları yüksek kalitede sonuçlar üretirler fakat çok yavaştırlar ve çok alan kullanırlar. Bu köşe yok etme yöntembilimi aslında tekrarlı küçülme ile yakından ilişkilidir. Özellikle Şekil 2.6'da resimlendirilen köşe yok etme, basit bir şekilde alt kenar küçültülerek de elde edilebilirdi. Genellikle köşe kaldırarak kenar küçültme [34], çevrenin dikdörtgensel olarak bir düzleme izdüşümünü almaktan [33] daha sağlamdır. Bu durumda çevreyi üst üste getirmeden (overlap) izdüşümün alınabileceği bir düzlem bulmak için sıkıntı yaşanmayacaktır.

7.5.1.4. Tekrarlı küçültme

Son önemli algoritma şekli de tekrarlı kenar küçültmeye dayanır [14, 21, 38-47]. Bir kenar küçültüldüğünde uç noktaları tek noktayla değiştirilir ve kenarlara dönüştürülen üçgenler kaldırılır (Şekil 7.8). Geometri açık şekilde saklanmadığı takdirde kenar küçültme algoritmaları yüzeydeki delikleri kapatarak geometriyi kapalı olarak değiştirebilirler. Hoppe [48], kenar küçültmesini yüzey basitleştirmeyi

elde etmek için temel mekanizma olarak kullanan ilk kişidir. Garland [43] ve diğerleri [14] sadece köşeleri değil de herhangi köşe çiftlerini küçültmeye izin veren kenar küçültme genellemesini ileri sürmüşlerdir. Bu genelleme birleşmeyen bölgelerin birleşmesine olanak verirken ayrıca geometrik basitleştirme ile de sonuçlanmıştır. Ayrıca dikkat edilmelidir ki küçültmenin esas işlemi, anlık komşu noktanın büklüm (manifold) olmasını gerektirmez. Esasen küçültme herhangi basitleştirilmiş karmaşığa uygulanabilir.



(a) Önce (b) Sonra

Şekil 7.8. (vi, vj) kenarı kısaltılıyor; iki yüz ile bir köşe noktası atılıyor.

Bu yüzden küçültmeye dayalı algoritmalar büklüm olmayan yüzeylerle de uyum içinde çalıştıkları için köşe yok etme algoritmalarından daha elverişlidir. Hoppe'un ilerici ağ üretme algoritması (progressive meshes) [38] enerji fonksiyonunu minimize etmeye dayalıdır. Başlıca bileşenlerinden birisi E_{avg} [Bkz. 38, Bölüm 2.8]'e çok benzeyen geometrik hata terimidir. Algoritma hem orijinal yüzey hem de yaklaşık yüzeyde örnek nokta kümelerini içerir. Bu noktalar ve karşılıklı yüzeylerdeki en yakın noktalar arasındaki mesafe geometrik hatayı belirler. Bu algoritma şu an kullanılan metotlar içinde en yüksek kalitedeki sonuçlardan birini üretmektedir. Fakat bu kesinliğin bedeli çok uzun işlem zamanıdır. Hoppe 70,000 yüzeyli bir model için işlem zamanının yaklaşık bir saat olduğunu rapor etmiştir [49]. Orijinal algoritma geometriyi yüzeydeki delikleri kapatarak ve (Popovic ve Hoppe [14]) bağlanmayan bölgelerin birleştirilmesi yoluyla genişleterek basitleştirebilir. Gueziec [41, 42] tarafından geliştirilen algoritma, yaklaşık yüzey etrafında orijinal yüzeyin belli bir hacim içinde kalması garantilenmiş şekilde tolerans hacmi içerir. Bu hacim yaklaşık modelin her köşesinde yer alan küreler ile tanımlanmıştır.

Modelin yüzeyi üzerindeki bu kürelerin dışbükey birleşimi tolerans hacimden oluşan yağlı olarak isimlendirilen üçgenler yaratır. Yaklaşık modelin köşeleri objenin

hacmini muhafaza edecek şekilde yerleştirilmiştir. Bu algoritma her ne kadar yavaş olarak görülse de Hoppe'un algoritmasından hızlıdır ve iyi kaliteli sonuçlar üretir. Ronfard ve Rossignac [40] oldukça etkili basitleştirme algoritması geliştirmişlerdir. Yaklaşık modeldeki her köşe düzlemler setine ilişkilendirilmiştir ve bu köşedeki hata bu setteki düzlemlere olan uzaklıkların karelerinin maksimumu ile tanımlanmıştır. Bu setler, köşeler birlikte küçültüldüğünde birleştirilirler çünkü düzlemlere olan uzaklıkların ölçümü üçgenlere olan uzaklıkların ölçümünden daha kolaydır, bu hata ölçümü Hoppe'nkinde çok daha kolaydır. Doğruluğu kusursuz olmasa da Ronfard ve Rossignac [50] göstermişlerdir ki sınırlanmış Hausdorff hatası onların ölçümünden de elde edilebilir. Sonuçlanan yaklaşık modellerin çoğunlukla iyi kalitede oldukları görülmüştür ve daha kesin algoritmalara kıyasla oldukça etkileyicidir.

Yakın zamanda Lindstrom ve Turk [21] tarafından geliştirilen "hafızasız" algoritma diğer algoritmaların aksine bu açıdan çok ilginçtir çünkü kararları tamamen o anki yaklaşık modele dayalı olarak yapmaktadır. Orijinal şekil ile ilgili herhangi bir veri alınmaz. Küçülme için köşe seçimi ve kalan köşenin pozisyon tayini, nereye yerleştirileceği, temel olarak hacmin korunumuna dayanan lineer kısıtlamalar kullanılarak yapılır. Esasen bu hata metriği geliştirmiş olduğumuz yöntem ile paralellik göstermektedir. Bildirilen sonuçlar iyi kalitede sonuçlar oluşturacağına işaret etmektedir ve özellikle hafıza kullanımında bu oldukça etkilidir. Hoppe'un [48] ağı en uygun şekle sokma (mesh optimization) algoritması ilerici ağ yapılandırma algoritmasının [38] önceki formudur. Basit küçülme yerine açık araştırma yapar ve sonuç olarak süreç daha uzun zaman alır fakat çok kaliteli sonuçlar verir. Olası yaklaşık model uzayını incelemek için kenarlarda küçülme, kenar ayırma ve kenarlarda dönme operatörleri kullanılır. Johnson ve Hebert [51] hem geometrik olarak aslına uygun hem de hemen hemen homojen kenar uzunluğuna sahip modeller üretmek için tekrarlı olarak kenarlarda küçülme ve kenar ayrımı uygulaması kullanmıştır. Tekrarlı küçülmenin yararlarından bir tanesi yarattığı sıra düzensel yapıdır. Bu da doğal olarak kullanışlı ve çok çözünürlüklü yüzey gösterimi sağlar [38, 39, 49, 52]. Dikkat edilirse benzer sıra düzenler köşe yok etme algoritmaları kullanarak da oluşturulabilir [34, 53].

8. AĞ BASİTLEŞTİRME UYGULAMASI

8.1. Basitleştirme Algoritmasının Amacı

Bu çalışmada gerçekleştirilmiş olan basitleştirme algoritmasının amacında, çok-çözünürlüklü modellere ulaşmanın ötesinde, modellerin özellikleri farklı bölgeleri için farklı çözünürlüklü bölgeler elde edilmesi yatmaktadır. Aslında bölgeler arasındaki bu farklılık, modelin özelliği önemli, bir anlamda kıvrımlı ve diğer bölgelerden ayırt edici özelliklere sahip bölgeler ile özelliği daha önemsiz, düz ve herhangi ayırt edilebilecek özelliğe sahip olmayan bölgeler arasındaki farklılıktır. Bizim için önemli olan ise, ağ basitleştirme uygulamasına karşı, işte bu özelliği önemli olan kıvrımlı bölgelerdeki poligonların, diğer bölgelerdeki poligonlardan daha dirençli yapılabilmesidir. Bu mümkün olabilirse uygulama sonucunda elde edilecek model detay seviyesindeki önemli bölgelerin, diğer bölgelerine göre daha yüksek çözünürlüklü olması gerçeklenecek ve model hala orijinal durumundaki özelliklerinin pek çoğunu koruyacaktır.

Peki, bölgeler arasındaki bu farklılık nasıl tespit edilecektir? Bu güne kadar geliştirilmiş olan basitleştirme algoritmalarının çoğunda, basitleştirilecek ağın poligonlarının, kenarlarının ya da bu poligonları oluşturan köşe noktalarının arasında belirlenecek bir hata metriği kullanılarak bir seçim yapılmaktadır. Bu seçim sonucunda yinelemeli olarak poligon yok etme (polygon decimation), kenar küçültme (edge contraction) [14, 21, 38-48, 54] ve ya köşe yok etme (vertex decimation) [22, 33-37] metotları ile veya herhangi başka bir yolla sırasıyla poligonlar, kenarlar ve ya köşe noktaları aşama aşama modelden atılarak model verisi ve dolayısıyla çözünürlüğü azaltılmaktadır. Diğer bazı basitleştirme algoritmaları ise herhangi bir hata metriği kullanmadan basitleştirme sağlarlar. Örnek olarak köşe kümeleme (vertex clustering) işlemi esnasında ise sınırlanan bir kutu oluşturarak, özgün modelin üzerinde ızgara biçiminde bölümlere ayrılır ve her bölüme denk gelen köşeler bir köşe noktası ifade edecek şekilde yeniden yapılandırılarak köşe noktalarının birleştirilmesiyle modelin genel yapısı korunmuş

olur [3, 15, 16, 31, 32]. Fakat bu türden bir algoritma ile çözünürlüğü farklı bölgeler oluşturulamayacak, sonuç model detay seviyesi oldukça düşük bir model tasviri sunacak ve hatta model topolojisi korunamayacaktır. İteratif olarak ve bir hata metriği kullanılarak köşe noktalarının, kenarların veya poligonların atılması ile gerçekleştirilen basitleştirme algoritmaları ile amaçlanan farklı bölgeleri farklı çözünürlüklü ve çok-çözünürlüklü, yüksek kaliteli modeller üretilebilmektedir. O halde gerçekleştireceğimiz algoritma bu tür özelliklere sahip olmalıdır.

8.2. Yinelemeli Poligon Yok Etme

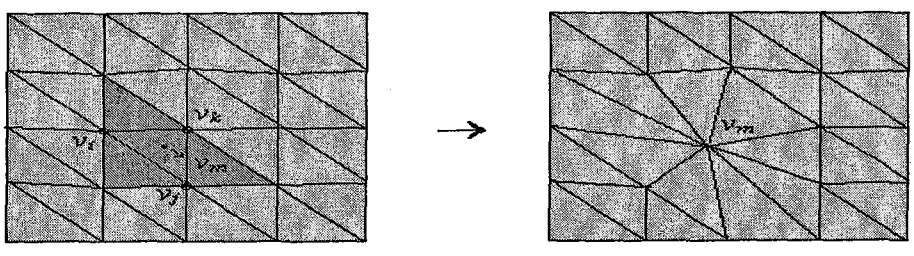
Geliştirilen ağ basitleştirme algoritması yinelemeli poligon yok etme algoritmasıdır. Bu algoritma orijinal yüzey ile başlamakta ve yinelemeli olarak modelden poligonları ve köşe noktalarını atmaktadır. Her yineleme, basit bir poligon yok etme uygulaması içerir. Bir poligon yok etme evresi basit anlatımla dört aşamada gerçekleşir:

1. Seçilen poligonu yok et ($P_i \downarrow$).
2. Poligonun bir köşesini merkez değeri ile değiştir ($v_i \leftarrow v_m$).
3. Seçilen poligonun köşelerini bu poligonla ortak kullanan tüm poligonları, bu köşeleri içermek yerine ilk köşeyi içerecek şekilde düzenle

($v_j, v_k \subset P_j$) $\rightarrow$ ($v_i \subset P_j$).

4. Yok edilen poligonun diğer ikisi noktasını yok et ($v_j, v_k \downarrow$) ve en az iki köşesi aynı olan (üçgen özelliğini kaybeden) poligonları yok et

($v_i, v_i \subset P_k$) $\rightarrow$ ($P_k \downarrow$).



(a) Önce (b) Sonra

Şekil 8.1. Poligon yok etme (soldaki şekilde koyu renk ile gösterilen poligonlar yok ediliyor.).

