

İçindekiler

Bilgisayar Grafiği (Computer Graphics).....	3
SORU-1: Graph Veritabanı Karşılaştırmaları yapınız.....	3
SORU-2: Hesaplamalı Geometri (Computational Geometry) hakkında bilgi veriniz.....	15
SORU-3: Sierpinski Üçgeni (Sierpinski Triangle) hakkında bilgi veriniz.....	18
SORU-5: Bezier Eğrileri (Bezier Curves) hakkında bilgi veriniz.....	24
SORU-6: Rendering (Görselleştirme) hakkında bilgi veriniz.....	26
SORU-7: Splines (Şeritler) hakkında bilgi veriniz.....	39
SORU-8: Texture Filtering (Doku Süzme) hakkında bilgi veriniz.....	63
SORU-9: Eşyönsüz Süzme (ANISOTROPIC FILTERING) hakkında bilgi veriniz.....	67
SORU-10: 3 Boyutlu Şekil Dönüşümleri hakkında bilgi veriniz.....	68
SORU-11: Izgara Tarama (Raster Scan) hakkında bilgi veriniz.....	74
SORU-12: Şeklin Eğilmesi (Shearing) hakkında bilgi veriniz.....	76
SORU-13: Yansıma (Reflection) hakkında bilgi veriniz.....	78
SORU-14: Ters Şekil Değiştirme Matrisleri hakkında bilgi veriniz.....	81
SORU-15: Homojen Koordinatlarla Şekil Değiştirme hakkında bilgi veriniz.....	93
SORU-16: OpenGL İsim Dizisi hakkında bilgi veriniz.....	96
SORU-17: OpenGL Nesne Seçimi (Object Picking) hakkında bilgi veriniz.....	97
SORU-18: OpenGL ile Bağlı Hareket hakkında bilgi veriniz.....	97
SORU-19: OpenGL ile Malzeme Özellikleri (Material Properties) hakkında bilgi veriniz.....	98
SORU-20: OpenGL ile Aydınlatma (Lighting) hakkında bilgi veriniz.....	99
SORU-21: Mafsallı Tasarım (Articular Design) hakkında bilgi veriniz.....	100
SORU-22: Varlık-Ağaç Modelleme (Tree Model) hakkında bilgi veriniz.....	102
SORU-23: Varlık-Durum Tablosu (Symbol Instance Table) hakkında bilgi veriniz.....	104
SORU-24: Malzeme Rengi (Material Color) hakkında bilgi veriniz.....	105
SORU-25: Işık Kaynakları (Light Sources) hakkında bilgi veriniz.....	106
SORU-26: Lambert kosinüs teoremi (Lambert's cosine theorem) hakkında bilgi veriniz.....	106
SORU-27: OpenGL ile Arkayüz (Opengl Backface) hakkında bilgi veriniz.....	108
SORU-28: Derinlik Vurgusu (Depth Cueing) hakkında bilgi veriniz.....	109
SORU-29: Hat Tarama Algoritması (ScanLine Algorithm) hakkında bilgi veriniz.....	110
SORU-30: A-Hafızalama (A-Buffer) hakkında bilgi veriniz.....	110
SORU-31: Derinlik Hafızalama (Depth Buffering , z-buffer) hakkında bilgi veriniz.....	111
SORU-32: Arka Yüz Algılama (Back Face Detection) hakkında bilgi veriniz.....	112
Soru:33: OpenGL ile Perspektif hakkında bilgi veriniz.....	112
SORU-34: OpenGL ve Kamera Görüntüsü (Camera Viewing) hakkında bilgi veriniz.....	114

SORU-35: Şev Yansıması (Oblique Projection) hakkında bilgi veriniz.....	115
SORU-36: İzometrik İzdüşüm (Isometric Projection) hakkında bilgi veriniz.....	116
SORU-37: Eksensel İzdüşüm (Axonometric Projection) hakkında bilgi veriniz.....	116
SORU-38: Dik İzdüşüm (Orthogonal, Orthographic Projection) hakkında bilgi veriniz.....	117
SORU-39: Paralel İzdüşüm (Parallel Projection) hakkında bilgi veriniz.....	118
SORU-40: İzdüşüm (Projection) hakkında bilgi veriniz.....	118
SORU-41: çoklu şekil değiştirmeler (multiple transformations) hakkında bilgi veriniz.....	118
SORU-42: OpenGL Şekil Değiştirme İşlemleri (Transformations) hakkında bilgi veriniz.....	121
SORU-43: Cohen-Sutherland Doğru Kesme Algoritması (Line Clipping Algorithm) hakkında bilgi veriniz.....	122
SORU-44: Homojen Koordinatlar (Homogenous Coordinates Form) hakkında bilgi veriniz.....	123
SORU-45: OpenGL Geometrik Nesneler (Geometric Objects) hakkında bilgi veriniz.....	125
SORU-46: Ölçeklendirme (Scaling) hakkında bilgi veriniz.....	125
SORU-47: 2 Boyutlu Döndürme (2D Rotation) hakkında bilgi veriniz.....	126
SORU-48: 2 Boyutlu Şekil Dönüşümleri (2D Transformations) hakkında bilgi veriniz.....	127
SORU-49: 2 boyutlu Taşıma (2D Translation) hakkında bilgi veriniz.....	127
SORU-50: OpenGL ile menülerin tasarımı hakkında bilgi veriniz.....	128
SORU-51: OpenGL ile kullanıcı ile iletişimi (user interaction) hakkında bilgi veriniz.....	130
SORU-52: DEV CPP ile OpenGL Derleme hakkında bilgi veriniz.....	134
SORU-53: Çokgenler ve OpenGL hakkında bilgi veriniz.....	134
SORU-54: Taşıma Algoritması (Flood Filling Algorithm) hakkında bilgi veriniz.....	136
SORU-55: Sınır Doldurma Algoritması (Boundary Filling Algorithm) hakkında bilgi veriniz.....	137
SORU-56: Çokgenlerin Doldurulması (Filling Polygons) hakkında bilgi veriniz.....	138
SORU-57: Çokgenlerin Üçgene Çevrimi (Splitting Polygons to Triangles) hakkında bilgi veriniz..	139
SORU-58: Çift Tamponlama (Double Buffering, Çift Arabellek) hakkında bilgi veriniz.....	139
SORU-59: Uzaysal Çözünürlük (Spatial Resolution) hakkında bilgi veriniz.....	140
SORU-60: OpenGL'in Çalışma Sırası hakkında bilgi veriniz.....	141
SORU-61: Çokgen (Poligon, Polygon) hakkında bilgi veriniz.....	142
SORU-62: Open GL hakkında bilgi veriniz.....	146
SORU-63: Yumuşatma (Antialiasing) hakkında bilgi veriniz.....	148
SORU-64: Ortanokta Çember Çizimi (Circle Drawing with Midpoint Algorithm) hakkında bilgi veriniz.....	149
SORU-65: Kutup Koordinatları ile Çember Çizimi (Circle Drawing with Polar Coordinates) hakkında bilgi veriniz.....	150
SORU-66: Pisagor Yöntemi İle Çember (Pythagorean Theorem in Circle Drawing) hakkında bilgi veriniz.....	152

SORU-67: Bresenham Doğru Çizim Algoritması (Bresenham's Algorithm) hakkında bilgi veriniz.	153
SORU-68: Doğrudan Çizim Algoritması (Direct Draw Algorithm , DDA) hakkında bilgi veriniz....	154

Bilgisayar Grafiği (Computer Graphics)

SORU-1: Graph Veritabanı Karşılaştırmaları yapınız.

Özet

Bu doküman AllegroGraph, Virtuoso, 4store, Bigdata, Mulgara, Oracle ve OWLIM veritabanlarının karşılaştırılmasını içermektedir. Doküman 3 kısımda hazırlanmıştır. İlk kısımda belirtilen veritabanları hakkında genel bilgi verilmiş, ikinci kısımda da karşılaştırma matrisi sunulmuş ve son kısımda da veritabanlarının performansları değerlendirilmiştir.

1. **Veritabanları**
2. **AllegroGraph**

AllegroGraph kapalı-kodlu, modern, yüksek performanslı, kalıcı bir graph veritabanıdır. AllegroGraph hafızayı etkin şekilde kullanır ve aynı zamanda disk-bazlı saklama yapar. Bu sayede üstün performansı devam ettirirken milyarlarca 4lüye kadar ölçeklenebilir. AllegroGraph birçok istemci uygulamasından SPARQL, RDFS++ ve prolog'u destekler.

- İşlemlerin ACID (atomicity, consistency, isolation, and durability) özelliklerinin gerçekleştirimi
- Yerleşik Prolog
- Bedava ve ücretli sürümler
- Desteklenen İşlemler (Transactions): Commit, Rollback, Checkpointing
- Tam ve Hızlı Kurtarma
- %100 okuma eşzamanlılığı, neredeyse tam yazma eşzamanlılığı
- Çevrimiçi yedekleme, tam zamanında kurtarma, yedekleme (replication), sıcak bekleme (warm standby)
- İleri seviye metin indisi
- Tüm istemciler REST protokolüne dayanır: Java Sesame, Java Jena, Python, C#, Clojure, Perl, Ruby, Scala ve Lisp
- Desteklenen platformlar: Windows, Linux, Mac OS X (32 ve 64 bit)
- Çoklu-proses
- Kullanıcı tanımlı indisler
- Triple seviyesinde güvenlik
- Özel tarayıcı: Gruff

AllegroGraph RDF için W3C standartlarını karşılamak için geliştirilmiştir, dolayısıyla, bir RDF veritabanı olarak düşünülebilir. SPARQL protokolü için bir referans uygulamasıdır. SPARQL, tıpkı SQL'in ilişkisel veritabanlarında olduğu gibi, RDF için yani bağlı veriler için standart bir sorgulama dilidir.

2. Virtuoso

Virtuoso geleneksel RDBMS, ORDBMS, sanal veritabanı, RDF, XML, serbest metin, web uygulama sunucusu ve dosya sunucusu fonksiyonlarını tek bir sistemde sunan hibrid bir katman (middleware) ve veritabanı motorudur. Bahsedilen fonksiyonlar için adanmış sunucular yerine Virtuoso “evrensel bir sunucudur”. Bu sayede tekil çoklu-thread sunucu sürecini sağlayarak çoklu protokolleri gerçekleştirir. Virtuoso'nun açık-kaynak kodlu versiyonu OpenLink Virtuoso olarak da bilinir.

Virtuoso, işletim sisteminin thread desteği ve çoklu CPU yapısından faydalanmak üzere tasarlanmıştır. İstemciler arasında paylaşılan ve ayarlanabilen bir thread havuzuyla, tekil bir süreçten oluşur. Çoklu-thread tekil bir indis ağacında birbirleriyle en az etkileşimde olacak şekilde çalışabilir. Veritabanı sayfalarının 1 cache'i tüm threadler arasında paylaşılır ve eski kirli sayfalar diske “arka plan” süreci olarak yazılır.

Veritabanı her zaman temiz bir “checkpoint” durumuna sahiptir. Bu sayede temiz yedeklemeler yapılabilir ve işlem (transaction) kayıt defteri sayesinde herhangi bir zamandaki (checkpoint) duruma geri dönülebilir.

Virtuoso dinamik kilitlemeyi sağlar ve kilitleme alt seviyelerde olabileceği gibi kullanım durumuna göre sayfa seviyesinde de olabilir. Kilitleme sadece başka bir işlem aynı sayfada kilit tutmadığı zamanda olur, bu sayede, hiçbir zaman çıkmaz (deadlock) durumu yaşanmaz.

Virtuosos’un eşsiz hibrid mimarisi tek bir ürün içerisinde farklı sunucu özelliklerini sunmasını sağlar. Bu özelliklerden bazıları şöyledir:

- İlişkisel, RDF ve XML veri yönetimi
- Serbest metin içeriği yönetimi ve tam metin indisleri
- Doküman web sunucusu
- Bağlı veri sunucusu
- Web uygulama sunucusu
- Web servis deployment (SOAP or REST)
- Platform desteği: Windows (32 & 64 Bit), Mac OS X, Linux (32 & 64 Bit), Solaris (32 ve 64 Bit versiyonları için SPARC ve x86_64).
- Güvenlik (SSL, şifreleme, kimlik doğrulama)
- Replikasyon
- Kümeleme ve yüksek erişebilirlik
- Geçişli Clojure sorguları
- Sesame, Jena ve Redland için veri sağlayıcıları
- Sorgu dili desteği: SQL, SPARQL ve XQuery
- Transaction desteği

Virtuoso, çevik girişimler ve kişiler için yenilikçi ve çoklu-model veri sunucusudur. Veri yönetimi, veriye erişim ve entegrasyonda agnostik ve rakipsiz bir çözüm sunmaktadır.

3. 4Store

4Store, RDF verisi tutan bir veritabanı depo ve sorgu motorudur. Garlik tarafından birincil RDF platformu olarak 3 yıl boyunca kullanılmış ve güvenli ve güçlü olarak kendini ispat etmiştir.

4Store’un ana gücü performansına, ölçeklenebilir ve kararlı oluşuna dayanır. RDF saklamak ve SPARQL sorgusunun ötesinde çok fazla özellik sunmasa da, ölçeklenebilir, güvenli, hızlı ve etkin bir RDF deposu aranıyorsa, 4Store listede olması gereken yazılımlardan birisidir.

GNU genel açık lisans ile sunulan 4Store, ANSI C99 koduyla yazılmıştır ve UNIX-türevi sistemlerde çalışmak üzere tasarlanmıştır. Ancak tekil Mac OS X makinelerde çalıştığı da bilinmektedir. Kümeleme desteği için Avahi multicast DNS kütüphanesine ihtiyaç duyar ki bu kütüphane Mac OS X’te rahatça bulunan bir kütüphane değildir.

Bilindiği üzere RDF depolarında bulunan bazı özellikler, tipik bilgisayar ortamlarına deploy edildiklerinde güvenlik riski taşımaktadır. 4Store’un saldırılara karşı herhangi bir risk taşımaması için önemli bir çaba sarf edilmiştir. 4Store’un özellikleri özetle şu şekildedir;

- Açık kaynak-kodlu (GPL v3)
- Ölçeklenebilir
- Güvenli
- Hızlı
- Güçlü

4. Bigdata

Bigdata opsiyonel işlemleri (transactions), yüksek eşzamanlılığı ve yüksek G/Ç oranları destekleyen, yatay olarak ölçeklenebilen bir depolama ve üründür. Bigdata en başından beri 100den 1000lere kadar makineye sahip dağıtık veritabanı mimarileri için tasarlanmıştır ancak tekil-sunucu sisteminde de yüksek performansla çalışmaktadır.

Bigdata mimarisi, kümeler üzerinde veriye hassas (data-sensitive) dağıtık hesaplama, indisleme ve yüksek seviye sorguları için yüksek performanslı bir platform sağlar. Semantik web veritabanı katmanını büyük ilgiyi görmüş olsa da, Bigdata mimarisi de birçok veri modeli, iş yükü ve uygulama için uygundur.

Bigdata, RDFS ve sınırlı OWL'yi destekleyen yüksek performanslı bir RDF veritabanı içermektedir. İndislerin dinamik anahtar-aralık parçalanabildiği bir kümede dağıtık işlem yapabilen tek RDF veritabanıdır. Bu sayede deploy edilmiş olan footprint, her yeni makine eklendiğinde verinin tekrar yüklenmesine gerek kalmaksızın, veri ölçeğiyle birlikte büyüyebilir. Bigdata RDF veritabanı özellikle çok büyük semantik veri setleri için tasarlanmıştır. Esnek veri modeli içerisinde RDF, özellikle neredeyse-gerçek-zamanlı veri entegrasyonuna uyumlu bir semantik web teknolojisidir.

Bigdata'nın temel özellikleri şöyledir:

- SPARQL desteği
- RDFS+.
- Hızlı yükleme ve sorgu
- Triple ve Quad desteği
- Tam metin indislemesi
- Petabyte ölçeği
- Dinamik sharding
- Açık kaynak kod ve lisans opsiyonları
- Yüksek performans
- Yüksek eşzamanlılık
- Yüksek erişilebilirlik

Bigdata sürekli olarak geliştirilmektedir. Uzaysal indisleme, analitik sorgular ve yeni sorgu optimizasyonları planlar dahilinde yer almaktadır.

5. Mulgara

Mulgara, Kowari projesinin triplestore'udur. Açık-kaynak kodlu, ölçeklenebilir ve işlem-güvenlidir (transaction-safe). Mulgara durumlar iTQL ve SPARQL sorgu dilleri ile sorgulanabilir.

İlişkisel sistemlerde metadata ile uğraşırken karşılaşılan yüksek sayıdaki tablo bağlantıları (join) sebebiyle Mulgara ilişkisel veritabanına dayalı değildir. Bunun yerine Mulgara, metadata yönetimi için optimize edilmiş yepyeni bir veritabanıdır. Mulgara metadata'yı, W3C RDF standardı gibi, özne-yüklem-nesne şeklinde tutar. Mulgara'dan metadata RDF ve Notation 3 formunda export veya import edilebilir. [6]

Mulgara'nın genel özellikleri şöyledir:

- RDF desteği
- Sunucu başı çoklu veritabanı modelleri
- Basit, SQL-tarzı sorgu dilleri
- Tam metin arama özelliği
- Yüksek saklama kapasitesi

- Çoklu-işlemci desteği
- Düşük hafıza gereksinimi
- Tam işlem desteği
- Kümeleme ve depo seviyesi fail-over
- Desteklenen protokoller: Jena, JRDF, SOAP, SDK
- Web tabanlı yönetim konsolu
- Desteklenen platformlar: Windows, Unix ve Linux, Solarix, Mac OS X, IRIX (32 ve 64 bit)

Mulgara açık kaynak kodlu olup, kaynak kodlar herkesin erişimine açıktır.

6. Oracle

Oracle Database 11g Semantic Technologies açık, standart-tabanlı, ölçeklenebilir, güvenli, güvenilir ve performanslı bir RDF yönetim platformudur. Graph veri modeline dayanır ve ilişkisel veri tiplerinde olduğu gibi RDF verisi (triples) yazılır, indislenir ve sorgulanır.

Uygulama geliştiriciler Oracle Database 11g'nin gücünü kullanarak semantik olarak güçlendirilmiş iş uygulamaları tasarlayıp geliştirebilir. Oracle Database 11g'nin önemli özellikleri şöyledir:

- Doğal RDF/OWL veri yönetimi
- Ölçeklenebilir ve güvenli platform
- +10milyar triple
- SQL, Jena, Sesame ve SPARQL desteği

7. OWLIM

OWLIM Java ile yazılmış, doğal RDF gerçekleştirimine sahip olan bir semantik depo çözümüdür. Farklı işletim ortamları için 2 farklı versiyonu vardır:

SwiftOWLIM: Hafıza-içi (in-memory) Rdf veritabanı, anlam-motoru ve sorgu cevaplama motorudur. Optimize edilmiş indisleri ve veri yapılarını kullanarak on milyonlarca RDF ifadesini standart masaüstü bilgisayarda işleyebilir. Hafızada çalışmasının da etkisiyle dünyanın en hızlı semantik deposudur ve 1.000 dolarlık bir makinede saniyede 50.000 veri yükleyebilmektedir. SwiftOWLIM kullanıma açıktır (bedavadır).

BigOWLIM: SwiftOWLIM'in kurumsal versiyonu olan BigOWLIM işlemci başına ücretlendirilmektedir. BigOWLIM, kuruluş seviyesinde bir çözümdür ve on milyarlarca ifadeyi kontrol edebilmektedir. BigOWLIM birkaç depolama ve sorgulama optimizasyonu kullanarak, milyarlarca sayıdaki bağlı açık veride bile çok başarılı ekleme ve silme performansları göstermektedir. Her ne kadar teorik olarak belirli bir limit olmasa da 64GB hafızası olan bir makinede 20 milyar ifadeyi kaldırabilmektedir. Bu sayıdan fazlasında performans ciddi oranda düşmektedir.

OWLIM'in önemli özellikleri şöyledir:

- Java ile geliştirilmiş
- Sesame ve Jena desteği
- Hem lisanslı hem de ücretsiz sürüm
- +10 milyar triple
- Küme içerisinde dikey ölçeklenebilme
- Yedekleme
- Geo-Uzaysal (Geo-Spatial) eklenti desteği
- RDFS, OWL ve QL desteği

OWLIM birçok projede ve yazılımda kullanılmakta olup kullanıcılar tarafından genellikle olumlu yorumlar almaktadır.

1. Karşılaştırma Matrisi

Özellik	Alle gro Gra ph	OWLIM	Or acl e	Mu lga ra	Big data	4Store	Virtuoso
Açık- Kayn ak Kod	✓	✓	-	✓	✓	✓	✓
Lisan s	✓	✓	✓	-	-	-	✓
Üreti ci Firm a	Fran z	Ontotext	Ora cle	Co mm unit y / Ko war i For k	Syst ap	garlik	OpenLink
Kulla nan firma lar	Pfize r, NAS A, Koda k, Mitr e, Ford , Citi, Nov artis, Cisc o, Ado be, KTF, KDDI , Geo BC	BBC, Governace, Raytheon, TSO, Nexcom, Charisma, Korea Telecom, ICT, Cola Spain	Cis co, Lill y, Pfiz er, We stla w, SIB, Cle vel an d Cli nic, Pol ar Lak e, Ray the on, Ly mb ia	Pfiz er, Cisc o, SIB, We stla w, Hut chi nso n, Uni ver sity of Mic hig an ve Tex as	Cam brid ge Sem anti cs, Info rbix, NDA com pani es (fort une 500)	DataPatrol, eRDF, OpenPSI, Market Blended Insight	Elsevier, Globo, UCB, Kodak, BBC, Lifeway, CGI Federal, Nato, NASA, Atos Origin
Dökü	Tam	Orta	Dü	Ort	Orta	Open Source	İyi

mantasyon ve Yardım	destek Orta		şük (satın al: yüksek ve pahalı)	a		(Az)	
Proje Lideri	Peter Norvig	Barry Bishop	?	David Wood / Norman Gray	SYSTAP, LLC	Steve Harris	Orri Erling
Yazıldığı Dil	Common Lisp	Java	?	Java	Java	C	?
İşletim Sistemi Gereksinimleri	W L M (32-64)	W L M (32-64)	W L M (32-64)	W L M (32-64)	W L M (32-64)	Tercihen: Linux – 64	W L M (32-64)
Bellek Gereksinimi	150 m tripl e için 40gb yeterli	>=32 (ciddi kullanım için)	?	Düşük >= 64 MB	Deployment' a göre değişiyor (scalable)	>= 4GB	>= 64 MB
Tripl e Sayısı	5mil – 500 mil – sınırsız (lisansa göre)	>= 20 milyar Veriye göre değişebilir	> 100 milyar	?	50 milyar	2-10 milyar	Sınırsız (cluster'da)
Tripl	1	1 milyar / 13	3	?	1	http://www4.wiwiwiss.fu-	1 milyar / 8 saat

e Yükle me Süres i	milyar / 36 dk. (32 core Intel E5520, 2.0 GHz, with 1 TB RAM, Red Hat v6.1.)	saat http://www.ontotext.com/owlim/benchmark-results/owlim-5	milyar / 105 dakika (64 core, 512 gb ram, 128 paralel thread)		milyar / 1 saat (16 düğümli cluster'da)	berlin.de/bizer/BerlinSPARQLBenchmark/results/V6/index.html#resultsExplore	http://www.openlinksw.com/dataspace/dav/wiki/Main/VOSVirtuoso6FAQ
Desteklenen Sorgu Dili	SPARQL, TwinQL, Prolog	SPARQL ve SeRQL	SPARQL or SQL	SPARQL, iTQL	SPARQL	SPARQL 1.0	SQL, SPARQL, XQuery ve XPath
Versiyon Numarası	v4.X	v4	11g	V2.1	v1.2.0	v1.1.3	v6.3
Danışmanlık	Franz.com	Ontotext.com	Terforce	-	SYSTAP, LLC	-	Openlinksw.com
Client API	SPARQL over http, Java Pool, Python, Lisp, Clojure	Sesame, Jena ve ORDI	Jena, Joski, ARQ, TD B, SD B, Sesame, Pellet, D2 RQ, Jett	iTQL Java Bean, Jena, JRD F, Shell	Sesame, SPARQL 1.1	SPARQL over HTTP, Java, C, Python, PHP, Ruby ve Perl	Jena, Sesame, Redland

			y, Cyt osc ape , GA TE, Pro tég é				
Mail Grub u / Foru m	✓ / -	✓ / -		✓ / -	✓ / ✓	✓ / -	✓ / ✓
Geçiş li Kural Tanı mı	✓	✓	✓	✓	✓	-	✓
Çokl u Kulla nıcı (Eşza manlı lık)	✓	✓	✓	✓	✓	✓	✓
Saniy ede İşley ebildi ği Sorg u Sayıs ı	Veri ye göre deği şiyor 4000 0 / sani ye	Veriye göre değişiyor 6500 / saniye	?	?	Veri ye göre deği şiyor 56,0 00 Que ry Mix – Mini Mac (Inte l)	Veriye göre değişiyor ancak çok hızlı değil (100/saniye)	Veriye ve konfigürasyona göre değişiyor
Dağı tıklık Dest eği	✓	✓	✓	✓	✓	✓	✓
Trans actio	✓	✓	✓	✓	✓	-	✓

n							
Lisans Ücreti	Dev. - 4000 \$ (4 cpu'ya kadar) Ent. - 4000 \$ / cpu	SE - 1200€ / cpu EE - 3200€ / cpu	?	-	(Aylık) 10.000\$ / 50 deployment	-	0 - 70.000\$ http://virtuoso.openlinksw.com/pricing/
GeoSpatial Destegi	✓	✓	✓	?	✓	?	✓

Tablo-4.1: Karşılaştırma matrisi

1. Performans

Her ne kadar Graph veritabanları için bazı Benchmark sonuçları olsa da gerçek dünya verileri ile bilgisayar ortamı verileri genellikle birbirini tutmamaktadır. Aşağıda bazı sonuçlara yer verilmiş olsa da gerçek sonuçlar, tek bilgisayar üzerinde yapılarak elde edilebilir. Sağlıklı bir karşılaştırma yapabilmek için aynı sistem üzerinde testlerin yapılması, elde edilen sonuçların geçerliliğini arttıracaktır.

- Test 1 (Reveltyx Değerlendirme Raporu)
- Nitel Değerlendirme

	Area	Notes	Virtuoso	OWLIM	Oracle	Mulgara	Parliament	AllegroGraph
General	Usability	Overall perception of usability						
	Support	Overall perception of support						
	Licensing/Cost	Assessment of licensing/cost						
Functional	System	System architecture, etc						
	Enterprise	Enterprise capabilities						
	Data Import	Ability to import data						
	API	APIs available						
	Querying	Query functionality						
	Inferencing	Inferencing support						
	Interoperability	Jena/Sesame interop						
Performance	Operational	Monitoring/canceling						
	Speed	Single-user, SP2						
		Single-user, BSBM						
	Throughput	Multi-user, SP2						
	Correctness	SP2 queries/data						
		BSBM queries/data						
LEGEND: Green is 'very good', red is 'not so good', and yellow is in-between. Gray is unknown.								

Şekil-4.1: Nitel değerlendirme

- Genel Değerlendirme

Area	Feature	Virtuoso	OWLIM	Oracle	Mulgara	Parliament	AllegroGraph
Usability	Ease of Installation						
	Ease of Development						
	Ease of Administration						
	Overall Usability						
Support	User Forum/Mailing List						
	Defect Tracking System						
	Documentation						
	General Support						
Licensing & Cost							
LEGEND: Green is 'very good', red is 'not so good', and yellow is in-between.							

Şekil-4.2: Genel değerlendirme

- Fonksiyonel Değerlendirme

Area	Feature	Virtuoso	OWLIM	Oracle	Mulgara	Parliament	AllegroGraph
System	Architecture						
	Underlying Persistence						
	Standards Compliance						
	Extensibility						
	Overall System						
Enterprise Capabilities	Concurrent Access						
	Transactions Support						
	Security Support						
	Overall Enterprise						
Data Import	Batch Loading						
	Data Formats Supported						
	Materialization on Load						
	Overall Import						
API	SPARQL Protocol						
	Jena API						
	Native API						
	Overall API						
Querying	SPARQL 1.0 Support						
	Indexing Support						
	Query Plan Access						
	Overall Querying						
Inferencing	Inferencing Support						
	Distinguish Inferred Data						
	Overall Inferencing						
Interoperability	Jena Interoperability						
	Sesame Interoperability						
	Overall Interoperability						
Operational	Query/Connection Timeouts						
	Query/Connection Cancel						
	Query/Connection Monitoring						
	Overall Operational						
LEGEND: Green is 'very good', red is 'not so good', and yellow is in-between. Gray is unknown.							

Şekil-4.3: Fonksiyonel değerlendirme

- Performans Testleri

Uygulanan Testler

Test Name	Benchmark	# Query Variations	Ontology Sizes	Concurrent Connections	Notes
Single user ('Baseline')	SP2Bench	14	10k, 50k, 250k, 1m, 5m	1	
Single user	BSBM	12	10k, 250k, 1m, 25m, 100m	1	
Concurrency	SP2Bench derivative	4 (with derivatives)	10k, 50k, 250k, 1m, 5m	1, 4, 16, 64	Query 2 data ignored

Tablo-4.2: Uygulanan testler

Genel Görünüm

System Under Test		Test Environment	Test Condition Summary				
System	API		Total Test Conditions	With Data	All Repeats Good	All Repeats Rejected	Some Repeats Rejected
AllegroGraph	SPARQL	EC2	70	70	37	22	11
Mulgara 2.1.9	Native	EC2	70	70	46	18	6
Oracle	SPARQL	EC2	70	66	41	10	15
OWLIM	SPARQL	EC2	70	70	48	14	8
Parliament	SPARQL	EC2	70	70	47	16	7
Virtuoso	SPARQL	EC2	70	70	52	10	8
Mulgara 2.1.8	Native	EC2	70	70	45	18	7
Oracle	Jena	EC2	70	70	39	12	19
Virtuoso	Native	EC2	70	70	51	14	5
AllegroGraph	SPARQL	Local	12	12	5	7	0
Mulgara 2.1.9	Native	Local	18	18	5	13	0
Oracle	Jena	Local	70	59	48	11	0
OWLIM	SPARQL	Local	18	18	9	9	0
Parliament	SPARQL	Local	24	24	18	5	1
Virtuoso	SPARQL	Local	70	41	30	3	8

Tablo-4.3: Genel görünüm

SP2 Performans Analizi (Top 3: Virtuoso – OWLIM – Oracle)

Query	Weight	Virtuoso	OWLIM	Oracle	Mulgara	Parliament	AllegroGraph
1	3	1	3	5	3	2	6
2	4	4	2	3	1	5	6
3a	4	3	3	2	1	5	6
3b	4	1	1	1	5	1	6
3c	4	1	1	1	5	1	6
4	4	4	2	1	4	4	6
5a	4	1	2	3	4	6	6
5b	4	3	2	4	1	5	6
6	2	1	6	2	5	3	3
7	1	4	6	2	4	2	4
8	3	2	1	6	4	3	5
9	4	2	5	3	1	4	6
10	2	5	2	6	3	4	1
11	3	1	6	4	2	5	3
Rank Average		2.36	3	3.07	3.07	3.57	5
Wt Rank Average		2.26	2.7	2.93	2.93	3.7	5.35

Tablo-4.4: SP2 performans analizi

Single User (BSBM) Performans Analizi (Top 3: OWLIM – Virtuoso – Mulgara)

Query	Frequency	BigOWLIM	Virtuoso	Mulgara	Parliament	AllegroGraph	Oracle
1	1	3	1	4	2	5	6
2	6	1	2	3	5	4	6
3	1	3	1	4	2	5	6
4	1	3	1	4	2	5	6
5	1	1	2	6	3	4	5
6	1	1	2	3	4	5	6
7	4	1	3	2	4	4	6
8	2	2	1	3	5	4	6
9	4	2	1	6	3	3	5
10	2	1	2	3	4	5	6
11	1	2	1	4	3	5	6
12	1	1	3	2	4	5	6
Composite Rank	25	1	2	5	3	4	5
Wt Rank Average		1.52	1.8	3.56	3.84	4.16	5.8

Tablo-4.5: BSBM performans analizi

- **Test 2**

Donanım:

- İşlemci: Intel i7 950, 3.07GHz (4 cores)
- Bellek: 24GB
- Hard Disk: 2 x 1.8TB (7,200 rpm) SATA2.

Yazılım:

- İşletim Sistemi: Ubuntu 10.04 64-bit, Kernel 2.6.32-24-generic
 - Filesystem: ext4
 - Seperate partitions for application data and data bases.
- Java Version and JVM: Version 1.6.0_20, OpenJDK 64-Bit Server VM (build 19.0-b09)

Test Prosedürü:

Sistemlerin performansları BSBM veri setlerinin Turtle temsilcilerinin yüklenmesiyle ölçülmüştür. Teste giren sistemlerde “inferencing” yapmamıştır. Sistemlerin sorgu performansları 500 BSBM karışık sorguları ile yapılmıştır ve SPARQL protokolü kullanılmıştır. Ağ gecikmelerini engellemek için tüm testler aynı bilgisayar yapılmıştır. [11]

Sonuçlar:

Sistem	100 Milyon	200 Milyon
4Store	26:42	1:12:04
Bigdata	1:03:47	3:24:25
BigOWLIM	17:22	38:36
Virtuoso	1:49:26	3:59:38

Tablo-4.62: Sonuçlar

SORU-2: Hesaplamalı Geometri (Computational Geometry) hakkında bilgi veriniz.

Esas itibarıyla mühendis kelimesinin kökü olan, hendese kelimesini büyük ölçüde karşılayan geometri kelimesinin başına hesaplamalı gibi bir kelime getirmek anlamlı değildir. Çünkü zaten geometri hesaplamalı bir çalışma alanıdır. Buradaki hesaplamalı kelimesi daha çok bilgisayar ile işlenen anlamını taşımaktadır. Yani hesaplamalı geometri ile kasıt (computational geometry) bilgisayar ile işlenen geometri çalışmalarıdır. İngilizcede kullanılan computational kelimesi bu anlamda, bilgisayarı (computer) çağırması açısından daha doğru bir tercihtir.

Geometri ve bilgisayar çalışmalarının kesişim noktası olan hesaplamalı geometri konusuna bilgisayar bilimleri açısından bakıldığında, geometrik problemleri çözen algoritmaların geliştirilmesi ve iyileştirilmesi (optimisation) olarak düşünülebilir.

Hesaplamalı geometri çalışmalarının ilk çıkışı, bilgisayar grafikleri konusunda yapılan çalışmalardır. Örneğin bir bilgisayar animasyonu veya bilgisayar destekli çizim tasarım ve üretim yazılımları (CAD / CAM) hesaplamalı geometrinin doğuşuna ve gelişmesine öncülük etmiştir.

Günümüzde, ayrıca robot çalışmaları, coğrafi bilgi sistemleri (geographic information systems, GIS) veya entegre devre tasarımı (integrated circuit , IC) gibi konularda da hesaplamalı geometriden yoğun olarak istifade edilmektedir.

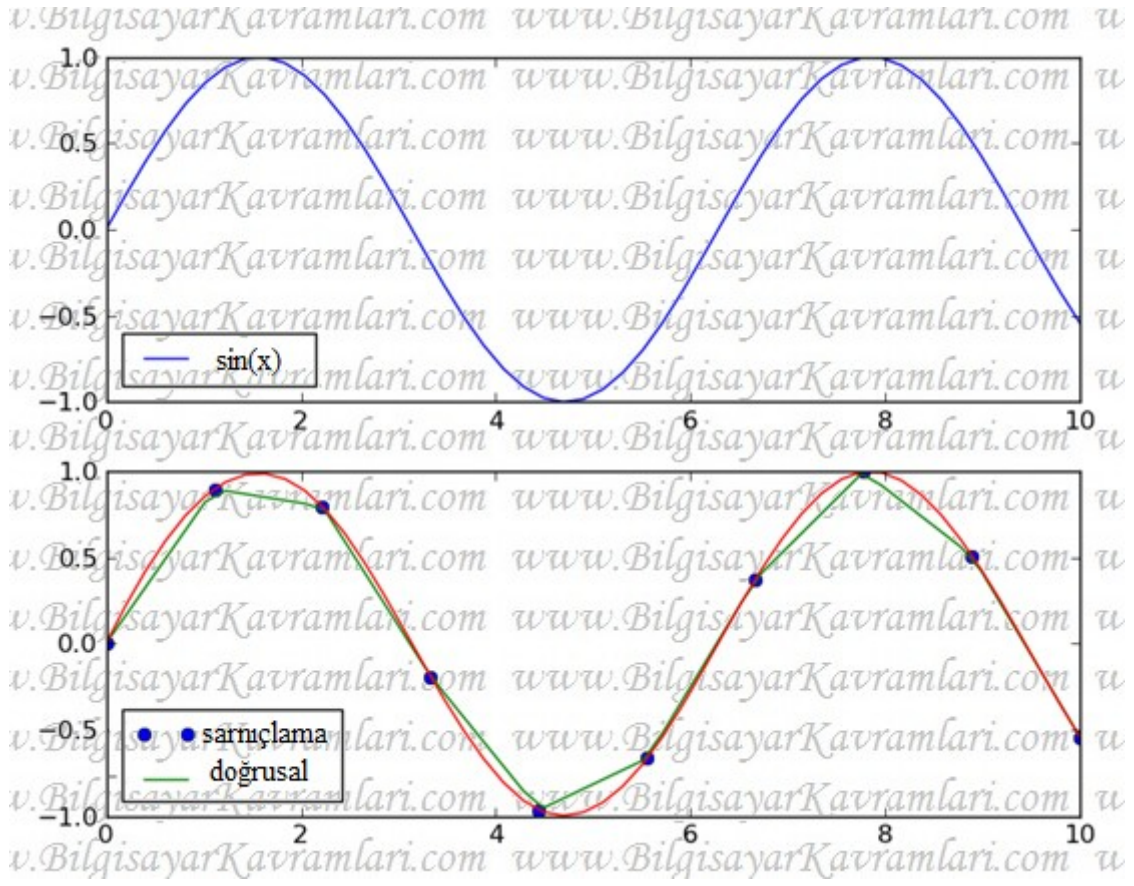
Yapı olarak çoğu bilgisayar bilimleri çalışmasında görüldüğü üzere iki farklı usul görülebilir.

Terkîbî hesaplamalı hendese (combinatorial computational geometry): Bilgisayar bilimlerinde bulunan terkipî çalışmaların (combinatorial) bir alt grubudur. Genel olarak birden fazla ihtimali içersen sonlu sayıdaki elemalı kümelerin (finite element sets) üzerinde çalışan ve bu kümeler üzerinde ihtimal veya ittihat (birleştirme) (combination) işlemleri yapan çalışmalara, birleştirme anlamında terkip (combinatorial) ismi verilmektedir. Bu alan, bilgisayar bilimleri açısından da önemli bir yere sahiptir.

Sayısal hesaplamalı hendese (numerical computational geometry): Bilgisayar bilimlerinde, karşılaşılan problemlerin çözülmesi için geliştirilen diğer bir yöntem de sayısal analiz yöntemlerini kullanmaktır. Buna göre sistemden sonlu sayıda noktanın analizi yapılmaktadır. Örneğin sarnıçlama yöntemi ile seçilen noktalar üzerinde hesaplama yapıp sisteme genellemek gibi yöntemler kullanılır.

Yukarıdaki iki farklı hesaplamalı geometri usulünün farkını anlatmak için bir misal verelim.

Örneğin aşağıda (üstteki mavi çizimle gösterilen) fonksiyon, basit bir $\sin(x)$ fonksiyonudur.



Şekil-4.4: Fonksiyon

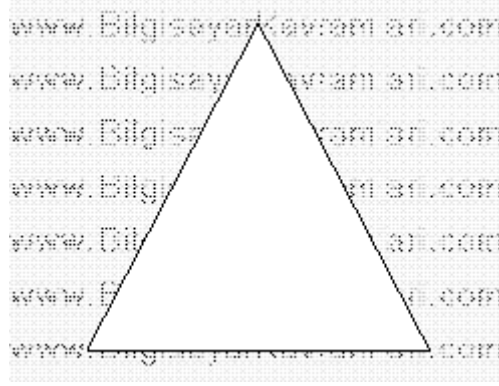
Bu fonksiyonun analitik değerinin $\sin(x)$ olduğunu bilmeden, fonksiyona çeşitli değerler vererek şekli çizilirse, alttaki resimde görüldüğü üzere doğrusal bir fonksiyon bulunur. Bu fonksiyon, orjinal fonksiyona yakın ancak tam değildir. Örneğin integral almak gibi basit geometrik işlemler, yukarıdaki ilk fonksiyon üzerinden analitik olarak yapılabileceği gibi, ikinci fonksiyondaki örnek noktalar üzerinden de yapılabilir.

SORU-3: Sierpinski Üçgeni (Sierpinski Triangle) hakkında bilgi veriniz.

Orijinal ismi Sierpiński olan Polonyalı matematikçi tarafından 1915 yılında tanımlanan bu üçgen, yapı olarak özyineli (recursive) iç üçgenlerden oluşur.

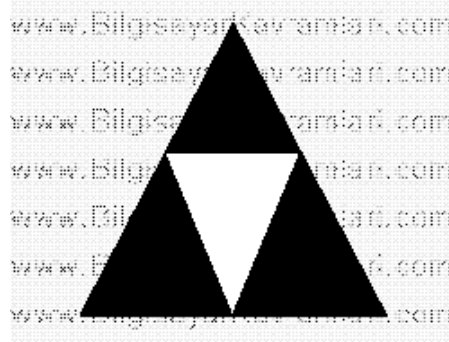
Kısaca üçgenin her parçası, üç adet alt üçgen oluşturacak şekilde bölünür.

Örneğin aşağıdaki eşkenar üçgeni ele alalım:



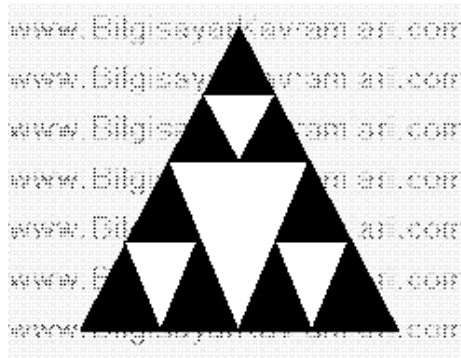
Şekil-4.5: Eşkenar üçgen

Bu üçgenin üç köşesinde üç ayrı üçgen oluşturarak işlemimize başlayabiliriz.



Şekil-4.6: Eşkenar üçgen

Yukarıdaki şekilde görüldüğü üzere 3 adet siyah eşkenar üçgen oluştu. Bu üçgenlerin her birisini ilk başta ele aldığımız üçgen gibi bir adım daha ilerleyerek üç parçaya bölmeye çalışalım:



Şekil-4.7: Eşkenar üçgen

Görüldüğü üzere bölme işlemi sonucunda her alt üçgeninin üç köşesinde, ilk adımda yapılan işleme benzer şekilde 3 ayrı üçgen oluşturulmuştur.

Bu işlem sonsuza kadar devam ettirilebilir. Ancak bilindiği üzere bilgisayar bilimlerinde sonsuz yoktur. Dolayısıyla bu işlem için bir limit belirlenir ve belirlenen limite kadar işlem devam ettirilir.

Sierpinski üçgeninin kodlanması

Yukarıda anlatılan üçgeni, Scheme dilinde programlamaya çalışalım. Bu kodlama sırasında PLT Scheme kullanılacaktır.

Öncelikle gerekli olan bazı bilgilerimizi hatırlayalım. Çizim için draw.ss teachpackinin yüklü olması gerekiyor.

Bunun için Language menüsünden Add Teachpack seçeneğini seçip draw.ss paketini yüklüyoruz.

Ardından basit bir üçgenin scheme ile nasıl çizileceğine bakalım.

Ne yazık ki üçgen çizen hazır bir fonksiyonumuz bulunmuyor. Bunun için üçgeni oluşturan üç ayrı doğruyu ayrı ayrı çizdiren bir fonksiyonu bizim yazmamız gerekiyor.

```
(define (draw-triangle a b c)( and (draw-solid-line a b)(draw-solid-line b c)(draw-solid-line c a)))
```

Yukarıdaki draw-triangle fonksiyonu üç ayrı nokta almaktadır. Bu noktaları birleştiren üç ayrı doğru fonksiyon içerisinde çizilmektedir. Bilindiği üzere draw-solid-line fonksiyonu verilen iki nokta arasında bir doğru çizer. Bu noktalar make-posn yapısında olmalıdır. Örneğin çizim ekranında (ki ekranın sol üst köşesi 0,0 olarak kabul edilir) 100, 150 koordinatlarından 200, 250 koordinatlarına bir doğru çizmek istiyorsak aşağıdaki şekilde komut vermemiz gerekir:

```
(draw-solid-line (make-posn 100 150) (make-posn 200 250))
```

Üçgen çizen fonksiyonumuz olduğuna göre artık sierpinski üçgenini kodlamaya geçebiliriz.

```
sp.scm (define ...) Save
(define (sierpinski a b c n)
  (cond
    [(= n 0) true]
    [else
     (local ((define a-b (mid-point a b))
              (define b-c (mid-point b c))
              (define c-a (mid-point a c)))
      (and
        (draw-triangle a b c)
        (sierpinski a a-b c-a (- n 1))
        (sierpinski b a-b b-c (- n 1))
        (sierpinski c c-a b-c (- n 1))))]))
;;www.bilgisayarkavramlari.com

(define (mid-point a-posn b-posn)
  (make-posn
    (mid (posn-x a-posn) (posn-x b-posn))
    (mid (posn-y a-posn) (posn-y b-posn))))

(define (mid x y)
  (/ (+ x y) 2))
```

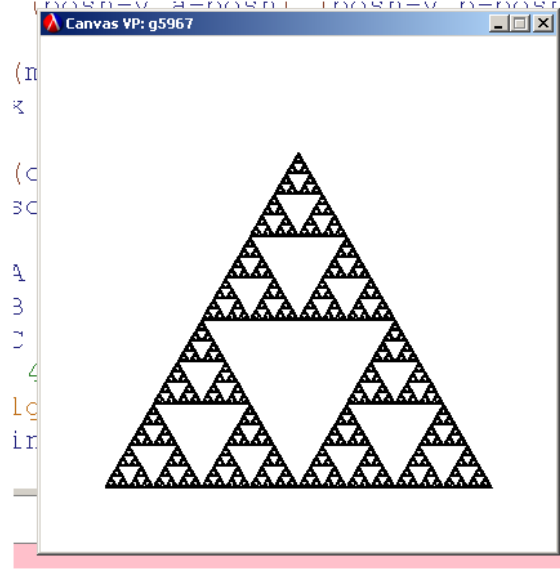
Şekil-4.8: Kodlar

```
(define A (make-posn 50 350))
(define B (make-posn 350 350))
(define C (make-posn 200 90))
  (start 400 400)
;;www.bilgisayarkavramlari.com
  (sierpinski A B C 10)
```

Görüldüğü üzere, üçgenin içerisine çizileceği üç nokta belirlenmiştir. Burada noktaları belirlerken, üçgenin eşkenar üçgen olması için ufak bir hesaplama yapılmıştır. Çizim tuvalimiz, 400 400 olarak verilmiş yani bize ekranda 400e 400 pixellik (imgecik) bir alan açılmaktadır. Bu alanı nispeten ortalayan bir üçgende A B ve C noktalarının birleştirilmesi ile teşekkül etmiş olur.

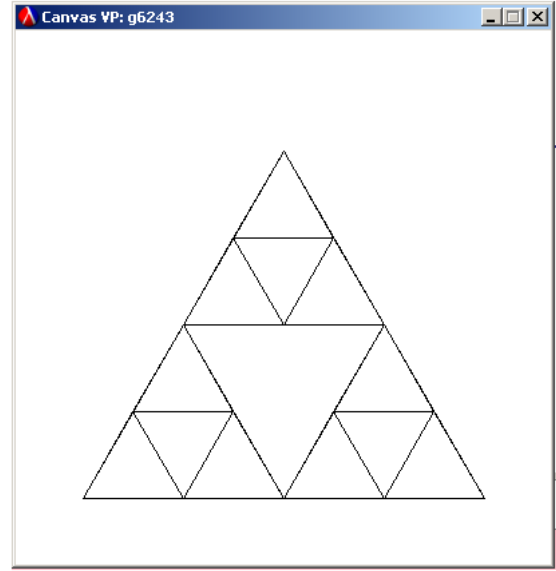
Son olarak sierpinski üçgenini çizen özyineli (recursive) fonksiyonumuzu bu üç noktanın koordinatları ve 10 adım gidileceğini belirten 10 sabit sayısı ile çağırıyoruz.

Sonuç olarak ekrana çizilen görüntü aşağıdaki şekildedir:



Şekil-4.11: Eşkenar üçgen

Yukarıdaki şekilde, 10 adım ilerlenmiş ve 10 adım sonunda detaylı bir görüntü oluşmuştur. Örneğin aynı üçgeni 3 parametresi ile, yani 3 adım ilerleyecek şekilde çağırıyordık:



Şekil-4.12: Eşkenar üçgen

Yukarıda verilen biraz daha ilkel halini görecektik.

SORU-4: Bezier Eğrisi Çizen Kod hakkında bilgi veriniz.

Soru: Üç noktanın koordinatlarını alarak bu noktalardan geçen eğriyi bezier algoritması ile ekrana çizen kodu yazınız.

Çözüm:

Bezier eğrisini çizebilmek için ikinci dereceden bir denkleme ihtiyaç duyulur. Burada denklemin ikinci derece olmasının sebebi 3 nokta ile çizim yapılmasının istenmesidir.

Öncelikle programımızda kullanacağımız basit temel fonksiyonları kodlayalım:

İki nokta verildiğinde bu iki noktanın ortasındaki noktayı hesaplayan fonksiyonu kodlayalım:

```
(define (mid-point a-posn b-posn)
```

```
(make-posn
```

```
(mid (posn-x a-posn) (posn-x b-posn))
```

```
(mid (posn-y a-posn) (posn-y b-posn))))
```

Yukarıda iki sayının orta değerini (aritmetik ortalamasını) hesaplayan mid fonksiyonunu kullandık. Bu fonksiyonu basitçe iki sayının toplamının yarısı olarak aşağıdaki şekilde tanımlayabiliriz:

```
(define (mid x y)
```

```
(/ (+ x y) 2))
```

Artık bezier fonksiyonumuzu kodlayabiliriz. Buradaki püf noktası sürekli olarak noktaların arasındaki noktaların hesaplanmasıdır. Bu açıdan bir fraktal üretimine benzetilebilir:

```
(define (bezier p1 p2 p3) (cond
```

```
[(too-small? p1 p2 p3) true]
```

```
[else (and (draw-solid-disk (mid-point (mid-point p1 p2) (mid-point p2 p3)) 3)
```

```
(bezier (mid-point (mid-point p1 p2) (mid-point p2 p3)) (mid-point p2 p3) p3)
```

```
(bezier p1 (mid-point p1 p2) (mid-point (mid-point p1 p2) (mid-point p2 p3)))
```

```
])))
```

Yukarıdaki kod 3 nokta alıp bu noktalar arasındaki mesafe çizilemeyecek kadar küçük oluncaya kadar (bu mesafe aynı zamanda çizimdeki noktaların seyrekliğini de belirtmektedir) ara nokta hesaplamakta ve hesaplanan bu ara noktaları çizmektedir. Yukarıdaki fonksiyonda bitişi belirten too-small? Fonksiyonunu aşağıdaki şekilde tanımlayabiliriz:

```
(define (too-small? p1 p2 p3)(cond[(< (abs(- (posn-x p1) (posn-x p2))) 10) true]
```

```
[else false]
```

))

Burada noktalar arası mesafe 10 veya daha düşükse artık yeni nokta üretimi duracaktır.

Çözüm çalıştırması:

```
(define p1 (make-posn 50 50))
```

```
(define p2 (make-posn 150 150))
```

```
(define p3 (make-posn 250 100))
```

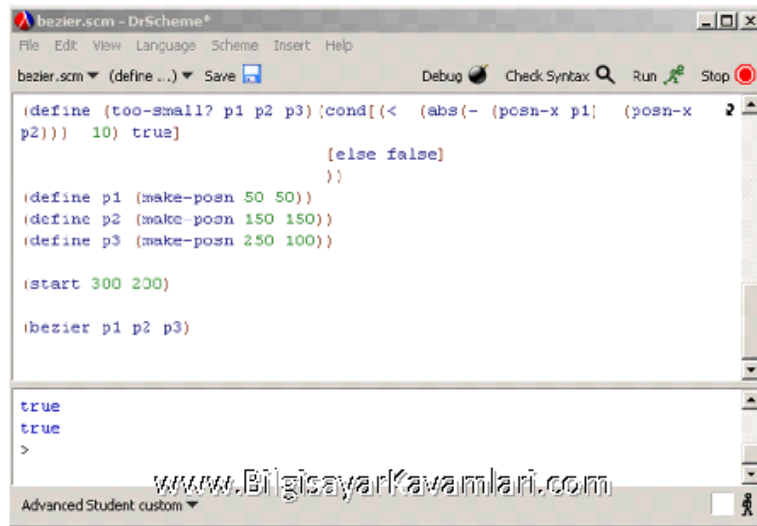
Üç nokta tanımlanmıştır

```
(start 300 200)
```

Çizim alanı açılmıştır

```
(bezier p1 p2 p3)
```

Ve son olarak fonksiyonumuza üç nokta parametre verilmiştir.



```
bezier.scm - DrScheme*
File Edit View Language Scheme Insert Help
bezier.scm (define ...) Save Debug Check Syntax Run Stop
(define (too-small? p1 p2 p3) (cond [(< (abs(- (posn-x p1) (posn-x p2))) 10) true]
                                     [else false]
                                     ))
(define p1 (make-posn 50 50))
(define p2 (make-posn 150 150))
(define p3 (make-posn 250 100))
(start 300 200)
(bezier p1 p2 p3)
true
true
>
```

Şekil-4.13: Kodlar

Kodun çalıştıktan sonra ekranda oluşturduğu çizim aşağıdaki şekilde görülebilir



Şekil-4.14: Çıktı

SORU-5: Bezier Eğrileri (Bezier Curves) hakkında bilgi veriniz.

Bilgisayar bilimlerinin bir çalışma alanı olan bilgisayar grafiklerinde kullanılan eğri biçimidir. Bezier eğri çiziminin özelliği parametrik olarak noktalar ile çalışmaları ve verilen noktalara göre bir eğri çizmesidir. Literatürde bézier curve veya bezier eğrisi olarak geçmektedir. İsmi bu hesaplama yöntemini ilk bulan Fransız matematikçiden gelmektedir.

Bezier eğrilerinin boyutlarından bahsetmek doğrudur. Buna göre bir bezier eğrisinin derecesi yada boyutu, bu eğriyi meydana getiren nokta sayısı ile ölçülebilir. Örneğin iki noktayı parametre olarak çalışan bir algorithma sonuç olarak bir doğru çıkacaktır.

Bu çizim için P_0 ve P_1 olmak üzere iki nokta aldığımızı düşünürsek, eğrinin formülü (daha doğrusu doğrunun formülü) aşağıdaki şekilde yazılabilir

$$B(t) = P_0 + t(P_1 - P_0)$$

Daha açık bir ifadeyle iki noktadan birini başlangıç (P_0) ve birini bitiş (P_1) noktası olarak kabul edecek olursak. Bitiş noktasına giden yol üzerindeki her nokta, başlangıç noktasına eklenen ve bitiş noktası istikametindeki bir fark kadar ötelenmiş noktadır.

Bu gösterimi açacak olursak :

$$(1-t)P_0 + tP_1$$

Olarak yazmak da mümkündür. Bu yazılım bezier denklemleri açısından daha anlamlıdır. Bunu daha yüksek seviyeli denklemlerde anlayacağız.

Bu yazı şadi evren şeker tarafından yazılmış ve bilgisayar kavramları.com sitesinde yayınlanmıştır. Bu içeriğin kopyalanması veya farklı bir sitede yayınlanması hırsızlıktır ve telif hakları yasası gereği suçtur.

Şimdi iki yerine 3 nokta ile teşkil edilen bir bezier eğrisini aynı mantıkla formülleştirmeye çalışalım.

Buna göre eğrimizde sırasıyla P_0 , P_1 ve P_2 noktaları olacaktır.



Şekil-4.15: Eğri

Yukarıdaki şekilde 3 noktadan oluşan ikinci derece eğri gösterilmiştir. Ayrıca eğrinin daha iyi anlaşılabilmesi için 3 noktayı birleştiren klavuz çizgileri de bulunmaktadır. Eğri aslında yukarıdaki iki çizimin arasında kalan şekil olup, eğrinin orijinali aşağıda açık olarak verilmiştir:



Şekil-4.16: Eğri

Yukarıdaki bu eğriyi elde etmek için kullanacağımız formül şu şekilde oluşturulur:

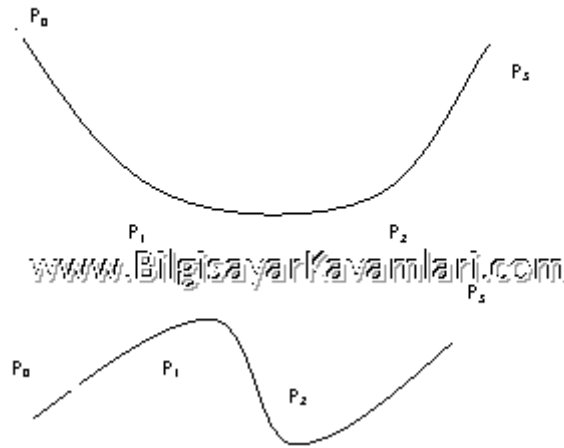
$$B(t) = (1-t)^2 P_0 + 2(1-t)t P_1 + t^2 P_2$$

Yukarıdaki denklemde dikkat edilirse binom üçgenindeki (pascal triangle) bulunan katsayılar kullanılmıştır. Bunun sebebi iki bilinmeyenli ikinci derece denklem açılımı olmasıdır.

Benzer şekilde üçüncü derece bezier eğrisi için de binom üçgeninin üçüncü satırı kullanılır:

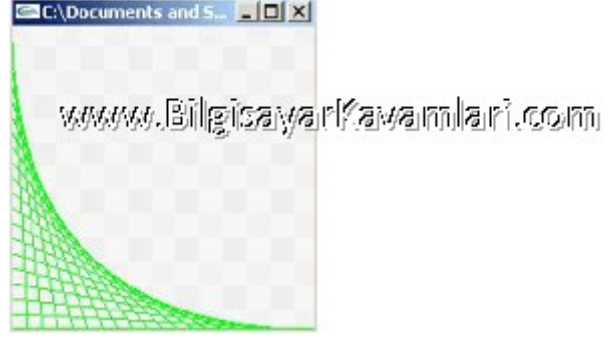
$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t)t^2 P_2 + t^3 P_3$$

Yukarıdaki denklemden de anlaşılacağı üzere t ve $(1-t)$ değişkenlerinin 4 nokta üzerindeki dağılımları birer katsayı olarak gelmektedir. 4 nokta ile çizilebilen bezier eğrilerine aşağıda örnekler verilmiştir:



Şekil-4.17: Eğri

Bezier eğrilerinin bir avantajı da bu eğrilerin adım adım çizilebilmesidir. Yani eğrinin oluşturulması sırasında kesikli matematik (discrete math) kullanılabilir. Örneğin 2. Derecen olan bezier eğrisi aşağıdaki şekilde çizilebilir:



Şekil-4.18: Çıktı

Yukarıda sadece doğrular çizilerek bir eğri oluşturulmuştur. Buradaki 3 nokta çizim alanı olan karenin 3 köşe noktasıdır ve bu noktalar kullanılarak doğru kaydırılmış ve şekilde eğri ortaya çıkmıştır.

Yukarıdaki formül binom dağılımı kullanılarak ilerletilebilir ve istenilen dereceden eğri çizimi elde edilebilir.

SORU-6: Rendering (Görselleştirme) hakkında bilgi veriniz.

1. Rendering nedir?

Bilgisayar programları yardımıyla bir modelden resim yaratma sürecine “rendering” (görselleştirme) adı verilir. Model, 3 boyutlu nesnelerin katı bir biçimde tanımlı bir dil veya veri yapısının tanımıdır. Model, geometri, bakış açısı, (texture) doku özelliği, ışıklandırma ve gölgelendirme bilgileri içerebilir. Resim ise dijital resim veya taramalı/ızgara grafik resimdir.

Rendering aynı zamanda videonun en son çıktısını oluşturmak için video düzenleme dosyasında yapılan etkileri hesaplama işlemi sürecine verilen addır.

3 boyutlu bilgisayar grafiğinin en büyük alt başlıklarından biri olan rendering, modellere ve animasyonlara nihai görünüşlerini vermeden önceki son adımdır. 1970’ten beri bilgisayar grafiği alanında yaşanan ilerlemelerden ötürü başlı başına bir konu olmuştur.

Rendering, farklı dağılımdaki özellikler ve tekniklerle mimaride, video oyunlarında, simülatörlerde, film ve TV özel efektlerinde kullanılmaktadır. Bazısı daha büyük modelleme ve animasyon paketleriyle bütünleşik, bazısı tek başına çalışan, kimisi de açık kaynak projeler olmak üzere ürün bazında pek çok değişik rendering mevcuttur. Esasen rendering, basit fizik, görsel algılama, matematik ve program geliştirme gibi farklı disiplinlerin karışımına dayanan dikkatlice oluşturulmuş bir programdır.

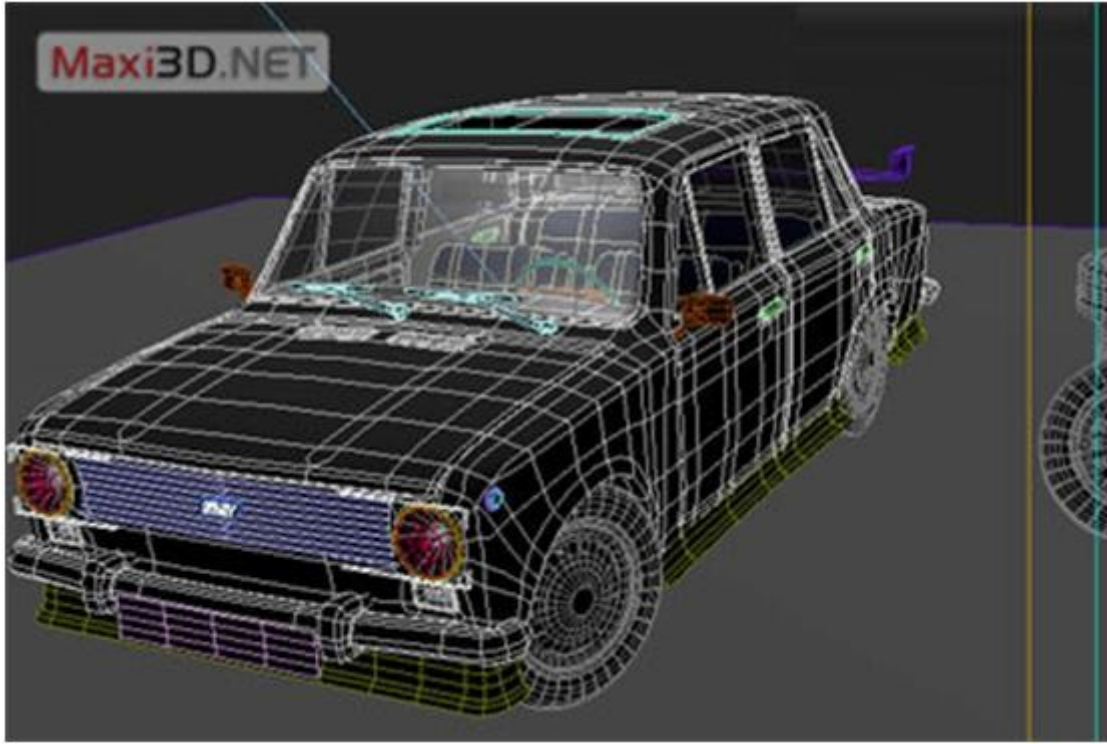


Şekil 1. Rendering

1. Kullanımı

Nesnelerin ön- resim hali tamamlandığında (genelde wireframe olarak çizilirler), Bitmap dokusu veya süreçsel dokular, ışıklar, şişkinlik haritalaması ve göreceli konumlandırma rendering ile nesnelere eklenir. Sonuç, müşterinin gördüğü tamamlanmış resimdir.

Aşağıdaki resimlerde rendering uygulanmamış ve uygulanmış araçlar bulunmaktadır.



Şekil 2 . Rendering uygulanmamış araç



Şekil 3. Rendering uygulanmış araç

1. Özellikleri

Rendering uygulanmış bir resim, görülebilen birkaç özelliğe göre incelenebilir. Rendering alanındaki araştırmalar ve geliştirmeler aşağıda verilen özellikleri iyileştirmek ve daha etkili

hale getirmek amacıyla yapılmaktadır. Bunların bazıları doğrudan belirli algoritma ve tekniklerle ilişkiliyken, öte yandan diğerleri birlikte oluşturulur.

- *Gölgeleme*: Işıklandırmayla yüzeyin rengi ve parlaklığının nasıl değiştiğini irdeler.
- *Doku haritalama*: Yüzeylere ayrıntı uygulanmasıdır.
- *Şişkinlik haritalama*: Yüzeylerdeki küçük boyuttaki şişkinlikleri benzetimdir.
- *Sis*: Işığın havadan veya net olmayan bir atmosferden geçerken nasıl donuklaştığını irdeler.
- *Gölgeler*: Işığı engellemenin etkileridir.
- *Yumuşak gölgeler*: Kısmen engellenmiş ışık kaynaklarından ötürü değişken karanlıktır.
- *Yansıma*: Ayna benzeri veya oldukça cilalı yansımadır.
- *Saydamlık*: Işığın katı cisimleri geçerkenki keskin aktarımıdır.
- *Yarı saydamlık*: Yüksek oranda dağılmış ışığın katı cisimleri geçerkenki aktarımıdır.
- *Kırılma*: Saydamlıktan ötürü ışığın eğilmesidir.
- *Kırınım*: Bir cismi geçerken ışığın eğilmesi, yayılması ve karışmasıdır.
- *Kısmi aydınlanma*: Yüzeylerin ışık kaynağından doğrudan aydınlanmak yerine, diğer yüzeylerden yansıyan ışıklar ile aydınlanmasıdır.
- *Caustics*: Yunanca yanık anlamına gelen “Caustics”, kısmi aydınlanmanın bir çeşididir. Parlak bir nesneden yansıyan ışık veya saydam bir nesnenin içinden geçen ışığın odaklanmasıyla, başka bir nesne üzerinde parlaklıkların oluşmasıdır.
- *Alan derinliği*: Odadaki bir nesnenin önü veya arkasında çok yakında bulunulması durumunda, cisimlerin çok bulanık gözükmektedir.
- *Hareket bulanıklığı*: Nesnelerin yüksek hızda hareketi veya kameranın hızı yüzünden cisimlerin bulanık gözükmektedir.
- *Gerçekçi olmayan fotoğraf rendering’i*: Bir tablo veya çizilmiş gibi gözükmesi amacıyla, sahnelere artistik stilde rendering uygulanmasıdır.

1. Teknikler

Pek çok rendering tekniği araştırılmasının ardından, kullanılan yazılımlar son resmi elde etmekte farklı tekniklere başvurmaktadır. Sahnedeki bütün ışınları takip etmenin uygulanabilirliği mümkün olmamakla beraber çok fazla zaman alır. Resim oluşturacak büyüklükteki bir sahne örneği bile, eğer akıllıca sınırlandırılmamışsa ışınların takibi yine uzunca bir zaman alır.

Bütün ışınları takip etmenin zorluğundan ötürü, birbirinden serbest ışık ulaşımında daha etkili 4 serbest teknik oluşmuştur: ızgaralama (hat tarama rendering’i), ışın kalıplaması, ışınsallık ve ışın takibi. Iızgaralama, sahnedeki nesneleri gelişmiş görsel efektler kullanmadan geometrik olarak resim düzlemine yansıtır. Işın kalıplaması tekniği sahneyi belirli bir noktadan görülyormuş gibi düşünür ve gözlemlenen resmi sadece geometri ve çok basit yansıma yoğunluğu kanunlarına göre hesaplar, bazen işlem hatalarını azaltmak amacıyla Monte Carlo tekniklerine de başvurabilir. (Bu yazı Tuğçe Doğan tarafından yazılmış olup bilgisayarkavramlari.com sitesinde yayınlanmaktadır, lütfen alıntı yapılması durumunda gerekli atıfta bulununuz). Işınsallık, yüzeyden yansıyan ışınların dağılışını simüle etmek için sonlu eleman matematiğini kullanmaktadır. Işın takibi, ışın kalıplamasına benzer özellikler içerir; ancak daha gelişmiş görsel simülasyon uygular. Işın takibi tekniğinde çoğunlukla Monte Carlo teknikleri daha yavaş fakat daha gerçekçi sonuçlar elde etmek için kullanılır.

Çoğu gelişmiş yazılım, yukarıda belirtilen tekniklerden iki ya da daha fazlasını harmanlayarak kullanıcılarına cüzi fiyatta yeterince iyi görüntüler sağlar.

1. Hat tarama ve ızgaralama

Bir resmin yüksek seviye temsili, piksellerin farklı alanlarındaki elemanları zorunlu olarak içerir. Bu elemanlara ilkel adı verilir. Örnek olarak, taslak çiziminde çizgi parçaları ve eğriler ilkel iken, 3 boyutlu sıvalamada uzaydaki üçgenler ve çokgenler ilkel olabilir.

Eğer rendering için piksel- piksel yaklaşımı(resim mertebesi) belli bir görev için çok yavaş veya uygulanamaz ise, ilkel- ilkel yaklaşımı(nesne mertebesi) daha yararlı olabilir. Bu noktada, her bir ilkel için döngüye girilir. Bu ilkelerin hangi pikselleri etkilediği belirlenir ve bu pikseller değiştirilir. Bu işleme “ızgaralama” adı verilir ve günümüzdeki tüm grafik kartlarının kullandığı rendering yöntemidir.

Izgaralama (ilkel- ilkel yaklaşım), büyük oranda piksel- piksel rendering’ten hızlıdır. İlk etapta, resmin büyük alanları hiçbir ilkel içermeyebilir; ızgaralama bu alanları yok sayacaktır, fakat piksel- piksel rendering bu noktaları geçmek durumundadır. İkinci etapta, ızgaralama önbellek tutarlılığını geliştirebilir ve resimdeki komşu olmaya meyilli, tek bir ilkel barındıran piksellerden faydalanarak gereksiz işi azaltabilir. Bu nedenlerden ötürü ızgaralama, etkileşimli rendering gerektiği durumlarda genellikle seçilen yaklaşımdır. Bununla beraber, piksel- piksel yaklaşımı çoğunlukla yüksek kalitede resimler üretir ve bu yaklaşımın çok yönlülüğü, ızgaralama alanındaki varsayımları içermediğinden ötürü daha fazladır.