Poligon yok etme uygulamasının ilk adımında seçilen poligon (Şekil 8.1(a) da ortadaki koyu renkli poligon) ve bu poligonla 2 ortak değer içeren 1. dereceden komşu poligonları (Şekil 8.1(a) da ortadakine komşu koyu renkli diğer poligonlar) yok edilerek yüzey geometrisi değiştirilmektedir. İkinci adım, ağın bağlantılarının düzenlenmesine hazırlık aşaması olarak nitelendirilebilir ki bu aşamada seçilen poligonun üç köşe noktası kullanılarak poligonun merkez noktasını belirlenmekte ve ileriki aşamada kullanılmak üzere poligonun bir köşesi bu değeri içerecek şekilde düzenlenmektedir. Üçüncü adımda ağın poligonsal düzeninin sağlanması için tüm poligonlar yeniden düzenlenmektedir. Önceden, seçilen poligon ile tek bir değeri ortak olarak kullanan komşu poligonlar, bu poligon yok edildiği için yeniden düzenlenerek ağ bağlantılarını ve yüzey topolojisini koruyacaklardır. Son adımda ise tamamen bozulan poligonlar (seçilen poligon ile önceden 2 değeri ortak olarak kullanan poligonlar, bu ortak değerlerin seçilen poligonun merkezini içerecek şekilde değiştirilmesi dolayısıyla üçgen özelliklerini kaybetmeleri) ile artık kullanılmasına gerek olmayan iki köşe noktası modelden atılacaktır.

Bir poligon yok etme evresinde gerçekleşen olaylar basit biçimde aktarılmıştır. Elbette çözünürlüğü ayarlanabilir ağ basitleşme tekniği bir poligon yok etme evresinden ibaret değildir. Bir sonraki bölümde ağ basitleştirme algoritmasının detayları incelenecektir.

8.3. Basitleştirme Algoritmasının Detayları

Gerçeklenen algoritmanın temel yapısında hata metriği olarak kullanılacak değerin elde edilmesinde model poligonlarının yüzey normallerinin Newell yöntemi kullanılarak bulunması yatmaktadır [5]. Tekrar hatırlarsak Newell yöntemi ile model normali, kenar vektörlerinin çapraz çarpımı belirlenerek hesaplanmaktadır. Bu çapraz çarpım sonra normalize edilerek normalize normal vektörü elde edilir. Model normallerinin ağ basitleştirme tekniği içerisindeki önemini konu ilerledikçe daha iyi anlayacağız ancak bizim için en büyük önemi, model poligonlarının yüzey normallerinin belirlenmesi işleminin, her poligonun komşu poligonları ile oluşturduğu yüzey ilişkisini vermesidir.

Eğer bir poligonun yüzey normali ile komşusunun yüzey normali arasındaki açı ne denli büyükse, o bölge, özelliği o denli önemli bölge olarak düşünülür. Bunun nedeni komşu iki poligonun normalleri arasındaki açının büyümesinin anlamına eştir ki bu açının büyük olması, bu poligonların oluşturduğu yüzeyin düzlemsel olmasının aksine kıvrımlı, bir değer sunan, model hakkında bilgi veren bir bölge olduğunu ifade etmektedir. Bu tür bölgelerdeki poligonların, kenarların ve köşe noktalarının öneminin diğer bölgelere nazaran büyük olduğu düşünülerek basitleştirme algoritmasındaki eleme işleminden daha az etkilenmesi, algoritmaya daha dirençli hale getirilmesi gerekir. Bu noktada kullanılacak olan metriğin bu normaller arasındaki açı değeri ile ilişkili olduğu ortaya çıkarılabilmektedir.

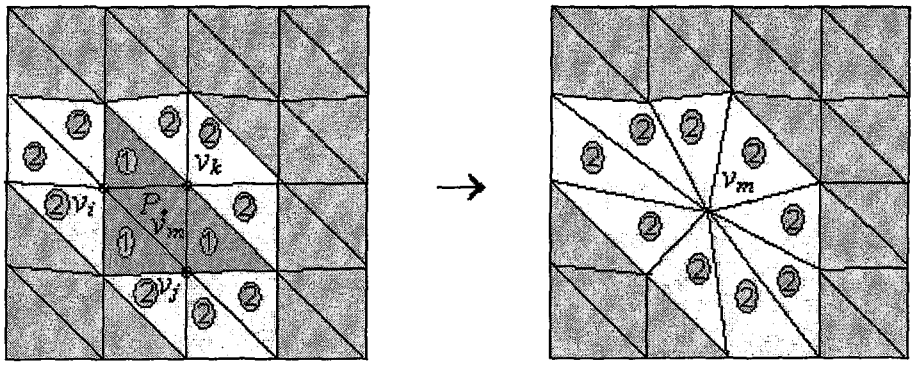
Geliştirilen ağ basitleştirme algoritmasının kullandığı metrik, her poligonun yüzey normali ile komşu poligonlarının yüzey normalleri arasındaki açı değeridir. Bu açı değerine göre elenecek yüzler seçilecektir. Her poligon için yüzey normalinin kendisine komşu olan tüm poligonların yüzey normalleri ile yaptığı açılar karşılaştırılarak yapılan maksimum açı tespit edilir. Bu değer poligonun ve komşularının oluşturduğu poligonsal bölgenin düzlemsellik değeri (aksine kıvrımlılık da denebilir) olarak düşünülebilir ve bu açı ne kadar büyük ise bölge o büyüklükte kıvrımlılık içerir. Belirlenen bu maksimum açı, o poligona açı değeri olarak atanacaktır. Basitleştirme sürecinin her aşamasında tüm poligonlara bir maksimum açı değeri atanarak, açı değeri en küçük olan poligondan en büyük olan poligona doğru bir sıralama gerçekleştirilir.

Basitleştirme işlemi sürecinde her poligonun açı değeri sınır olarak verilen bir eşik değeri ile karşılaştırılarak bu poligonun basitleştirme sürecine dâhil olup olmaması gerektiğine karar verilir. Eğer açı değeri, eşik değerinden küçük ise o poligon sürece katılır, aksi takdirde katılmaz. Bu karşılaştırmanın yapılmasının nedeni belirtildiği gibi nesnelerin önemli sayılan kıvrımlı yerlerinin basitleştirme işlemi esnasında en az düzeyde etkilenerek model ne kadar az veri ile ifade edilirse edilsin bu bölgelerin belirgin görünürlüğünün zarar görmesinin önemli ölçüde önüne geçilmesidir.

Sürece katılan poligonlar sırasıyla en küçük açı değerine sahip poligondan itibaren belirli bir eşik değerine kadar veya tamamı basitleştirme işlemine tabi tutulur. Basitleştirmeye dâhil olacak olan poligon bulunduktan sonra poligon yok etme evresi

gerçekleştirilir. Daha önce de değinildiği gibi ilk olarak elenecek poligonun merkezi bulunur ve poligonun ilk köşe değeri, merkez değeri ile değiştirilir. Bu poligon ile birlikte bu poligona iki köşe değeri ile bağlı olan üç poligon (1. derece komşu) ve poligonun diğer iki köşesi bu esnada yok edilir. Bu poligona tek bir köşe değeri ile bağlı olan poligonlar (2. derece komşu), poligonun köşe değerlerinden birini içermek yerine merkez değerini içerecek şekilde yeniden düzenlenir. Bu düzenlenen poligonlar basitleştirme sürecine bu aşamada yeniden katılmamaları için işaretlenir. Basitleştirme işlemi istenen düzeye ulaşılıncaya kadar aşama aşama yapılmakta ve her aşamada mevcut tüm poligonlar ağ basitleştirme sürecine aktif veya pasif olarak katılacaklardır.

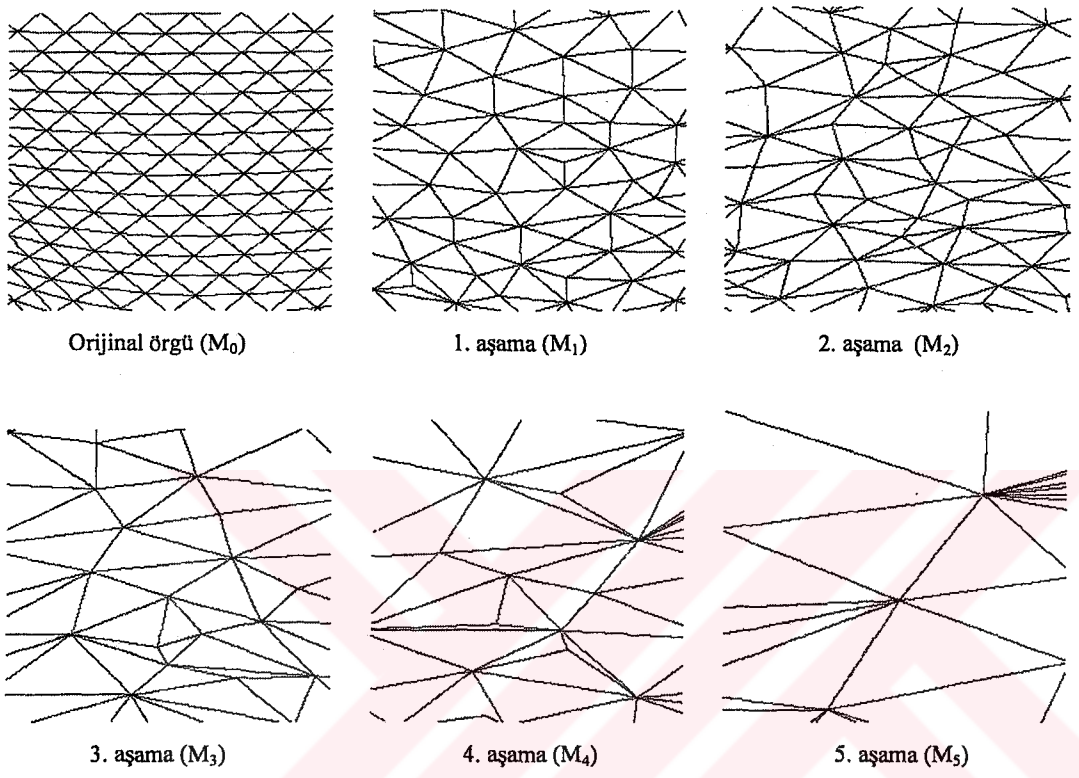
Şekil 8.2'de bir poligon yok etme evresinden önceki ve sonraki poligon temelli ağ yapısı gösterilmektedir. Şekillerden de görüldüğü üzere seçilen poligonun kendisi ve bu poligona 1. dereceden komşu poligonlar modelden atılmış ve poligona 2. dereceden komşu olan poligonlar yeniden düzenlenerek ağsal yapı korunmuştur. Sağdaki şekilde "2" ile gösterilen poligonlar basitleştirme sürecinin bu aşamasında düzenlenen bölgenin tekrar bozulmaya uğramaması için basitleştirme sürecinden çıkarılacaklardır. Bu çıkarım basit bir poligon işaretleme işlemi ile gerçekleştirilir. Bir sonraki seçilecek poligon, "2" ile gösterilen poligonların dışındaki herhangi diğer poligonlar arasından minimum açı değerine sahip olan poligon olacaktır.



(a) Önce (b) Sonra

Şekil 8.2. Poligon yok etme ("1" ile gösterilen poligonlar "p" poligonuna 1. dereceden, "2" ile gösterilen poligonlar ise 2. dereceden komşu poligonlardır.).

Şekil 8.3'de orijinal ağ örgüsünün 5 aşamada geçirdiği basitleştirme süreci gösterilmektedir. Orijinal ağ M_0 ve her detay seviyesindeki ağ yapısı detay seviyesini alt indis olarak gösterecek şekilde M_n ile ifade edilmektedir ($M_0 \rightarrow M_1 \rightarrow M_2 \ldots \rightarrow M_n$).



Şekil 8.3. Basitleştirme aşamaları.

8.4. Basitleştirme Algoritmasının Gerçeklenmesi

8.4.1. Kullanılan Donanım ve Yazılım

Ağ basitleştirme ve benzeri algoritmalar, bilgisayar grafiklerini işlediklerinden donanım olarak yüksek hızlı bir işlemciye ve büyük bir hafızaya sahip bilgisayara gereksinim duymaktadırlar. İşlemcinin hızının yüksek olması ve hafıza alanının genişliği gerçekleştirilen algoritmanın işleme zamanını önemli ölçüde düşürecek ve verimli olarak kullanılmasını sağlayacaktır. Bu tip algoritmaların performansını etkileyen diğer ve önemli bir faktör ise kullanılan yazılım dilidir. Düşük seviyeli diller ile yazılan programlar göreceli olarak düşük, yüksek seviyeli diller ile yazılan

programlar ise yüksek hızlı çalışılabilir olmaları nedeniyle kullanılacak yazılım için yüksek seviyeli bir dil tercih edilmelidir.

Gerçekleştirdiğimiz ağ basitleştirmesi uygulaması için kullandığımız donanım Intel Pentium 4 1.6 Ghz işlemcili, 256 MB RAM'e sahip bir bilgisayardır. Performans bu özelliklere göre değerlendirilmelidir. Yazılım olarak kullandığımız temel dil ise "C" 'dir. C programlama dilleri arasında yüksek seviyeli bir dil olarak bilinmektedir. Grafiksel modelleri görüntüleyecek temel platform olarak "OpenGL" kullanılmakta ve hazırlanan C kodları OpenGL kodları ile harmanlanmaktadır. Harmanlanan kodlar, Microsoft ürünü Visual C++ 6.0 programı üzerinde koşturulmuştur.

8.4.2. Ağ Basitleştirme Uygulama Notları

Modelleme kısmında model verilerinin tanımlamaları yer almaktaydı. Orijinal model verileri iki adet çok boyutlu dizide saklanarak görüntüleme sağlanmaktaydı. Şekil 8.4'te orijinal model verilerinin saklanmaları için oluşturulan yapılar ve dinamik olarak boyutlandırılmaları görülmektedir.

Ağ basitleştirme sürecinde orijinal modellerden farklı detay seviyelerinde farklı çözünürlüklü modeller elde edilecektir. Bu modellerin içerdiği öğeler, orijinal modelden atılan poligon ve köşe noktaları nedeniyle azaltılacak, yeniden düzenlenen öğelerle değiştirileceğinden bu modeller için de ayrı yapılar tanımlanarak, sahip olacakları öğelerin miktarları ile boyutlandırılacaklardır. Unutulmaması gereken bu noktada sadece tanımlamaların yapılacağı, dinamik boyutlandırmanın ise her ağ basitleştirme sürecinin sonucunda oluşacak değerlere göre yapılacağıdır.

```
typedef struct tagVERTICE
{       float t, u, v;          } VERTICE;

typedef struct tagPOLYGON
{       int x, y, z;            } POLYGON;

sector1.vertice = new VERTICE[numvertices];
sector1.polygon = new POLYGON[numpolygons];
```

Şekil 8.4. Model öğeleri için oluşturulan yapılar ve boyutlandırılmaları.

Peki, tüm düzenlemeler hangi matris üzerinde gerçekleşecektir? Eğer düzenlemeler orijinal matris üzerinde yapılırsa, orijinal model değişeceğinden ağ basitleştirme sürecinden sonra geri dönülemeyecektir. Bu nedenle değişikliklerin yapılacağı geçici matrisler tanımlanarak ve her defasında bir üst modelin boyutlarına eşit olarak boyutlandırılarak bu problem aşılabilir. Geriye dönük modellerin görüntülenmesine gerek duyulmaz ise bu evre atlanabilir.

Aşağıda 6 aşamada gerçekleştirilecek olan ağ basitleştirme uygulamasında kullanılacak olan farklı detay seviyelerindeki model verilerini saklamak üzere tanımlanan yapılar görüntülenmektedir. Poligonlar için POLYGON ve köşe noktaları için VERTICE olmak üzere ($M_0 \rightarrow M_1 \rightarrow M_2 \ldots \rightarrow M_6$) detay seviyelerini belirtmek için de {A, B, ... , G} olacak şekilde sonlandırma harfleri kullanılarak tanımlamalar yapılmaktadır. İfade edildiği gibi her aşama sonunda bir alt detay seviyesine ulaşılacak ve sadece bu detay seviyesine ait boyutlandırma işlemi gerçekleştirilecektir. Elbette geçici yapılar kullanacaksanız, bu yapıların boyutlandırılması her zaman bir üst model değerleri ile basitleştirme sürecine başlanmadan yapılmalı, mevcut model değerleri bu geçici matrislere kopyalanmalı, değişikliklerin tamamı bu matrisler üzerinde yapılmalıdır. Atılan ve düzenlenen değerler sonucunda bir alt seviyedeki model detay seviyesine ulaşıldığında bu değerlerin tamamını içerecek şekilde bir alt model yapıları boyutlandırılarak, bu değerler bu matrislere atanacaktır. Geçici model yapıları ile işimiz bittiğinde ise hafıza alanını boşaltmak için bu matrisler yok edilmelidir. Bir örnekle bu durumu açıklayalım. Örneğin orijinal modelden (M_0: VERTICE_A, POLYGON_A) bir alt

```
typedef struct tagVERTICE_A
{      float t, u, v;              } VERTICE_A;
...
...
typedef struct tagVERTICE_G
{      float t, u, v;              } VERTICE_G;
-------------------------------------------------------
typedef struct tagPOLYGON_A
{ int x, y, z;                     } POLYGON_A;
...
...
typedef struct tagPOLYGON_G
{ int x, y, z;                     } POLYGON_G;
```

Şekil 8.5. Çoklu detay seviyeleri için oluşturulan yapılar.

detay seviyedeki modele (M_1: VERTICE_B, POLYGON_B) ulaşmak için basitleştirme sürecinin bir aşama gerçekleştirilmesi gerekmektedir. Bu durumda, orijinal model yapılarının boyutları ile aynı büyüklükte boyutlandırılacak olan geçici model yapılarına (GECICI_VERA, GECICI_POLA) orijinal modelin tüm değerleri kopyalanacak, basitleştirme işlemlerinin tamamı bu geçici yapı üzerinde gerçekleştirilecektir. Geçici model yapısından, atılan ve düzenlenen model verileri sonucunda bir alt detay seviyedeki model verisi elde edilecektir.

```
typedef struct tagGECICI_POLA
{ int x, y, z; int pointer;      } GECICI_POLA;
...
...
typedef struct tagGECICI_POLG
{ int x, y, z; int pointer;      } GECICI_POLG;
-----------------------------------------------
-
typedef struct tagGECICI_VERA
{float t, u, v;     int pointer; } GECICI_VERA;
...
...
typedef struct tagGECICI_VERG
{float t, u, v;     int pointer; } GECICI_VERG;
```

Şekil 8.6. Geçici olarak kullanılacak yapıların tanımlanması.

Bu yeni detay seviyesindeki model, daha önceden tanımlanmış fakat boyutlandırılmamış olan model yapısında temsil edileceğinden, bu noktada içereceği öğelerin miktarı kadar boyutlandırılacaktır. Artık orijinal model verileri (M_0) ve bir alt detay seviyedeki model (M_1) verileri elimizde olduğundan dolayı geçici yapılara ihtiyacımız kalmayacak ve hafızada alan kapladığından yok edilmesi gerekecektir. Şekil 8.7'de oluşturulan yapıların yok edilmesi gösterilmektedir.

```
delete [] sector1.gecici_verA;
delete [] sector1.gecici_polA;
```

Şekil 8.7. Geçici olarak kullanılacak yapıların yok edilmesi.

Şekil 8.8'de, tanımlanan model yapıları ile geçici olarak kullanılacak model yapılarının boyutlandırılması, orijinal model verilerinin işlenmesi için geçici model

yapısına kopyalama işlemi gösterilmektedir. Kopyalama işleminin ardından tüm işlemler bu geçici yapı üzerinde gerçekleştirilecektir.

```
sector1.vertice_A = new VERTICE_A[numvertices_A];
sector1.polygon_A = new POLYGON_A[numpolygons_A];

sector1.gecici_polA = new GECICI_POLA[numpolygons_A];
sector1.gecici_verA = new GECICI_VERA[numvertices_A];

for (int dongu3 = 0; dongu3 < numvertices_A; dongu3++) {
sector1.gecici_verA[dongu3].t = sector1.vertice_A[dongu3].t;
sector1.gecici_verA[dongu3].u = sector1.vertice_A[dongu3].u;
sector1.gecici_verA[dongu3].v = sector1.vertice_A[dongu3].v; }

for (int dongu4 = 0; dongu4 < numpolygons_A; dongu4++) {
sector1.gecici_polA[dongu4].x = sector1.polygon_A[dongu4].x;
sector1.gecici_polA[dongu4].y = sector1.polygon_A[dongu4].y;
sector1.gecici_polA[dongu4].z = sector1.polygon_A[dongu4].z; }
```

Şekil 8.8. Geçici olarak kullanılacak yapıların tanımlanması.

Orijinal köşe noktaları sayısını "numvertices_A" ile ve orijinal poligon sayısını da "numpolygons_A" ile tanımlayarak orijinal model yapılarını "Vertice_A[numvertices_A]" ve "Polygon_A[numpolygons_A]" ile boyutlandırıyoruz. Aynı boyutlarda "Gecici_verA[numvertices_A]" ve "Gecici_polA[numpolygons_A]" yapılarını da boyutlandırarak bir for döngüsüyle kopyalama işlemi gerçekleştirilmektedir. C'de yapılar sınıf/yapı/birleşim (class/struct/union) formatında olmak zorundadır. Bu formatı sağlamak için köşe noktaları (x, y, z) koordinatlarını belirtmek üzere sırasıyla t, u ve v birleşim ifadelerini, poligonun hangi köşe noktalarının birleşiminden oluştuğunu belirtmek üzere x, y ve z (poligonu oluşturan köşe indisleri) birleşim ifadelerini kullanmaktayız.

Basitleştirme sürecinde kullanacağımız tüm yapılar mevcut bulunduğuna göre sürece poligonlara atayacağımız normal değerleri belirleyecerek başlayabiliriz. Belirledeğimiz normal değerlerini saklamak için bir yapı daha tanımlamak zorundayız. Normal değerleri vektör ifade edecek ve x, y, z koordinatları bulunacaktır. Bu değerleri saklamak için 3 değer içeren bir dizi (normal[3]) kullanılabilir. Normalleri kullanarak saptanacak metrik, normaller arasındaki açı

olduğuna göre, yapı, açı değerini de içermelidir. Aşağıda normalleri saklamak üzere tanımlanan yapı gösterilmektedir. Elbette normalleri poligonlara atayacağımız için NORMALVEKTOR_A yapısı da poligon sayısı ile boyutlandırılacaktır (Şekil 8.9).

```
typedef struct tagNORMALVEKTOR_A
{ double aci, normal[3];                } NORMALVEKTOR_A;
...
...
typedef struct tagNORMALVEKTOR_G
{ double aci, normal[3];                } NORMALVEKTOR_G;

sector1.normalvektor A = new NORMALVEKTOR A[numpolygons A];
```

Şekil 8.9. Normal vektörlerini saklayacak yapı ve boyutlandırılması.

Normal vektörleri Newell yöntemi kullanılarak kolayca bulunabilir. Yapılması gereken sadece bir for döngüsü içerisinde her poligon için sırasıyla poligonu oluşturan köşe noktaları indisleri çekilecek, çekilen köşe noktaları indisleri ile köşe nokta koordinatlarına ulaşılacak, bu koordinatları kullanarak kenar vektörleri saptanacak ve kenar vektörlerinin çapraz çarpımları normalize edilerek poligonun normali saptanacaktır. Saptanan değerler sector1.normalvektor_A[indis].normal[3] yapısı içerisinde karşılık gelen poligon için sonradan kullanılmak üzere atanacaktır. Tüm normaller belirlendikten sonra tüm poligonların komşu poligonları ile yaptıkları maksimum açı değerlerini belirlemek üzere bir fonksiyon yazılması gerekmektedir. Komşu poligonlar kolaylıkla ortak olarak kullanılan köşeleri belirleyerek bulunabilir. Bir poligon eğer komşu poligon olarak saptanmışsa, normal ifadesi normalvektor yapısından çekilerek iki normal vektörü arasındaki açı değeri bulunabilir.