Izgaralamanın eski sürümünün en karakteristik özelliği bütün yüzeye tek bir renk ile rendering uygulanmasıdır. Başka bir seçenek olarak, ızgaralama daha karmaşık bir yöntem izlenerek yapılabilir: İlk olarak yüzeyin köşelerine rendering uygulayıp daha sonra bu köşelerin birleştikleri noktaların renklerinin karıştırılarak piksellerin ızgaralanmasıdır. Izgaralamanın bu sürümü, eski yöntemin yerini grafiklerin karmaşık dokumalar olmadan akışını sağlamasından ötürü almıştır. Izgaralamanın bu daha yeni yöntemi grafik kartının gölgeleme fonksiyonlarını daha fazla kullanmak ile beraber, daha fazla performans sağlamaktadır. Çünkü bellekteki daha basit dokumalar daha az yer kaplamaktadır. Kimi zaman tasarımcılar bazı yüzeylerde eski ızgaralama yöntemini tercih ederken, yeni yöntemi açığa bağli olarak kullanarak hızı arttırıp genel etkiyi kismadan aynı sonucu alabilmektedirler.

1. Işın kalıplaması

Işın kalıplaması, ayrıntının çok önemli olmadığı veya işlemsel aşamada daha iyi performans elde etmek adına ayrıntıların sahtesini yapmanın daha etkili olduğu, gerçek zamanlı simülasyonlar olan 3 boyutlu bilgisayar oyunları ve çizgi filmlerde kullanılmaktadır. Bu, genel olarak çok sayıda film karesinin animasyon olarak gösterilmesi gerektiği durumlardır. Meydana gelen yüzeyler, karakteristik bir düz yüzey görünüşe ve sahnedeki nesnelerin tümü mat bir bitime sahiptir.

Modellenen geometri piksel- piksel, hat- hat bakış noktasından dışarı doğru, sanki kalıplar bu bakış noktasından ışıyormuşçasına ayrıştırılır. Bir nesne ile kesişme yaşandığında, o noktadaki renk değeri çeşitli teknikler ile değerlendirilir. En basit halinde, kesişim noktasındaki nesnenin renk değeri o pikselin değeri olur. Renk, dokuma haritasından da belirlenebilir. Daha incelikli diğer yöntem ise renk değerinin bir aydınlanma faktörü ile ayarlanmasıdır; ancak bu yöntemde, simule edilmiş ışık kaynağı ile ilişkisi hesaplanmaz. İşlem hatalarını azaltmak amacıyla farklı yönlerdeki çeşitli ışınları ortalaması alınır.

Görsel özelliklerin kaba simülasyonlarına ek olarak başvurulabilir: nesneden çıkan ışının bakış açısına doğru basit bir hesaplama yapılır. Diğer bir hesaplama ışık kaynağından çıkan ışınların açısını bulmak için yapılır. Buna ek olarak, belirtilen yoğunluktaki ışık kaynaklarının

da yardımıyla pikselin değeri hesaplanır. Başka bir simülasyon, ışınsallık algoritması kullanılarak çizilmiş aydınlanmayı kullanır.

1. Işınsallık

Doğrudan aydınlanmış yüzeylerin dolaylı ışık kaynağı gibi kullanılarak diğer yüzeylerin aydınlatılmasının simülasyonun denendiğı tekniğı “ışınsallık” denir. Bu sayede daha gerçekçi gölgeleme sağlanır ve daha iyi bir kapalı alan ambiyansı yakalanır. Bunun tipik örneğı gölgelerin oda köşelerinin kucaklama şeklidir.

Simülasyonun görsel ilkesi, verilmiş bir yüzeydeki belirli bir noktadan yayılmış ışınların geniş spektrumlu yönlerde yansması ve bunun çevresindeki alanı aydınlatmasına dayanır.

Simülasyon tekniğı zorluk bakımından farklılık gösterebilir. Çoğu rendering kabaca bir ışınsallık tahminine sahiptir ve bunu, bütün sahneyi ambiyans denilen faktör ile hafifçe aydınlatarak yapar. Bununla birlikte, gelişmiş bir ışınsallık tahmini ile yüksek kalitede bir ışın takip algoritması beraber kullanıldığında resimlerin, özellikle de kapalı alan sahnelerinde, ikna edici bir gerçeklik sergilediğı görülür.

Gelişmiş ışınsallık simülasyonlarında, yinelemeli sonlu eleman algoritmaları ışığı modeldeki yüzeyler arasında belirli bir yineleme limitine ulaşana kadar ileri- geri yansıtır. Bu sayede, bir yüzeyin renklendirilmesi komşusunun da renklendirilmesini etkiler. Tüm modelin aydınlanma sonuçları (bazen boş alanları de dahil olmak üzere) depolanır ve bunlar, ışın kalıplaması ve ışın takibi modelleri hesaplamalarında ek girdiler olarak kullanılır.

Hızlı ışınsallık hesaplamalarının standartlaştırılmasından önce, kimi grafik sanatçıları yanlış ışınsallık adı verilen teknik kullanmışlardır. Bu teknikte, dokuma haritasının ilgili köşelerinin, birleşme yerlerinin ve girintilerinin alanlarını karartılır ve bunlar kendini aydınlatma ve dağılım haritalaması yardımıyla hat tarama sıvalaması için uygulanır. Şu anda bile, gelişmiş ışınsallık hesaplamaları duvardan, zeminden ve tavandan yansıyan ışığın oluşturduğu odanın ambiyansını, bu karmaşık nesnelerin yarattığı ışınsallığın katkısını incelemeyen hesaplar veya karmaşık nesneler, ışınsallık hesabı sırasında benzer boyutta ve dokuda daha basit nesneler ile yer değiştirilir.

Eğer sahnedeki nesnelerin ışınsallığında ufak bir değişiklik olursa, aynı ışınsallık verisi pek çok film karesi için tekrardan kullanılabilir ve bu sayede ışınsallık, ışın kalıplamasındaki düz yüzeyliğı geliştirirken zaman başına sıvalanan genel film karesini ciddi bir şekilde etkilemez. Bu sebepten, ışınsallık öncü gerçek zamanlı sıvalama tekniğı olmuştur ve son zamanlarda çekilen uzun metrajlı 3 boyutlu animasyonların oluşturulmasında kullanılmıştır.

2.4. Işın takibi

Işın takibi tekniğinde amaç, zerrecik olarak yorumlanan ışığın doğal akışını simule etmektir. Işın takibi tekniklerine, rendering denklemini tahmin etmek amacıyla Monte Carlo teknikleri uygulanır. En çok kullanılan ışın takip tekniklerinden bazıları: Patika takibi, Çift Yönlü Patika Takibi veya Metropolis Işık Ulaşımıdır. Bunlara ek olarak melez yöntemler ve Whitted Tarzı Işın Takibi gibi yarı gerçekçi teknikler de mevcuttur. Yöntemlerin çoğu ışığı doğrusal olarak yayılmasına izin vermekle beraber, uygulamaların göreceli uzay-zaman etkilerinin simülasyonu üzerinde etkileri mevcuttur.

Işın takibi ile oluşturulacak bir ürünün son hali için, her bir piksel için çeşitli ışınların çekimi yapılır. Optiğin bilinen yasalarından “geliş açısı ile yansıma açısı birbirine eşit olmalıdır” ile kırılma ve yüzey pürüzlülüğüyle ilgilenen daha ileri seviye optik yasalarını kullanarak, bu ışınlar ilk nesne ile kesişim yerine ardışık yansımalara kadar takip edilir.

Işın bir ışık kaynağı ile karşılaşana dek veya daha muhtemeli, belirli bir seviyedeki yansıma sayısı değerlendirildiğinde, o noktadaki yüzey aydınlanması yukarıda anlatılan teknikler ile yapılır. Daha sonra, çeşitli yansımalar arasındaki yolda meydana gelen değişiklikler, bakılan noktada bir değer tahmin etmek amacıyla değerlendirilir. Bu işlem her örnek ve her piksel için tekrarlanır.

Dağılımsal ışın takibinde, her bir kesişim noktasında ışınlardan çok sayıda üretilir. Öte yandan patika takibinde ise, her bir kesişim noktasında Monte Carlo deneylerinin istatistiksel doğasından faydalanarak sadece bir tane ışın ateşlenir.

Temelindeki işleyiş yüzünden, ışın takibi gerçek zamanlı uygulamalarda kullanılabilirliği çok yavaş olduğundan tercih edilmezdi. Hatta yakın geçmişe kadar herhangi kalitedeki bir kısa film için bile çok yavaştı. Buna rağmen, ardışık özel efektlerde ve reklamların yüksek kalite gerektiren kısa bölümlerinde ışın kalıbının kullanımına rastlanabilir.

Öte yandan, ayrıntının yüksek olmadığı veya ışın takibinin özelliklerine bağımlı olmadığı iş bölümlerinin hesaplamalarını düşürmek için yapılan iyileştirme çalışmaları, yakın zamanda ışın takibi tekniğinin daha geniş kullanımına yönelik sonuçlar doğurdu. Şu anda, prototip(ilk örnek) aşamasında bile olsa, ışın takibini hızlandıran donanım ekipmanları ve gerçek zamanlı ışın takibi donanımı ve yazılımı kullanan oyun demoları piyasada mevcuttur.

1. *Optimizasyon*
2. *Sahne Geliştirme sırasında sanatçının kullandığı optimizasyonlar*

Yüksek sayıda hesaplamalardan ötürü devam etmekte olan bir işe, genellikle sadece o süre içinde yapılmış iş kadar ayrıntılı rendering işlemi uygulanır. Bu sebeple, modellemenin ilk safhalarında hedeflenen çıktının ışınsallık içeren ışın takibi olduğu durumlarda bile, tel kafes (wireframe) ve ışın kalıplaması kullanılabilir. Sahnenin sadece yüksek ayrıntılı parçalarının renderinglenmesi ve geliştirilen o sahnedeki önemli olmayan nesnelerin yok sayılması oldukça yaygındır.

1. *Gerçek zamanlı rendering için genel optimizasyonlar*

Gerçek zaman rendering’i için, bir veya daha fazla yaygın yakınlaştırmayı sadeleştirmek uygundur ve bunlar, en iyi sonucu almak için önceden karar verilmiş parametrelere ayarlanmış ilgili sahnedeki hassas parametrelere ayarlanır.

1. *Örnekleme ve filtreleme*

Hangi yöntemi seçerse seçsin tüm rendering sistemlerinin uğraşmak zorunda olduğu örnekleme sorunu vardır. Temel olarak, rendering süreci resim uzayından resimlere, sonlu sayıda piksel kullanarak sürekli bir fonksiyon resmetmeye çalışır. Örnekleme teoreminin sonucu olarak, tarama frekansı nokta oranının iki katı olmalıdır ki bu da resim çözünürlüğüyle orantılıdır. Daha basit terimlerle, bu durum bir resmin bir piksel küçük ayrıntıları sergileyemeyeceğini ifade etmektedir.

Eğer orta halli bir rendering algoritması kullanılırsa, resim fonksiyonundaki yüksek frekanslar resmin son halinde kenarlarda bulunacak çirkin çentikli görüntülere(aliasing) neden olacaktır. Çentikli görüntüler kendilerini nesnelerin köşelerinde testeremsi bir yapı olarak gösterecektir. Çentikli görüntüyü ortadan kaldırmak adına, tüm rendering algoritmalarının yüksek frekansları yok etmek için resim fonksiyonunu filtrelemesi gerekmektedir. Bu işleme “çentik yok etme” (antialiasing) adı verilir.

1. *Matematiksel işleyiş*
2. *Rendering denklemi*

Bu denklem rendering için anahtar teorik konudur. Rendering’in algılanması zor yönünün en özet ve düzgün halidir. Diğer tüm eksiksiz algoritmalar bu denklemin belirli biçimlerinin çözümleri olarak görülebilir.

$$L_o(x, \vec{w}) = L_e(x, \vec{w}) + \int_{\Omega} f_r(x, \vec{w}', \vec{w}) L_i(x, \vec{w}') (\vec{w}' \cdot \vec{n}) d\vec{w}'$$

Yukarıdaki denklemin anlamı, belirli bir konum ve yönde giden ışık (L_o), emilen ışık (L_e) ve yansıyan ışığın toplamına eşittir. Yansıyan ışık ise bütün yönlerden gelen ışığın toplamının yüzey yansıması ve ışığın gelme açısıyla çarpımına eşittir. Bir etkileşim noktasıyla dışarı doğru olan ışığı içeri doğru olan ışık ile bağdaştırması nedeniyle, bu denklem sahnedeki tüm ışığın ulaşımını sağlamaktadır.

1. *Çift yönlü yansıma dağılım fonksiyonu*

Çift Yönlü Yansıma Dağılım Fonksiyonu, ışığın yüzey ile temasını basit bir model ile ifade eder:

$$f_r(x, \vec{w}', \vec{w}) = \frac{dL_r(x, \vec{w})}{L_i(x, \vec{w}') (\vec{w}' \cdot \vec{n}) d\vec{w}'}$$

Işık teması sıklıkla daha basit modeller ile hesaplanır: dağılım yansıması ve ayna benzeri yansıma. Modellerin ikisi de Çift Yönlü Yansıma Dağılım Fonksiyonu olabilir.

1. *Görsellik*
2. *Geometrik görüş*

Rendering, geometrik görüş olarak da bilinen ışık fiziğinin zerre yönüyle özel bir şekilde ilgilidir. En basit seviyede, ışığı etrafta yansıyan zerreler olarak ele almak uygun bir basitleştirmedir. Işığın dalga yönü pek çok sahnede yok sayılacak boyuttadır ve bunların simülasyonu çok zordur. Işığın dalgasal yönünün görülebildiği kayda değer fenomenler, CD ve DVD’lerin üzerindeki renkler ile LCD’lerdeki kutuplaşmadır. Her iki etki, eğer gerekirse, yansıma modelinin görünüm kökenli değişimiyle yapılabilir.

1. *Görsel algılama*

Her ne kadar az ilgi çekse de, insanoğlunun görsel algılamasının anlaşılması rendering için önemlidir. Bunun asıl nedeni resim gösterimlerinin ve insan algısının sınırlarının olmasıdır. Bir rendering işlemi sınırsıza yakın çeşitlilikte parlaklık ve renk simülasyonu yapabilir ancak

bunlar günümüz sinema ekranları ve bilgisayar monitörleriyle görüntülemeyebilir, bu sebeple kimi fazlalıklar atılır veya sıkıştırılır. İnsan algısının da belirli sınırları vardır ve gerçeklik yaratmak için geniş çeşitlilikte resimler verilmesine gerek yoktur. Bu durum resimlerin ekrana sabitlenmesi sorununu çözmekte yardımcı olur ve buna ek olarak, ne tür kısa yolların sıvalama simülasyonunda kullanılabileceğine dair yol gösterir. Bu konuya tonlama haritası adı verilir.

Rendering sırasında matematiksel anlamda lineer cebir, calculus, sayısal matematik, sinyal işleme ve Monte Carlo teknikleri kullanılır.

1. Donanımsal yön

Gerçek zamanlı 3 boyutlu rendering sahip olduğu yüksek işleme ve bellek erişimi oranında otürü fazla güç tüketir. İş yükündeki ve algısal toleranstaki değişkenlik yüzünden, “güç farkındalığı” bu güç tüketimini kayda değer bir şekilde optimize edebilir; bunun sonucu olarak geleceğin güç bakımından sınırlı cihazlarından PDA’lar, tabletler ve telefonlara geçişe kolaylık sağlayabilir.

Güç farkındalığı üstünde yapılan araştırmalardan birinde, bu konuyla ilgili önerilen düşük güç tüketimi “Yaklaşık Görsel Rendering (YGR)” e dayanmaktadır. YGR sistemi çeşitli algoritmaları ve önceden belirtilmiş kesin parametrelere dayalı işlemsel mekanizmalardaki değişimleri desteklemektedir. Görsel imgelerin sinyal ve gürültü modelleri ile insanın görsel algılaması hakkında önceden sahip olunan bilgi sayesinde en düşük güç tüketiminde elde edilecek kaliteyi sağlayan konfigürasyon seçilir.

YGR sistemi kullanılarak elde edilen güç tasarrufu, 3 boyutlu rendering sisteminin en çok güç tükettiği iki evre olan gölgeleme ve doku haritalamasında test edilmiştir. Bu maksatla CORDIC (iterative COordinate Rotation DIgital Compter) algoritması ile algılanan imgenin uzaysal bağıntısını dinamik olarak zenginleştiren bir program tercih edilmiştir. Tercih edilen algoritmanın ve programın güç tasarrufu tahminini yapmak için, donanım sentezi ve tanınmış 3 boyutlu değerlendirme deneyi ve Quake 2 bilgisayar oyunu kullanılmıştır. Gölgelemede %75.1, doku haritalamasında %73.8 ve CORDIC algoritmasının kullanıldığı rendering’te %72 oranında güç tasarrufu sağlanmıştır.

1. Gölgeleme

3 boyutlu modellerde veya illüstrasyonlarda derinliği karanlık seviyesinde değişiklik yaparak elde etmeye “gölgeleme” adı verilir.

1. Çizimde gölgeleme

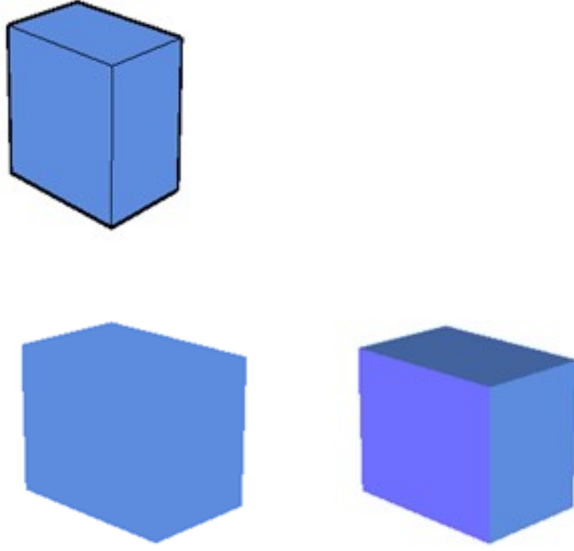
Kağıt üzerinde çizim yaparken karanlık seviyesini resmedebilmek için koyu renkli bölgelerin daha yoğun, açık renkli bölgelerin az yoğun medya uygulanmasına gölgeleme denir. Aralarında, değişen yakınlıktaki dik çizgilerin ızgarasal model içerisinde çizildiği çapraz bölmelemenin (cross hatching) bulunduğu pek çok gölgeleme yöntemi vardır. Çizgiler birbirine yaklaştıkça daha koyu bir bölge oluşur. Benzer mantıkla, çizgiler birbirinden uzaklaştıkça daha açık renkli bölge oluşur.

1. Bilgisayar grafiğinde gölgeleme

Bilgisayar grafiğinde gölgeleme, bir rengi ışığa olan açısı ve ışığa olan uzaklığına bağlı olarak gerçekçilik katmak için değiştirme sürecine “gölgeleme” denir.

1. Işık kaynağına olan açı

Gölgeleme, 3 boyutlu bir modelde yüzeylerin renklerini yüzeyin ışık kaynağı ile olan açısına göre değiştirilir.



Şekil 4. Gölgeleme

Yukarıda görülen 3 resimden en soldakinin bütün yüzeyleri rendering’e maruz bırakılmıştır; ancak hepsi aynı renktedir. Resmin görülmesini daha kolay kılmak adına kenar çizgilerine de rendering uygulanmıştır. Ortadaki resimde kenar çizgileri hariç rendering işlemi uygulanmıştır ve bu durumda bir yüzeyin nerede bitip diğerinin nerede başladığını söylemek zordur. En sağdaki resimde rendering işlemi gölgeleme ile beraber yapılmıştır ve resim daha gerçekçi olup yüzeylerin her biri kolayca görülebilmektedir.

1. Işık kaynakları

Çevre ışığı: Sahnedeki bütün nesneler eşit olarak ve gölgeleme eklenmeden aydınlatılır.

Yönsel ışık: Bu tarz ışık kaynakları, verilen yönden bütün nesneleri eşit olarak aydınlatır. Bu, sahneden sonsuz uzaklıkta ve büyüklükte bir ışık bölgesinin olması gibidir. Gölgeleme vardır; ancak mesafe azalması durumu yoktur.

Nokta ışık: Nokta ışığı tek bir noktadan başlar ve dışarı doğru tüm yönlerde yayılır.

Sahne ışığı (spotlight): Nokta ışık gibi tek bir noktadan kaynağını alır ve dışarı doğru konik bir yönde yayılır.

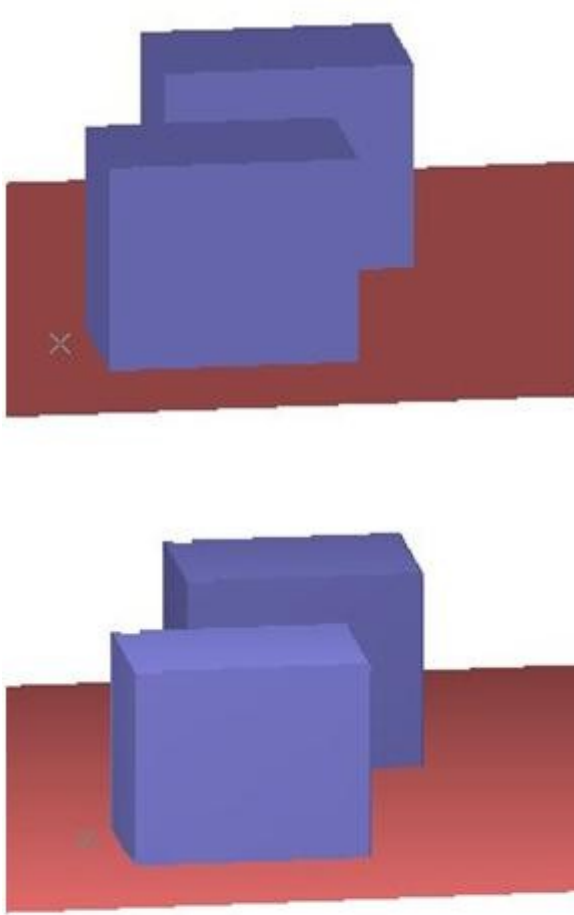
Alan ışığı: Tek bir düzlemden kaynağını alan “alan ışığı” verilen yöndeki tüm nesneleri, o düzlemden başlayarak aydınlatır.

Hacim ışığı: Çevrelenmiş bir uzay içindeki nesneleri aydınlatır.

Gölgelemenin ara değeri, yukarıda verilmiş ışık kaynaklarından gelen ışınların sahnedeki nesnelerle yaptığı açıya göre hesaplanır. Bu ışık kaynakları sıklıkla bir arada kullanılır. Bu noktada, rendering'i yapacak program bu ışınların nasıl bir arada kullanılacağına ara değerini hesaplar ve buna göre ekrana yansıtılmak üzere 2 boyutlu bir resim oluşturur.

1. Mesafe azalması

Teorik olarak paralel konumlu iki yüzey uzak bir ışık kaynağı ile eşit miktarda aydınlanır. Bir yüzey daha uzakta olsa bile göz bunun çoğunu aynı uzayda görür, bu sebepten eşit aydınlanma oluşur.



Şekil 5. Gölgelemede mesafe

Soldaki resimdeki iki kutunun ön yüzelerinin renkleri tamamıyla aynıdır. İki yüzün birleştiği yerde hafif farklılık gözlemleniyor olsa da; dikey kenarın iki yüzeyin birleşim yerinden alçakta olmasından ötürü bir göz yanılmasıdır.

Sağdaki resimde öndeki kutunun yüzeyi daha açık, arkadaki kutunun yüzeyi daha koyudur. Aynı zamanda zemin uzağa doğru gittikçe açık renkten koyuya kaymaktadır. Bu mesafe

azalması etkisi sayesinde ek ışık kaynağı eklemek zorunda kalmadan daha gerçekçi resimler oluşturulabilir.

Mesafe azalması değişik yöntemlerle hesaplanabilir:

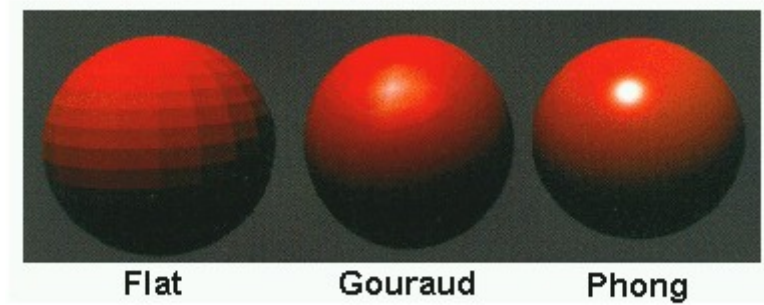
Doğrusal- Işık kaynağından x birim uzaklıkta olan bir nokta için gelen ışık x birim daha az parlaktır.

İkinci dereceden denklem- Bu yöntem gerçek hayatta ışığın işleyişiyle aşağı yukarı aynıdır. Biri diğerinden ışık kaynağına iki kat daha uzakta olan bir nokta, dört kat daha az aydınlanmış olur.

N' inci kuvvet- Işık kaynağından x birim uzaklıkta olan bir nokta, ışığın $1/(x^n)$ kadarını alır.

Bunlar dışında herhangi bir matematiksel fonksiyon da kullanılabilir.

1. Düz yüzey ve pürüzsüz yüzey gölgelemesi



Şekil 6. Gölgeleme çeşitleri

1. Düz yüzey gölgelemesi

Düz yüzey gölgeleme 3 boyutlu bilgisayar grafiğinde bir aydınlatma tekniğidir. Bu teknikte nesnenin her bir çokgeninin, çokgenin normali, konumu ve ışık kaynağının yoğunluğu dikkate alınarak gölgeler. Daha gelişmiş gölgeleme tekniklerinin çok pahalı olduğu durumlarda hızlı rendering için kullanılırdı. Ancak 20. yüzyılın sonlarına doğru piyasadaki çok pahalı olmayan grafik kartlarının zaten hızlı olan pürüzsüz gölgelemeyi sağlamaya başlamasıyla birlikte, görüntü kalitesi olarak başarılı olmayan düz yüzey gölgelemenin hız sebepleriyle tercih edilmesini gereksiz kıldı.

Düz yüzey gölgelemenin en büyük dezavantajı az köşegenli modellere fasetali bir görüntü vermesidir. Bazen bu durum kutusal modellerde avantajlı olabilmektedir. Sanatçılar kimi zaman yarattıkları katı modelin köşegenlerine bakmak için düz yüzey gölgelemeyi kullanır. Daha gerçekçi ve gelişmiş aydınlatma ve gölgeleme teknikleri Gouraud gölgelemesi ve Phong gölgelemesidir.

1. Gouraud gölgelemesi

Gouraud gölgelemesi: 3 boyutlu grafikte, 1970 yılında Henri Gouraud tarafından geliştirilen ve gölgeli bir yüzeyi her bir üçgenin köşelerindeki renk ve aydınlamayı dikkate alarak hesaplayan bir tekniktir. Bir sonraki başlıkta bahsedilecek olan Phong gölgelemesinden daha hızlı olan Gouraud gölgelemesi, en basit rendering tekniğidir. Üçgenlerin noktalarındaki yüzey normalleri, üçgenin yüzeyi boyunca ortalaması alınmış RGB değerlerini oluşturmak için kullanılır.

1. Phong gölgelemesi

Phong gölgelemesi: 1970'lerin ortalarında Phong Bui- Thong tarafından geliştirilmiş, gölgeli bir yüzeyi her bir pikseldeki renk ve aydınlanmayı kullanarak hesaplayan bir tekniktir. Gouraud gölgelemesinden daha gerçekçi sonuçlar verir; ancak daha fazla işleme ihtiyaç duyar. Üçgenlerin noktalarının yüzey normalleri her bir pikselin yüzey normalini hesaplamak için kullanılır ve sonuç olarak her bir piksel için daha hassas RGB değerleri yaratılır.



Şekil 7. Nükleer reaktörün Phong gölgelendirmesi ile 86 adet ışık kaynağı kullanılarak görselleştirilmesi

Sonuç

Görüntünün ham halinin bilgisayar tarafından işlenmesi işlemi olan rendering; mimarlık, bilgisayar oyunları, simülörler, sinema ve televizyon gibi digital ortamlara oldukça hakimdir. Tasarımın en çarpıcı noktalarından biri olan rendering yoğun ve uzmanlık gerektiren işlemleri ile çarpıcı sonuçlar elde edilmesini sağlamaktadır. Bu projede, rendering basamakları ayrıntılarıyla anlatılmış olup ek olarak gölgelendirme üzerinde durulmuştur.

Günümüzün gelişen teknolojisi ve bu teknolojiye ulaşımın kolaylaşması ile birlikte geçmişteki ilkel grafik kullanımının yerini çok daha gelişmiş görseller almıştır.

SORU-7: Splines (Şeritler) hakkında bilgi veriniz.

1.Spline

Çizim terminolojisinde Spline, noktalar kümesi tarafından belirlenen düzgün bir eğri oluşturmak için kullanılan esnek bir şerittir. Şerit boyunca dağıtılmış birçok küçük ağırlıklar, çizim masasında sabit konumda duran bir eğri oluşmasını sağlarlar. Spline'lar genelde endüstriyel tasarımda kullanılır. (otomobil, gemi, uçak, uzay aracı gövdeleri, ürün tasarımı gibi.)

1.1 Spline Eğrileri

Spline eğrileri de temelde bu yolla oluşturulur. Matematiksel olarak bu tür bir eğri, ilk ve ikinci türevleri, farklı eğri bölümlerinde sürekli olan kübik polinom fonksiyonu ile tanımlanır. Bilgisayar grafiklerinde Spline eğrileri, parçaların sınırlarında süreklilik şartlarını sağlayan polinom kısımları tarafından biçimlendirilen bileşik eğrilerdir.

1.2 Kontrol Noktaları

Spline eğrileri, kontrol noktası adı verilen koordinat konum kümeleri verilerek tanımlanırlar. Kontrol noktaları, eğrinin genel şeklini belirler ve sürekli parametrik polinom fonksiyon parçalarında kullanılır. Bunun iki yolu vardır.

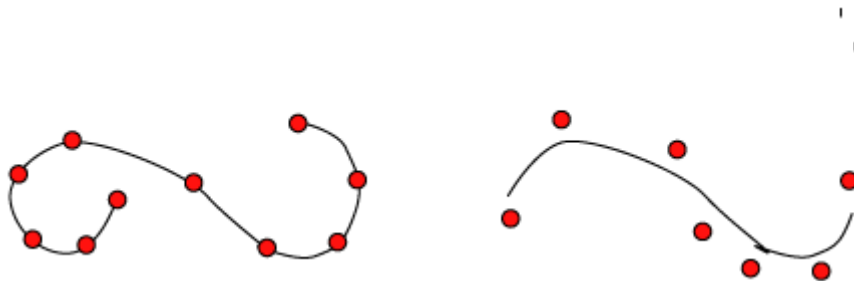
1.3 Spline Gösterimlerinin Sınıflandırılması

1. Aradeğerleme (Interpolation) Spline'ları

Eğri tüm kontrol noktalarından geçer.

1. Yaklaşım (Approximation) Spline'ları

Eğrinin tüm kontrol noktalarından geçme şartı yoktur



1.4 Spline Eğri ve Yüzeylerinin Önemi

- Spline eğrileri, tanımlanıp değiştirilebilirler.
- Yeni kontrol noktaları eklenebilir.
- Kontrol noktaları silinebilir.

- Etkileşimli grafik ve CAD paketlerinde kullanıcılar, tasarımları üzerinde fare ile değişiklikler yaparak istenen modelleri elde edebilirler.

2. Bezier Splines

- Fransız otomobil üreticisi Renault bünyesinde P. Bezier 1962 yılında geliştirmiştir.
- Bezier eğrileri, eğriler üzerinde daha çok manipülasyon yapabilmek amacıyla geliştirilmiştir.
- Karmaşık araba yüzey tasarımlarını gerçekleştirmek için kullanılan bir araçtır.
- Bezier eğrisi bir kontrol poligonu tarafından kontrol edilmektedir
- İkinci derece (quadratic) ve kübik (cubic) Bézier eğrileri en çok kullanılanlardır.
- Daha yüksek dereceli eğrileri kullanması daha masraflıdır
- Karmaşık şekiller gerektiği zaman düşük dereceli Bézier eğrileri birbirine yamandır.
- Ayrıca, Bézier eğrilerinin programlanması da oldukça kolaydır. Bu tür özelliklerinden dolayı, Bézier eğrileri bilgisayar grafiklerinde yaygın olarak kullanılır.

$n+1$ adet kontrol noktası tarafından kontrol edilen $B(t)$ eğrisi aşağıdaki genel Beziereğri formu ile verilmiştir;

$$B(t) = \sum_{i=0}^n b_{i,n}(t) P_i, \quad t \in [0, 1]$$

Burada

$$b_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad i = 0, \dots, n$$

n . derece Bernstein polinomları olarak adlandırılır.

Not: $(t^0 = 1 \text{ and } (1-t)^0 = 1)$

$$\binom{n}{i}$$

: binom katsayıdır (2 terimli)

Polinomun derecesi, kullanılan kontrol noktalarının sayısından bir azdır.

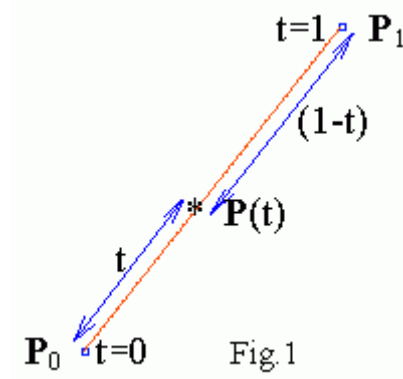
$$B(t) = \sum_{i=0}^n b_{i,n}(t) P_i, \quad t \in [0, 1]$$

- Kontrol noktaları Béziereğrisinin kontrol noktalarıdır.
- P_0 ile başlayan ve P_n ile son bulan noktaların çizgilerle birleşmesinden şekillenen poligon Bézier poligonu ya da kontrol poligonu olarak adlandırılır. Bézier poligonunun dış bükey yapısı Béziereğrisini içerir.
- Eğri P_0 'da başlar ve P_n 'de biter. Buna uç nokta interpolasyon özelliği (endpoint interpolation property) denir.

- Eğri sadece ve sadece kontrol noktaları aynı doğru çizgi üzerinde ise düz çizgi olabilir.
- Eğrinin başlangıcı (sonu) Bézierpoligonunun ilk (son) bölümüne teğettir.
- Eğri her hangi bir noktadan 2 eğri halinde parçalanabilir. Bu parçalanma ile herhangi birçok alt eğri elde edilebilir. Bunların her biri Bézier eğrisidir.

2.1 Linear Bezier Spline

Doğrusal bir Béziereğrisinin $t=[0,1]$ arası formu doğrusal interpolasyon formuna dönüşür.



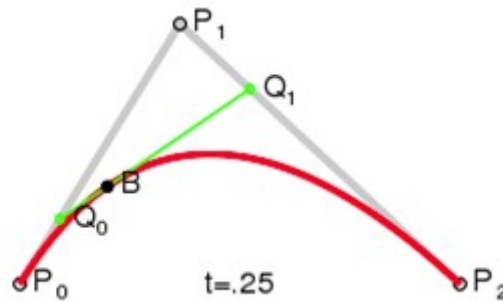
$$P(t) = (1-t)P_0 + tP_1, \quad 0 \leq t \leq 1$$

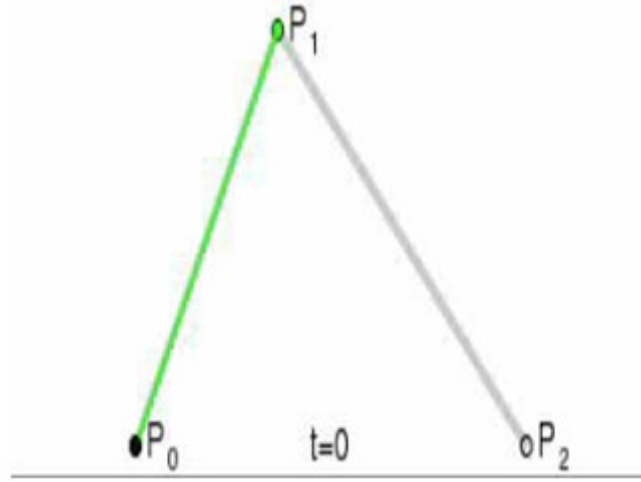
2.2 Quadratic Bezier Spline

Eğri P_0, P_1 & P_2 noktalarından geçer.

$$B(t) = (1-t)^2P_0 + 2t(1-t)P_1 + t^2P_2, \quad t \in [0, 1]$$

- 2. derece Bézier eğrileri için $t=[0-1]$ arasında Q_0 ve Q_1 adında ara noktalar oluşturulabilir.
- Q_0 noktası P_0 ile P_1 arasında değişir ve doğrusal bir Bézier eğrisi oluşturur.
- Q_1 noktası P_1 ile P_2 arasında değişir ve doğrusal bir Bézier eğrisi oluşturur.
- $B(t)$ noktası Q_0 'dan Q_1 'e değişir ve 2. derece bir Bézier eğrisi tanımlar.





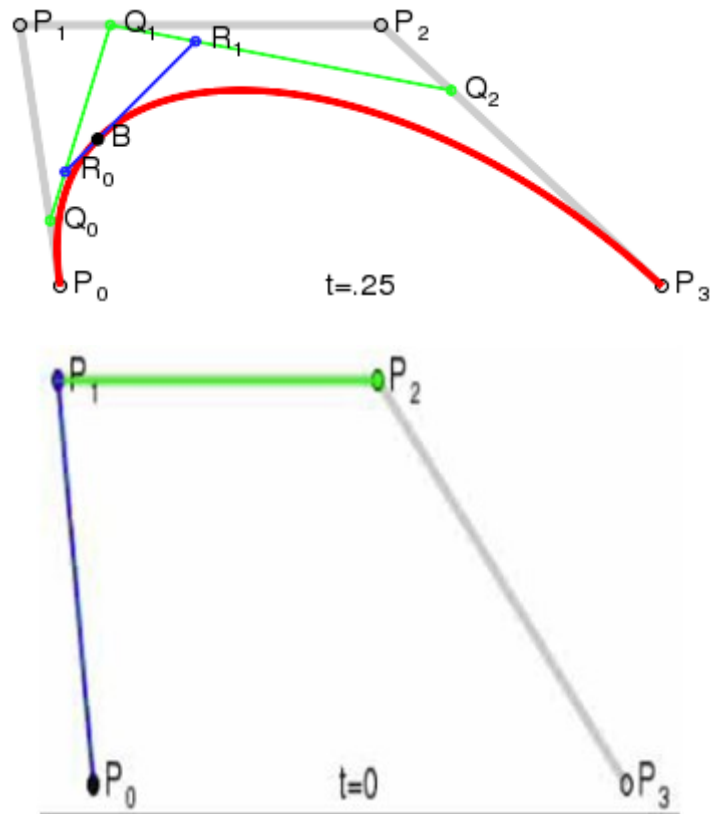
2.3 Cubic Bezier Spline

Eğri P0, P1, P2 & P3 noktalarından geçer.

$$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3t(1-t)^2\mathbf{P}_1 + 3t^2(1-t)\mathbf{P}_2 + t^3\mathbf{P}_3, t \in [0, 1]$$

Kubikform için;

- Doğrusal Béziereğrisi tanımlayan Q0, Q1 & Q2 ara noktaları ve
- 2. derece Béziereğrisi tanımlayan R0 & R1 noktaları oluşturulur.



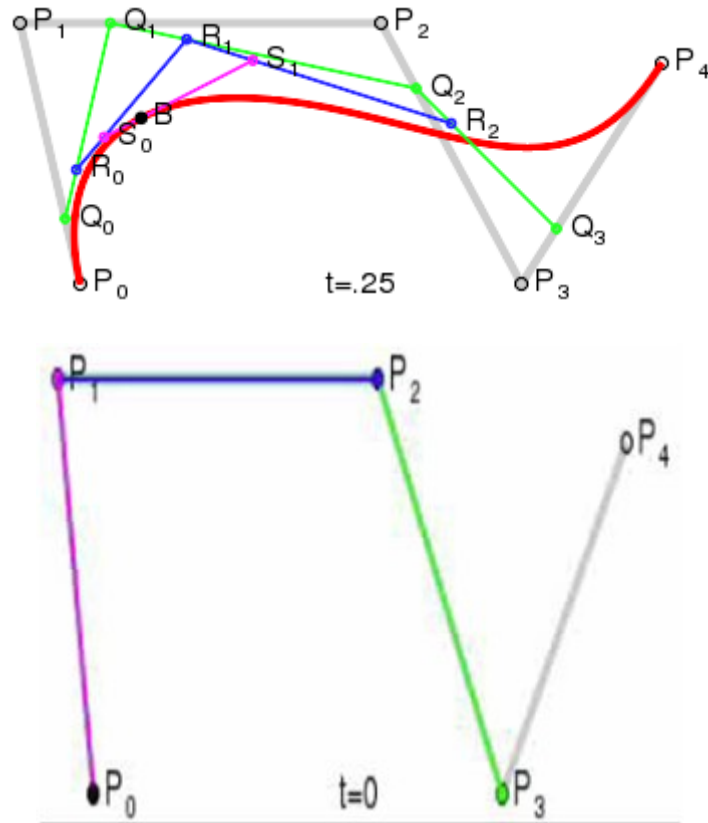
2.4 Yüksek Derece Eğriler

Dördüncü Derece Bezier Eğrisi

Eğri P0,P1, P2, P3 & P4 noktalarından geçer.

Dördüncü derece form için;

- Doğrusal Béziereğrisi tanımlayan Q0, Q1, Q2&Q3 ara noktaları ve
- 2. derece Bézier eğrisi tanımlayan R0,R1&R2 noktaları ve
- Kubik Bézier eğrisi oluşturan S0&S1 noktaları oluşturulur.



2.5 Polinom Formu

- Bazen Bézier eğrisini Bernstein polinomlarının toplamı olarak ifade etmek yerine polinom olarak ifade etmek daha kolaydır.
- Binom teoreminin eğrinin tanımına uygulanması ve tekrar düzenlenmesi aşağıdaki formu verir:

$$\mathbf{B}(t) = \sum_{j=0}^n t^j \mathbf{C}_{j\bar{n}}$$

$$\mathbf{C}_j = \frac{n!}{(n-j)!} \sum_{i=0}^j \frac{(-1)^{i+j} \mathbf{P}_i}{i!(j-i)!} = \prod_{m=0}^{j-1} (n-m) \sum_{i=0}^j \frac{(-1)^{i+j} \mathbf{P}_i}{i!(j-i)!}.$$

- Eğer CjB(t)'nin bir çok kez hesaplanmasından önce hesaplanırsa bu yöntem pratik ve verimli olur.
- Fakat yüksek dereceli eğriler sayısal kararsızlık konusunda sorunlu olabileceği için dikkatli olmak gerekir. (Bu durumda de Casteljau'nun algoritması kullanılmalıdır).

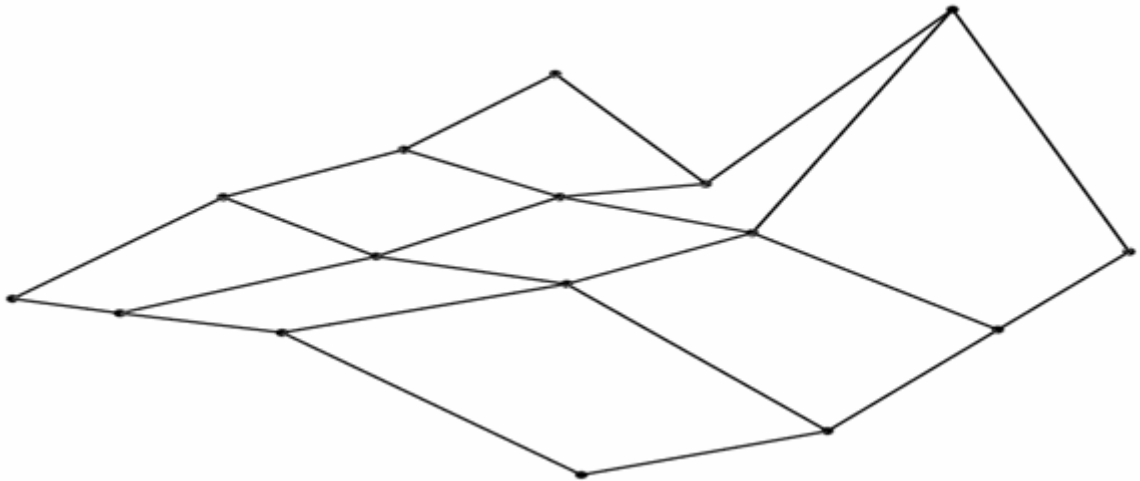
2.6 Bezier Eğrileri Dezavantajları

- Eğrinin kontrol noktalarından geçmemesi kontrolü ve hassasiyeti zora sokar.
- Bezier eğrisinde lokal değişiklik yapılamaz.

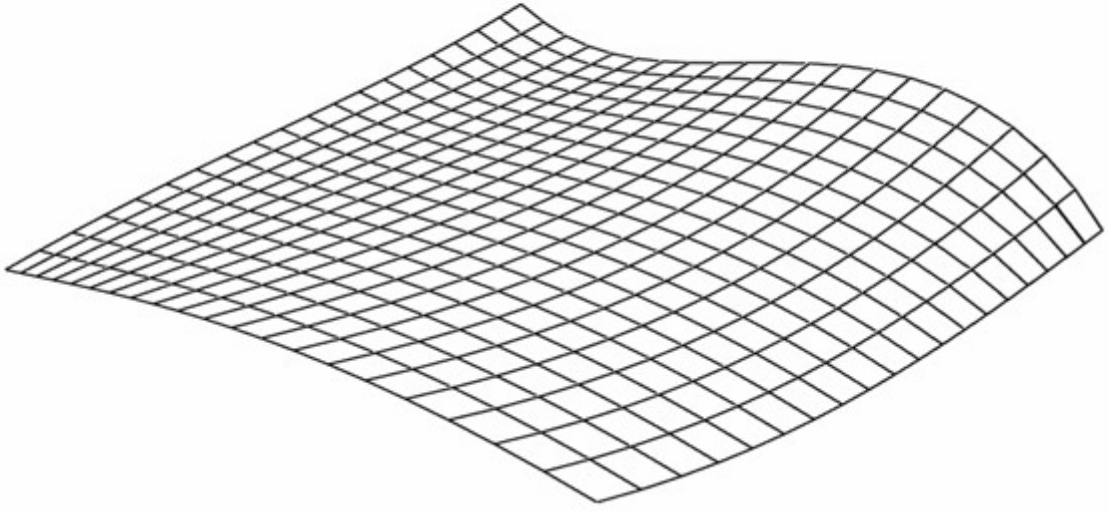
2.7 Bezier Yüzeyleri

Üç boyutlu Beziér yüzeylerinin (diğer adıyla Beziér yamalarının) tanımlanması için Beziér eğrilerinin gösteriminin genelleştirilmesi mümkündür. Matematiksel olarak, üç boyutlu yüzeyler iki eğrinin kartezyen çarpımından elde edilebilir. Dolayısıyla, $(m+1)(n+1)$ kontrol noktası tarafından belirtilen Beziér yüzeyinin gösterimi şöyle olacaktır:

$$P(u, v) = \sum_{i=0}^m \sum_{k=0}^n p_{i,k} BEZ_{i,m}(v) BEZ_{k,n}(u)$$



16 kontrol noktası ile tanımlanmış bir Beziér yüzeyi



16 kontrol noktasıyla tanımlanmış üç boyutlu Beziér yüzeyi

3. B-Spline Eğrileri

- B-Splineeğrileri Bezier eğrilerinin genelleştirilmiş halidir
- En önemli avantajı lokal kontrole izin vermesidir
- Bir temel teoriye göre verilen derece, düzgünlük ve alan bölümünde bütün eğri fonksiyonları B-Spline eğrilerin aynı derece, düzgünlük ve alan bölümünde doğrusal kombinasyonu olarak ifade edilebilir.
- B-Splineadı Isaac Jacob Schoenberg tarafından konulmuştur ve “basis spline”nın kısaltılmış formudur.
- B-Splineeğriler de Boor algoritması ile sayısal denge içinde değerlendirilebilirler.
- B-Spline eğrileri yüksek dereceliBézier,eğrilerinde karşılaşılan problemleri polinomun derecesini yükseltmeden önleyebilir.
- Elde edilen eğrinin derecesi kullanılan kontrol noktası sayısından bağımsızdır.

Verilen $n+1$ sayıda kontrol noktası;Pitarafından tanımlanan B-Splineeğri tanımı:

$$P(u) = \sum_{i=0}^n P_i N_{i,k}(u), \quad 0 \leq u \leq u_{\max}$$

Burada $N_{i,k}(u)$ Cox ve de Boor tarafından 1972’de önerilen B-Splinefonksiyonlarıdır.

k parametresi B-Splineeğrinin derecesini $(k-1)$ kontrol eder ve genellikle kontrol noktası sayısından bağımsızdır.

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

$$N_{i,1} = \begin{cases} 1, & u_i \leq u \leq u_{i+1} \\ 0, & \end{cases}$$

U_i parametrik düğüm (knots) ya da düğüm değerleri olarak adlandırılır. Açık bir eğri için şu formda verilmiştir:

$$u_j = \begin{cases} 0, & j < k \\ j - k + 1, & k \leq j \leq n \\ n - k + 2, & j > n \end{cases}$$

$$0 \leq j \leq n + k$$

$$0 \leq u \leq n - k + 2$$

(k-1) derecesinde eğriyi (n+1) sayıda kontrol noktası ile tanımlamak için (n+k+1) sayıda düğüm (knots) gereklidir.

$$n - k + 2 > 0$$

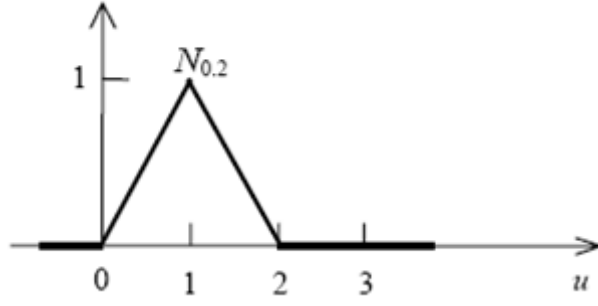
Bu eşitlik gösterirki;

Doğrusal eğri için en az 2

2.derece eğri için en az 3

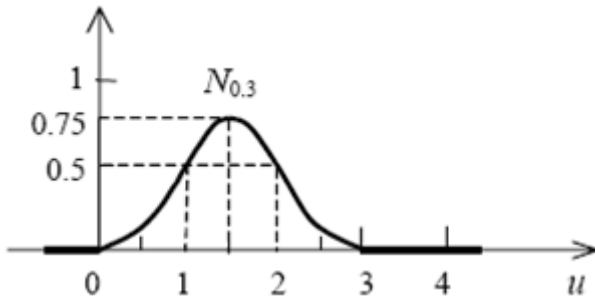
Kubik eğri için ise en az 4 kontrol noktası gerekir.

Aşağıdaki şekiller B-Splinefonksiyonlarının şekillerini göstermektedir:



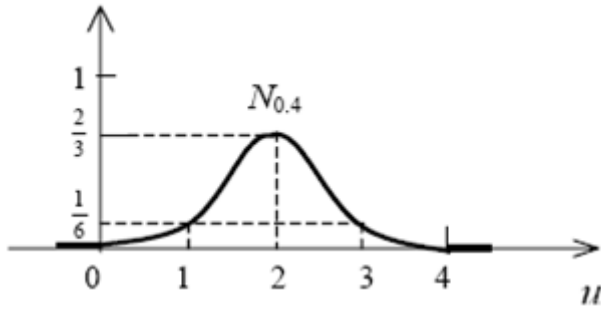
Doğrusal fonksiyon

$k=2$



2.derecefonksiyon

$k=3$

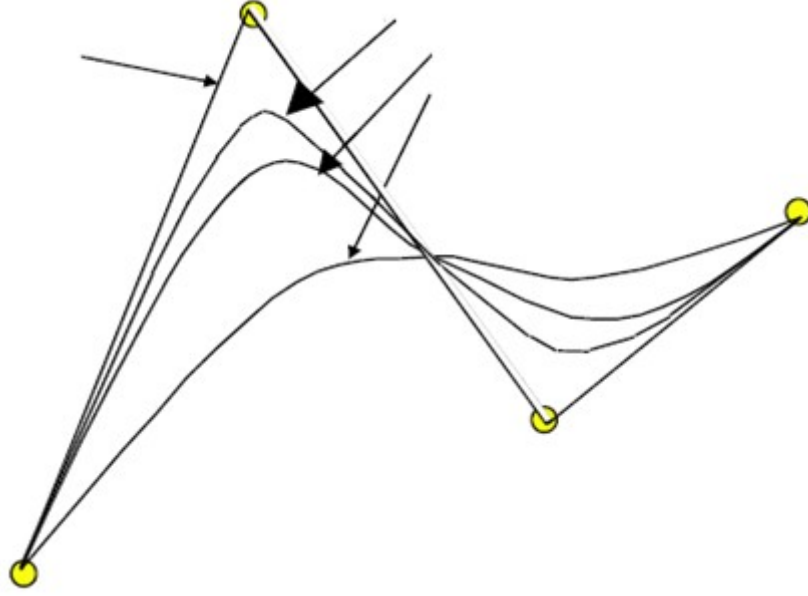


kubikfonksiyon

$k=4$

3.1 B-Spline Eğrileri Özellikleri

Kontrol noktası sayısı polinomunderecesinden bağımsızdır.



Sarı noktalar: verteks

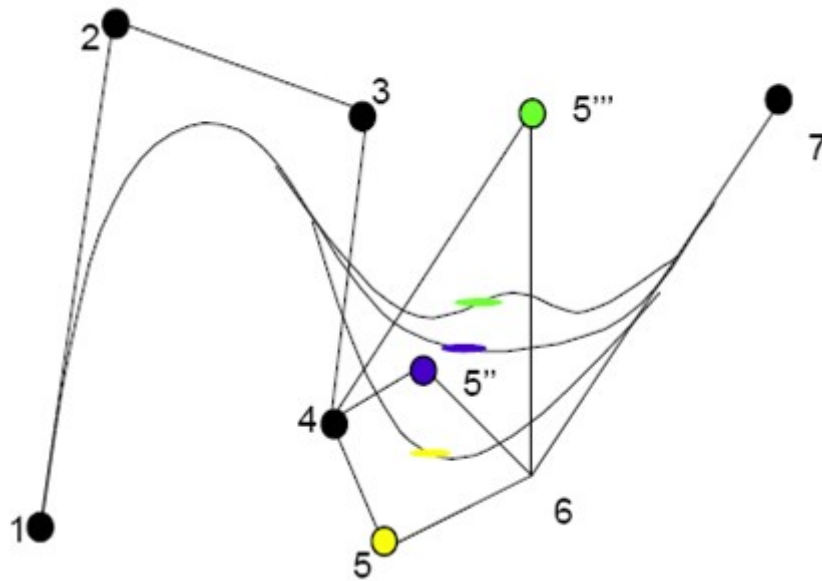
Sol üst ok: Doğrusal $k=2$

Sağ üst ok: 2.derece B-spline $k=3$

Sağ orta ok: Kubik B-spline $k=4$

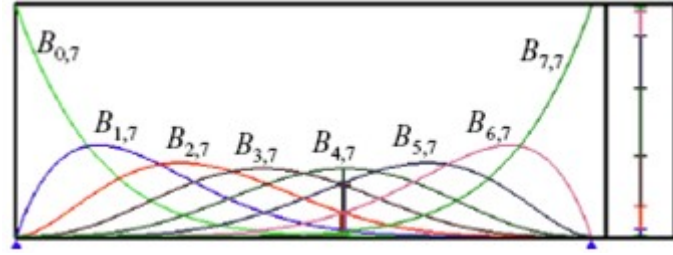
Sağ alt ok: 4.derece B-spline $k=5$

- B-Splinederecesi yükseldikçe yakın kontrol noktasına etkisi azalır.
- B-Splinedaha iyi lokal kontrole izin verir.
- Eğrinin şekli kontrol noktalarını hareket ettirerek ayarlanabilir.
- Local kontrol: Bir kontrol noktası sadece k bölümüne etki eder.

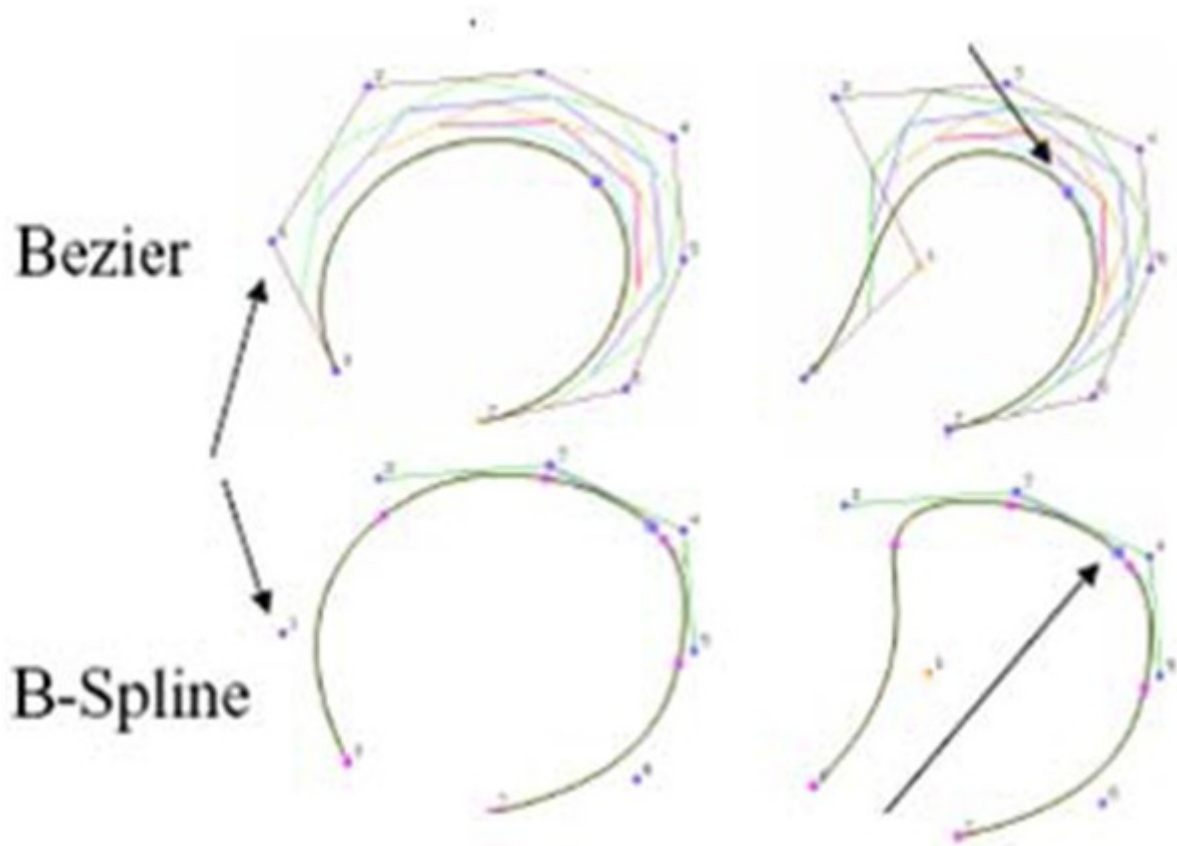
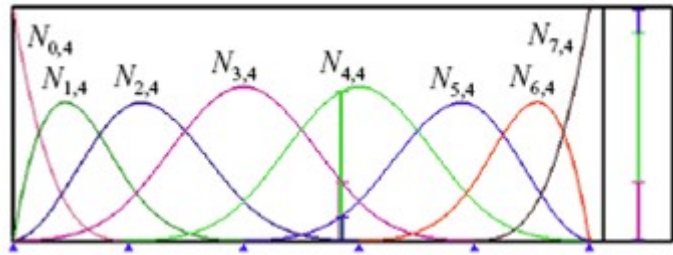


3.2 Bezier/B-Spline Eğrileri

BezierHarmanlama Fonksiyonları (BezierBlendingFunctions); $B_{i,n}$



B-splineHarmanlama Fonksiyonları (B-splineBlendingFunctions); $N_{i,k}$



Sağ alt okla gösterilen nokta hariç diğer 3 belirtilen nokta hareket ettiriliyor.

3.3 Uniform B-Spline Eğrileri

B-splineuniformolduğu zaman n dereceli B-splinefonksiyonlarıbirbirinin kaydırılmış kopyasıdırlar. Eğri boyunca, düğümler (knots) eşit mesafelerde yerleşmiştir.

4. Rasyonel Eğriler ve NURBS(Rational Curves and NURBS)

- Rasyonel polinomlar hem analitik hem de polinom eğrileri düzenli (uniform) tanımlayabilirler.
- Eğriler kontrol noktalarının ağırlıkları (weighting) değiştirilerek düzenlenebilirler.
- Genel kullanılan şekli NURBS (Non-Uniform Rational B-spline)'dür.

4.1 Rasyonel Bezier Eğrileri

Rasyonel Bézier formunda, rastgele şekiller elde edilebilmesi için kontrol noktalarına ayarlanabilir ağırlıklar eklenmiştir.

n + 1 kontrol noktası; P_i için rasyonel Bézier eğrisi:

$$\mathbf{B}(t) = \frac{\sum_{i=0}^n b_{i,n}(t) \mathbf{P}_i w_i}{\sum_{i=0}^n b_{i,n}(t) w_i}$$

Pay kısmında yeralan ağırlıklı BernsteinşeklindeBéziereğrisidir ve paydada ise Bernsteinpolinomlarınınağırlıklı toplamları yer almaktadır. Daha basitçe;

$$\mathbf{B}(t) = \frac{\sum_{i=0}^n \binom{n}{i} t^i (1-t)^{n-i} \mathbf{P}_i w_i}{\sum_{i=0}^n \binom{n}{i} t^i (1-t)^{n-i} w_i}$$

formunda ifade edilebilir.

4.2 Rasyonel B-Spline Eğrileri

Bir rasyonel eğri 2 polinomun birbirine oranı olarak tanımlanmıştır. Rasyonel eğride kontrol noktaları homojen koordinatlarda tanımlanmıştır.

$$x \cdot h = \sum_{i=0}^n (h_i \cdot x_i) N_{i,k}(u)$$

$$y \cdot h = \sum_{i=0}^n (h_i \cdot y_i) N_{i,k}(u)$$

$$z \cdot h = \sum_{i=0}^n (h_i \cdot z_i) N_{i,k}(u)$$

$$h = \sum_{i=0}^n h_i N_{i,k}(u)$$

dolayısıyla rasyonel B-Spline eğrisi:

$$\mathbf{P}(u) = \sum_{i=0}^n \mathbf{P}_i R_{i,k}(u), \quad 0 \leq u \leq u_{\max}$$

formunda ifade edilebilir.

$R_{i,k}(u)$ rasyonel B-Spline taban fonksiyonlarıdır.

$$R_{i,k}(u) = \frac{h_i N_{i,k}(u)}{\sum_{i=0}^n h_i N_{i,k}(u)}$$

Yukarıdaki eşitlik gösterir ki; $R_{i,k}(u)$ rasyonel olmayan taban fonksiyonları;

$N_{i,k}(u)$ 'nın genelleştirilmiş halidir.

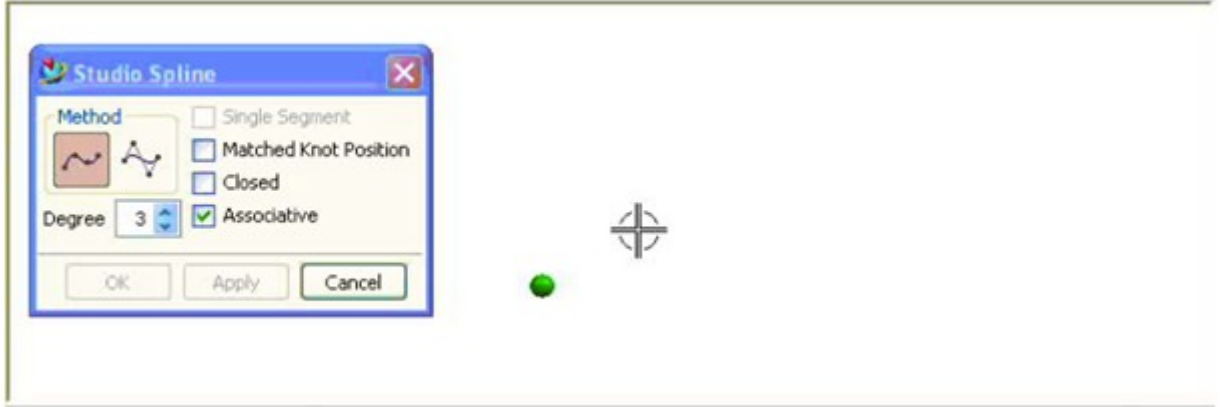
$R_{i,k}(u)$ eşitliğine $h_i=1$ yerleştirilmesi durumunda rasyonel olmayan formla

yaklaşık aynı özellikleri gösterir.

4.3 NURBS

- Düzgün olmayan, rasyonel B-Spline formülü olup, matematik modelidir ve genellikle eğriler ve yüzeyler oluşturmak ve göstermek için bilgisayar grafiğinde kullanılır.
- NURBS eğrisi , derecesi, bir set ağırlıklı kontrol noktaları ve düğüm vektörü (knot vector) ile tanımlanır.
- NURBS eğrileri ve yüzeyleri hem B-spline hem de Bézier eğrileri ve yüzeylerinin genelleştirilmiş halidir.
- En önemli farkı kontrol noktaları ağırlıklarıdır ki bu NURBS'ü rasyonel eğri (rational curve) yapar.

- NURBS eğrileri sadece bir parametrik yön içerir (genellikle s ya da u olarak adlandırılır). NURBS yüzeyleri iki parametrik yön içerir.
- NURBS eğrileri konik eğrilerin tam modellenmesini sağlar.



NURBS eğrisinin genel formu;

$$C(u) = \sum_{i=1}^k \frac{N_{i,n} w_i}{\sum_{j=1}^k N_{j,n} w_j} \mathbf{P}_{ii}$$

k: kontrol noktası (Pi) sayısı

wi: ağırlıklar (weights)

Payda normalleme faktörüdür (normalizing factor).

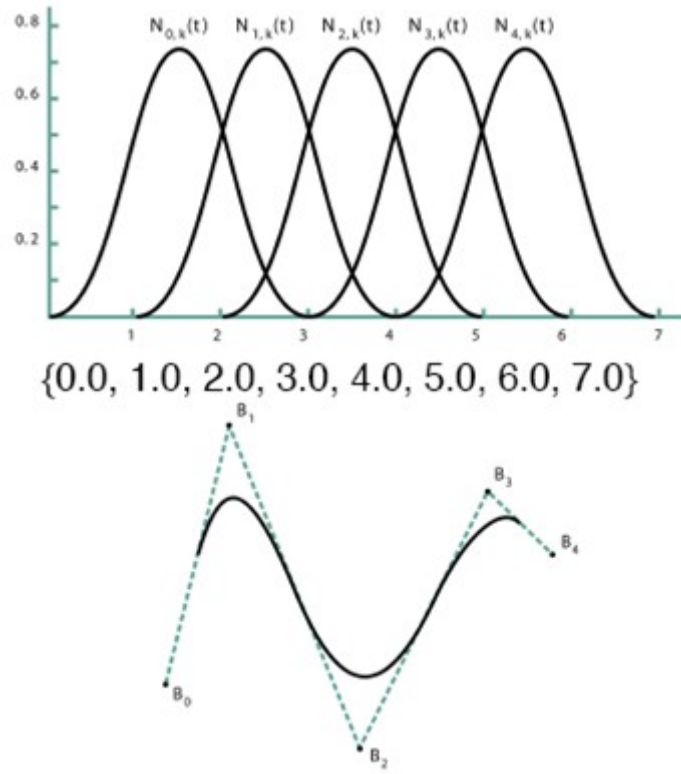
NURBS genel formu aşağıdaki gibi ifade edilebilir:

$$C(u) = \sum_{i=1}^k R_{i,n} \mathbf{P}_i$$

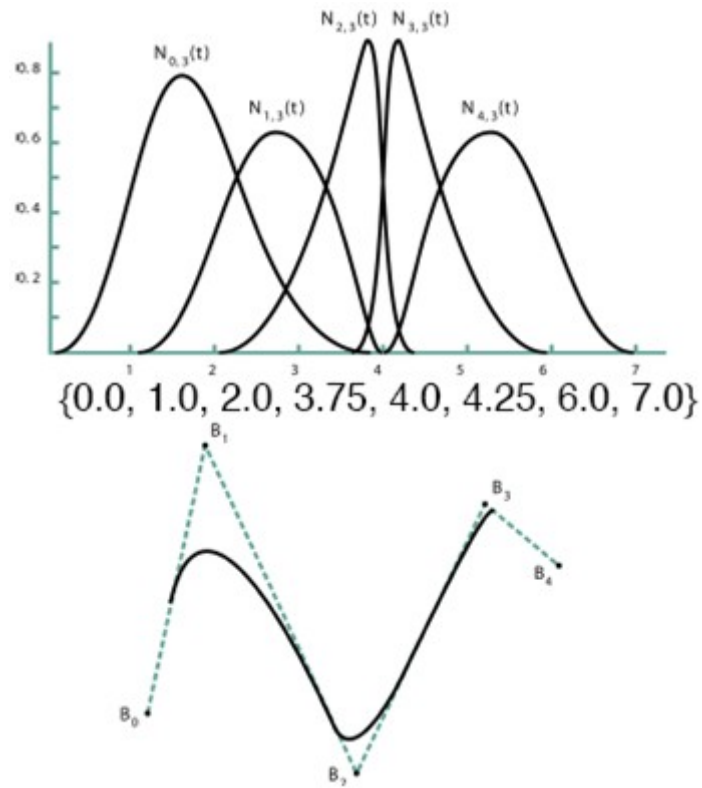
$$R_{i,n} = \frac{N_{i,n} w_i}{\sum_{j=1}^k N_{j,n} w_j}$$

Rinfonksiyonları, rasyonel taban fonksiyonları (rational basis functions) olarak adlandırılır.