Verilen $[x_1 \ . \ . \ . \ x_n]^T$ ve $[y_1 \ . \ . \ . \ y_n]^T$ vektörleri için noktasal çarpım "$x_1y_1 + \ldots + x_ny_n$" olarak tanımlanır ve "v" ve "w" olan iki vektörün noktasal çarpımı "v • w" ile gösterilir. İki vektör arasındaki açı, noktasal çarpım (dot product) kullanılarak hesaplanır. Aşağıdaki formül, "v" ile "w" vektörleri arasındaki "θ" açısını vermektedir.

$$\theta = \cos^{-1}\left(\frac{v \cdot w}{\|v\|\|w\|}\right)$$

Şekil 8.10. İki vektör arasındaki açının hesaplanması.

Ağ basitleştirme uygulamasının kullandığı metrik, bir poligonun komşuları ile yaptığı maksimum açı olmalıdır ki bir poligonun bulunduğu bölgedeki maksimum kıvrımlılık değeri bir ölçüde bulunabilsin. Şekil 8.11'de bir poligonun komşuları ile yaptığı maksimum açının bulunmasını sağlayan algoritma yer almaktadır.

```
For i  = (1 : Poligon Sayısı ) Do

       Poligon Seç;
       Maksimum = 0;
       For i  = (1 : Poligon Sayısı ) Do
              Komşu Poligon Seç
              Komşu Poligon Normalleri Arasındaki Açıyı, Açı Formülünü
                           Kullanarak Belirle

              If (Maksimum < Açı)
              Maksimum = Açı;
       Poligona Açı Değerini Ata;
```

Şekil 8.11. Ağ basitleştirmede kullanılan metriklerin atanması.

Poligonların tümüne metrik değerleri atandıktan sonra poligon yok etme evresi gerçekleştirilecektir. Bu noktada gerçekleştirilmesi gereken işlemlerin tamamını bir algoritma ile özetlemek gerekirse Şekil 8.12'deki algoritma bu amaç için kullanılabilir.

Ağ Basitleştirme Algoritması

```
For i  = (1 : poligon sayısı ) Do
         Normalleri belirle;
For i1  = (1 : poligon sayısı) Do
          Açı değerini hesapla
For i2 = (1 : poligon sayısı)  Do
          If (Açı < sınır)
          Bu poligonu kuyruğa koy
For j  = (1 : index)  Do
    If (isaret != 0 && isaret != 1)
         For i3 = (1 : poligon sayısı) Do
              Açı değeri en küçük poligonu seç
              Sıfır ile işaretle! (Yok Et)
              Bu poligonun üç köşesini hafızadan çek
              1.köşe =  (1. köşe + 2. köşe + 3. köşe) / 3
              2. ve 3. köşeyi sıfır le işaretle (Yok et)
        For i4 =  (1 : poligon sayısı ) Do
                Poligonun köşe değerlerine bak
                If  (köşe = 1. köşe ||  2. köşe)
                    köşe = 1. köşe değeri
         For i5  =  (1 : poligon sayısı) Do
                  1.  derece komşuları bul
                  Sıfır ile işaretle! (Yok Et)
                  2. derece komşuları bul
                  1 ile işaretle (Dokunulamaz)
```

Şekil 8.12. Ağ basitleştirme algoritması.

Şekil 8.12'deki algoritma ağ basitleştirme programı içerisinde yalnızca bir aşamada yapılması gereken işlemleri özetlemektedir fakat bir sonraki aşamaya geçilmesi durumunda yapılması gerekenler hakkında bilgi içermemektedir. Yapılması gereken işlemleri yalnızca ilk aşama hariç tutulmak üzere ön hazırlık olarak adlandırabiliriz. Orijinal model ile basitleştirme sürecine yalnızca model tanımlamaları ve boyutlandırmaların tamamlanmasının ardından başlanabiliyordu. Fakat bir aşama basitleştirme uygulaması gerçekleştirilmesi durumunda elde edilen model detay seviyesinin özelliklerinin tamamı belirlenmeksizin yeni bir basitleştirme aşaması gerçekleştirilemeyecektir. Bu nedenle bu noktada bu özelliklerin belirlenmesine değinilecektir. Yukarıda özetlenen ağ basitleştirme algoritmasının bir aşama gerçekleştirilmesi neticesinde bir alt detay seviyesine ulaşılacaktır. Fakat daha önce de ifade edildiği gibi elde edilen model detay seviyesi, geçici yapılar altında, modelden atılan ve değişime uğratılan öğeler ile birlikte bulunmaktadır. Her ne kadar bu model detay seviyesi bu haliyle görüntülenebilmekte ise de eğer istenen azaltma sağlanmamışsa uygulamanın bir aşama daha gerçeklenmesi gerekecektir. Bu durumda elde edilen alt model seviyesi diğer veriler ile karışık olarak bulunduğundan uygulama ya gerçeklenemeyecek ya da pek çok zorlukla karşılaşılacaktır.

```
Numpolygons_new, Numvertices_new = 0;

For i1  = (1 : poligon sayısı) Do
         Poligonun İşaretini Kontrol Et
         If (işaret != 0);
         Numpolygons_new++;

For i2  = (1 : köşe noktası sayısı) Do
        Köşe noktasının işaretini Kontrol Et
         If (işaret != 0);
         Numvertices_new++;

       sector1.polygon_new = new POLYGON_NEW[Numpolygons_new];
       sector1.vertice_new = new VERTICE_NEW[Numvertices_new];
sayac = 0;

For i3 = (1 : poligon sayısı) Do
       If (isaretci != 0)
       sector1.vertice_new[sayac].t = sector1.gecici_ver[i3].t;
       sector1.vertice_new[sayac].u = sector1.gecici_ver[i3].u;
       sector1.vertice_new[sayac].v = sector1.gecici_ver[i3].v;
       sayac += 1;
sayac = 0;

For i4 = (1 : köşe noktası sayısı) Do
       If (isaretci != 0)
       sector1.polygon_new[sayac].x = sector1.gecici_pol[i4].x;
       sector1.polygon_new[sayac].y = sector1.gecici_pol[i4].y;
       sector1.polygon_new[sayac].z = sector1.gecici_pol[i4].z;
       sayac += 1;
```

Şekil 8.13. Ağ basitleştirme ara aşamalar için ön hazırlık.

O halde aynı algoritmanın yeniden kullanılabilir olmasını sağlamak amacıyla geçici yapılar içerisinde elde edilen model detay seviyelerini her aşama sonunda diğer verilerden ayıklayarak kendi detay seviyelerini ifade edecek yapılar içerisinde tanımlamalıyız. Bu ayıklama, algoritmanın takip edilmesi ile yapılabilir. Şekil 8.13'te, ilk aşama haricindeki diğer aşamalar için yapılması gereken ön hazırlık kodları bir algoritma ile verilmektedir.

Şekil 8.13'teki algoritmayı adım adım incelemeye çalışalım. İlk iki satırda yeni aşamaya girdi olarak girecek model detay seviyesinin öğe (köşe noktaları ve poligonlar) sayılarının belirlenmesi amacıyla sıfırlandığı görülmektedir. İlk for döngüsü içerisinde, bir önceki basitleştirme aşamasında yok edilmeyen (işareti sıfır olmayan) poligonların sayılarını belirlemek için bir if karşılaştırması yapılmaktadır. Ağ basitleştirme algoritmasında yok edilen poligon ve köşe noktaları sıfır ile işaretlendiğinden işareti sıfırdan farklı elemanlar sayılarak poligon sayısı bulunmaktadır. İkinci for döngüsü aynı mantıkla köşe noktalarının sayılarını belirlemek için çalıştırılmaktadır.

Yeni model detay seviyesinin içerdiği öğelerin miktarı belirlendiğine göre bu ara modeli bir yapı altında tanımlayarak boyutlandırabiliriz. Bu işlemler ikinci ve üçüncü for döngüleri arasında yapılmaktadır. Yapılar için kullanılan polygon_new ile vertice_new ara model detay seviyelerini belirtmek üzere new ile sonlandırılmıştır. Bunların daha önce (M0→ M1→ M2 ... → M6) farklı model detay seviyelerini belirtmek üzere {A, B, ... , G} harfleri ile sonlandırmıştık. Programda new yerine yine bu harflerle veya herhangi birbirinden farklı rakamlarla belirtebiliriz.

Algoritmada üçüncü ve dördüncü for döngüleri, bir üst model yapısı içerisinden atılan model öğelerinin haricindeki öğelerin, tanımlanan yeni model detay seviyesi yapısı içerisine yüklenmeleri işlevini gerçekleştirmektedirler. Üçüncü for döngüsü, elde edilen model detay seviyesinin köşe koordinatlarının bu ara model detay seviyesinin köşe noktalarını belirten çok boyutlu dizi yapısına aktarılması, dördüncü for döngüsü ise aynı şekilde ara modelin poligonlarını oluşturan köşe noktalarının, poligonları belirten çok boyutlu dizisine aktarılması işlemini gerçekleştirmektedir.

Ara model detay seviyeleri her aşama sonunda yukarıda belirtilen algoritma kullanılarak yapılandırılmakta ve eğer gerekirse gerçekleştirilecek olan yeni bir basitleştirme aşaması için hazır hale getirilmektedir. Ağ basitleştirme süreci istenen çözünürlükte ve detay seviyesinde model elde edilene dek anlatıldığı şekilde her aşamada ön hazırlıklar yapılarak ve basitleştirme algoritması işletilerek tamamlanacaktır. Elde edilen çok-çözünürlüklü modeller OpenGL kullanılarak rahatlıkla ayrı ayrı, aynı ekranda yanyana veya sıra ile, kullanıcı etkileşimi de sağlanarak görüntülenebilmektedir.

9. ARTIRIM UYGULAMASI

Çok-çözünürlük şemasının bütününü oluşturabilmek, yalnızca orijinal modelden en az çözünürlüklü modele ağ basitleştirme algoritmasıyla ulaşılması ile mümkün olmayacak, ayrıca basitleştirme sürecinde etkilenen ağ yapısını korumak ve en az çözünürlüklü modelden orijinal modele ulaşmak için geliştirilecek bir artırım mekanizması ile mümkün olacaktır. Her bir alt model detay seviyesinden bir üst detay seviyelerine ulaşılması amacıyla kullanılan veri, geliştirme katmanı vazifesini görecektir. Artırım metodunda, düşük çözünürlüklü model verilerine ek olarak bir aşama üst çözünürlüklü modelin tüm verilerini yeniden göndermenin aksine yok edilmiş poligon ve köşe noktaları değerleri ile değiştirilmesi gereken değer verileri kullanıcıya gönderilerek ve kullanıcının bu değerleri kullanması ile bir üst çözünürlüklü modeli görüntülemesine olanak sağlanır. Her aşama için ek veriler ve değiştirilmesi gereken değerler gönderilerek, kullanıcının orijinal modeli görüntüleyebilmesi sağlanacaktır.

9.1. Algoritmanın Amacı

Artırım algoritması, ağ basitleştirme algoritmasında en yüksek çözünürlüklü modelden her aşamada bir düşük çözünürlüklü modellerin elde edilmesiyle kabul edilebilecek en düşük çözünürlüklü model elde edilinceye kadar gerçekleştirilen işlemleri tersine gerçekleştirerek, her aşamada bir üst çözünürlüklü modellerin elde edilmesiyle orijinal modelin elde edilmesini sağlayacak şekilde çalışan bir yöntemdir. Bu sayede en düşük çözünürlüklü model verilerine sahip kullanıcılar her bir üst çözünürlüklü modeli, her aşamada bu modelleri tek başlarına görüntülemeye yetmeyecek kadar düşük miktarda verinin ilave edilmesiyle görüntüleme imkânına sahip olacaklardır.

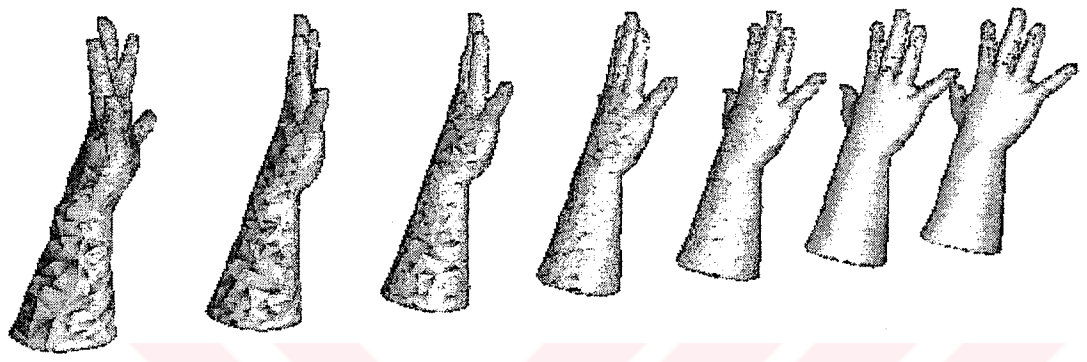
Artırım algoritması kullanıcılara, en az çözünürlüklü modeli kendi bilgisayarlarına aktarmaları sayesinde nesnenin ilk görüntüsünü elde etmelerini diğer tüm verileri yüklemelerini gerektirmeden sağlayacaktır. Eğer kullanıcı diğer detay seviyelerindeki modelleri görmek isterse, modele artırım uygulayabilir.

Şüphesiz ki kullanıcı, en az çözünürlüklü modelden orijinal modeli elde etmek için artırım işlemi gerçekleştirecek ise her geliştirme katmanında eklenecek verilerin haricinde düşük çözünürlüklü modellerden bir üst katman modeli elde edilirken değişime uğraması gereken poligon ve köşe noktaları bilgilerinin de kapsanacağı düşünülerek toplamda daha fazla veriyi bilgisayarına yüklemesi gerekecektir. Bununla birlikte artırım, yalnızca kullanıcının bir üst model detay seviyesini elde etmek istemesi ve yeterli yükleme hızına sahip olması durumunda gerçekleştirilebilir ve çoğu durumda düşük çözünürlüklü modele düşük miktarda verinin eklenmesi yeterli olacaktır. Çok-çözünürlüklü şablonunun diğer bir avantajı ise kullanıcının çözünürlük güncellemelerini, sadece geliştirme katmanlarının alınışıyla birlikte gerçekleştirmesidir.

Kullanıcılara ayrıca en az çözünürlüklü model ile en yüksek çözünürlüklü modelleri yüklemesi arasında seçim yapmasına izin verecek iki seçenekli bir seçim de sunulabilir. Fakat bu durumda en yüksek çözünürlüklü modelin tüm verilerinin görüntüle işleminden önce yüklenmesi gerekecektir.

Şekil 9.1'de "Sağ El" modeli üzerinde artırım yöntemi ve Tablo 9.1'de artırım sürecinde bu modelin içerdiği veri miktarları gösterilmektedir. En az çözünürlüklü modeli ("model g") görüntülemek için gerekli olan veri miktarı 106 KB'dir. Bir üst çözünürlüklü modele ("model f") artırım yapmak için geliştirme katmanının 56 KB'lik veri içermesi gerekmektedir. Böylece bir üst çözünürlüklü modeli, herhangi bir ilave yapılmaksızın görüntülemek için gereken 128 KB'lik veriye kıyasla düşük çözünürlüklü modelden artırarak görüntülemek için toplamda 162 KB'lik veri gerekmektedir. Geliştirme katmanı en az çözünürlüklü modelden "model e" detay seviyesindeki modeli oluşturmak için 69 KB'lik ek veri içerecek ve en az çözünürlüklü modelden "model e"yi artırım yaparak oluşturmak için toplamda 231 KB'lik veri gerekecektir. Bu yolla gidilerek, en az çözünürlüklü modelden artırım

gerçekleştirerek en üst çözünürlüklü modeli ("model a") elde etmek için toplamda 783KB'lik veri gerekmektedir. Bu miktar hemen hemen en üst çözünürlüklü modeli kendi başına, artırım yapmaksızın temsil etmek için gereken 413KB'lik verinin iki katıdır. Bir kodlama tekniği ile aradaki bu farkın sıkıştırılarak düşürülebilinmesine rağmen bir çok-çözünürlük şablonuna sahip olmak için hala kabul edilebilecek bir öneme sahiptir.



Şekil 9.1. "Sağ el" modelinin artırım sürecindeki detay seviyeleri.

a) Orijinal model : 7664 Köşe ve 15324 Poligon : 413 KB
b) % 33 Basitleştirilen Model : 5930 Köşe ve 11856 Poligon : 318 KB
c) % 41 Basitleştirilen Model : 4588 Köşe ve 9172 Poligon : 245 KB
d) % 53 Basitleştirilen Model : 3644 Köşe ve 7284 Poligon : 194 KB
e) % 62 Basitleştirilen Model : 2954 Köşe ve 5900 Poligon : 156KB
f) % 69 Basitleştirilen Model : 2440 Köşe ve 4870 Poligon : 128 KB
g) % 74 Basitleştirilen Model : 2038 Köşe ve 4064 Poligon : 106 KB

Model_g	**Model_f**	**Model_e**	**Model_d**	**Model_c**	**Model_b**	**Model_a**
106 KB	162 KB	231 KB	319 KB	433 KB	589 KB	783 KB
Ek_gf	**Ek_fe**	**Ek_ed**	**Ek_dc**	**Ek_cb**	**Ek_ba**	
56 KB	69 KB	88 KB	114 KB	156 KB	194 KB	

Tablo 9.1. "Sağ el" modelinin artırım sürecindeki verileri.

Tablo 9.1'de ifade edilen "Ek" verileri, hangi düşük çözünürlüklü modelden, hangi daha yüksek çözünürlüklü modele artırım yapılacağını belirtmek üzere "Ek" kelimesine ilave edilen iki harfle birlikte gösterilmektedir. Örneğin "Ek_gf" verisi "model g" en alt çözünürlüğe sahip modelinden, "model_f" bir üst çözünürlüklü modele artırım yapılırken modele ne kadarlık veri eklenmesi gerektiğini gösterecektir. Aynı mantıkla sonuncu ek veri, "Ek_ba" "model b" detay seviyesindeki modelden "model a" orijinal modele artırım yapılması esnasında gerekli ek veri miktarını göstermek için kullanılan ifadedir.

9.2. Artırım Algoritmasının Detayları

Artırım algoritmasında düşük çözünürlüklü modellere bazı verilerin ilave edilmesi ile üst çözünürlüklü modellere ulaşıldığı ifade edilmektedir. Fakat gerçekte bu ek veriler ne tür bilgileri içerecektir? Bunu anlamak için yapılması gereken şey basitleştirme algoritmasının nasıl çalıştığı konusunu iyi biçimde anlamaktan geçmektedir.

Ağ basitleştirmesi algoritmasında her aşamada bir yüksek çözünürlüklü modelden bazı verileri atarak ve bazılarını da değiştirerek daha az çözünürlüklü modelleri elde etmekteydik. İşte bu noktada artırım algoritmasının atılan ve değiştirilen bu verilerin bir düşük çözünürlüklü model verileri ile birleştirilmesi anlamına geldiği anlaşılmaktadır. Peki, bu birleştirme nasıl gerçekleştirilecek?

Yapılacak şey aslında çok da zor olmayan bir düşünceyi gerçekleştirmektir. Artırım algoritmasında ihtiyaç duyulan bu atılan veriler ile değiştirilen verilerin orijinal değerleri ile hangi verilerin değiştiğini bize bildirecek işaretçi verilerinin hafızanın belli bir bölgesinde tutulması ve artırım mekanizmasının her aşamasında gereken verilerin hafızanın ilgili bölümünden çekilmesi ile düşük çözünürlüklü modellerden orijinal modelin yeniden elde edilmesi sağlanmaktadır. Elbette gerekli verileri oluşturacak işlemlerin basitleştirme uygulaması içerisinde gerçekleştirilmesi gerekmektedir. Şekil 9.2'de basitleştirme uygulamasının her aşamasında bu ek verilerin oluşturulması için algoritmaya eklenmesi gereken işlemleri özetleyen ve Şekil 9.3'te bu ek verilerin kullanılmasıyla artırım işleminin gerçekleştirilmesi

işlemlerini özetleyen algoritmalar sunulmaktadır. Bu algoritmaların incelenmesi ile uygulamanın tam anlamıyla anlaşılacağını düşünüyorum.

```
degvertsay = 0;                    // değiştirilen köşe sayısı
ekvertsay = 0;                     // eklenecek köşe sayısı
dongu1 = dongu2 = 0;

For i1 = (1 : köşe sayısı) Do
      Köşenin işaretini kontrol et!

      If (işaret == 1)
         Degvertsay ++;            // değişen köşe sayısını 1 artır


      Else If (işaret == 0)
         Ekvertsay ++;             // eklenecek köşe sayısını 1 artır

sector1.ek_vertx = new EK_VERTx[ekvertsay];
sector1.degvertx = new DEG_VERTx[degvertsay];

For i2 = (1 : köşe sayısı) Do
      Köşenin işaretini kontrol et!

      If (işaret == 1)
        indis = sector1.geciciverx[i2].indis;
        sector1.degvertx[dongu1].koord = sector1.verticeX[i2].koord;
        sector1.degvertx[dongu1].indis = indis;
        dongu1 ++;

      If (işaret == 0)
        sector1.ekvertx[dongu2].koord = sector1.verticeX[i2].koord
        dongu2 ++;
```

Şekil 9.2. Eklenecek ve değiştirilecek köşelerin elde edilmesi.

İlk olarak yukarıdaki algoritmada gerçekleştirilen işlemleri açıklayarak başlayalım: Artırım algoritmasında önceden yok edilmiş ("0" ile işaretlenen) ve koordinat değerleri değiştirilmiş ("1" ile işaretlenen) köşe noktalarını düşük çözünürlüklü model verileri ile birleştirerek bir üst çözünürlüklü model elde edilecektir. Bu nedenle ilk olarak ağ basitleştirme algoritmasında bir düşük çözünürlüklü modele basitleştirme yapılırken yok edilen ve değiştirilen köşe noktalarının sayılarını belirlememiz gerekmektedir. Şekil 9.2'deki algoritmadaki ilk for döngüsü bu amaçla gerçekleştirilerek bu sayıların bulunmasını amaçlamıştır.

Basitleştirme algoritmasındaki köşelerin ilgili matris yapısındaki işareti kontrol edilerek, köşe nokta sayıları belirlenmektedir. Kontrol edilecek işaret değeri, ilgili matris yapısı olarak kullanılacak geçici matris yapıları üzerinde daha önceden

tanımlanan bir değişkene basitleştirme sürecinde modelden atılan ve değiştirilen veriler için bir işaret değeri atanması ile sağlanabilir. Geçici matris yapılarının kullanılmasının bir avantajı da burada karşımıza çıkmaktadır. Normal köşe ve poligon yapıları üzerinde modelin orijinal değerleri tutulurken, geçici yapılar üzerinde hangi değerlerin atıldığı ve hangi değerlerin değiştirildiğini bildirecek değişkenler kullanılarak artırım gerçekleştirilecektir. Bir başka deyişle eğer basitleştirme yanında artırım uygulaması da yapacaksak mutlaka "geçici" olarak adlandırılmasa dahi başka bir adlandırma altında yeni yapılar kullanılmak zorundadır. Biz daha önce bu yapıları, ileride kullanılacağı düşünülerek "geçici" yapılar adı altında tanımladık.

Geçici yapıların kullanılması ile basitleştirme sürecinde modelden atılan verilerin "0" ile işaretlendiği düşünülerek kontrol edilecek işaret değerinin "0" olması durumunda eklenecek köşe sayısı bir arttırılarak ve değiştirilen değerlerin "1" ile işaretlendiği düşünülerek değerin "1" olması ile değiştirilmesi gereken köşe noktaları sayısı bir arttırılarak "ek" ve "değişen" köşe sayıları belirlenmektedir. Bu yapılar üzerinde Bu değerin "0" ile "1"den farklı bir değer olması durumunda ise herhangi bir işlem yapılmamaktadır. Belirlenen veriler sayesinde eklenecek ve değiştirilecek verileri içerecek matris yapıları ("ekvertx" ve "degvertx") oluşturularak, boyutlandırılmaktadır. İki for döngüsü arasında, boyutlandırma işlemi gösterilmektedir. Adlandırmada kullanılan "ek" kelimesi kendi anlamında, "deg" kelimesi "değişen" anlamında, "vert" kelimesi "köşe" anlamında ve son olarak "x" harfi ise herhangi bir model seviyesi için kullanılabilecek harf veya rakamlar yerine kullanılmıştır.