4.4 NURBS Örnekleri



Uniform düğüm (knot) vektörü



Uniform olmayan (nonuniform) düğüm (knot) vektörü

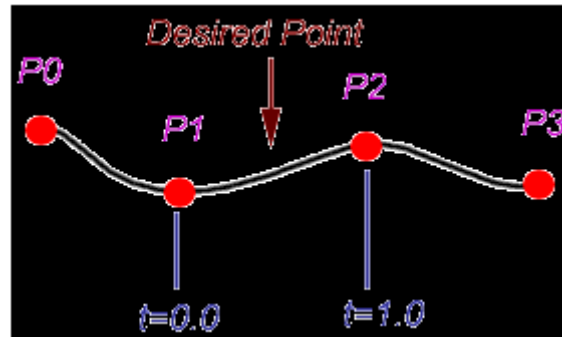
4.5 NURBS'ün Gelişimi

- Boeing: TigerSystem-979
- SDRC: Geomod-1993
- University of Utah: Alpha-1-1981
- Endüstri Standartları : IGES, PHIGS, PDES,Pro/E, vb.

4.6 NURBS Avantajları

- Rasyonel olmayan (non-rational)B-spline olduğu kadar rasyonel (rational) ve rasyonel olmayan (non-rational)Bezier eğrileri ve yüzeyleri için gerçek genelleştirme sağlar.
- Hem standart analitik şekillerin matematiksel formlarını (koniler, 2. derece eğriler, döndürme yüzeyler, vb.) hem de serbest şekilli eğriler ve serbest şekilli yüzeyleri hassasiyetle modeller.B-splineeğriler sadece konik eğrileri modeller.
- NURBS eğrisini parametrenin değişik değerlerinde hesaplayarak eğri 2 ya da 3 boyutlu kartezyen uzayda tanımlamak mümkündür.
- Aynı durum NURB yüzeyleri için de geçerlidir.
- Kontrol noktası ve ağırlık kullanımı ile birçok şeklin tasarımı esnekliğini sağlar.
- Ağırlıkları artırmanın eğriyi kontrol noktasına çekme etkisi vardır.
- Güçlü bir alet takımı vardır. (düğüm ekleme/detaylandırma/kaldırma/derece yükseltme/bölme/vb.)
- Döndürme ve öteleme gibi perspektif transformasyonlarda değişmezdir (invariant). Bu işlemleri sadece kontrol noktalarına uygulamak yeterlidir.
- Makul hızda ve dengeli hesaplanabilirlerdir.
- Şekilleri saklarken az hafızaya gereksinim gösterirler.

5. Catmull-Rom Splines



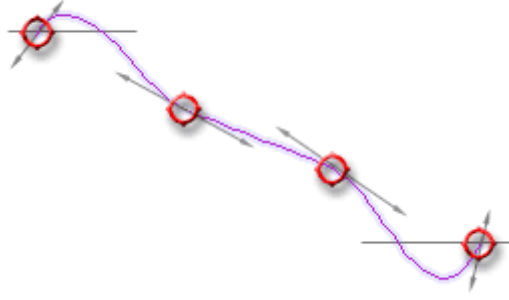
Eğri bütün kontrol noktalarından geçmelidir.Bir noktaı hesaplamak için Şekilde görüldüğü gibi o noktanın iki yanındaki noktalar gereklidir.P1,P2,P3,P4 gibi kontrol noktaları ve t değeriyle noktanın yeri şu şekilde hesaplanabilir.

$$q(t) = 0.5 * (2 * P_1 +$$

$$(-P_0 + P_2) * t +$$

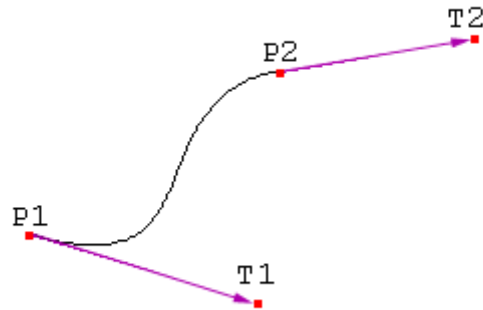
$$(2*P_0 - 5*P_1 + 4*P_2 - P_3) * t^2 +$$

$$(-P_0 + 3*P_1 - 3*P_2 + P_3) * t^3)$$



6. Hermite Splines

Oluşturmak için her kontrol noktasının tanjant vektörlerini gerektirir.



P1:Eğrinin başlangıç noktası.

P2:Eğrinin bitiş noktası

T1:Eğrinin başlangıç noktasından nasıl çıktığının tanjantı

T2:Eğrinin bitiş noktasına nasıl vattığının tanjantı

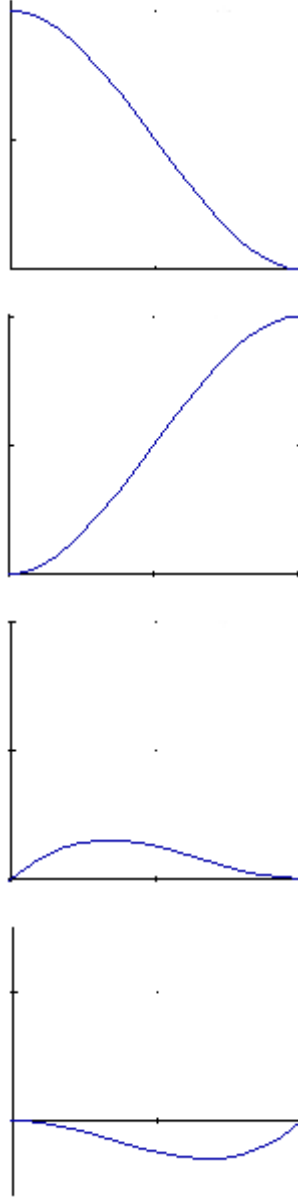
Bu 4 vektör ve Hermite Fonksiyonları bir araya getirildiğinde;

$$h1(s) = 2s^3 - 3s^2 + 1$$

$$h2(s) = -2s^3 + 3s^2$$

$$h3(s) = s^3 - 2s^2 + s$$

$$h4(s) = s^3 - s^2$$

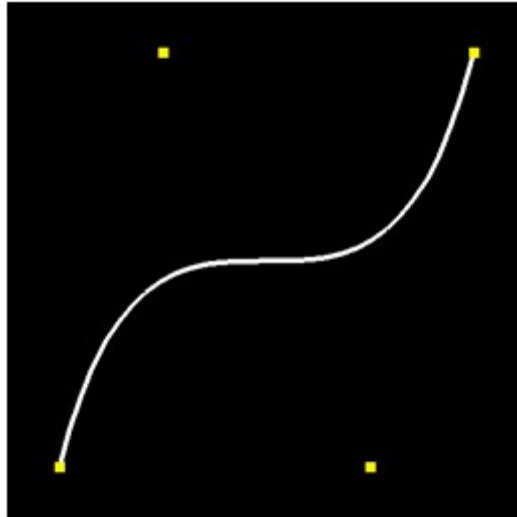


7. Beta-splines:

Otomatik süreklilik ve tanjant ile kavis uzunluğu üzerinde tam manasıyla kontrol sağlar.

8. Programlama Örnekleri

C++(Bu C++ programı ekrana bezier eğrisi çizer.4 nokta var.2'si kontrol noktası)



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

GLfloat ctrlpts[4][3] = {
    {-0.4, 0.4, 0.0},
    {-0.1, 2.0, 0.0},
    { 0.1,-2.0, 0.0},
    { 0.4, 0.4, 0.0}
};

void display()
{
    GLint k;
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 1.0);
    glBegin(GL_LINE_STRIP);
    for (k= 0; k<= 50; k++)
        glEvalCoord1f((GLfloat) (k)/50.0);
    glEnd();
    glPointSize(5.0);
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_POINTS);
    for (k= 0; k< 4; k++)
        glVertex3fv(&ctrlpts[k][0]);
    glEnd();

    glutSwapBuffers();
}

void init()
{
    glClearColor(1.0,1.0,1.0,0.0);
    glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4, *ctrlpts);
    glEnable(GL_MAP1_VERTEX_3);
}

int main(int argc, char ** argv)
{
    glutInit(&argc, argv);
```

```

glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
glutInitWindowSize(400,400);
glutInitWindowPosition(50, 50);
glutCreateWindow("Bezier Eğrisi");
init();
glutDisplayFunc(display);
glutMainLoop();
return 0;
}

```

1. C++(Mouse ile işaretlenen noktalardan geçen bezier eğrisi çizdirmek.)

```

#include "gl/glut.h"

```

```

static int Genislik = 500;
static int Yukseklik =500;

```

```

GLfloat cpts[4][3];
int ncpts = 0;

```

```

void Baslat(void)
{
glClearColor(1.0,1.0,1.0,0.0);
glEnable(GL_MAP1_VERTEX_3);
}

```

```

void Goster(void)
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0, 0.0, 0.0);
glPointSize(5.0);
}
void EgriCiz(){
glColor3f(0.0, 0.0, 0.0);
glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4, &cpts[0][0]);
glMapGrid1f(30, 0.0, 1.0);
glEvalMesh1(GL_LINE, 0, 30);
glFlush();
}

```

```

void FareDinle(GLint button,GLint action,GLint x,GLint y)
{
float wx, wy;
if(button==GLUT_LEFT_BUTTON && action==GLUT_DOWN){
wx = (2.0 * x) / (float)(Genislik - 1) - 1.0;
wy = (2.0 * (Yukseklik - 1 - y)) / (float)(Yukseklik - 1) - 1.0;
if (ncpts ==4)
return;
cpts[ncpts][0] = wx;

```

```

cpts[ncpts][1] = wy;
cpts[ncpts][2] = 0.0;
ncpts++;
glBegin(GL_POINTS);
glVertex3f(wx, wy, 0.0);
glEnd();
glFlush();
}
else
if(button==GLUT_RIGHT_BUTTON)
EgriCiz();
}

void main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(Genislik,Yukseklilik);
glutInitWindowPosition(100, 150);
glutCreateWindow("İnteraktif Eğri Çiz");
glutDisplayFunc(Goster);
glutMouseFunc(FareDinle);
Baslat();
glutMainLoop();
}

```

8.3 Java(İnteraktif 2D Bezier splines)

```

import java.awt.*;
import java.awt.event.*;
import java.util.StringTokenizer;
public class Bezier extends java.applet.Applet
    implements MouseMotionListener{
Image buffImage;      Graphics buffGraphics;
int n = 4, n1, w,h,h1,w2;
double[] Px,Py;
public void drawFun(){
    double step = 1./w2, t = step;
    double[] B = new double[n1], Bo = new double[n1], Bold = new double[n1];
    B[1] = Bo[1] = h1;
    Color[] iColor = {Color.gray, Color.red, new Color(0f,.7f,0f),
        Color.blue, Color.magenta, new Color(0f,.8f,.8f), new Color(.9f,.9f,0f) };
    for (int k = 1; k < w2; k++){
        System.arraycopy(B,1,Bold,1,n);
        System.arraycopy(Bo,1,B,1,n);
        for (int j = 1; j < n; j++) // basis functions calculation
            for (int i = j+1; i > 0; i--)
                B[i] = (1-t)*B[i] + t*B[i-1];
        for (int m = 1; m <= n; m++){
            buffGraphics.setColor(iColor[m % 7]);

```

```

        buffGraphics.drawLine(w2+k-1, h1-(int)Bold[m], w2+k, h1-(int)B[m] );}
    t += step;
}
}
public void drawSpline(){
    double step = 1./w2, t = step;
    double[] Pxi = new double[n], Pyi = new double[n];
    int X,Y, Xold = (int)Px[0], Yold = h1-(int)Py[0];
    buffGraphics.clearRect(0,0, w2, h);
    buffGraphics.setColor(Color.blue);
    for (int i = 0; i < n; i++){
        X = (int)Px[i]; Y = h1-(int)Py[i];
        buffGraphics.drawRect(X-1,Y-1, 3,3);}
    if ( n > 2 ){
        int Xo = Xold, Yo = Yold;
        for (int i = 1; i < n; i++){
            X = (int)Px[i]; Y = h1-(int)Py[i];
            buffGraphics.drawLine(Xo,Yo, X,Y);
            Xo = X; Yo = Y;}
    }
    buffGraphics.setColor(Color.red);
    for (int k = 1; k < w2; k++){
        System.arraycopy(Px,0,Pxi,0,n);
        System.arraycopy(Py,0,Pyi,0,n);
        for (int j = n-1; j > 0; j--) // points calculation
            for (int i = 0; i < j; i++){
                Pxi[i] = (1-t)*Pxi[i] + t*Pxi[i+1];
                Pyi[i] = (1-t)*Pyi[i] + t*Pyi[i+1];}
        X = (int)Pxi[0]; Y = h1-(int)Pyi[0];
        buffGraphics.drawLine(Xold,Yold, X,Y );
        Xold = X; Yold = Y;
        t += step;
    }
}
}
public void init() {
    w = Integer.parseInt(getParameter("width"));
    h = Integer.parseInt(getParameter("height")); h1 = h-1; w2 = w/2;
    String s = getParameter("N"); if (s != null) n = Integer.parseInt(s);
    n1 = n+1;
    Px = new double[n]; Py = new double[n];
    s=getParameter("pts");
    if (s != null){
        StringTokenizer st = new StringTokenizer(s);
        for (int i = 0; i < n; i++){
            Px[i] = w2*Double.valueOf(st.nextToken()).doubleValue();
            Py[i] = h1*Double.valueOf(st.nextToken()).doubleValue();}
    }
    else{
        Px[0] = .1*w2; Px[1] = .1*w2; Px[2] = .9*w2; Px[3] = .9*w2;
        Py[0] = .1*h1; Py[1] = .9*h1; Py[2] = .9*h1; Py[3] = .1*h1;}
    buffImage = createImage(w, h);
}

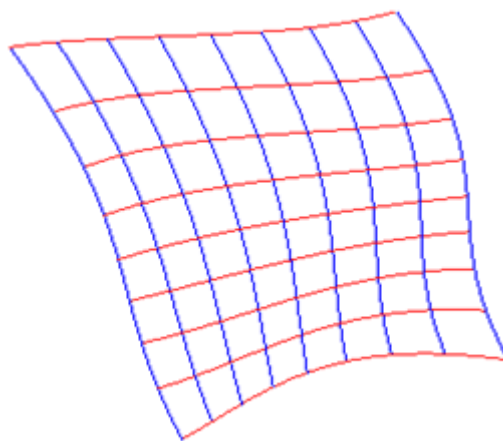
```

```

buffGraphics = buffImage.getGraphics();
setBackground(Color.white);
buffGraphics.clearRect(0,0, w, h);
addMouseListener(this);
drawFun();
drawSpline();
}
public void destroy(){ removeMouseListener(this); }
public void mouseMoved(MouseEvent e){ //1.1 event handling
public void mouseDragged(MouseEvent e) {
    int x = e.getX(); if (x < 0) x = 0; if (x > w2-3) x = w2-3;
    int y = h1 - e.getY(); if (y < 0) y = 0; if (y > h1) y = h1;
    int iMin = 0;
    double Rmin = 1e10, r2,xi,yi;
    for (int i = 0; i < n; i++){
        xi = (x - Px[i]); yi = (y - Py[i]);
        r2 = xi*xi + yi*yi;
        if ( r2 < Rmin ){ iMin = i; Rmin = r2;}}
    Px[iMin] = x; Py[iMin] = y;
    drawSpline();
    repaint();
}
public void paint(Graphics g) {
    g.drawImage(buffImage, 0, 0, this);
    // showStatus( " " + x + " " + y);
}
public void update(Graphics g){ paint(g); }
}

```

8.4 C++



```

GLfloat ctrlpoints[4][4][3] = {

```

```

{{-1.5, -1.5, 4.0}, {-0.5, -1.5, 2.0},
{0.5, -1.5, -1.0}, {1.5, -1.5, 2.0}},
{{-1.5, -0.5, 1.0}, {-0.5, -0.5, 3.0},
{0.5, -0.5, 0.0}, {1.5, -0.5, -1.0}},
{{-1.5, 0.5, 4.0}, {-0.5, 0.5, 0.0},
{0.5, 0.5, 3.0}, {1.5, 0.5, 4.0}},
{{-1.5, 1.5, -2.0}, {-0.5, 1.5, -2.0},
{0.5, 1.5, 0.0}, {1.5, 1.5, -1.0}}
};

```

```

void display(void)

```

```

{

```

```

int i, j;

```

```

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

```

```

glPushMatrix ();

```

```

glRotatef(15.0, 1.0, 1.0, 1.0);

```

```

for (j = 0; j <= maxJ; j++) {

```

```

glColor3f(1.0, 0.0, 0.0);

```

```

glBegin(GL_LINE_STRIP);

```

```

for (i = 0; i <= 30; i++)

```

```

glEvalCoord2f((GLfloat)i/30.0, (GLfloat)j/8.0);

```

```

glEnd();

```

```

glColor3f(0.0, 0.0, 1.0);

```

```

glBegin(GL_LINE_STRIP);

```

```

for (i = 0; i <= 30; i++)

```

```

glEvalCoord2f((GLfloat)j/8.0, (GLfloat)i/30.0);

```

```

glEnd();

}

glPopMatrix ();

glFlush();

}

void init(void)

{

glClearColor (0.0, 0.0, 0.0, 0.0);

glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 4,

0, 1, 12, 4, &ctrlpoints[0][0][0]);

glEnable(GL_MAP2_VERTEX_3);

glEnable(GL_DEPTH_TEST);

glShadeModel(GL_FLAT);

}

```

SORU-8: Texture Filtering (Doku Süzme) hakkında bilgi veriniz.

Bilgisayar grafiklerinde, doku süzme civardaki doku imgeciklerinin rengini kullanarak (doku kaplanmış) texture mapped imgecikler için doku rengini belirlemek için kullanılan bir yöntemdir. Kısaca, doku imgeciklerini daha küçük imgecikler haline bölerek beraber harmanlar. Doku süzme için bir başka terim ise doku yumuşatmadır. Doku süzmenin birçok yöntemi vardır. Bu yöntemlerin miktarı, hesap karmaşıklığı ve görüntü kalitesi arasında farklı koordinasyonlar yaratır.

Doku filtrelemesi bir küme ayrık örneklemdaki en yakın noktaların değerini bulma girişimi olduğu için bir ara değeri bulma(iç değerlendirme) biçimidir.

1 SÜZME İHTİYACI

Doku kaplama(texture mapping) süreci sırasında, bir “texture lookup” doku üzerindeki her imgeciğin (pixel) merkezinin nereye düştüğüne bakar. Doku kaplanmış yüzey keyfi uzaklıkta olabileceği için ve izleyen kişinin açısı ve yönelimine bağlı olduğu için genellikle bir imgecik (pixel) direkt olarak başka bir doku imgeciğine karşılık gelmeyebilir. En iyi imgecik (pixel) rengine karar verebilmek için bazı filtreleme şekilleri uygulanmalıdır. Yetersiz veya hatalı süzme, görüntüde hatalar oluşmasına yol açar. Bir imgecik ve o imgeciği ekranda temsil eden doku imgecik/imgecikleri birbirlerine farklı şekillerde karşılık gelebilirler. İzleyenin görüş açısından doku kaplanmış yüzeyin konumuna bağlı olarak her durum için farklı tür bir süzme

gerekebilir. Dünyada kare bir düzlemin üzerinde bir doku kaplanmış kare, bazı görüş açlarına bir ekran imgeciğinin boyutu bir doku imgeciğinin boyutuyla tıpatıp aynıdır. Bir yandan, texture magnification(doku büyütme) olarak bilinen bir işlem ile ekran imgeciklerinden daha büyük olan doku imgecikleri uygun bir şekilde büyütülürler. Öte yandan, her doku imgeciği bir imgecikten küçüktür ve bu yüzden bir imgecik birçok doku imgeciğini kapsar. Bu durumda, texture minification vasıtasıyla kaplanmış doku imgecikleri baz alınarak uygun bir renk seçilmelidir. Grafik Application Programming Interface(API)-Uygulama Programı Arabirimi(UPA) olan OpenGL, prgramcının minification ve magnification filtreleri için değişik seçenekler ayarlamasına imkan sağlar.

İmgeciklerin ve doku imgeciklerinin tıpatıp aynı boyutta olduğu halde bile, bir imgecik bir doku imgeciği ile ister istemez uymayabilir, hizalanamamış olabilir ve dört komşu doku imgeciğine kadar, bu imgeciklerin bazı kısımlarını kaplıyor olabilir. Bundan ötürü bazı filtreleme şekilleri hala gereklidir.

2 MIPMAPPING

Üç boyutlu bilgisayar grafikleri doku süzmesinde, mipmap'ler önceden hesaplanmış, ana dokuya eşlik eden en iyi şekildeki görüntü derlemeleridir ve insan etkileşiminden dolayı bilgisayar ekranında gördüğümüz çizgiler, eğriler ve simgelerin kenarlarının tırtıklı gözükmesini azaltmak ve rendering hızını arttırmak içindir. Yaygın olarak üç boyutlu bilgisayar oyunları, uçuş simülatörleri ve diğer üç boyutlu görüntüleme sistemlerinde kullanılır. Teknik “mipmapping” olarak bilinir. MIP harfleri Latince “multum in parvo” kelimelerinin kısaltmasıdır ve “küçük alanda daha çok” manasına gelir. Hafızada daha fazla alana ihtiyaç duyarlar.

Minmapping, texture minification sırasında filtreleme işlerinden bazılarını kurtarmak için kullanılan standart bir tekniktir. Texture magnification sırasında, herhangi bir imgecik için düzeltilmesi gereken doku imgeciği sayısı dört veya daha azdır, oysa minification sırasında doku kaplanmış çokgen uzaklaştıkça potansiyel olarak dokunun tamamı tek bir imgeciğe düşebilir. Bu doku kaplanmış imgeciklerin hepsini okumayı ve imgecik rengine doğru bir şekilde karar verebilmek için değerlerini birleştirmeyi gerektirecektir. Yüksek maliyetli bir işlemdir. Bu durumu minmapping dokuyu ön filtreleyerek ve onu tek bir piksele varan küçük boyutlarda saklayarak önler. Doku kaplanmış yüzey uzaklaştıkça, uygulanmış doku ön filtrelenmiş daha küçük hale dönüşür. Minmap'ın farklı boyutları ‘seviyeler’ de tanımlanır. 0.seviye en büyük boyuttadır(izleyiciye daha yakın mesafede kullanılır) ve artarak giden diğer boyutlar mesafelerin daha büyümesiyle kullanılırlar.

3 FİLTRELEME METOTLARI

3.1 NEAREST NEIGBOR INTERPOLATION (EN YAKIN KOMŞU ARADEĞERLEME)

Nearest-neighbor interpolation en hızlı ve ilkel filtreleme metotlarından biridir. İmgecik rengi için, imgeciğin merkezine en yakın doku imgeciğinin rengini kullanır. Hızlı olduğu kadar, yakınlaşmada kaplama bloklama olarak bilinen artifactsler çoğunlukla gözükür.

3.2 NEAREST-NEIGHBOR WITH MIPMAPPING

Bu metot nearest neighbor interpolation ile beraber mipmapping kullanır. Öncelikle mesafeye göre en yakın mipmap seviyesi seçilir, daha sonra imgecik rengini elde edebilmek için

merkezi en yakın doku imgeciğinden örnek alınır. Aliasing ve shimmering problemlerini biraz azaltır.

3.3 BILINEAR FILTERING (İKİ DOĞRUSAL SÜZME)

Bilinear filtering bir adım daha ilerisidir. Bu metotta imgecik merkezine en yakın dört doku imgeciğinden örnek alınır (en yakın minmap seviyesinde), ve bunların renkleri uzaklığa göre ağırlıklı ortalama ile birleştirilir. Bdoku imgecikleri arasındaki geçişleri yumuşatır. Bilinear filtering neredeyse her zaman minmapping ile birlikte kullanılır, ama onsuzda kullanılabileceği gibi aynı aliasing ve shimmering problemlerini yaşar.



3.4 TRILINEAR FILTERİNG (ÜÇ DOĞRUSAL SÜZME)

Trilinear filtering, minmaplenmiş bilinear filtrelenmiş görüntülerde oluşan bozuklukların çoğunu giderir. Bir mipmap seviyesinden diğerine sunuş halindeki geçişlerde görüntülerin sınırlarındaki kalitede gözle görünür bir düzelme sağlar.

Trilinear filtering, iki en yakın mipmap seviyelerinde (bir yüksek ve bir düşük kalite) bilinear filtering ve “texture lookup” yaparak daha sonra sonuçları doğrusal aradeğerleyerek bu sorunu çözer. İzleyiciye olan uzaklık arttıkça doku kalitesinde bir dizi ani düşüş yerine, yumuşak bir bozulma ile sonuçlanır. Tabi ki, seviye 0’ a en yakın sadece bir mipmap seviyesi mevcuttur ve algoritma bu nokta da bilinear filtering’e dönüşür.



3.5 ANISOTROPIC FILTERING (EŞYÖNSÜZ SÜZME)

Eşyönsüz süzme yürürlükteki tüketilen üç boyutlu grafik kartlarında bulunan en yüksek kalitedeki süzme yöntemidir. Üç doğrusal ve iki doğrusal süzmelerde kare şeklinde örnek bir doku ancak tam karşıdan doğru bir açıyla izlendiğinde düzgün gözükür, bu sorundan dolayı eşyönsüz süzme yayılmıştır.

Bu sorun doku kaplanmış yüzeye farklı bir açıyla bakılmasında aşırı buğulanmayı doğurur. Örneğin; çok genel bir durum, bir mesafe boyunca zeminin uzaklaşma durumudur. Eşyönsüz süzme bu sorunu bakış açısına göre doğru bir yamuğa örnekleyerek çözer. Yeni oluşan örnek daha sonra son rengi üretebilmek için üç doğrusal süzülür.



SORU-9: Eşyönsüz Süzme (ANISOTROPIC FILTERING) hakkında bilgi veriniz.

Eşyönsüz süzme, üç boyutlu bilgisayar grafiklerinde kamera açısına bağlı olarak dokunun iz düşümünün dik olmadığı durumlarda ve farklı açılardan bakıldığında yüzeyler üzerindeki görüntü kalitesini yükseltmek için kullanılan bir metottur.

İki doğrusal ve üç doğrusal süzme’de olduğu gibi aliasing etkilerini ortadan kaldırırken, bir yandan buğulanmayı azaltma ve uç görüntüleme açılarında detayların korunması tekniklerinde iyileştirme sağlar.

1990’ların sonlarına doğru tüketici seviyesi grafik kartlarında standart özellik haline gelmiştir ve oldukça yaygındır. Günümüzde eş yönsüz süzme modern grafik donanımında yaygındır; hem sürücü ayarları veya grafik uygulamalarında kullanıcılar tarafından ve programlama arayüzlerinde video oyunları tarafından seçilir kılınmıştır.

2 EŞYÖNSÜZLÜĞÜN DESTEKLEDİĞİ ORANLAR

Kaplama sırasında farklı derece veya oranda eşyönsüz süzme uygulanabilir ve geçerli donanım kaplama uygulamaları bu orana bir üst sınır daha ekler. Bu derece, süzme süreci tarafından desteklenen maksimum eşyönsüzlük oranını gösterir. Örneğin; 4:1 eşyönsüz süzme 2:1’in eğik dokuları keskinleştirmesinden çok daha fazla keskinleştirecektir. Dokular iki kat daha keskin olacaktır. Ancak, bütün sahnede 4:1 süzme kullanmaya gerek yoktur, sadece daha eğik ve genellikle uzak imgecikler (pixel) bu keskinleştirmeye ihtiyaç duyar. Bunun anlamı,

eşyönsüz süzmenin derecesi katlandıkça, görünür kalite açısından azalan getiri vardır, çok çok az sayıda kaplanmış imgecik bundan etkilenir ve çıkan görünümün farkı izleyiciye belli belirsiz gelir. 8:1 ve 16:1 eşyönsüz süzülmüş iki sahne karşılaştırılırsa, sadece çok az sayıda eğimi çok yüksek olan imgecikler, daha çok uzak mesafe geometrisinde olanlar yüksek dereceli eşyönsüz süzmedeki sahnede görünür biçimde keskin gözükebileceklerdir ve bu az sayıda 16:1 süzülmüş imgecikler den alınan frekans bilgisi 8:1 oranının sadece iki katıdır. Daha az imgecik yüksek eşyönsüzlüğün veri getirimlerine ihtiyaç duyar bu yüzden, güç azalması da azalır. Sonuç olarak bu azalmanın neticesi fazladan donanım karmaşıklığı gibi şeyleri beraberinde getirir. Bu durum da donanım tasarımında eşyönsüzlük kalitesine bir üst sınır konulmasına neden olur. Uygulamalar ve kullanıcılar bu işlemlerden sonra sürücü ve yazılım ayarlarından bu eşiğe kadar ayarlayıp kullanmakta özgürdürler.

3 UYGULAMA

Doğru eşyönsüz süzme, eşyönsüzlüğün herhangi bir yönelme anı için her bir imgecik temel alınarak dokuyu yön bağımlı olarak dokuyu eşyönsüz olarak inceler. Grafik donanımında, genellikle doku eşyönsüz olarak örneklendiğinde dokunun merkezine yakın birkaç doku imgeciği örnekleme alınır ama o imgecikteki izdüşümü alınmış şekle göre örnek desen eşlenir. Her doku imgeciği örnekleme kendi içinde, sürece daha fazla örnekleme ekleyen süzülmüş mipmap örneğidir. On altı adet üç doğrusal eşyönsüz örnekleme, saklanmış dokudan 128 adet örneğe ihtiyaç duyar.(Üç doğrusal mipmap süzme dört örneğe ihtiyaç duyar, ve iki mipmap seviyesi olduğu için iki katı alınacaktır daha sonra eşyönsüz örnek 16 adet üç doğrusal süzülmüş doku imgeciği örnekleme ihtiyacı duyar.)

4 PERFORMANS VE ENİYİLEME

Gerekli örneklem sayısı, eşyönsüz süzmeyi fena halde yoğun bantgenişliği ihtiyacına sürükleyebilir. Çoklu dokular için genelde, her doku örnekleme dört veya daha fazla byte'tır böylece her eşyönsüz imgecik , her ne kadar doku sıkıştırma ile bu düşürülmeye çalışılsa da, doku belleğinden 512 byte gerektirebilir. Bir ekran rahatlıkla bir milyondan fazla imgecik içerir,ve istenilen çerçeve hızı(frame rate) saniyede 30- 60 çerçeve veya daha fazla hızdadır, böylece doku belleğinin bantgenişliği çok hızlı ve çok yüksek olabilir (saniyede yüzlerce gigabyte'a varan). Daha iyi performans için bir takım etkenlerin durumu yumuşatması iyi bir şey. Doku imgeciği örneklemlerinin kendileri belleğe alınmış doku örneklerini paylaşırlar. 16 eşyönsüz süzmede bile, 16 sınıfın hepsi her zaman gerekli değildir çünkü sadece mesafece uzak imgeciklerin doldurulmasında yüksek bir eşyönsüzlük gerekir.

SORU-10: 3 Boyutlu Şekil Dönüşümleri hakkında bilgi veriniz.

Bilgisayar grafiklerinde bir şeklin dönüştürülmesi sırasında aşağıdaki 3 işlemten birisi yapılabilir:

içerik

3 boyutlu uzayda şekil taşıma

3 boyutlu uzayda şekil döndürme

Eksene paralel doğru etrafında şekil döndürme

Eksene paralel olmayan doğru etrafında şekil döndürme

3 boyutlu şekil ölçekleme

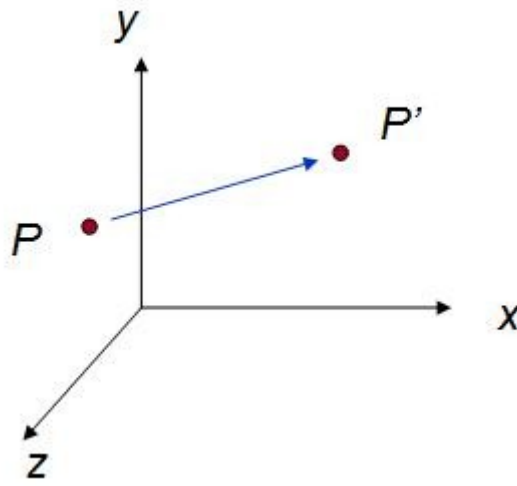
- Taşıma (Translation)

- Döndürme (Rotation)
- Ölçekleme (Scaling)

Yukarıdaki bu işlemlerin 2 boyutlu şekil dönüşümlerinin (2D Transformations) yanında 3 boyutlu olarak işlenmesi de mümkündür. Bu yazının amacı yukarıda listelenen işlemlerin sırasıyla anlatılmasıdır.

3 Boyutlu uzayda şekil taşıma

Şekil taşıma işlemi basitçe bir şekli oluşturan noktaların değerlerinin belirli bir x,y ve z boyutu değişimine tabi tutulması olarak düşünülebilir. Bu durumu basit bir nokta ile anlayabiliriz. Örneğin (x,y,z) değerlerine sahip bir noktanın taşınması demek aslında noktanın (x+tx,y+ty,z+tz) koordinatlarında kodlanması demektir. Buradaki tx,ty ve tz değerleri taşıma işlemi sırasındaki yer değiştirme miktarlarıdır ve taşıma işleminin yönüne göre eksi değer alabilirler.



Yukarıdaki şekilde P noktası P' noktasına taşınmıştır. Bu işlem için homojen koordinat sistemi (homogenous coordinate system) kullanılacak olursa noktalarımızı aşağıdaki şekilde gösterebiliriz:

$$P = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad P' = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}$$

Yukarıdaki vektör (yöney) gösterimlerindeki 1 değerleri homojen koordinat sisteminden gelen değerdir. Ayrıca yukarıdaki x,y,z değerlerini 3 boyutlu kartezyen uzayda (cartesian space) bir nokta belirlemek için kullanılan değerler olarak düşünmek mümkündür.

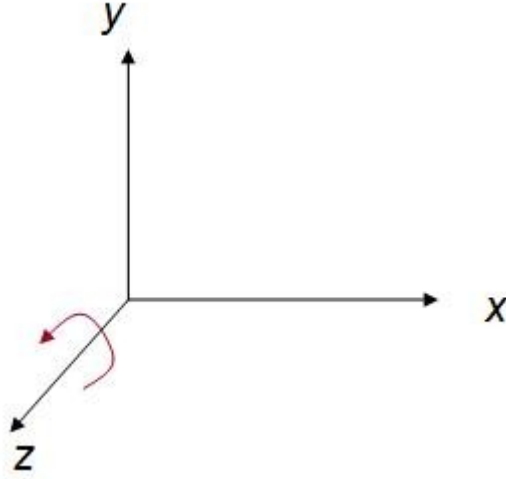
Yukarıdaki iki nokta arasındaki dönüşüm işlemi için aşağıdaki dönüşüm masfufunu (matrix) kullanmak mümkündür.

$$T = \begin{pmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Yukarıdaki dönüşüm masfufunu kullanarak $P' = T P$ masfuf çarpımını yapmak mümkündür.

3 boyutlu uzayda şekil döndürme

Döndürme (rotation) işlemi taşıma (translation) işlemine benzer şekilde bir nesnenin bir doğru etrafında dönmesidir. Temel döndürme işlemleri eksen etrafında yapılan işlemlerdir. Örneğin aşağıdaki şekilde görüntülenen z eksen etrafında döndürme işlemini inceleyelim:



Yukarıdaki şekilde görüntülenen döndürme aslında 2 boyutlu döndürme (2d rotation) olarak düşünülebilir çünkü döndürme işlemi sırasında sadece şeklin x ve y değerlerinde değişikli olacak z değeri sabit kalacaktır. 3 boyutlu döndürmenin 2 boyuttan tek farkı ise şekli oluşturan noktaların 3 boyutlu olması dolayısıyla da x,y ve z değerlerine sahip olmasıdır.

Bu durumda bir noktanın döndürülmeden önceki ve döndürüldükten sonraki değerlerini aşağıdaki şekilde gösterecek olursak:

$$P = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad P' = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}$$

Bu noktalar arasındaki geçişi formüllediğimizde:

$$x' = x \cos\theta - y \sin\theta$$

$$y' = x \sin \theta + y \cos \theta$$

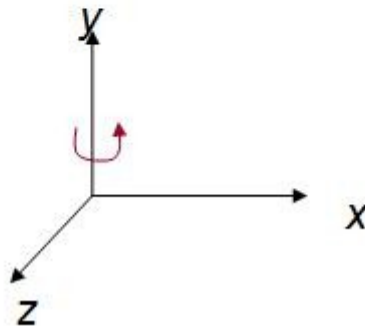
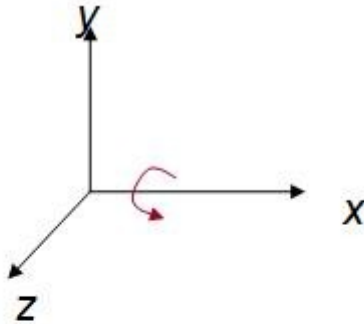
$$z' = z$$

şeklinde gösterebiliriz. Yani bir noktanın orjin (ordinat) merkez alınarak z eksenini etrafında döndürülmesi işlemi sonucunda oluşan yeni nokta değerleri 2 boyutlu şekil dönüşümüne benzer şekilde yukarıda verilen formülden hesaplanabilir.

Bu formülü bir dönüşüm masfufu indirgemek gerekirse aşağıdaki şekilde bir masfuf elde edilir:

$$R = \begin{pmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Yukarıdaki masfuf kullanılarak $P' = T P$ çarpımı sonucunda istenilen denklemler elde edilmiş olur. Döndürme işlemi z eksenini etrafında yapılabileceği gibi y ve x eksenleri etrafında da yapılabilir.



Yukarıdaki şekillerde gösterilen bu döndürme işlemleri için aşağıdaki masfuflar (matrices) kullanılabilir:

$$\begin{aligned}y' &= y.\cos\theta - z.\sin\theta \\z' &= y.\sin\theta + z.\cos\theta \\x' &= x\end{aligned}$$

$$R = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

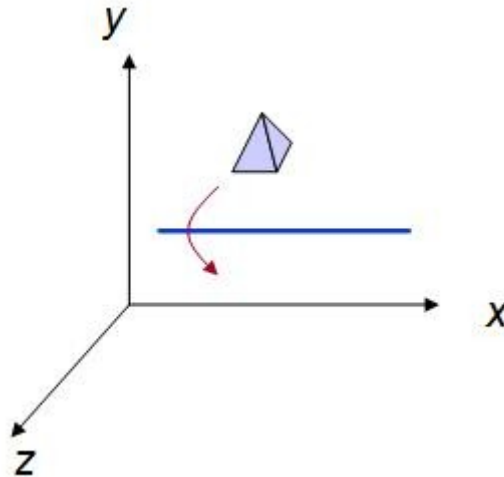
$$\begin{aligned}z' &= z.\cos\theta - x.\sin\theta \\x' &= z.\sin\theta + x.\cos\theta \\y' &= y\end{aligned}$$

$$R = \begin{pmatrix} \cos \theta & 0 & -\sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Yukarıdaki denklemler ve masfuflar sırasıyla x ve y eksenleri etrafında yapılan döndürme işlemleridir.

Eksene paralel doğru etrafında döndürme

Yukarıdaki bölümde açıklanan bir eksen etrafında döndürme işlemine benzer olarak bir doğru etrafında da döndürme işlemi gerekebilir. Bu durumda şayet doğru bir eksene paralelse, döndürme işlemi basit bir taşıma işlemi ilave edilerek eksen etrafında döndürme işlemiyle çözülebilir.



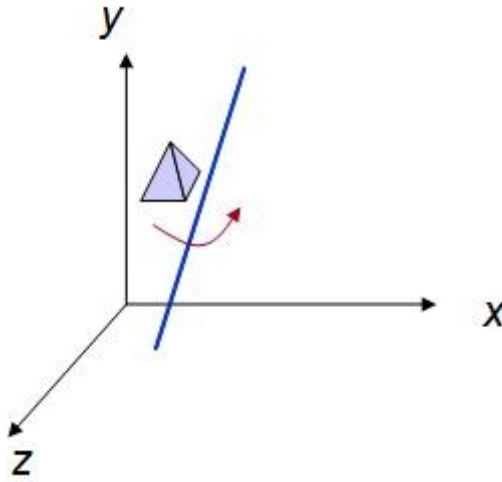
Yukarıdaki şekilde x eksenine paralel bir doğru etrafında döndürme işlemi tasvir edilmiştir. Bu durumda doğruyu eksene taşımak ve benzer şekilde şekli de eksene taşıma ve döndürmek sonra da geri taşımak bir çözüm olur.

$$P' = T^{-1} \cdot R_x(\theta) \cdot T \cdot P$$

Yukarıda verilen bu dönüşüm işlemleri sırasıyla yapılırsa, yani P noktası önce taşınırsa (T) sonra x eksenı etrafında döndürülürse ($R_x(\theta)$) ve sonra eski yerine taşınırsa (T^{-1}) şekil aslında eksene paralel olan bu doğru etrafında döndürülmüş olur.

Eksene paralel olmayan doğru etrafında şekil döndürme

Yukarıdaki bölümde anlatıldığı gibi her zaman bir doğru eksene paralel olmayabilir. Bu durumda eksene paralel olmayan bir doğru için önce etrafında döndürme yapılacak doğrunun eksenlerden birisine paralel hale getirilmesi gerekir.



Örneğin yukarıdaki şekilde, herhangi bir eksene paralel olmayan bir doğru etrafında şeklimizi döndürmek isteyelim.

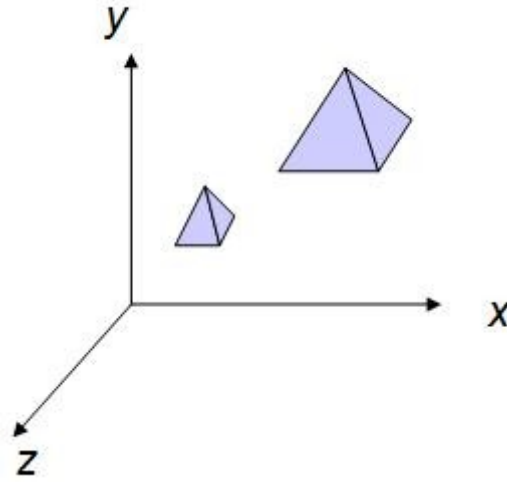
Bu işlem 5 adımda gerçekleştirilebilir:

1. Doğrunun merkezden geçecek şekilde taşınması
2. Doğrunun eksenlerden birisine paralel halde döndürülmesi
3. Şeklin eksen etrafında döndürülmesi
4. Doğrunun eski eğimine döndürülmesi
5. Doğrunun merkezden eski yerine taşınması

Yukarıda sıralandığı üzere nokta (veya şekli oluşturan noktalar) sırasıyla 5 dönüştürme işlemine tabi tutulur ve sonuçta şekil verilen doğru etrafında döndürülmüş olur.

3 boyutlu şekil ölçekleme

Bir şeklin ölçeklenmesi, şekli oluşturan noktaların x,y ve z değerlerinin belirli bir katsayı ile çarpılması ile mümkündür. Bu işlem sırasında şekil hem yer değiştirmekte hem de şekli oluşturan noktalar birbirinden uzaklaşıp yaklaştığı için şekil büyümekte veya küçülmektedir.



Yukarıdaki şekilde de görülen bu durum için aşağıdaki dönüşüm masfufu kullanılabilir:

$$S = \begin{pmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

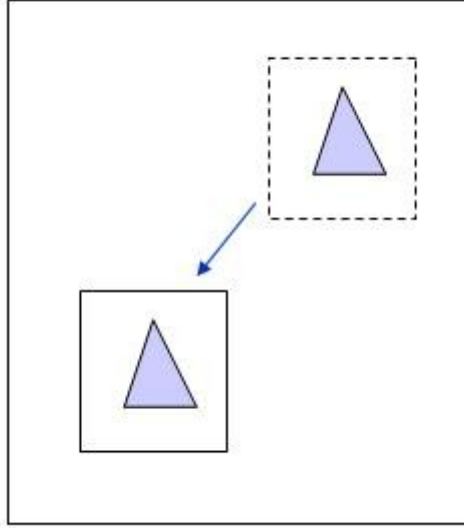
Yukarıda verilen bu masfuf ile bir nokta aşağıdaki şekilde çarpıldığında elde edilen yeni P' noktası merkeze göre verilen değerlerde ölçeklenmiş bir noktadır:

$$P' = T P$$

SORU-11: Izgara Tarama (Raster Scan) hakkında bilgi veriniz.

Bilgisayar grafiklerinde bir şeklin ekranda gösterilmesi sırasında kullanılan yöntemin ismidir. Basitçe ekranı imgecik (pixel) matrisinden oluşan bir ızgara gibi düşünebiliriz. Örneğin 1024 x 768 boyutlarındaki bir ekranın yine aynı boyutlardaki bir ızgara olarak düşünülmesi mümkündür.

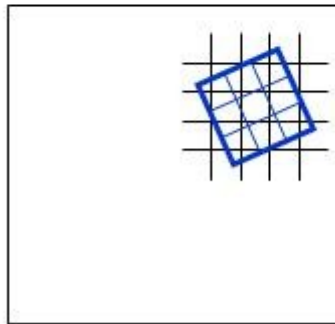
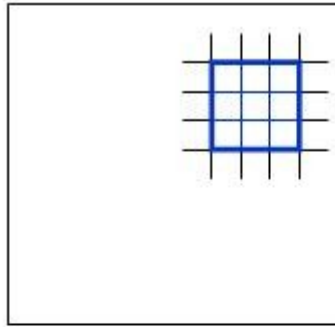
Izgara (raster) kullanılarak şekiller üzerindeki dönüşüm işlemleri yapılabilir. Örneğin şeklin taşınması işlemi için basit hafızada matris bilgilerinin kopyalanması yeterlidir.



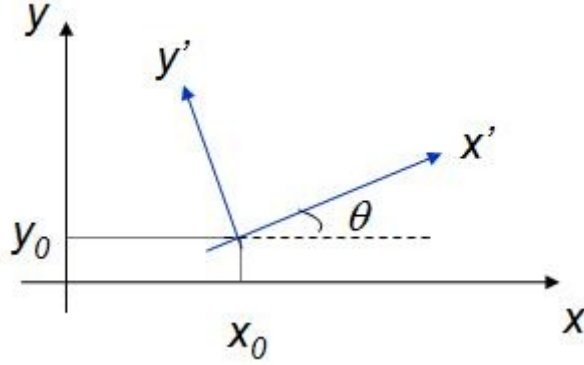
Yukarıdaki şekilde ilk konumdaki üçgen kesilerek yeni konuma taşınmıştır. Aslında bir taşıma işlemi olan bu olay hafızadaki ilgili imgeciklerin (pixel) yerinin değiştirilmesi olarak düşünülebilir.

Benzer şekilde döndürme (rotation) , ölçekleme (scaling), yansıma (reflection) ve eğme (shearing) işlemleri de aslında hafızadaki bir alanın yeniden hesaplanarak gösterilmesi olarak düşünülebilir.

Örneğin döndürme (rotation) işlemi için aşağıdaki dönüşüm kullanılabilir:



Yukarıdaki şekilde görüldüğü üzere şekli oluşturan 3×3 boyutlarındaki alan döndürülmüştür. Bunun sonucunda bu alanda bulunan şekil döndürülmüş olur. Aslında şeklin döndürülmesi veya şeklin çizildiği alanın döndürülmesi aynı şeyler olmasına karşılık ortada iki farklı koordinat sistemi oluşmaktadır.

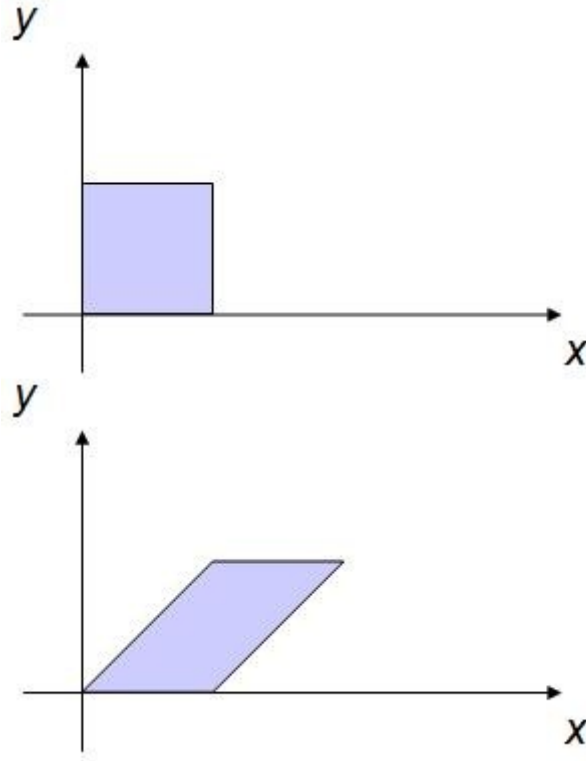


Yukarıdaki şekilde görüldüğü üzere x, y kartezyen uzayındaki x', y' alt uzayı (raster space) sadece şeklin ilgili alanının yeniden çizileceği bölge için geçerli olan koordinat sistemidir. Bu durumda şekil normal çizilirken çizim alanının ilgili dönüşüme tabi tutulması gerekir.

İki uzay arasında (Bütün şekilleri kapsayan ana uzay ile dönüşüm uygulanan alt uzay (Raster space)) bir dönüşüm matrisi elde edilebilir ve bu alan için dönüşüm matrisi uygulanarak çizim yapılabilir.

SORU-12: Şeklin Eğilmesi (Shearing) hakkında bilgi veriniz.

Bilgisayar grafiklerinde bir şeklin herhangi bir ekseninde eğilmesine verilen işlemidir. Temel şekil değiştirme (Transformation) işlemlerinden birisidir. Aşağıdaki temsili şekilde gösterilmiştir:



Yukarıdaki ilk şekilde olan kare, ikinci şekilde gösterildiği üzere eğilmiştir. Eğme işlemini aşağıdaki dönüşüm matrisi (transformation matrix) ile yapabiliriz.

$$\begin{pmatrix} 1 & shx & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Yukarıdaki bu dönüşüm matrisini (x,y) kartezyen koordinatlarına sahip temsili bir noktaya uygulayacak olursak

$$P' = T \times P$$

şeklinde çarpma işlemi ile dönüştürmek için iki matrisi çarpalım ve sonuçta:

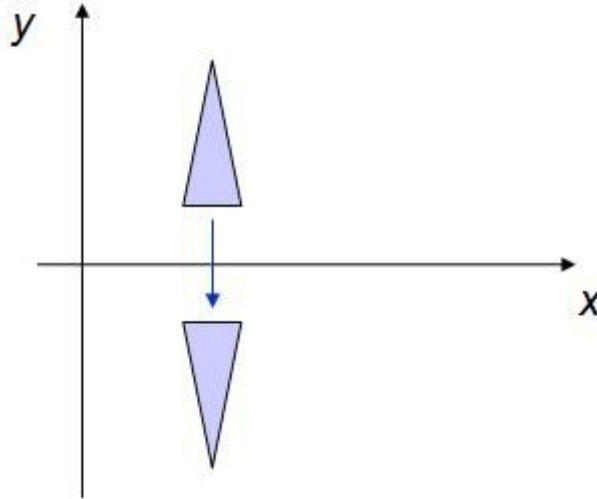
$$x = x + y shx$$

$$y = y$$

sonucuna ulaşırız. Burada görüldüğü üzere şekli oluşturan noktaların x değerlerinde y değerine bağlı bir ilave değer oluşmaktadır. Noktanın y değeri ne kadar büyükse ilave bu değer de o kadar artmaktadır. Sonuçta x değeri y değerinin shx kadar çarpımı kadar kaymakta ve şekilde eğilmektedir.

SORU-13: Yansıma (Reflection) hakkında bilgi veriniz.

Bilgisayar grafiklerinde, kartezyen uzayda (cartesian space) bulunan bir noktanın herhangi bir eksene veya doğruya göre yansımaları almak mümkündür. Örneğin aşağıdaki üçgeni ve x eksenine göre yansımaları ele alalım:



Yukarıdaki üçgen aslında üçgeni oluşturan 3 noktanın teker teker yansımalarının alınması ve sonuçta elde edilen 3 ayrı yansımış noktanın yeniden üçgen oluşturması olarak düşünülebilir.

Dolayısıyla tek bir noktanın yansımaları almak ve bu yansıma işlemini 3 noktaya ayrı ayrı uygulamak ve sonuçta çıkan noktalardan üçgen oluşturmak şeklin yansıması olur.

Tek bir noktanın yansımaları almak için bu işte kullanılacak bir dönüşüm matrisi elde etmek gerekir. Homojen koordinat sistemini kullanan dönüşüm matrislerine uygun bir şekilde yukarıdaki yansımayı, aşağıdaki matris ile gerçekleştirebiliriz:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Yukarıdaki matriste dikkat edilirse birim matristeki (identity matrix) y değeri -1 olmuştur. Bunun anlamı koordinat sistemindeki bütün y değerlerinin - değerini almasıdır. Yani yukarıdaki dönüşüm matrisi

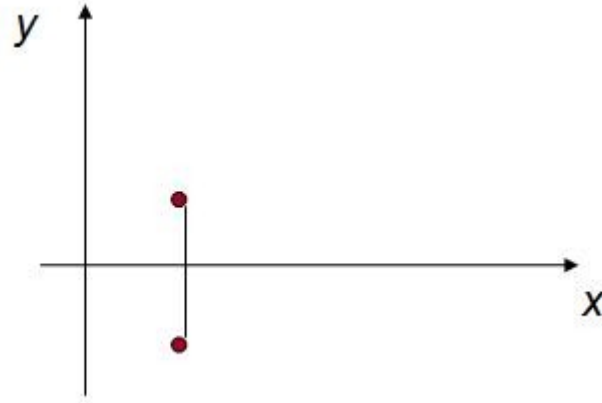
x
y
1

şeklinde bir nokta ile çarpma işlemine tabi tutulursa

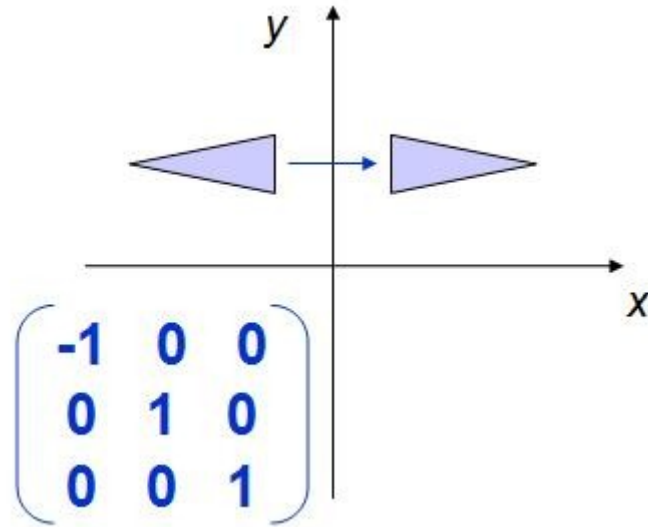
$$x = x$$

$$y = -y$$

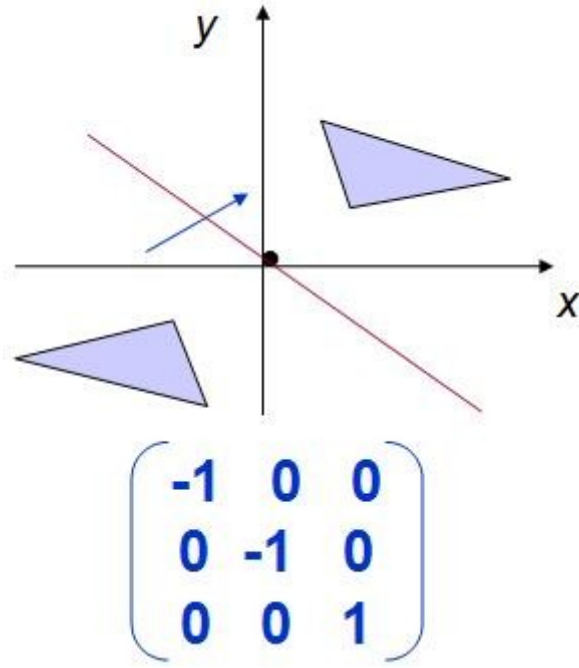
sonuçları elde edilir. Daha basit bir ifade ile aşağıdaki şekilde gösterilen 2,1 noktası 2,-1 noktasına dönüşmektedir.



Sonuç olarak y değerinin eksi değerine dönüştürülmesi x ekseninde bir yansıma elde edilmesini sağlar. Benzer şekilde y ekseninde yansıma elde etmek için de x değerlerinin eksisinin alınması gerekir. Bu işlemi yapan matris ve örnek şekil aşağıda verilmiştir:



Örneğin bu iki dönüşüm matrisi birleştirilirse ve hem x hem de y değerlerinin eksisi alınırsa bu durumda hem x hem de y eksenlerine göre yansıma elde edilmiş olur:



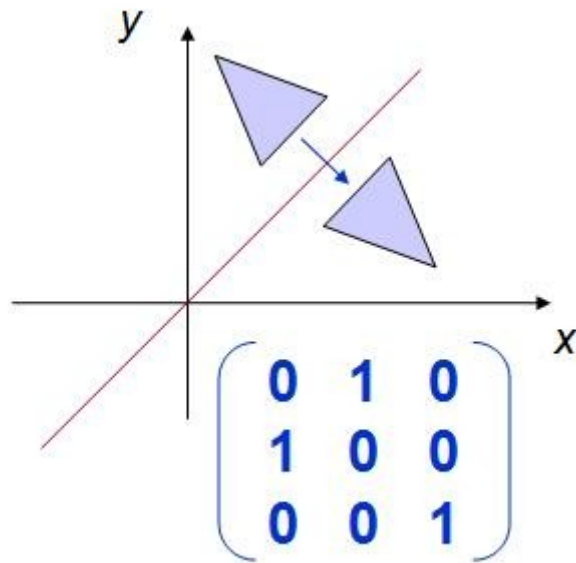
Yukarıdaki şekilde görülen ve matriste ifade edilen yansıma bu şekilde bir yansımadır. Aslında bu yansıma yukarıdaki kırmızı doğru ile gösterilen

$$y = -x$$

doğrusuna göre alınmış bir yansıma olarak da düşünülebilir. Benzer şekilde

$$y = x$$

doğrusuna göre yansıma almamız da mümkün olabilir:



Yukarıdaki şekilde ve matriste verilen yansıma ise bu şekilde alınmış $x = y$ doğrusuna göre şeklin yansımasıdır.

SORU-14: Ters Şekil Değiştirme Matrisleri hakkında bilgi veriniz.

Bilgisayar grafiklerinde kullanılan ve bir şeklin değişmesine (transformation) yarayan matrisleri üç grupta toplayabiliriz:

İçerik

Taşıma işleminin geri alınması

Döndürme işleminin geri alınması

Ölçekleme işleminin geri alınması

Örnek uygulama

- Taşıma (Translation)
- Döndürme (Rotation)
- Ölçekleme (Scaling)

Bu üç gruptaki matrislerin aynı zamanda tersini almak da mümkündür. Yani herhangi bir değişim işlemine maruz kalan şeklin geri dönüşerek ilk haline dönmesi ancak bu dönüşüm işleminin tersi olan matrisin dönüşmüş şekle uygulanması ile mümkündür. Bu işlemi üç alt başlıkta yukarıdaki dönüşüm işlemlerinin sırasıyla inceleyeceğiz.

Taşıma işlemi ve tersinin alınması

[Taşıma işlemi](#) için kullanılan dönüşüm matrisi aşağıdaki şekildedir:

$$\begin{pmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{pmatrix}$$

Bu matris ile

$$P' = T \times P$$

matris çarpım işlemi yapıldığında P' değerini veren yeni nokta için

$$x' = tx + x$$

$$y' = ty + y$$

sonucu çıkar. Buradaki tx ve ty değerleri sırasıyla x ve y eksenlerindeki hareketi göstermektedir.

İşlemin geri alınabilmesi için yukarıdaki matrisin tersi olan aşağıdaki matris ile taşınmış noktanın çarpılması mümkündür:

$$\begin{pmatrix} 1 & 0 & -tx \\ 0 & 1 & -ty \\ 0 & 0 & 1 \end{pmatrix}$$

Yukarıdaki yeni matrise T' dersek aşağıdaki eşitlikler doğru olur:

$$P' = T \times P \rightarrow P = T' \times P'$$

$$P = T' \times T \times P$$

görüldüğü üzere $T' \times T$ çarpımı birim matrisi (identity matrix) vermektedir ve bir matris önce T sonra T' ile çarpıldığında ilk haline dönmektedir.

Bu durumu işleme dökerek deneyelim:

$P' = T \times P$ işlemini yapacak olursak

$$\begin{pmatrix} tx + x \\ ty + y \\ 1 \end{pmatrix}$$

matrisini elde etmiş oluruz. Bu yeni matrisi (yani P' noktasını) T' matrisi ile çarpalım:

$$x = tx + x - tx$$

$$y = ty + y - ty$$

sonucunu bulmuş oluruz ve yukarıdaki işlem sonucunda

$$x = x$$

$$y = y$$

şeklinde ilk koordinatlara dönmek mümkündür.

Döndürme işleminin geri alınması

Taşıma işlemine benzer şekilde döndürme işleminin de geri alınması mümkündür. Bunun için aşağıdaki döndürme matrisinin:

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

tersinin alınması gerekir. Yukarıdaki matrisin tersi aşağıda verilmiştir:

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

Yukarıdaki bu iki matrisin birbirinin tersi olduğunu yine işlem üzerinde görelim. Öncelikle ilk dönüşüm matrisimizi herhangi x y koordinatlarına sahip bir P noktası üzerinde uygulayalım

$$P' = T \times P$$

işlemini yapacak olursak :

$$x = x \cos \theta - y \sin \theta$$

$$y = x \sin \theta + y \cos \theta$$

şeklinde yeni koordinat değerlerini elde etmiş oluruz. Şimdi bu yeni koordinat değerleri üzerinde ters dönüşüm matrisimizi uygulayalım. Yani

$$\begin{pmatrix} x \cos \theta - y \sin \theta \\ x \sin \theta + y \cos \theta \\ 1 \end{pmatrix}$$

matrisi ile yukarıdaki ters dönüşüm matrisini çarpalım. Sonuçta aşağıdaki değeri elde ederiz:

$$x = (x \cos \theta - y \sin \theta) \cos \theta + (x \sin \theta + y \cos \theta) \sin \theta$$

$$y = -(x \cos \theta - y \sin \theta) \sin \theta + (x \sin \theta + y \cos \theta) \cos \theta$$

Yukarıdaki bu işlemleri ilerletecek ve çarpanları parantez içerisine dağıtacak olursak:

$$x = (x \cos \theta \cos \theta - y \sin \theta \cos \theta) + (x \sin \theta \sin \theta + y \cos \theta \sin \theta)$$

$$y = -(x \cos \theta \sin \theta - y \sin \theta \sin \theta) + (x \sin \theta \cos \theta + y \cos \theta \cos \theta)$$

parantezleri açıp toplama işlemlerini yaparsak

$$x = x \cos\theta \cos\theta - y \sin\theta \cos\theta + x \sin\theta \sin\theta + y \cos\theta \sin\theta$$

$$y = -x \cos\theta \sin\theta + y \sin\theta \sin\theta + x \sin\theta \cos\theta + y \cos\theta \cos\theta$$

sonuçları ortak paranteze alırsak

$$x = x(\cos\theta \cos\theta + \sin\theta \sin\theta)$$

$$y = y(\sin\theta \sin\theta + \cos\theta \cos\theta)$$

Trigonometrik kurallardan hatırlanacağı üzere $\cos^2\theta + \sin^2\theta = 1$ olur. Dolayısıyla aşağıdaki sonuca varılmış olur:

$$x = x$$

$$y = y$$

Görüldüğü üzere işlem geri alınmış ve θ açısı kadar döndürülmüş olan şekil geri döndürülmüş ve ilk değerini almıştır.

Ölçekleme işleminin geri alınması

Yukarıdaki taşıma ve döndürme işlemlerine benzer şekilde ölçekleme işlemi de geri alınabilir. Öncelikle ölçekleme matrisimizi hatırlayalım:

$$\begin{pmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Yukarıdaki bu matrisin tersi alınacak olursa:

$$\begin{pmatrix} 1/sx & 0 & 0 \\ 0 & 1/sy & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

yukarıdaki matris bulunur. Bu matrisi de işlem ile deneyerek ters almada kullanılabileceğini görelim.

Öncelikle

$P' = T \times P$ işlemini yapıp P' noktasının koordinatlarını bulalım:

$$x' = x \cdot sx$$

$$y' = y \cdot s_x$$

sonuçları bulunur. Bu sonuçların oluşturduğu

$$\begin{pmatrix} x & s_x \\ y & s_y \\ 1 & 1 \end{pmatrix}$$

matrisini ters dönüşüm matrisimiz ile çarpalım. Sonuç aşağıdaki şekilde çıkacaktır:

$$\begin{pmatrix} x & s_x \\ y & s_y \\ 1 & 1 \end{pmatrix} \begin{pmatrix} s_x & / & s_x \\ s_y & / & s_y \\ 1 & 1 \end{pmatrix}$$

Görüleceği üzere $s_x/s_x = 1$ ve $s_y/s_y = 1$ değerleri yazılacak olursa

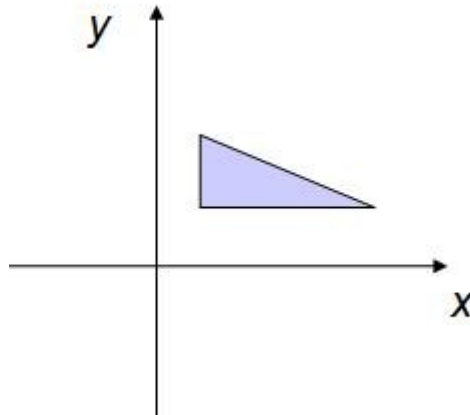
$$x = x$$

$$y = y$$

şeklinde ölçekleme işlemi geri alınmış olur.

Örnek

Yukarıdaki konunun daha iyi anlaşılabilmesi için aşağıdaki soruyu çözmeye çalışalım:



Yukarıdaki şekilde bulunan (3,2) (3,3) (5,2) koordinatlarına sahip üçgen ordinat (0,0 noktası) etrafında 45 derece (saat yönünün tersi istikametinde) döndürülüyor. Ardından $y = -x$ doğrusuna göre yansıması (reflection) alınıyor ve $t_x = 2$ $t_y = 1$ olacak şekilde taşınıyor. Son olarak şekil 2 katına çıkarılıyor (büyütülüyor) yukarıdaki bu işlemler sonucunda üçgenin koordinatlarının ne olacağını adım adım hesaplayınız. Yukarıdaki işlemleri geri alan matrisleri yazıp bu matrisleri uyguladıktan sonra sonuçta üçgenin Orijinal noktalarına geri döndüğünü gösteriniz. ($\sin 45 = 0.7$ ve $\cos 45 = 0.7$ olarak alınız)

Çözüm:

Döndürme işlemi için kullanılan matris

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

Şeklindedir. Bu matris ile noktalarımız oluşturan matrislerin (vektörlerin) çarpımı döndürülmüş nokta koordinatlarını verecektir. Toplam 3 adet noktamız bulunuyor bunlar homojen koordinat olarak yazılacak olursa:

N1:

$$\begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$

N2:

$$\begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix}$$

N3:

$$\begin{bmatrix} 5 \\ 2 \\ 1 \end{bmatrix}$$

Noktalarıdır. Her noktanın döndürülmüş halini bulmak için döndürme (rotation) matrisi ile ayrı ayrı çarpıyoruz.

N1':

$$\begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} \cos 45 & -\sin 45 & 0 \\ \sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 3x0.7 - 2x0.7 \\ 3x0.7 + 2x0.7 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 3.5 \\ 1 \end{bmatrix}$$

N2':

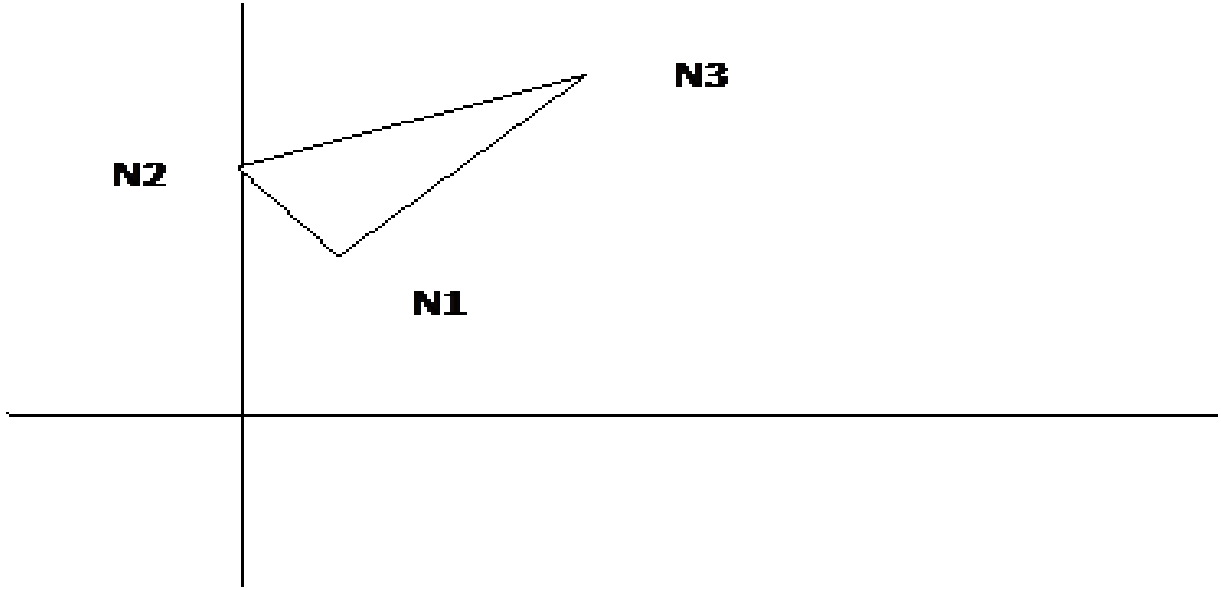
$$\begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix} \begin{bmatrix} \cos 45 & -\sin 45 & 0 \\ \sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 3x0.7 - 3x0.7 \\ 3x0.7 + 3x0.7 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 4.2 \\ 1 \end{bmatrix}$$

N3':

$$\begin{bmatrix} 5 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} \cos 45 & -\sin 45 & 0 \\ \sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 5x0.7 - 2x0.7 \\ 5x0.7 + 2x0.7 \\ 1 \end{bmatrix} = \begin{bmatrix} 2.1 \\ 4.9 \\ 1 \end{bmatrix}$$

Noktalarıdır. Bu noktaların oluşturduğu üçgen merkezden geçen z eksenini etrafında 45 derece döndürülmüş üçgen olarak kabul edilebilir.

Yukarıdaki bu noktalardan oluşan üçgen çizilirse aşağıdaki şekilde bir üçgen elde edilir:



Yukarıdaki üçgen dikkat edilirse ilk üçgenenin 45° döndürülmüş halidir.

Yukarıdaki üçgenin $y=-x$ doğrusuna göre yansımasını alalım.
Bu yansıma için aşağıdaki dönüşüm matrisi kullanılmalıdır.

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Matrisidir. Dolayısıyla 3 nokta ayrı ayrı yukarıdaki matris ile çağrılacaktır.

Noktalarımız aşağıda verilmiştir.

N1'

$$\begin{bmatrix} 0.7 \\ 3.5 \\ 1 \end{bmatrix}$$

N2'

$$\begin{bmatrix} 0 \\ 4.2 \\ 1 \end{bmatrix}$$

N3'

$$\begin{bmatrix} 2.1 \\ 4.9 \\ 1 \end{bmatrix}$$

Yukarıdaki noktalardan oluşan üçgenin $y=-x$ doğrusuna göre yansımaları alabiliriz. Bunun için bütün noktaları yansıma matrisi ile çarpıyoruz:

N1'':

$$\begin{bmatrix} 0.7 \\ 3.5 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.7 \\ -3.5 \\ 1 \end{bmatrix}$$

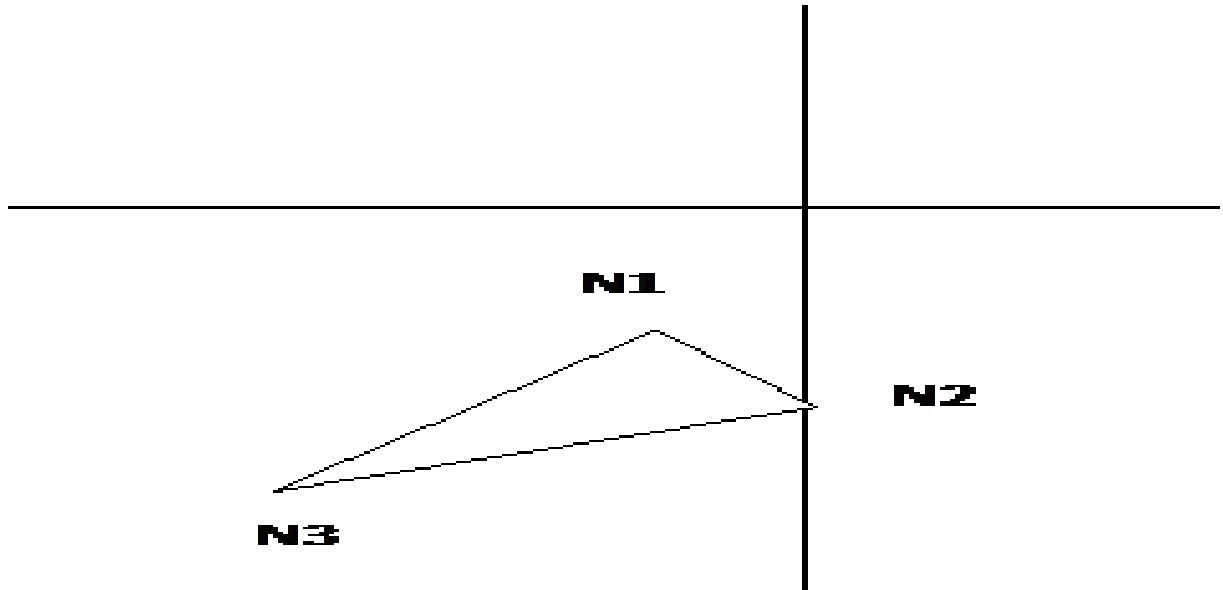
N2'':

$$\begin{bmatrix} 0 \\ 4.2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -4.2 \\ 1 \end{bmatrix}$$

N3'':

$$\begin{bmatrix} 2.1 \\ 4.9 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -2.1 \\ -4.9 \\ 1 \end{bmatrix}$$

Yukarıdaki üçgenin yansıması alınmış hali aşağıdaki temsili resimde gösterilmiştir:



Yukarıdaki şekilde üçgenin yansıması alınmış hali görülmektedir.

Taşıma işlemi için aşağıdaki dönüşüm matrisi kullanılabilir.

$$\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix}$$

Bu genel matristeki tx=2 ve ty=1 olacak şekilde bütün noktalarla çarpılır.

N1''':

$$\begin{bmatrix} -0.7 \\ 3.5 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.7 + 2 \\ -3.5 + 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.3 \\ -2.5 \\ 1 \end{bmatrix}$$

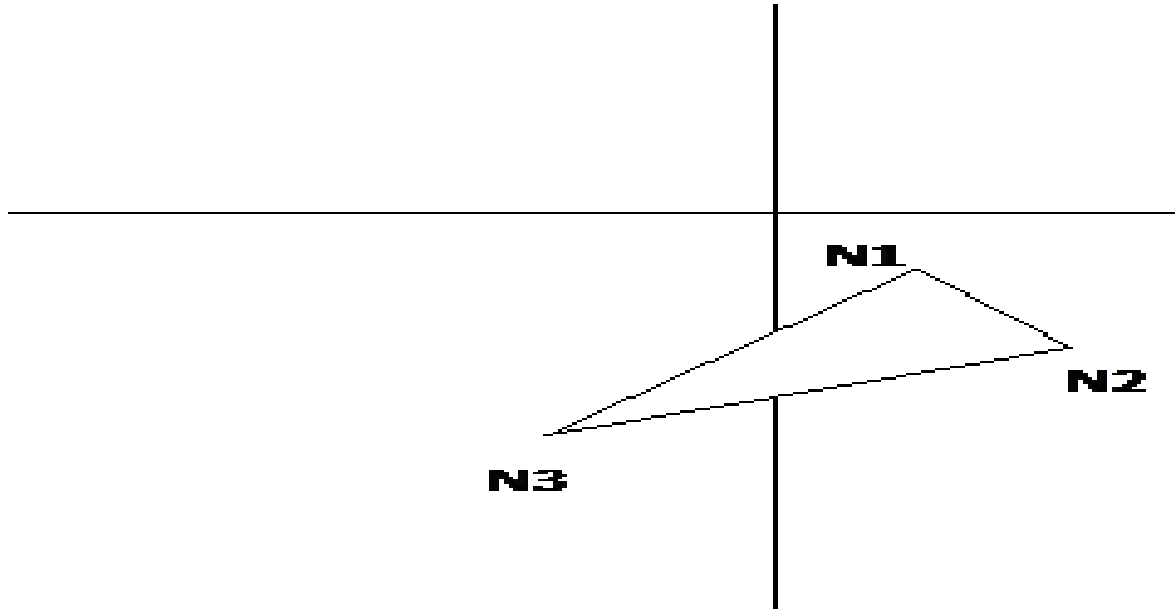
N2''':

$$\begin{bmatrix} 0 \\ -4.2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 + 2 \\ -4.2 + 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ -3.2 \\ 1 \end{bmatrix}$$

N3''':

$$\begin{bmatrix} -2.1 \\ -4.9 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -2.1 + 2 \\ -4.9 + 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.1 \\ -3.9 \\ 1 \end{bmatrix}$$

Olarak bulunur. Yeni noktalardan teşekkül eden üçgeni aşağıdaki temsili resimde görmek mümkündür.



Son olarak soruda üçgenin iki misline çıkarılması istenmiştir. Bu işlem için de aşağıdaki ölçekleme dönüşüm matrisi kullanılabilir.

$$\begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Sorumuzda şeklin 2 misline çıkarılması istendiği için $s_x = 2$ ve $s_y = 2$ olarak alınmalıdır. Bu dönüşüme göre nokta koordinatlarımız aşağıdaki şekilde hesaplanabilir:

N1'''';

$$\begin{bmatrix} 1.3 \\ -2.5 \\ 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2.6 \\ -5 \\ 1 \end{bmatrix}$$

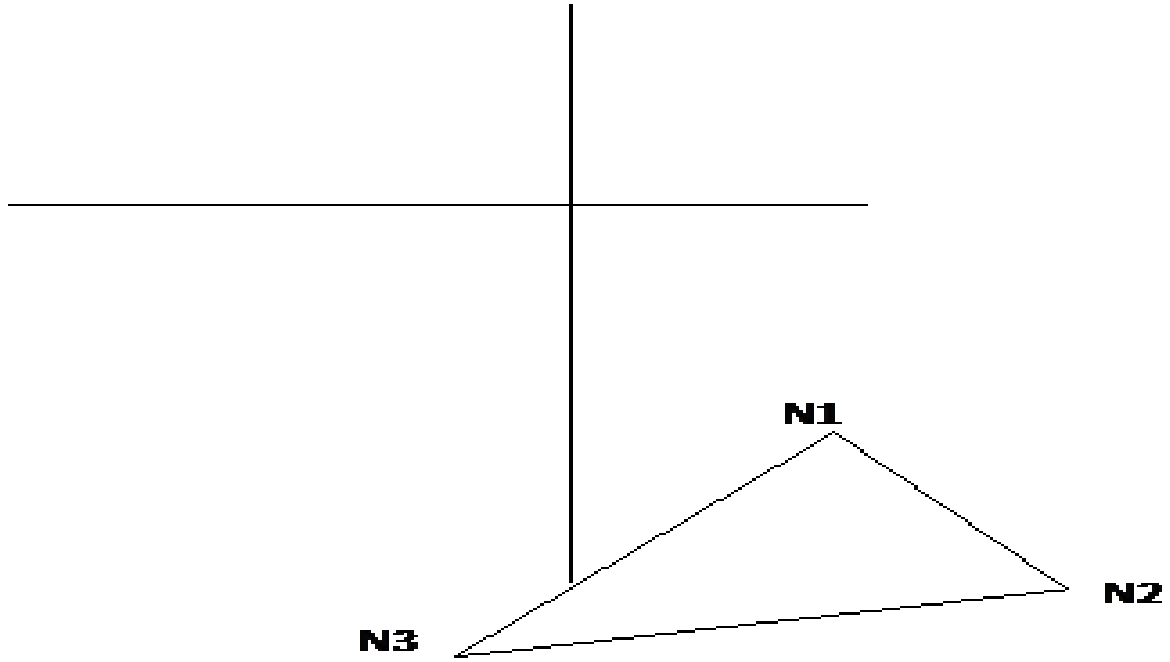
N2'''';

$$\begin{bmatrix} 2 \\ -3.2 \\ 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 4 \\ -6.2 \\ 1 \end{bmatrix}$$

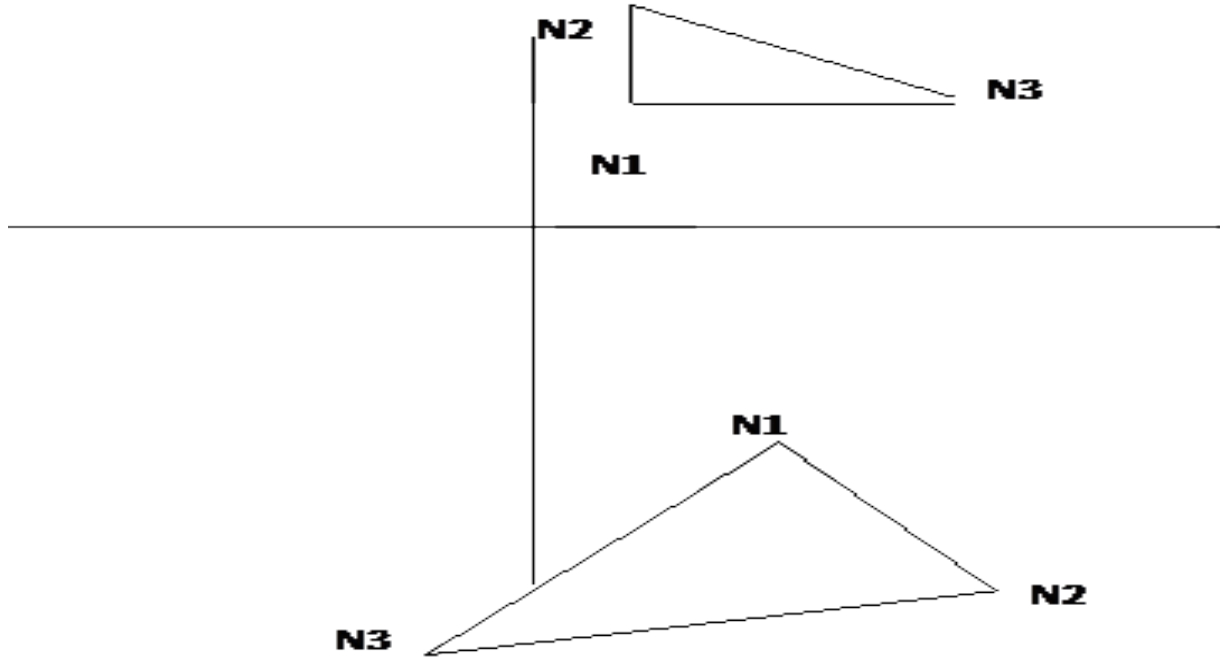
N3'''';

$$\begin{bmatrix} -0.1 \\ -3.9 \\ 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.2 \\ -7.8 \\ 1 \end{bmatrix}$$

Yeni üçgenimiz aşağıdaki temsili resimde tasvir edilmiştir:



Resmin son hali ile ilk halini aynı şekilde gösterecek olursak:



Yukarıdaki görüldüğü gibi üçgen 450 döndürülmüş ardından $y=-x$ doğrusuna göre yansıtması alınmış ardından $x=3$, $y = 1$ değerlerinde taşınmış ve son olarak 2 misline çıkarılmıştır.

Sorunun ikinci kısmında bütün bu işlemlerin geri alınması için ters dönüşüm matrislerinin kullanılması istenmiştir. Dönüşüm işlemlerini sondan başa doğru geri alacak olursak son işlem olan ölçekleme ile başlamamız gerekir. Ölçekleme işleminin ters dönüşüm matrisi aşağıda verilmiştir:

$$\begin{bmatrix} 1/sx & 0 & 0 \\ 0 & 1/sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Sorumuzda şeklin 2 misline çıkarılması istendiği için $sx = 2$ ve $sy = 2$ olarak alınmalıdır. Bu dönüşüme göre nokta koordinatlarımız aşağıdaki şekilde hesaplanabilir:

N1''':

$$\begin{bmatrix} 2.6 \\ -5 \\ 1 \end{bmatrix} \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1.3 \\ -2.5 \\ 1 \end{bmatrix}$$

N2''':

$$\begin{bmatrix} 4 \\ -6.2 \\ 1 \end{bmatrix} \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 \\ -3.2 \\ 1 \end{bmatrix}$$

N3''':

$$\begin{bmatrix} -0.2 \\ -7.8 \\ 1 \end{bmatrix} \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.1 \\ -3.9 \\ 1 \end{bmatrix}$$

Ölçekleme işleminden hemen önce taşıma işlemi yapılmıştır.

Bu işlemin geri alınması için taşımanın ters dönüşüm matrisi kullanılmalıdır:

$$\begin{bmatrix} 1 & 0 & -tx \\ 0 & 1 & -ty \\ 0 & 0 & 1 \end{bmatrix}$$

Dönüşümün tersi için $tx = 2$ ve $ty = 1$ dönüşüm değerleri yerine konulursa noktalar aşağıdaki şekilde hesaplanabilir:

N1'':

$$\begin{bmatrix} 1.3 \\ -2.5 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1.3 - 2 \\ -2.5 - 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.7 \\ -3.5 \\ 1 \end{bmatrix}$$

N2'':

$$\begin{bmatrix} 2 \\ -3.2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 - 2 \\ -3.2 - 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -4.2 \\ 1 \end{bmatrix}$$

N3'':

$$\begin{bmatrix} -0.1 \\ -3.9 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.1 - 2 \\ -3.9 - 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -2.1 \\ -4.9 \\ 1 \end{bmatrix}$$

Noktaları bulunmuş olunur. Bu ters alma işleminden sonra yansıma işleminin tersini alabiliriz. Yansıma dönüşümlerinin tersi yine kendisidir bu durumda yansıma işleminde kullandığımız aşağıdaki matrisi yine kullanır ve bulduğumuz nokta vektörleri ile çarparsak ters dönüşüm elde etmiş oluruz.

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Noktaların her birisi için çarpımları ve sonuçları aşağıda verilmiştir

N1':

$$\begin{bmatrix} -0.7 \\ -3.5 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 3.5 \\ 1 \end{bmatrix}$$

N2':

$$\begin{bmatrix} 0 \\ -4.2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 4.2 \\ 1 \end{bmatrix}$$

N3':

$$\begin{bmatrix} -2.1 \\ -4.9 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2.1 \\ 4.9 \\ 1 \end{bmatrix}$$

Son olarak ilk başta uyguladığımız dönüşüm matrisi olan döndürme işleminin tersini alarak orijinal şekli elde edebiliriz. Döndürme işleminin tersi matris aşağıda verilmiştir:

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

Yukarıdaki matrisi son bulduğumuz noktalara uygulayalım:

N1':

$$\begin{bmatrix} 0.7 \\ 3.5 \\ 1 \end{bmatrix} \begin{bmatrix} \cos 45 & \sin 45 & 0 \\ -\sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.7 \times 0.7 + 0.7 \times 3.5 \\ -0.7 \times 0.7 + 3.5 \times 0.7 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$

N2':

$$\begin{bmatrix} 0 \\ 4.2 \\ 1 \end{bmatrix} \begin{bmatrix} \cos 45 & \sin 45 & 0 \\ -\sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.7 \times 0 + 0.7 \times 4.2 \\ -0.7 \times 0 + 3.5 \times 4.2 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix}$$

N3':

$$\begin{bmatrix} 2.1 \\ 4.9 \\ 1 \end{bmatrix} \begin{bmatrix} \cos 45 & \sin 45 & 0 \\ -\sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.7 \times 2.1 + 0.7 \times 4.9 \\ -0.7 \times 2.1 + 3.5 \times 4.9 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \\ 1 \end{bmatrix}$$

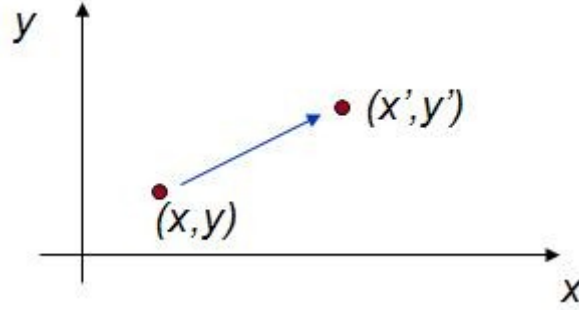
Yukarıdaki işlemler sonucunda elde edilen sayıların kusurlu kısımlarının yuvarlanmış halleri görülmektedir. Bu kusur sin45 ve cos45 değerlerini 0.7 almaktan kaynaklanmaktadır aslında değer daha hassas alınırsa hata miktarı azalır.

SORU-15: Homojen Koordinatlarla Şekil Değiştirme hakkında bilgi veriniz.

Bilgisayar grafiklerinde bir şeklin değiştirilmesi aşağıdaki işlemlere indirgenebilir:

- Taşıma (Translation)
- Döndürme (Rotation)
- Ölçekleme (Scaling)

Bu işlemlerin hepsi kendisine özgü masfuf (Matris) işlemleri olarak görülebilir. Örneğin [taşırma işlemleri](#) 2 boyutlu kartezyen uzaydan (cartesian space) bir noktanın verilen x ve y değışimini kadar taşınması demektir.



Yukarıdaki şekilde verilen x,y değerlerine sahip nokta x',y' koordinatlarına taşınmıştır. Bu işlemleri

$$x' = tx + x$$

$$y' = ty + y$$

şeklinde yazarsak tx,ty değerleri noktanın taşınma miktarı olarak düşünülebilir.

Bu [taşırma işlemleri](#) [homojen koordinat](#) kullanarak yapmamız da mümkündür. Bunun için [homojen koordinatlarda \(homogenous coordinates\)](#) bir dönüşüm masfufu (transformation matrix) oluşturmamız gerekir.

Temel olarak dönüşüm matrislerindeki amaç noktaya çarpan olarak gelebilmesidir. Yani sonuçta

$$P' = T \times P$$

şeklinde bütün T dönüşüm matrislerini orjinal noktamız olan P ile çarparak P' noktasını elde etmek isteriz.

Yukarıdaki taşırma işleminde bu durum toplama olarak yapılmaktadır. YAnı

$$P' = T + P$$

şeklinde formülize edilmiştir. Öyleyse yukarıdaki bu [tx,ty] matrisini homojen koordinatlara uygun hale getirmemiz gerekir.

$$\begin{pmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{pmatrix}$$

Yukarıdaki bu taşıma matrisi, homojen koordinatlara sahip bir dönüşüm matrisi olarak görülmektedir. Yukarıdaki matrisi P noktası ile çarparsak elde edeceğimiz sonuç formülü de aslında toplama işlemidir.

Yukarıdaki bu işlemi

$$P' = T \times P$$

olarak formülize edip çarpacak olursak:

$$x = x + tx$$

$$y = y + ty$$

sonucuna varırız (klasik matris çarpımı ile)

Yukarıdaki taşıma işlemine benzer şekilde döndürme (Rotation) ve ölçekleme (scaling) işlemleri de matris çarpımı şeklinde yapılabilir. Örneğin döndürme işlemi için aşağıdaki matris çarpımında kullanılabilir:

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

Yukarıdaki matris de bir önceki taşıma matrisine benzer şekilde çarpan olarak düşünülebilir ve

$$P' = T \times P$$

formülündeki T matrisi olarak düşünülebilir. Bu formülde ilgili işlemler yapılırsa

$$x = x \cos \theta - y \sin \theta$$

$$y = x \sin \theta + y \cos \theta$$

sonucu döndürme (rotation) konusunda anlattığımız şekilde beklentimiz doğrultusunda bulunmuş olunur.

Benzer şekilde ölçekleme matrisi olarak da

$$\begin{pmatrix} \underline{SX} & 0 & 0 \\ 0 & \underline{SY} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

yukarıdaki matris kullanılabilir.

Örneğin yukarıdaki ölçekleme matrisi için

$$P' = T \times P$$

işlemini gerçekleştirecek olursak P' matrisi için aşağıdaki sonuç çıkar:

$$x = SX \cdot x$$

$$y = SY \cdot y$$

Dolayısıyla beklentimiz doğrultusunda ölçekleme (scaling) konusunda anlattığımız dönüşüme uygun bir sonuç elde etmiş oluruz.

SORU-16: OpenGL İsim Dizisi hakkında bilgi veriniz.

OpenGL programlaması sırasında ekrandaki bir nesnenin seçilmesi mümkündür. Bu seçme işlemi sırasında sanki ekranda verilen bir (x,y) koordinatından (ekran 2 boyutlu olduğu için görüntüler 3 boyutluda olsa sonuçları 2 boyutludur (detayı için izdüşüm (projection) konusunu okuyunuz) dolayısıyla sadece x ve y koordinatlarından) ekranda bir ışın fırlatılmış gibi, ışının geçtiği bütün nesnelerin isimlerini almaya yarar.

Dolayısıyla ekrandaki seçilen x,y koordinatında görüntülen bütün nesnelerin isimlerini alan bir isim yığını (stack) OpenGL'de tanımlıdır. Bu yığının sıfırlanması ve ilk değerini alması aşağıdaki şekilde mümkündür:

```
glInitNames();
glPushName(0);

glPushMatrix(); // Görüntünün değişmemesi
// için çizime başlanmadan önceki dönüşümler
// kaydediliyor

// ilk nesneye 1 ismi veriliyor ve ilk
// nesne ekrana çiziliyor
glLoadName(1);
ilkNesneyiCiz();
// ikinci nesne benzer şekilde çiziliyor
glLoadName(2);
ikinciNesneCiz();
// son nesneyi çiz
glLoadName(3);
ucuncuNesneCiz();
// goruntu donusumune geri donuyoruz
glPopMatrix();
```


Yukarıda yorumlanmış olarak verilen kodu kullanarak çizim yapmanız ve çizilen nesnelerin isimlerini yığına (stack) koymanız mümkündür.

SORU-17: OpenGL Nesne Seçimi (Object Picking) hakkında bilgi veriniz.

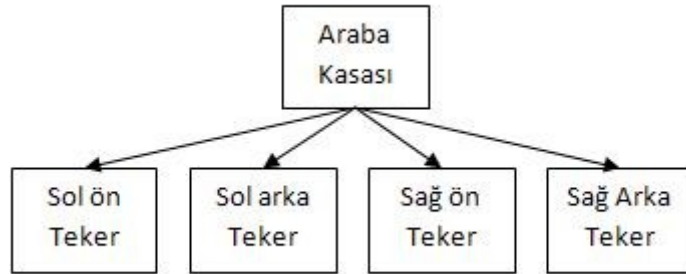
Bu yazının amacı bilgisayar grafiklerinde kullanılan OpenGL kütüphanesi marifetiyle ekranda bulunna 3 boyutlu nesnelerden birisinin fare ile tıklandığında nasıl algılandığını anlatmaktır.

Ekrandan seçme işlemi yapılabilmesi için aşağıdaki 6 adımın izlenmesi gerekir:

1. Öncelikle ileride kullanılmak üzere seçilen nesneleri tutan ve glSelectBuffer isimli fonksiyon ile ilk değerini alacak olan bir dizinin tanımı
2. Seçme işleminin aktif olabilmesi için GL_SELECT mod'unu glRenderMode() fonksiyonuna parametre vermek.
3. Seçilmiş nesneleri tutan yığının (Stack) ilklenmesi için glInitNames ve glPushName fonksiyonlarının çağırılması
4. Seçme işlemi sırasında kullanılacak olan görüntü alanının (viewing Volume) belirlenmesi. Bu görüntü alanı çizim sırasında kullanılan alandan farklıdır.
5. Çizim işlemlerinin yapılması ve bu sırada çizilen nesnelere isin verilmesi
6. Seçme durumundan çıkarak seçilmiş nesnenin işlenmesi

SORU-18: OpenGL ile Bağlı Hareket hakkında bilgi veriniz.

Bilgisayar grafiklerinde bir nesnenin diğer nesnelere bağlı olarak hareket etmesi mümkündür. Bunu sağlamak için bir varlık ağaç yapısı üzerinden bütün nesneler birbirine bağlanır. Ardından bağlı nesneleri çağıran bir dolaşma (traverse) fonksiyonu özyineli (recursive) olarak çalışır ve bütün nesneleri doğru sıra ile ve doğru şekil dönüşümleri ile (transformation) çizer.



Bu ağaç yapısının tanımlanması aslında C dilindeki yapılar (structs) ile mümkündür. Ağacı oluşturan her düğümün (node) yapısı aşağıda verilmiştir:

Düğüm Yapısı:

```
typedef struct treenode
{
    GLfloat m[16];
    void (*f)();
    struct treenode *sibling;
    struct treenode *child;
} treenode;
```

Yukarıdaki kodda bir şekli oluşturan veri bilgisi tanımlanmıştır. Bu veri ünitesinde bir dönüşüm matrisi (m) bir fonksiyon göstericisi (function pointer) ve komşu ve çocuklarını gösteren birer gösterici (pointer) bulunmaktadır.

Atama işlemi:

Aşağıdaki kod örneğinde yukarıda tanımlanmış olan yapının (struct) içerisine ilgili değerler atanmıştır (örnek olarak)

```
treenode torso_node, head_node, lua_node, ... ;
/* use OpenGL functions to form transformation matrices */
glLoadIdentity();
glRotatef(theta[0], 0.0, 1.0, 0.0);
/* move modelview matrix to m */
glGetFloatv(GL_MODELVIEW_MATRIX, torso_node.m)
torso_node.f = torso_draw; /* torso_draw() draws torso */
torso_node.sibling = NULL;
torso_node.child = &head_node;
```

Dolaşma Fonksiyonu:

Son olarak yukarıda tanımlanan ve içerisine değer atanan yapıyı dolaşan ve dolaşırken ekrana basan fonksiyon aşağıda verilmiştir:

```
void traverse(treenode *root)
{
if(root == NULL) return;
glPushMatrix();
glMultMatrix(root->m);
root->f();
if(root->child != NULL)
traverse(root->child);
glPopMatrix();
if(root->sibling != NULL)
traverse(root->sibling);
}
```

SORU-19: OpenGL ile Malzeme Özellikleri (Material Properties) hakkında bilgi veriniz.

Bilgisayar grafiklerinde bir nesnenin görüntüsünün oluşmasını belirleyen iki temel unsurdan birisi ışık kaynağı iken diğeri de görüntülenecek olan nesnenin ışığı yansıtma özelliğidir.

OpenGL içinde bu özelliği belirlemeye yarayan glMaterial fonksiyonu bulunmaktadır ve alabileceği parametreler aşağıda listelenmiştir:

Parametre ismi	İlk Değeri	Açıklama
GL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	Malzemenin eşit yansıma (ambient)

		renği
GL_DIFFUSE	(0.8, 0.8, 0.8, 1.0)	Malzemenin yayılma (diffuse) rengi
GL_AMBIENT_AND_DIFFUSE		Malzemenin hem yansıma hem yayılma (ambient and diffuse) rengi
GL_SPECULAR	(0.0, 0.0, 0.0, 1.0)	Malzemenin yönlendirilmiş ışık yansıma rengi (specular)
GL_SHININESS	0.0	Malzeme üzerindeki parlama oranıdır. Bu yönlendirilmiş ışıklarda olabilir (specular)
GL_EMISSION	(0.0, 0.0, 0.0, 1.0)	Malzemenin yaydığı renktir.
GL_COLOR_INDEXES	(0,1,1)	Malzemenin yansıtacağı ışık renkleridir. (ambient, diffuse, specular parametreleri)

Örneğin aşağıda bu fonksiyonun kullanımı gösterilmiştir:

```
float[] mat_ambient = {0.7f, 0.7f, 0.7f, 1.0f};
```

```
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
```

Yukarıdaki kod ile şeklin sadece ön yüzünün yayılma tipi aydınlatma yapacağı ve ışığın renginin gri tonda (0.7,0.7,0.7) olacağı belirtilmiştir.

SORU-20: OpenGL ile Aydınlatma (Lighting) hakkında bilgi veriniz.

Bilgisayar grafiklerinde bir nesnenin görülmesini ve görüntülenirkenki aydınlığını iki unsur belirler. Bunlardan birincisi ortamın aydınlık miktarı ve aydınlatılma şekli iken ikincisi nesnenin malzemesinin ışığı yansıtma oranı ve şeklidir.

OpenGL programlama ortamında bu iki özellik için de fonksiyonlar bulunur. Aşağıda OpenGL’de tanımlı ışıklandırma seçenekleri verilmiştir:

Parametre ismi	İlk değeri	Anlamı
GL_LIGHT_MODEL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	Ambient (yansıma) ışığıdır ve değerler RGBA olarak verilir.
GL_LIGHT_MODEL_LOCAL_VIEWER	0.0 veya GL_FALSE	Yönlendirilmiş ışık (specular reflection) açısının nasıl hesaplandığı verilir
GL_LIGHT_MODEL_TWO_SIDE	0.0 veya GL_FALSE	Tek yüzlü veya çift yüzlü aydınlatma seçeneklerinden birisini seçmeye yarar

Yukarıdaki bu seçenekleri glLightModel fonksiyonuna parametre olarak vermek, ilgili parametrenin değerini atamak için yeterlidir. Örneğin:

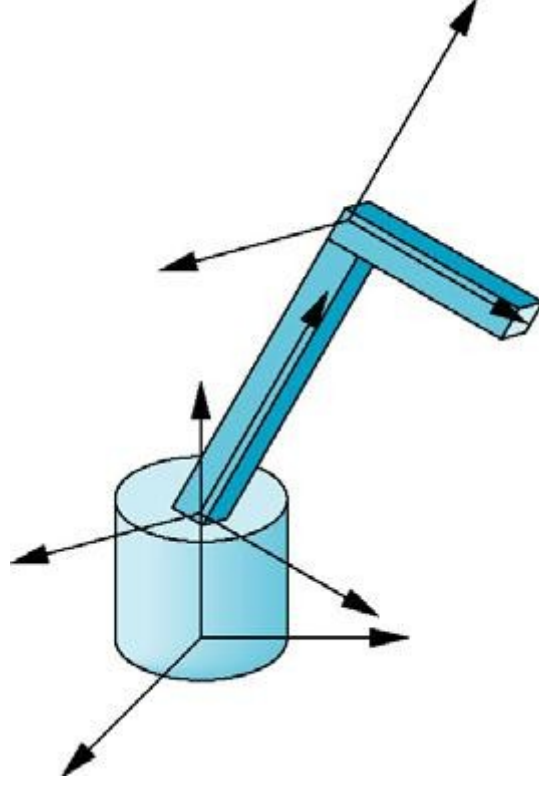
```
GLfloat ambient_light[] = {1.0, 0.3, 0.45, 1.0};
```