İkinci for döngüsünde değişen köşe noktalarını yükleyeceğimiz "degvertx" yapısı ile önceden yok edilen köşe değerlerini yeniden eklemek üzere üzerine yükleme yapacağımız "ekvertx" yapısı, basitleştirilen modelin ilgili verileri ile doldurulmaktadır. Bu veriler genelde sadece köşe nokta koordinatları olmakla beraber değişen köşe noktaları için hangi köşelerin yok olduğunu belirten "indis" değerleri de ayrıca gereklidir. Tüm veri yüklemesi tamamlandığında, iki model seviyesi arasında istenen her anda artırım yapılması sağlanacaktır.

Yukarıda ifade edilen süreç yalnızca köşe noktalarının artırılmasını önermektedir fakat köşe noktaları için yapılan işlemlerde olduğu gibi yok edilen poligon ve değiştirilen poligon değerleri için de aynı süreç gerçekleştirilerek ağ yapısı korunacaktır. Poligonlar için yapılması gereken işlemleri kısaca özetlemek istersek, köşe noktaları için oluşturulan "ekvertx" ve "degvertx" matris yapıları yerine "ekpolx" ve "degpolx" ile adlandırılan yapılara, eklenecek ve değiştirilecek poligonlar atanacak ve artırım esnasında yine bu veriler kullanılarak süreç tamamlanacaktır.

Artırım yöntemine giriş olarak alınacak ek değerler ile değişen değerler artık rahatlıkla düşük çözünürlüklü modeller ile birleştirilerek bir üst çözünürlüklü model elde edilir. Şekil 9.3'te bir alt çözünürlüklü modelden bir üst çözünürlüklü modelin elde edilmesini sağlayan algoritma verilmektedir.

```
v_siraF = numvertices_G;

for (int loop1 = 0; loop1 < numvertices_G; loop1++)
{
   sector1.new_verF[loop1].t = sector1.vertice_G[loop1].t;
   sector1.new_verF[loop1].u = sector1.vertice_G[loop1].u;
   sector1.new_verF[loop1].v = sector1.vertice_G[loop1].v; }
}


for (int loop2 = 0; loop2 < degvertsay; loop2++)
{
   indis = sector1.degvert[loop2].indis;

   sector1.new_verF[indis].t = sector1.degvert_F[loop2].t;
   sector1.new_verF[indis].u = sector1.degvert_F[loop2].u;
   sector1.new_verF[indis].v = sector1.degvert_F[loop2].v;
}

for (int loop2 = 0; loop2 < degvertsay; loop2++)
{
   v_siraF++;

   sector1.new_verF[v_siraF].t = sector1.ekvert_F[loop2].t;
   sector1.new_verF[v_siraF].u = sector1.ekvert_F[loop2].u;
   sector1.new_verF[v_siraF].v = sector1.ekvert_F[loop2].v;
}
```

Şekil 9.3. Bir yüksek çözünürlüklü modele artırım algoritması.

Şekil 9.3'teki kod ile en düşük çözünürlüklü model ("model_g"), bir üst çözünürlüklü modele ("model_f") artırılmaktadır. Kodun ilk satırında, en düşük

çözünürlüklü modelin ("model_g") köşe sayısı "v_siraF" değişkenine atanıyor. Bu işlemin yapılmasının nedeni kullanılacak yeni model yapısı içerisindeki dizilişte, bir üst çözünürlüklü modele ulaşabilmek için gereken "ek" köşe noktalarının "model_g" köşe noktaları verilerinin bitiminden başlayarak "model_f" köşe noktaları sayısına kadar sıralanmasıdır. İlk for döngüsünde "new_verF" ile adlandırılan matris yapısını görmekteyiz. Bu yapı, artırım sürecinde düşük çözünürlüklü modellerden elde edilecek olan üst çözünürlüklü model verilerini içerecek ve basitleştirme sürecinde bu verileri temsil eden model yapısından sadece dizilişte farklılık içerecek şekilde düzenlenecektir. Bir anlamda bir üst çözünürlüklü model yapısı olan "vertice_f", artırımda sadece verilerinin diziliş sırası farklı olarak "new_verF" yapısı ile temsil edilecektir. İlk döngü ile düşük çözünürlüklü modelin tüm verileri bu yeni üst çözünürlüklü model yapısı içerisine sırayla yüklenmektedir. Artırım algoritmasında, üst çözünürlüklü modelin basitleştirme sürecinde değiştirilen verilerinin eski değerlerini yeniden kazanması gerekmektedir.

İkinci for döngüsünde basitleştirme sürecinde düşük çözünürlüklü modeli elde etmek amacıyla değerinde değişiklik yapılan köşe noktalarının, orijinal değerlerini yeniden kazanması amacıyla gerçekleştirilen veri transferi görülmektedir. Değerinde değişiklik yapılacak köşe noktalarının orijinal değerleri, daha önceden ileride kullanılmak üzere "degvertx" yapılarına aktarılmıştı. Artırım sürecinde yapılması gereken bu yapı içerisinde düzenlenen verilerin, doğru köşe noktalarına atanmasıdır. Doğru köşe noktalarının değiştirilmesi için ise daha önceden "degvertx" yapısında ifade edilen"indis" olarak belirtilen ve köşe noktalarını gösteren hafıza işaretçilerinin kullanılmasıyla gerçekleştirilir. Gereken verilerin değiştirilmesi sonucunda elde edilen yapının görüntülenmesi sonucunda model örgüsünde boşluklar olduğu görülecektir. Bunun nedeni, bu model yapısının sadece düşük çözünürlüklü model yapısındaki verilerin miktarı kadar verilere sahip olması ve bu veriler ile bir üst çözünürlüklü modeli görüntüleme çabasıdır ve gözlemlenen boşluklar ise, basitleştirme sürecinde düşük çözünürlüklü modeli elde ederken modelden atılan poligon ve köşe noktalarından kaynaklanmaktadır. Bu boşlukların giderilmesi için modelden atılan verilerin yeniden eklenmesi gerekmektedir. Daha önceden bu veriler "ek" ile adlandırılan yapılar içerisine yüklenerek, kullanıma hazır hale getirilmişti. Üçüncü for döngüsünde, bu boşlukları dolduracak köşe noktalarının diğer köşe

noktalarının bitişinden sıra ile köşe noktalarını temsil eden yapı içerisine yüklenmesi işlemi gerçekleştirilmektedir.

Şekil 9.3 ile ifade edilen kod ile düşük çözünürlüklü modelin köşe noktaları ile ek ve değiştirilmesi gereken köşe noktalarını içeren ilave veriler kullanılarak yüksek çözünürlüklü modelin köşe noktaları elde edilmektedir. Model artırımı, köşe noktalarının yanında, poligon değerlerinin de aynı köşe noktalarında yapılan artırım ile aynı süreçten geçirilerek artırılması sonucunda gerçekleştirilmiş olacaktır. Poligon artırımı için, köşe noktaları artırımını gösteren kod parçasında yapılacak küçük değişiklikler sonucunda elde edilecek bir kod kullanılacaktır. Yapılması gereken sadece köşe noktaları yerine poligonları ifade eden değişkenlerin, yapıların ve işaretçilerin kullanılmasıdır.

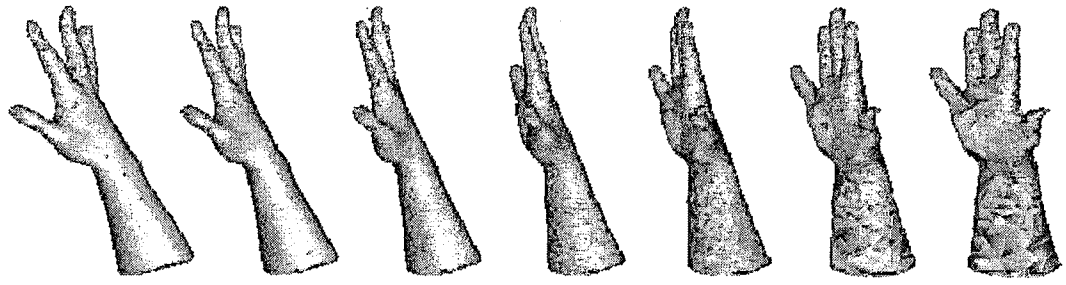
Sonuç olarak artırım sürecinde elde edilecek modeller, basitleştirme sürecinde elde edilen modeller (M0 → M1 → M2 → ... → Mn) ile aynı miktarda ve dizilişlerdeki farklılık haricinde aynı verileri içeren, aynı özellikleri gösteren, gösterimleri de aynı olan modeller (Mn → ... M2 → M1 → M0) olarak karşımıza çıkmaktadır. Basitleştirme ve artırım algoritmaları ile çok-çözünürlüklü model şablonu elde edilmekte ve geliştirilen görüntüleme sistemi ile kullanıcı etkileşimi sağlanarak istenen her an için istenen modeller birlikte ve ayrı ayrı görüntülenebilmekte, modeller arası geçişler yapılabilmekte ve modellerin farklı konum ve açılarda görüntüleri elde edilebilmektedir.

10. SONUÇ VE ÖNERİLER

Bu çalışmada, model çözünürlüğünün önerilen ağ basitleştirme ve artırım tekniği kullanılarak ayarlanabildiği bir üç boyutlu nesne modellemesi uygulaması gerçekleştirilmiştir.

Uygulama, geliştirilme aşamalarında çok yüksek oranda örneklendirilen nesne modellerinin bilgisayar ortamında etkileşimli ve etkin biçimde kullanılması için ve çok-çözünürlüklü yaklaşıklıkları ile özellikle nesnelerin belirli bölgelerinin orijinal nesne modelinden ayırt edilemeyecek düzeyde detay içerecek ve aynı zamanda hafızada orijinal modelden daha az yer kaplayacak şekilde gösterimlerinin elde edilmesini amaçlamıştır. Uygulamanın ne derece başarılı olduğu ancak bu amaçları karşılayıp karşılayamadığı ile ölçülecektir. Elbette uygulama hızının ve literatürdeki diğer algoritmalar ile yapılan kıyaslamaların da bu başarımda bir etkisi söz konusudur. Şimdi örnek modeller üzerinde bu değerlendirmeyi yapmaya çalışalım.

Şekil 10.1'de 7664 köşe noktası ve 15,324 üçgensel poligon içeren "Sağ El" modelinin (M0) basitleştirme uygulamasıyla elde edilen farklı çözünürlüklü bölgeler içeren çok-çözünürlüklü yaklaşıklıkları görüntülenmektedir. Bir analiz yapılması gerekirse modelin özelliği önemli bölgelerinin kıvrımlı olan daha açık biçimde bu model için parmakların bulunduğu bölgelerin saptanması ile başlanabilir. Model yaklaşıklıkları incelendiğinde parmakların bulunduğu bölgelerin çözünürlüğünün avuç içi, avuç dışı ve bilek bölgelerine nazaran daha büyük çözünürlükte oluştuğu görülmektedir. Uygulamanın farklı bölgelerde farklı çözünürlük sağladığı ve yüksek basitleştirme oranlarında dahi orijinal modelin özelliklerinin büyük oranda korunduğu söylenebilir. Bu verilerle algoritmanın başarılı olduğu söylenebilir. Algoritmanın başarısının ölçülebildiği diğer bir faktör ise uygulama zamanı olmaktadır. Şekil 10.1'deki modelin adım adım uygulama zamanları Tablo 10.1'de görüldüğü gibi oluşmaktadır.



M_0 M_1 M_2 M_3 M_4 M_5 M_6

Şekil 10.1. "Sağ El" modelinin basitleştirilmiş ağ örgüleri.

Model (Sağ El)	Köşe Noktası Sayısı	Poligon Sayısı	Basitleştirme Oranı(%)	Kapladığı Alan (KB)	Uygulama Zamanı(sn)
M_0	*7664*	*15324*	-	*413*	-
M_1	*5930*	*11856*	*33*	*318*	*7*
M_2	*4588*	*9172*	*41*	*245*	*+3*
M_3	*3644*	*7284*	*53*	*194*	*+2*
M_4	*2954*	*5900*	*62*	*156*	*+1*
M_5	*2440*	*4870*	*69*	*128*	*+0.5*
M_6	*2038*	*4064*	*74*	*106*	*+0.3*

Tablo 10.1. "Sağ el" modelinin çok-çözünürlüklü yaklaşıklıkları ve uygulama zamanları.

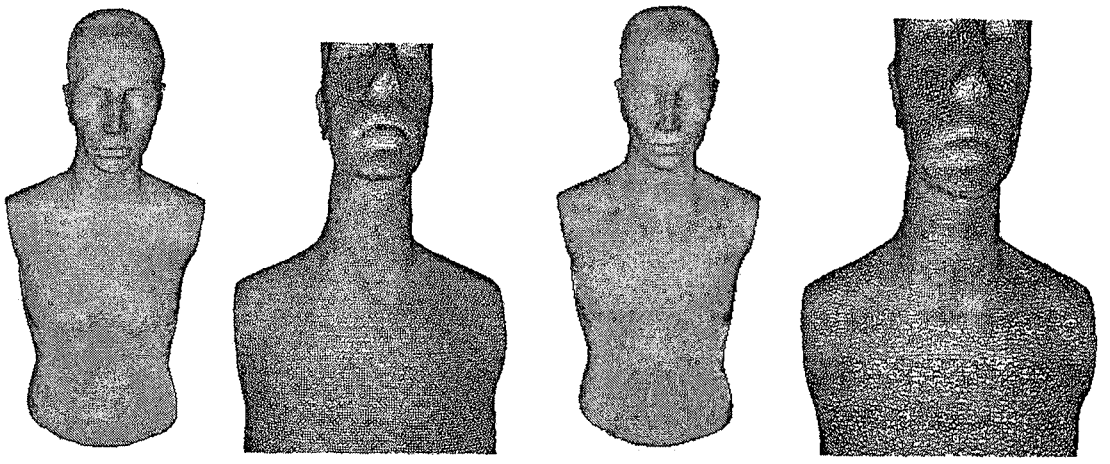
Benzer şekilde daha fazla veri içeren "Torso" nesne modelinin yaklaşıklıklarının elde edilmesi sürecini inceleyelim. Şekil 10.2'de gösterilen "Torso" nesnesinin orijinal modeli 57,380 köşe noktası ve 114,756 üçgensel poligondan oluşmaktadır. Model detay seviyeleri için "Sağ El" modeli ile elde edilen sonuçların benzerleri söylenebilir. Yeniden bir analiz yaparak başlamak gerekirse modelin özelliği önemli bölgelerinin burun, çene ve göz gibi kıvrımlı olan bölgeler olduğu saptanabilir. M3 model detay seviyesine kadar bu bölgelerin küçük ve sık poligonlarla ile gösterilerek hala önemli ölçüde korunduğu ve diğer bölgelerdeki poligonların büyüdüğü ve çözünürlüğünün bu bölgelerde daha çok azaldığı gözlenmektedir. Bu detay seviyesinde poligonlarda % 86 azaltma yapılmış olduğu dikkate alındığında burun, göz ve çene bölgelerinin % 86'dan daha az, diğer bölgelerdeki poligonlarda ise %

86'dan daha fazla basitleştirme yapıldığı kolaylıkla görülebilir. M5 model detay seviyesinde, % 96 basitleştirilmiş model için burun ve çene bölgelerinin algoritmaya hala direnç gösterdiği görülmekte fakat artık göz ve diğer bölgelerde çözünürlüğün giderek azalması ile bozulmalar olduğu görülmektedir. Fakat zaten amaçlanan farklı çözünürlüklü bölgeler elde edildiğine ve belirli bölgelerin bizim için önemli olduğu dikkate alındığında algoritmanın oldukça etkili olduğu düşünülebilir. Elbette yine uygulama zamanı dikkate alındığında yeni bir değerlendirme yapılmalıdır. Model yaklaşıklıklarının özellikleri ile üretilme zamanları Tablo 10.2'de görüldüğü gibi oluşmaktadır:

Model (Torso)	**Köşe Noktası Sayısı**	**Poligon Sayısı**	**Basitleştirme Oranı(%)**	**Hafıza Kullanımı(KB)**	**Uygulama Zamanı(sn)**
M_0	*57380*	*114756*	-	*3,514*	-
M_1	*33380*	*66756*	*41*	*2.102*	*1500*
M_2	*15554*	*31104*	*72*	*985*	*+780*
M_3	*7644*	*15280*	*86*	*495*	*+290*
M_4	*3918*	*7788*	*93*	*248*	*+35*
M_5	*2176*	*4264*	*96*	*142*	*+10*

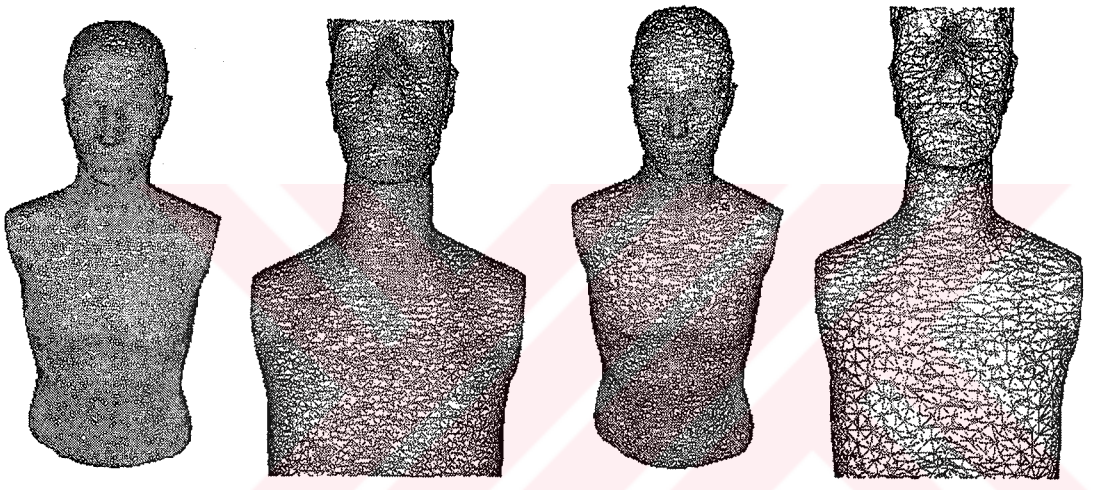
Tablo 10.2. "Torso" modelinin yaklaşıklıkları ve uygulama zamanları.

Tablo 10.1 ile 10.2 incelendiği zaman model verilerinin büyüklüğü ile uygulamanın performansı ciddi oranda azaldığı görülmektedir. "Torso" modelinin yüksek miktarlarda veri içermesi nedeni ile uygulama oldukça fazla zaman almakta, fakat "Sağ El" modeli daha az veri içerdiğinden çok kısa sürede tamamlanmaktadır. Uygulama zamanının, veri miktarına bağlı olmasına rağmen arada bir doğru orantı yoktur. Bunun nedeni algoritma içerisindeki tüm döngülerde bütün verilerin sürece katılması nedeniyle işleme sürelerinin katlamalı olarak artmasıdır.



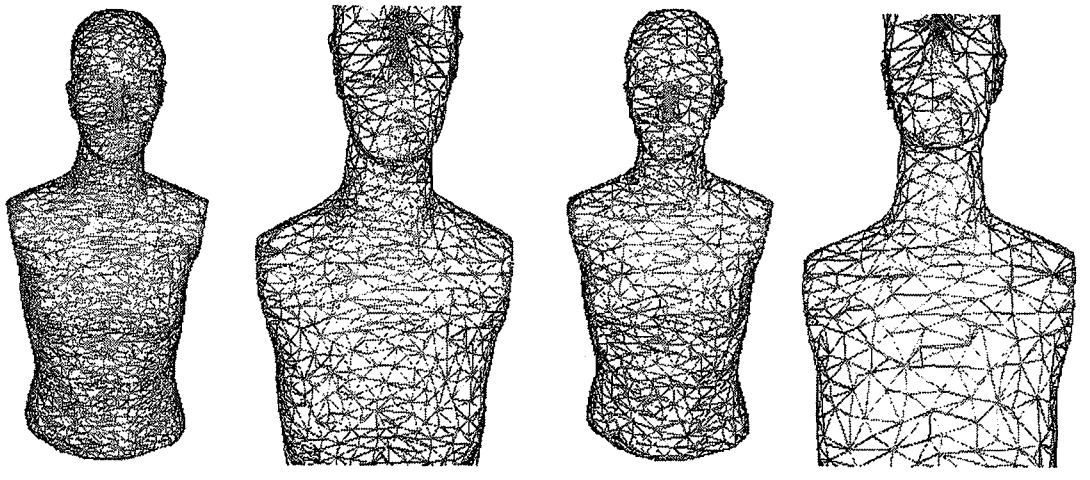
a) 57380 köşe, 114756 poligon, Orijinal (M_0)

b) 33380 köşe, 66756 poligon, %41 basitleştirme (M_1)



c) 15554 köşe, 31104 poligon, % 72 basitleştirme (M_2)

d) 7644 köşe, 15280 poligon, %86 basitleştirme (M_3)



e) 3918 köşe, 7788 poligon, % 93 basitleştirme (M_4)

f) 2176 köşe, 4264 poligon, % 96 basitleştirme (M_5)

Şekil 10.2. "Torso" modelinin basitleştirilmiş ağ örgüleri.

Uygulamanın amaca göre performansı değerlendirildiğinde oldukça başarılı olduğu söylenebilir. Uygulamanın zamana göre performansı literatürdeki diğer algoritmaları ile kıyaslama yapılması ile daha net biçimde ifade edilecektir. Bu tez ile raporlanan sonuçlar, algoritmanın Intel Pentium 4 1.6 Ghz işlemcili, 256 MB RAM'e sahip bir bilgisayarda koşturulması sonucu elde edilmiştir.

Hoppe [49] 70,000 yüzeyli bir model için işlem zamanının yaklaşık bir saat olduğunu, Gueziec [41, 42] geliştirmiş olduğu algoritmanın Hoppe'un algoritmasından biraz daha hızlı ve iyi kalitede sonuçlar ürettiğini rapor etmiştir. Garland M. [10] kuadrik – tabanlı poligon temelli yüzey basitleştirme algoritması için Şekil 10.3'te elde edilen sonuçları raporlamaktadır. Garland işleme zamanlarını Intel Pentium Pro 200 Mhz işlemcili ve 1 GB ana hafıza kullanarak elde ettiğini bildirmektedir.

Model	Köşe	Poligon	İşleme Zamanı
Face	14,445	28,906	2.652
Brontosaur	23,984	47,904	4.665
Bunny	34,834	69,451	7.047
Dragon	54,296	108,588	10.772
Femur	76,794	153,322	15.495
Teeth†	212,192	424,376	55.59
Buddha†	543,644	1,085,634	182.85
Turbine†	882,954	1,765,388	310.09

Şekil 10.3. Garland [10]'ın algoritmasının bazı modeller için raporladığı basitleştirme işlem zamanları (sn).

Şekil 10.3'te belirtilen işleme zamanları ile bu tez içerisinde sunulan algoritmaların işleme zamanları arasında bir karşılaştırma yapılması gerektiğinde Garland [10]'ın algoritmasının daha hızlı olduğu görülmektedir. Elbette daha kesin bir yargıya varmak için algoritmaların aynı özelliklere sahip donanımlar üzerinde koşturulmaları gerekir. Hoppe [49]'un 70,000 yüz içeren bir modelin basitleştirme algoritmasında raporladığı işleme sonuçları ile Tablo 10.2'deki 114,756 yüz içeren "Torso" modelinin basitleştirme işlem zamanları karşılaştırıldığında, bu tez ile önerilen algoritmanın daha hızlı olduğu görülmektedir.

Sonuç olarak, bu tez ile sunulan çözünürlüğü ayarlanabilir nesne modellemesi uygulamasında gerçekleştirilen ağ basitleştirme ve artırım algoritmalarının literatürdeki algoritmaların bazılarından hızlı ve bazılarından da yavaş çalışma hızına, görüntü kalitesi olarak istenilen kaliteye sahip olduğu görülmektedir.

Önerilen yöntemler, ileriki aşamalarda algoritma ve kod üzerinde fazla zaman sarfiyatına neden olan döngüler ve testler üzerinde gerçekleştirilecek düzeltmelerle geliştirilebilir ve hızlandırılabilir. Basitleştirme algoritmasında, her aşamada gerçekleştirilen yok etme sürecinde yüzey eğriliğinin hesaba katılması ile çok daha yüksek basitleştirme oranlarına ulaşılabileceği düşünülmektedir.

KİŞİSEL YAYINLAR

[1] ULUÇAY, Özgür and ERTÜRK, Sarp, "3-Dimensional Object Modeling with Mesh Simplification Based Resolution Adjustment" 12th IEEE Signal Processing and Communication Applications Congress, 28-30 April 2004, Kusadasi

[2] ULUÇAY, Özgür and ERTÜRK, Sarp, "3-Dimensional Object Modeling with Mesh Simplification Based Resolution Adjustment" , Second International Symposium on 3D Data Processing, Visualization & Transmission September 6-9 2004, Thessaloniki, Greece (Accepted)

KAYNAKLAR

[1] FOLEY, J., VAN DAM, A., FEINER, S., HUGHES, J., "Computer Graphics Principles And Practice Second Edition", Addison-Wesley Publishing Company.

[2] WOO, T., "A Combinatorial Analysis of Boundary Data Structure Schemata," Cg & A, 5(3), March 1985, 19 – 27.

[3] BAUMGART, B. G.,"A Polyhedron Representation For Computer Vision," Ncc 75, 589 -596

[4] CARLSSON, P., "3-D School - An Educational Program For Three-Dimensional Computer Graphics", Växjö University , Department of Mathematics, Statistics And Computer Science, Masda Report 9852, Issn 1400 - 1942, October 1998

[5] MCREYNOLDS, T. and BLYTHE, D., "Advanced Graphics Programming Techniques Using Opengl", 1998 April 26, 1998, Silicon Graphics, Siggraph '98 Course

[6] NEIDER, J., DAVIS, T., WOO, M., "The Official Guide To Learning Opengl", Release 1, Addison-Wesley Publishing Company Copyright 1994 By Silicon Graphics, Inc.

[7] MOLOFEE, J., "Nehe Opengl Tutorial", Neon Helium Productions.

[8] KILGARD, Mark J. , "The Opengl Utility Toolkit (Glut) Programming Interface, Api Version 3", November 13, 1996, Silicon Graphics, Inc.

[9] BAKER, Steve "Brief Mui User Guide".

[10] GARLAND, Michael, "Quadric-Based Polygonal Surface Simplification" May 9, 1999 Cmu-Cs-99-105, Copyright © 1999 Michael Garland.

[11] SEIDEL, Raimund "A Simple and Fast Incremental Randomized Algorithm For Computing Trapezoidal Decompositions and For Triangulating Polygons". Computational Geometry: Theory And Applications, 1(1):51 - 64, 1991.

[12] NARKHEDE, A. and MANOCHA, D. "Fast Polygon Triangulation Based On Seidel's Algorithm". In Alan W. Paeth, Editor, Graphics Gems V, Pages 394 - 397. Academic Press, Boston, 1995.

[13] LOW, Kok-Lim and TAN, Tiow-Seng, "Model Simplification Using Vertex Clustering". In 1997 Symposium On Interactive 3d Graphics. Acm Siggraph, 1997.

[14] POPOVIC, J. and HOPPE, H., "Progressive Simplicial Complexes". In Siggraph 97 Proc., Pages 217 - 224, 1997. .

[15] ROSSIGNAC, J. and BORREL, P., "Multi-Resolution 3d Approximations for Rendering Complex Scenes". In B. Falcidieno and T. Kunii, Editors, Modeling In Computer Graphics: Methods and Applications, Pages 455 - 465, Berlin, 1993. Springer-Verlag. Proc. of Conf., Genoa, Italy, June 1993.

[16] SCHAUFLER, G. and URZLINGER, W. St., "Generating Multiple Levels of Detail from Polygonal Geometry Models". In M. G¨Obel, Editor, Virtual Environments '95 (Eurographics Workshop), Pages 33 - 41. Springer Verlag, January 1995.

[17] LORENSEN, W. E. and CLINE, H. E., "Marching Cubes: A High Resolution 3d Surface Reconstruction Algorithm". Computer Graphics (Siggraph'87 Proceedings), 21(4):163 - 170, July 1987.

[18] CLARK, J. H. "Hierarchical Geometric Models For Visible Surface Algorithms". cacm, 19(10): 547 - 554, October 1976.

[19] HECKBERT, P. S. and GARLAND, M., "Survey of Polygonal Surface Simplification Algorithms", in Multiresolution Surface Modeling Course Notes, acm Siggraph, 1997.

[20] CIGNONI, P., MONTANI, C. and SCOPIGNO, R., "A Comparison of Mesh Simplification Algorithms". Computers & Graphics, 22(1):37 - 54, 1998.

[21] LINDSTROM, P. and TURK, G., "Fast And Memory Efficient Polygonal Simplification". In IEEE Visualization 98 Conference Proceedings, Pages 279 - 286, 544, Oct 1998.

[22] CIAMPALINI, A., CIGNONI, P., MONTANI C. and SCOPIGNO R., "Multiresolution Decimation Based On Global Error". The Visual Computer, 13(5):228 - 246,1997.

[23] COHEN, J., VARSHNEY, A., MANOCHA, D., TURK, G., WEBER, H., AGARWAL, P., BROOKS, F. and WRIGHT, W., "Simplification Envelopes". In Siggraph '96 Proc., Pages 119 - 128, August 1996.

[24] HE, T., HONG, L., VARSHNEY, A. and WANG, S. "Controlled Topology Simplification". IEEE Transactions on Visualization and Computer Graphics, 2(2):171-184, June 1996.

[25] EL-SANA J. and VARSHNEY, A. "Controlled Simplification of Genus for Polygonal Models". In IEEE Visualization 97 Conference Proceedings, Pages 403 - 410, 1997.

[26] DOUGLAS, D. H. and PEUCKER, T. K. "Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or Its Caricature". The Canadian Cartographer, 10(2):112 - 122, December 1973.

[27] FOWLER, R. J. and LITTLE, J. J., "Automatic Extraction of Irregular Network Digital Terrain Models". Computer Graphics (Siggraph '79 Proc.), 13(2):199 - 207, August 1979.

[28] COSMAN, M. A. and SCHUMACKER, R. A., "System Strategies to Optimize Cig Image Content". In Proceedings of The Image II Conference, Pages 463 - 480. Image Society, Tempe, Az, June 1981.

[29] STEED, Paul "The Art of Low-Polygon Modeling". Game Developer, Pages 62 - 69, June 1998.

[30] FUNKHOUSER, Thomas A. "Database and Display Algorithms for Interactive Visualization of Architectural Models". Phd Thesis, Cs Division, Uc Berkeley, 1993.

[31] LUEBKE, D., "Hierarchical Structures for Dynamic Polygonal Simplification". Tr 96-006, Department of Computer Science, University of North Carolina at Chapel Hill, 1996.

[32] LUEBKE D. and ERIKSON, C., "View-Dependent Simplification of Arbitrary Polygonal Environments". In Siggraph 97 Proc., Pages 199 - 208, August 1997.

[33] SCHROEDER, W. J., ZARGE, J. A. and LORENSEN, W. E., "Decimation of Triangle Meshes". Computer Graphics (Siggraph '92 Proc.), 26(2):65 - 70, July 1992.

[34] SCHROEDER, W. J., "A Topology Modifying Progressive Decimation Algorithm". In IEEE Visualization 97 Conference Proceedings, Pages 205 - 212,545, 1997.

[35] SOUCY, M. and LAURENDEAU, D., "Multiresolution Surface Modeling Based on Hierarchical Triangulation". Computer Vision and Image Understanding, 63(1):1 - 14, 1996.

[36] KLEIN, R., LIEBICH, G. and STRAFER, W. "Mesh Reduction with Error Control", In Proceedings of Visualization '96, Pages 311 - 318, October 1996.

[37] KOBBELT, L., CAMPAGNA, S. and SEIDEL, H. P., "A General Framework for Mesh Decimation", In Proc. Graphics Interface '98, Pages 43 - 50, June 1998.

[38] HOPPE, Hugues "Progressive Meshes". In Siggraph '96 Proc., Pages 99 - 108, August 1996.

[39] HOPPE, H., "Smooth View-Dependent Level-Of-Detail Control and Its Application to Terrain Rendering". In IEEE Visualization 98 Conference Proceedings, Pages 35 - 42,516, Oct 1998.

[40] RONFARD R. and ROSSIGNAC, J., "Full-Range Approximation of Triangulated Polyhedra". Computer Graphics Forum, 15(3), August 1996. Proc. Eurographics '96.

[41] GUEZIEC, A., "Surface Simplification with Variable Tolerance". In Second Annual Intl. Symp. on Medical Robotics and Computer Assisted Surgery (Mrcas '95)", Pages 132 - 139, November 1995.

[42] GUEZIEC, A., "Surface Simplification inside a Tolerance Volume". Technical Report, Yorktown Heights, Ny 10598, March 1996. IBM Research Report Rc 20440.

[43] GARLAND, M. and HECKBERT, P. S. "Surface Simplification Using Quadric Error Metrics", In Siggraph 97 Proc., Pages 209 - 216, August 1997.

[44] GIENG, T. S., HAMANN, B., JOY, K. I., SCHUSSMAN, G. L. and TROTTS, I. J., "Constructing Hierarchies for Triangle Meshes". IEEE Trans. on Visualization and Computer Graphics, 4(2):145{161, April - June 1998.

[45] ALGORRI Maria-Elena and SCHMITT, Francis "Mesh Simplification", Computer Graphics Forum, 15(3), August 1996, Proc. Eurographics '96.

[46] MELAX, S. "A Simple, Fast, and Effective Polygon Reduction Algorithm". Game Developer, Pages 44 - 49, November 1998.

[47] LAU, R., GREEN, M., TO, D. and WONG, J. "Real-Time Continuous Multi-Resolution Method for Models of Arbitrary Topology". Presence: Teleoperators and Virtual Environments, 7(1):22 - 35, February 1998.

[48] HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J. and STUETZLE, W., "Mesh Optimization". In Siggraph '93 Proc., Pages 19 - 26, August 1993.

[49] HOPPE, H. "View-Dependent Refinement of Progressive Meshes". In Siggraph 97 Proc., Pages 189 - 198, August 1997.

[50] RONFARD, R. and ROSSIGNAC, J., "Full-Range Approximation of Triangulated Polyhedra". Technical Report IBM Research Report Rc 20423, IBM T. J. Watson Research, Yorktown Heights, Ny, 1996.

[51] JOHNSON, A. and HEBERT, M., "Control of Polygonal Mesh Resolution for 3-D Computer Vision". Technical Report, Robotics Institute, Carnegie Mellon U., February 1997, cmu-ri-tr-96-20.

[52] XIA J. C. and VARSHNEY, A. "Dynamic View-Dependent Simplification for Polygonal Models". In Proceedings of Visualization '96, Pages 327 - 334, October 1996.

[53] DE FLORIANI, L., MAGILLO, P. and PUPPO, E., "Efficient Implementation of Multi-Triangulations". In IEEE Visualization 98 Conference Proceedings, Pages 43 - 50,517, Oct 1998.

[54] GARLAND M. and HECKBERT, P. S. "Simplifying Surfaces with Color and Texture Using Quadric Error Metrics". In IEEE Visualization 98 Conference Proceedings, pages 263 - 269, 542, October 1998.

ÖZGEÇMİŞ

5 Mayıs 1980 tarihinde Eskişehir'de doğdu. İlkokul ve ortaokul öğrenimini Balıkesir'de, lise öğrenimini Kayseri Melikgazi süper lisesinde tamamladı. 1997 yılında öğrenimine başladığı Kocaeli Üniversitesi, Mühendislik Fakültesi, Elektronik ve Haberleşme Mühendisliği Bölümü'nü Temmuz–2002 tarihinde tamamlayarak mezun oldu. Ekim–2002 tarihinde Kocaeli Üniversitesi, Fen Bilimleri Enstitüsü, Elektronik ve Haberleşme Ana Bilim Dalı'nda başladığı yüksek lisans öğrenimini Temmuz–2004 tarihinde bitirme durumundadır.

Ocak–2004 tarihinden itibaren Kocaeli Üniversitesi adına 2004K120720 sayı ve "E-devlet için bilgisayar destekli görsel dokümantasyon, arşiv ve yönetim sistemi gerçekleştirilmesi" isimli DPT projesi kapsamında görev yapmaktadır.