```
glLightModel(GL_LIGHT_MODEL_AMBIENT,ambient_light);
```

Yukarıdaki fonksiyon ile ortamı, yansıma ışığı (her tarafın eşit aydınlatıldığı ışık kaynağıdır, ambient) kırmızı ton ağırlıklı olarak aydınlatacaktır.

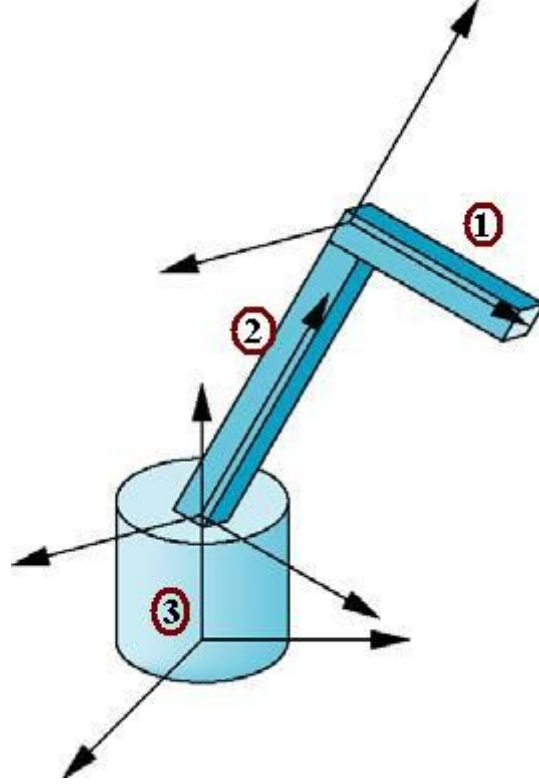
SORU-21: Mafsallı Tasarım (Articular Design) hakkında bilgi veriniz.

Bilgisayar grafiklerinde birbirine bağlı nesneleri modellemek için kullanılan bir yöntemdir. Buna göre birbirine bir mafsalla bağlı olan nesnelerin şekil değiştirme (transformation) matrisleri arasında bir bağlantı kurulmaya çalışılır:



Örneğin yukarıda tasvir edilen robot kolunun 3 ayrı parçası ve her parçayı bağlayan 2 ayrı mafsalı (joint) bulunmaktadır.

Yukarıdaki şekildeki her 3 nesnede ayrı ayrı hareket etmekte ancak bir nesnenin hareketi diğerlerin bağlamaktadır. Örneğin en altta bulunan silindirin dönmesi durumunda robot kolu oluşturan diğer parçalar da dönecektir. Dolayısıyla bir nesne bağlı olduğu bir üst nesnenin şekil değiştirme matrisini (transformation matrix) çarpan olarak içermektedir. Örneğin şekli aşağıdaki şekilde numaralandıracak olursak:



1 numaralı nesne, 2 numaralı nesnenin hareketlerine ve 2 numaralı nesne de 3 numaralı nesnenin hareketlerine bağlıdır. Ayrıca problemi basitleştirmek için 3 numaralı silindirin sadece kendi etrafında dönebildiğini 2 ve 1 numaralı nesnelerin ise sadece kesişim noktasında aşağı yukarı hareket edebildiğini kabul edelim. Yani her nesnenin sadece tek bir dönüşüm (transformation) matrisi bulunsun.

Bu durumda

$T_3 = R_3$, yani 3 numaralı nesnenin dönüşüm matrisi (transformation) sadece yaptığı dönüş (rotation) miktarı kadardır.

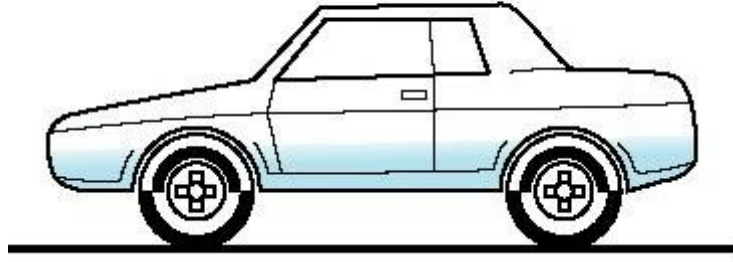
$T_2 = R_3 R_2$, yani 2 numaralı nesnenin dönüşüm matrisi, yapmış olduğu dönüş ile bağlı bulunduğu nesnenin yapmış olduğu dönüşün çarpımı kadardır.

$T_1 = R_3 R_2 R_1$ yani 1 numaralı nesnenin dönüşüm matrisi kendisinin bağlı bulunduğu bütün nesnelerin dönüşüm matrislerinin çarpımı kadardır.

SORU-22: Varlık-Ağaç Modelleme (Tree Model) hakkında bilgi veriniz.

Bilgisayar grafiklerinde nesneleri modellemek için kullanılan yapılardan birisidir. Buna göre modelleme işlemi için bir veri ağacı kullanılır ve bu veri ağacının her üyesi bir nesneden oluşur. Bu sayede nesneler arasındaki bağlantı tutulabileceği gibi nesneler gruplanarak daha üst nesne tanımları elde edilmiş olur.

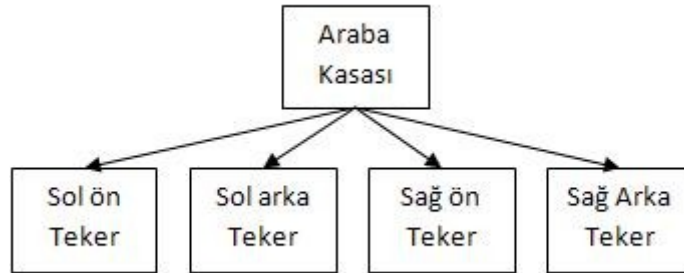
Örneğin aşağıdaki araba modelini ele alalım:



Yukarıdaki ara birden fazla nesnenin birleşmesinden oluşmaktadır. Olayı basitleştirmek için 4 tekerlek ve bir araba kasasından (şasi) oluştuğunu düşünelim. bu durumda aynı tekerlek nesnesi 4 farklı konumda ve 4 farklı şekil değiştirmeye (transformation) tabi tutularak modellenecektir.

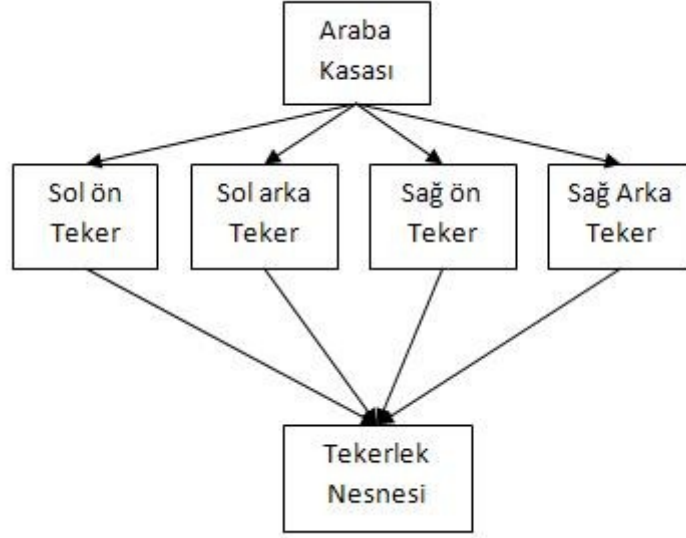
Bu modelleme sırasında kullanılabilir yöntemlerden birisi varlık-durum tablosudur (symbol instance table) ancak bu yöntemde şekiller arasında bir eşgüdüm (senkronizasyon, synchronisation) elde edilmesi oldukça güçtür. Örneğin arabanın hareket etmesi tekerlerin döndüğü miktara bağlıdır. Dolayısıyla araba kavramı ile teker arasındaki bağlantı tablo üzerinde ifade edilememektedir.

Çözüm olarak bir nesneyi ve bu nesneyi meydana getiren alt nesneleri aşağıdakine benzer bir ağaç yapısı içerisinde tutmak mümkündür:



Yukarıdaki modelleme sayesinde nesneyi oluşturan alt nesneler ile bir ilişki kurulmuştur. Yapısı itibari ile yönlü düz ağaçlardır (Directed Acyclic Graph).

Yukarıdaki bu yöntem kullanıldığında gösterici (pointer) yapısının da sağlamış olduğu imkanlar ile aslında aşağıdakine benzer bir veri modellemesi elde edilebilir:

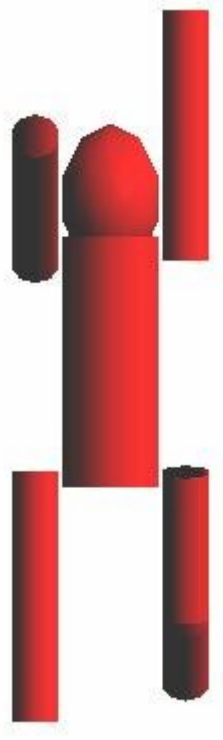


Yukarıdaki yapıda bir nesnenin farklı ortamlarda kullanılmasına karşılık her kullanım durumu aynı nesneyi işaret etmektedir

SORU-23: Varlık-Durum Tablosu (Symbol Instance Table) hakkında bilgi veriniz.

Bilgisayar grafiklerinde nesnelerin ekrana basılması anlık bir olaydır. Her anlık durumda basılan nesnenin durumu da farklı olabilir. Örneğin ekranda hareket eden bir küre her tazelemede farklı bir konumda görülecektir. İşte her varlığın anlık olarak durumunun durduğu tabloya varlık durum tablosu (Symbol-instance table) ismi verilir.

Temel olarak 3 çeşit şekil değiştirme (transformation) bulunmaktadır. Bunlar taşıma (translation), döndürme (rotation) ve ölçekleme (scaling, büyütme küçültme) değişimleridir. İşte birden fazla şekilden oluşan bir ortamda her şeklin anlık durumunu tutan tabloda bu 3 bilgi yeterlidir:



örneğin yukarıdaki temsili robot, 5 farklı silindir ve 1 küreden oluşmaktadır. Bu silindirlerin birbirinden tek farkı bulundukları konum, boyutları ve döndürme oranlarıdır. Şayet bu verileri bir tabloda tutmak istersek aşağıdakine benzer bir tablo çıkar:

Varlık	Ölçek (Scale)	Döndürme (Rotate)	Taşıma (Translate)
1	s_x, s_y, s_z	$\Theta_x, \Theta_y, \Theta_z$	d_x, d_y, d_z
2			
3			
4			
...			

SORU-24: Malzeme Rengi (Material Color) hakkında bilgi veriniz.

Bilgisayar grafiklerinde bir malzemenin renginin belirlenmesi bu malzeme üzerine düşen ışıktan ne kadar kırmızı yeşil ve mavi (red, green, blue) kodlarının yansıdığıdır.

Örneğin bir kırmızı küp üzerine beyaz ışık düştüğünü ve küpün bütün yeşil ve mavi renkleri emdiğini düşünelim. Bu durumda küp tam kırmızı renkte görüntülenecektir. Benzer şekilde kırmızı küpü kırmızı ışık ile aydınlatacak olursak küp yine kırmızı görülür. Ancak küpü yeşil veya mavi ışık ile aydınlatacak olursak küpten yayılan ışık bulunmayacak ve neticede küp siyah olarak görüntülenecektir.

Bilgisayar grafiklerinde kullanılan ışık kaynakları (light sources) gibi malzemelerinde çevresel (Ambient), dağılımsal (Diffuse) ve akis (specular) yansımaları olmaktadır. Basitçe yansıyan ışığın çeşitlerini ışık kaynağı çeşitleri olarak görmek mümkündür.

Çevresel yansımada ışık her yana eşit dağılırken dağılımsal (diffuse) ışık yansımalarında ışığın geldiği kaynak daha parlak görülmektedir. Örneğin kırmızı bir top üzerine beyaz ışık yansıması durumunda ışığın yansıdığı noktalar beyaz olarak görülmektedir.



Örneğin yukarıda bir beyaz top üzerine beyaz ışık düşmüş ve ışığın geldiği yönün göze yansıması parlak, ışığın düşmediği noktalar ise koyu görülmektedir.

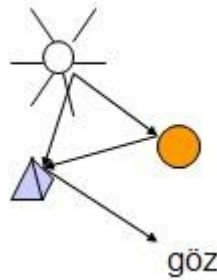
SORU-25: Işık Kaynakları (Light Sources) hakkında bilgi veriniz.

Bilgisayar grafiklerinde şekillerin aydınlatılması için kullanılan çeşitli ışık kaynakları bulunur. Temel olarak 3 çeşit ışık kaynağından bahsedilebilir. Bunlar çevresel (ambient), dağılımsal (diffuse), akis (specular) ışık kaynaklarıdır.

Kısaca çevresel ışık kaynağında ışığın yönünün belirlenmesi imkansızdır. Işık her yönden eşit geliyormuş gibi algılanır. Örneğin bir odadaki arkaplan ışığı vurulması böyle bir aydınlatmadır.

dağılımsal ışık kaynağında ise ışığın bir yönü ve dağılım oranı bulunmaktadır. Buna göre ışık kaynağına yakınlığa ve ışık vektörünün merkezine uzaklığa göre (ışığın vurduğu yöne ne kadar yakın olduğuna göre) aydınlanma oranı değişir.

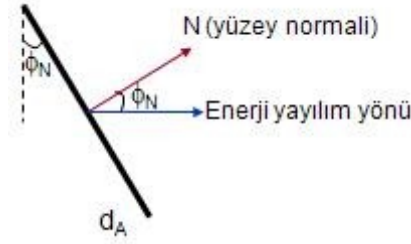
Akis ışık kaynağı (yansıma ışığı) ise bir çizimden aks eden ışık ile bir nesnenin aydınlanmasıdır. Bu durum aşağıdaki örnekte gösterilmiştir:



Göz tarafından normalde görülmeyecek olan yüzey, başka bir yüzeyden gelen yansıma sayesinde görülür olmuştur.

SORU-26: Lambert kosinüs teoremi (Lambert's cosine theorem) hakkında bilgi veriniz.

Lamberte göre eksenle Φ açısı yapan bir düzlemin üzerinden yayılan enerji miktarı düzlemin normal vektörü ile yapılan Φ açısının kosinüsüne eşittir.



Yukarıdaki şekilde bu durum gösterilmiştir. Bu duruma ilave olarak yayılım yönü bakan kişinin bakış açısı olarak da yorumlanabilir. Yani yüzeyin yaydığı enerjinin bakan kişiye doğru giden miktarı $\cos(\Phi)$ ile orantılıdır.

Yayılma yoğunluğu (intensity) = birim zamandaki yayılma enerjisi / yansıma alanı

formülü ile bulunabilir. Buna göre yukarıdaki şekildeki yayılma yoğunluğu:

$\alpha \cos(\Phi) / (d_A \cos(\Phi))$ olarak hesaplanabilir.

SORU-27: OpenGL ile sis (FOG) hakkında bilgi veriniz.

Bilgisayar grafiklerinde uzakta olan objelerin, uzaklık hissini arttırılması için puslu gösterilmesi mümkündür. Buradaki amaç şekillerin renklerinin daha gri tonlara yakın gösterilmesi ve bu sayede uzaklık hissini oluşturulmasıdır.

Aynı zamanda dumanlı ortamların modellenmesi için de kullanılabilir. Sis etkisi oluşturmak için aşağıdaki OpenGL kodlarından faydalanılabilir:

```
glEnable(GL_FOG);
{
    GLfloat fogColor[4] = {0.5, 0.5, 0.5, 1.0};

    fogMode = GL_EXP;
    glFogi(GL_FOG_MODE, fogMode);
    glFogfv(GL_FOG_COLOR, fogColor);
    glFogf(GL_FOG_DENSITY, 0.35);
    glFogf(GL_FOG_START, 1.0);
    glFogf(GL_FOG_END, 5.0);
}
```

Yukarıdaki koddaki komutlar sırasıyla aşağıda açıklanmıştır. Yukarıdaki kod bir şekil değiştirme (transformation) gibi şekil basılmadan önce uygulanabilir.

```
glEnable(GL_FOG);
```

Fog effecti komutlarını aktifleştirir.

```
GLfloat fogColor[4] = {0.5, 0.5, 0.5, 1.0};
glFogfv(GL_FOG_COLOR, fogColor);
```

Oluşturulacak olan sisin (dumanın, pusun) rengi ve alfa çarpanı. (yukarıdaki örnekte gri renktedir. Örneğin kırmızı renk için 1.0,0.0,0.0,1.0 parametreleri verilmelidir)

```
glFogfv (GL_FOG_COLOR, GL_EXP);
```

Kullanılacak olan sis'in şekiller üzerine nasıl uygulanacağını belirleyen matematiksel fonksiyondur. 3 farklı seçenekten birisi uygulanabilir : GL_LINEAR, GL_EXP veya GL_EXP2. Bu seçenekler aşağıda açıklanmıştır:

GL_LINEAR için kullanılan fonksiyon $f = (end - z) / (end - start)$

GL_EXP için kullanılan fonksiyon $f = e^{-(density \cdot z)}$

GL_EXP2 için kullanılan fonksiyon $f = e^{-(density \cdot z)^2}$

yukarıdaki formüllerde end: sis'in bitişi , start: sis'in başlangıcı, e evrensel logaritma kökü, density: sis'in yoğunluğu, dot (dot product yani skalar çarpım), z ise objenin kameraya olan uzaklığını ifade etmektedir.

```
glFogf (GL_FOG_DENSITY, 0.35);
```

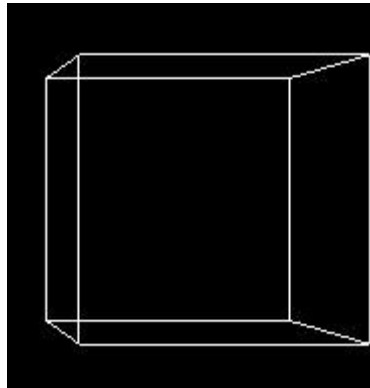
Uygulancak olan sisin yoğunluğu (ne kadar çok yoğun olursa şekiller o kadar az görülür.) sayı float tipindedir yani 0 ile 1 arasında değerler alabilir.

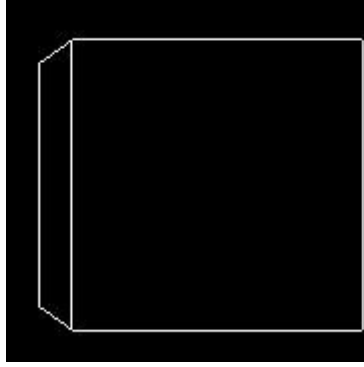
```
glFogf (GL_FOG_START, 1.0);  
glFogf (GL_FOG_END, 5.0);
```

Sis'in kameraya göre ne kadar uzaklıkta tanımlı olduğudur. Yani yukarıdaki satırlardan sonra sis 1 float birim uzaktan başlayarak 5 float birim uzağa kadar etkili olacaktır.

SORU-27: OpenGL ile Arkayüz (Opengl Backface) hakkında bilgi veriniz.

Bu yazının amacı OpenGL ile arkaplanda olan şekillerin gösterilmesi ve gizlenmesidir. Şekillerin arkaplan görüntüsünün gizlenmesinin ve gösterilmesi aşağıda gösterilmiştir.





İki şekil arasındaki fark birinci şekilde arkaplanda görüntülenen yüzlerin, ikinci şekilde görüntülenmemesidir. Bu işlemi yapan temel fark aşağıda verilen parametrelerdir.

Öncelikle gösterilecek olan polygon mod'unu ayarlamamız gerekiyor:

```
GLPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
```

Yukarıdaki satır sayesinde gösterilecek olan bilgi hem ön hem de arka yüzleri içermektedir (FRONT BACK)

İstenirse GL_FRONT (Sadece ön yüzleri göster) yada GL_BACK (Sadece arka yüzleri göster) seçenekleri seçilebilir.

Bu seçimin ardından CULLING (seçip alma) işleminin yapılması gerekir. Bu işlem sayesinde ayarlanmış olduğumuz moda uygun olarak sadece öndeki veya sadece arkadaki yüzeyler görüntülenir.

Yukarıdaki ilke kübü basmak için

```
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);  
glutWireCube (1.0);
```

ikinci kübü basmak için de:

```
glEnable(GL_CULL_FACE);  
glPolygonMode(GL_FRONT, GL_LINE);  
glutWireCube (1.0);  
glDisable(GL_CULL_FACE);
```

Satırları kullanılabilir

SORU-28: Derinlik Vurgusu (Depth Cueing) hakkında bilgi veriniz.

Bilgisayar grafiklerinde derinlik hissinin oluşturulması amacıya uzaktaki imgeciklerin (pixel) donuklaştırılmasını hedefleyen yöntemdir.

kabaca en uzak imgeciğin (pixel) 1.0 ve en yakın imgeciğin 0.0 değeri alabileceğini düşünürsek:

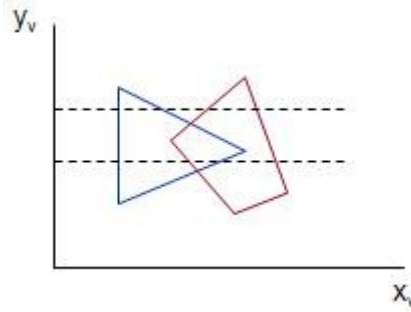
$$f_{\text{derinlik}}(m) = (d_{\text{azami}} - d) / (d_{\text{azami}} - d_{\text{asgari}})$$

Yukarıda verilen bu derinlik fonksiyonu verilen mesafe için bir oran elde etmektedir.

Bu fonksiyondan elde edilen katsayı ile imgeciğin değeri çarpılmaktadır.

SORU-29: Hat Tarama Algoritması (ScanLine Algorithm) hakkında bilgi veriniz.

Bilgisayar grafiklerinde görünen yüzeylerin belirlenmesi için kullanılan algoritmanın ismidir. Scan Line Rendering olarak da isimlendirilmektedir.



Yukarıdaki şekilde de görüldüğü üzere iki doğru taraması yapılmış ve bu doğruların hangi yüzeyleri gördüğü nokta nokta olarak kodlanmıştır.

Algoritma aşağıdaki adımlardan oluşmaktadır:

1. Yatay hatlarda tarama yapıldığı için x değeri arttırılarak görüntülenmesi gereken kenarları içeren bir liste hazırlanır (verilen tarama doğrultusu için)
2. Tarama sırasındaki her yüzey için (surface) bir bayrak tanımlanarak, yüzeyin tarama hattından görülüp görülmediği belirlenir.
3. Soldan dağa doğru imgecik (pixel) konumları işlenerek görülür durumdaki yüzeyler belirlenir

SORU-30: A-Hafızalama (A-Buffer) hakkında bilgi veriniz.

Bilgisayar grafiklerinde birden fazla yüzün aynı nokta üzerine görüntüsünün düşmesi (projection) durumunda hangi yüzün görüntüleneceğine karar vermekte kullanılan yöntemdir. Derinlik hafızalama yönteminin şeffaf yüzleri göstermedeki yetersizliği üzerine geliştirilmiştir. Buradaki amaç bir noktadan bakıldığında birden fazla yüzün görünmesi halinde nasıl bir görüntü oluşturulacağını belirlemektir.

A-Hafızalama yöntemi veri yapısı olarak bağlı liste (linked list) kullanır. Listenin her düğümünde (node) aşağıdaki şekilde 2 bilgi tutulur :

derinlik | yüzey bilgisi

Bu düğümler birbirine bağlanırlar ve sonuçta yukarıdaki veri aşağıdaki bilgiler içermiş olur:

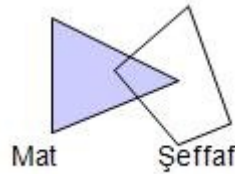
Derinlik :

- < 0 : birden fazla yüz görüntüleniyor demektir.
- ≥ 0 : tek yüz görüntüleniyor demektir.

Yüzey Bilgisi:

- Renk kodu (RGB yoğunluğu)
- Matlık (opacity)
- Derinlik (depth)
- Alan kaplama oranı (percent of area coverage)
- Yüzey tanımlayıcısı (diğer yüzeylerden ayırt eden özel bir sayı yada dizgi (surface identifier))

Yukarıdaki bu iki bilgiyi tutan bağılı liste sayesinde örneğin aşağıdaki şekilde gösterilen iki yüzey aynı anda görüntülenebilmektedir :

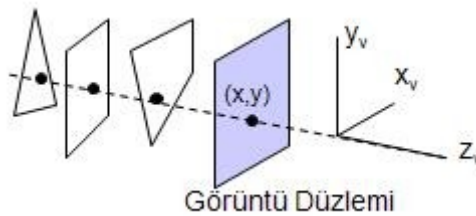


Yukarıdaki şekillerde kesişim noktasında şeffaf olan ve yakın olan şeklin arkadaki şekli engellememesi beklenmektedir. Bu durumda bu noktada iki şeklin bilgisi de tutulmalıdır.

SORU-31: Derinlik Hafızalama (Depth Buffering , z-buffer) hakkında bilgi veriniz.

Bilgisayar Grafiklerinde bir görüntünün oluşturulması sırasında aynı noktaya düşen birden fazla yüz (polygon) bulunuyorsa bu yüzlerden hangisinin nasıl görüntüleneceğini belirleyen hafıza yönetimidir.

Bu durumu aşağıdaki şekil üzerinden anlaşılabilir:



Şekilde bir görüntü düzlemindeki bir noktanın (x,y) oluşması için etkili olan 3 yüzey gösterilmiştir. Aslında bu yüzeylerin herbirisi ayrı ayrı ele alındığında (ortamımızda sadece bir tanesi bulunsaydı) görüntülenecekti. Ancak birden fazla yüzey olduğu için en yakındakinin görüntülenmesi beklenir.

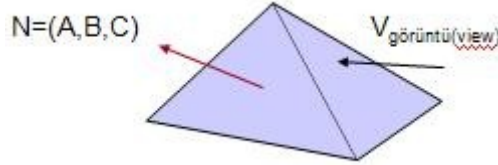
Bu problem bütün yüzeylerin, görüntü düzlemi üzerindeki iz düşümü alınarak en yakındakinin gösterilmesi şeklinde çözülür. Yani hafızada her yüzey için bir iz düşüm hesabı yapılır ardından z değeri (yani derinlik değeri) görüntüye en yakın olan görüntü alınır. z-hafızalama (z-buffer) ismi de buradan gelmektedir.

Alternatifi olan A-Hafızalama (A-Buffer)'dan en büyük farkı A-Hafızalamada kullanılan bağılı liste (linked list) yapısının tersine en yakındaki noktayı tutmasıdır. Derinlik hafızalamadaki bu durum, şayet saydamlık (transparency) kullanılıyorsa problem oluşturmaktadır.

SORU-32: Arka Yüz Algılama (Back Face Detection) hakkında bilgi veriniz.

Bilgisayar grafiklerinde bir şeklin herhangi bir yüzünün görüntülenip görüntülenmeyeceğine karar verilmesi işlemidir.

Aşağıdaki şekli ele alalım:



Yukarıdaki şekilde Görüntü vektörü (V_{view}) ve piramiti oluşturan yüzlerden birisinin Normal vektörü verilmiştir. Bir yüzün (polygon) görüntülenip görüntülenmemesi:

$Ax + By + Cz < 0$ veya $V \cdot N > 0$ formülü sağlanıyorsa bu yüz baktığımız yönün tersine çevrilmiş yani baktığımız yönde böyle bir yüz yok demektir.

Soru:33: OpenGL ile Perspektif hakkında bilgi veriniz.

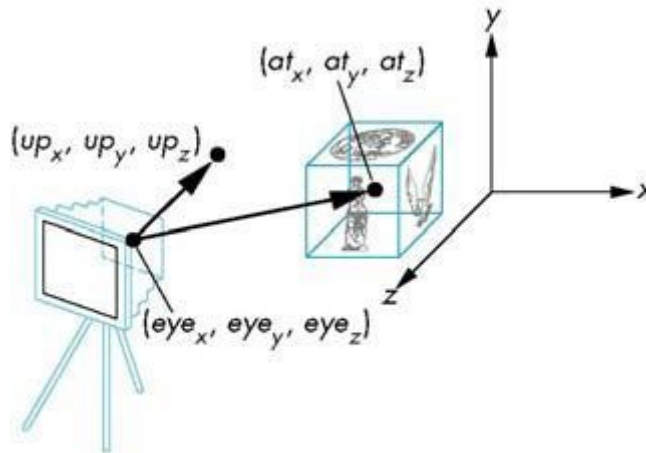
OpenGL kullanılarak perspektif değerlerinin 3 boyutlu uzayda uygulanması için aşağıdaki verilerin belirlenmesi gerekir:

- kameranın konumu
- kameranın açısı (Baktığı yön)
- perspektif şekli ve katsayıları

Yukarıdaki maddelerden ilki opengl ortamında, glLookAt fonksiyonu ile programlanmaktadır.

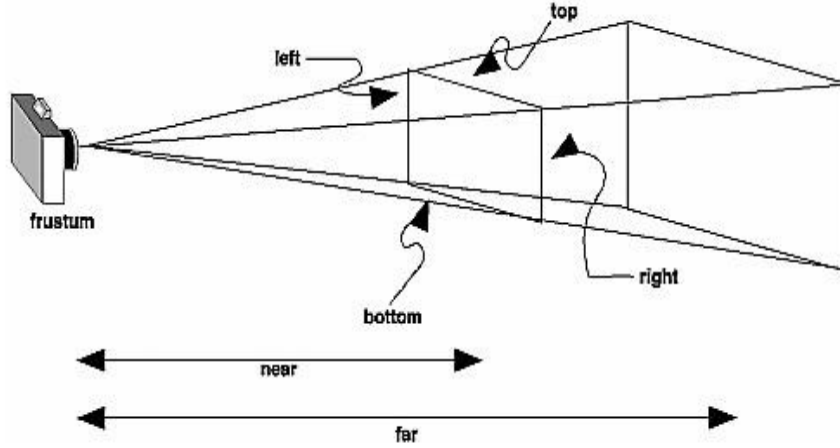
`glLookAt(eyex, eyey, eyez, atx, aty, atz, upx, upy, upz);`

Yukarıdaki değerleri verilen fonksiyonun kullanımı aşağıdaki şekilde gösterilmiştir:



Buna göre eye (x,y,z) değerleri kameranın konumunu, at(x,y,z) değerleri kameranın baktığı noktayı ve son olarak up(x,y,z) değerleri de kameranın baktığı yönle nasıl bir açı yaptığını (bu eksen etrafında ne kadar döndürüldüğünü) belirlemek için kullanılan kerteriz noktalarıdır.

Yukarıdaki maddelerden sonuncusu olan perspektifin çizim ortamında uygulanması da aşağıdaki şekilde gösterilmiştir:



Yukarıdaki şekilde, solda duran bir kameranın, sağdaki bir çizim ortamını nasıl görüntülediği tasvir edilmiştir. Burada 6 adet değer perspektifin hesaplanması ve kamera görüntüsünün oluşturulmasında önemli rol oynar:

- near (yakın perspektif ekranının uzaklığı)
- far (uzak perspektif ekranının uzaklığı)
- up (üst sınır)
- down (alt sınır)
- left (sol sınır)
- right (sağ sınır)

Yukarıda verilen bu değerlere göre bir çizim ortamının uzak ve yakın iki limit arasındaki perspektifini alıp bir kamera görüntüsü üzerine yansıtmak mümkün olabilir.

Bu işlem sırasında aşağıdaki matris (masfuf) kullanılır:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{\text{right-left}}{\text{top-bottom}} & 0 & A \\ 0 & 0 & \frac{\text{top-bottom}}{\text{top-bottom}} & B \\ 0 & 0 & 0 & C \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ D \\ -1 \end{pmatrix}$$

Yukarıdaki bu matris perspektifi hesaplarırken aşağıdaki komutu kullanmaktadır:

```
void glFrustum( GLdouble left,
                GLdouble right,
                GLdouble bottom,
```

```
GLdouble top,  
GLdouble zNear,  
GLdouble zFar)
```

Dolayısıyla perspektifin uygulanması istenen bir çizim alanında glFrustum fonksiyonu yukarıdaki parametrelerle çağırılarak 3 boyutlu uzayda iki ekran (yakın ve uzak ekranlar (clipping window)) arasındaki perspektifi hesaplamaktadır.

Bu hesaplanma sırasında uygulanacak olan perspektif ise gluPerspective fonksiyonu ile aşağıdaki şekilde verilir:

gluPerspective (theta, aspect, dnear, dfar)

Bu fonksiyondaki theta perspektifin yaptığı dönüş açısını, aspect en/boy oranını, dnear ve dfar ise sırasıyla yakın ve uzak perspektif panellerine olan uzaklığı vermektedir. gluPerspective ve glFrustum fonksiyonları aşağıdaki gibi beraber kullanılırlar:

```
glMatrixMode (GL_PROJECTION);  
glLoadIdentity ();  
gluPerspective(60.0, 1.0, 1.5, 20.0);  
glFrustum (-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);  
glMatrixMode (GL_MODELVIEW);
```

Yukarıdaki örnekte MatrixMode öncelikle projection'a ayarlanmış ardından Birim matris ile şekil değiştirme matrisi atanmış ve ilgili perspektif değerleri çizim ortamına uygulanmıştır.

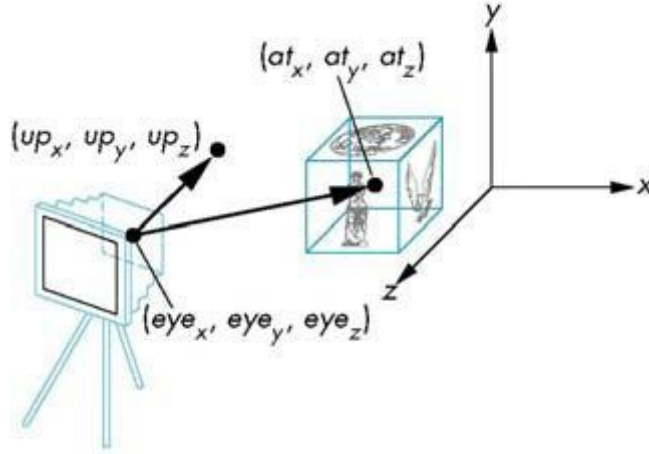
yukarıdaki kodda glFrustum fonksiyonuna verilen değerlere bakıldığında sol-sağ aralığı 1.0 ile -1.0 aralığında , üst-alt aralığı da 1.0 ile -1.0 aralığında olmaktadır. Dolayısıyla bir kare görüntü kameraya yansıtılacak ve şekillerin en/boy oranı 1'e 1 olarak görüntülenecektir. Ayrıca yakın düzlem 1.5 ve uzak düzlem 20 olarak verilmiştir. Bunun anlamı bu uzaklıklar arasında kalan şekiller gluPerspective fonksiyonunda verilen perspektif değerlerine göre yansıtılacaktır.

SORU-34: OpenGL ve Kamera Görüntüsü (Camera Viewing) hakkında bilgi veriniz.

OpenGL kütüphanesi ile uzayda istenilen bir noktaya kamerayı koymak ve bu noktadan istenilen bir yöne istenilen açı ile bakmak mümkündür. Bu işlemin 3 ögesi bulunur:

- Kameranın bulunduğu koordinatlar
- Kameranın baktığı yöndeki noktanın koordinatları
- Kameranın bu eksen üzerindeki açısı

Kısaca bu durum aşağıdaki şekilde görüntülenmiştir:



Yukarıdaki şekilde de gösterildiği üzere kamera verilen eye_x , eye_y ve eye_z koordinatlarına yerleştirilmiş ve kameranın odak çizgisi verilen at_x , at_y ve at_z koordinatlarına yönlendirilmiştir. Bu doğru üzerinde kamera istenildiği gibi döndürülebileceği için bu değeri belirlemek için kameranın bu eksenle yaptığı normal vektörü de up_x , up_y , up_z değerleri ile belirlenmiştir.

Bu kamera yerleştirme işlemi OpenGL kütüphanesinde aşağıdaki şekilde yapılabilir:

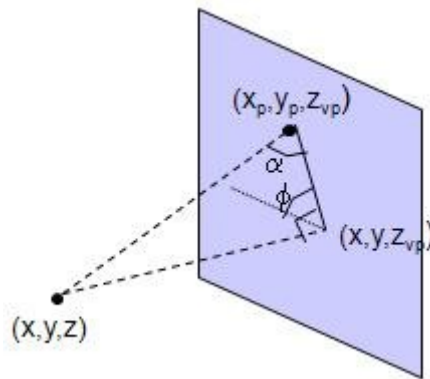
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
```

Yukarıdaki gluLookAt fonksiyonu yine yukarıdaki şekildeki nokta koordinatlarına göre şu şekilde açıklanabilir:

```
gluLookAt(eyex, eyey, eyez, atx, aty, atz, upx, upy, upz)
```

SORU-35: Şev Yansıması (Oblique Projection) hakkında bilgi veriniz.

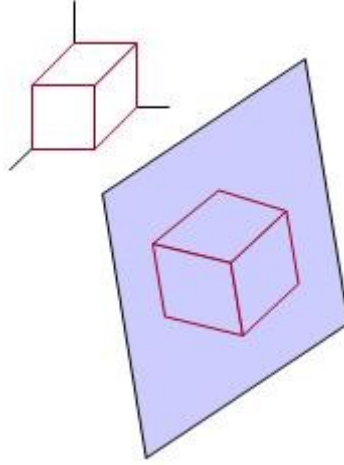
Bilgisayar grafiklerinde şev veya eğri izdüşümü olarak geçen yansıma şekline göre bir şeklin sanki bir ışık kaynağından gölgesi düşüyormuş gibi yansımadır. Buna göre şekil yansıdığı düzlem üzerine eğilmiş olduğu için bu izdüşümüne şev izdüşümü adı verilir.



Yukarıdaki şekilde (x,y,z) noktasının düzlemle açısı yapan çizginin düzlem üzerinde oluşturduğu izdüşüm gösterilmiştir. Buna göre çizginin başlangıç noktasından (x,y,z_p) bir dik çizilip, çizginin bitiş noktasından geçen ve düzleme düşen ikinci bir doğru elde edilir bu doğrunun düzlemi kestiği (x_p,y_p,z_p) noktası çizginin izdüşümünün bittiği noktadır.

SORU-36: İzometrik İzdüşüm (Isometric Projection) hakkında bilgi veriniz.

Bilgisayar grafiklerinde, bir paralel izdüşüm çeşidi olan dik izdüşümün alt çeşidi olan eksensel izdüşümün özel bir halidir. Buna göre bir izdüşüm düzleminin bütün eksenleri kesmesi durumudur:

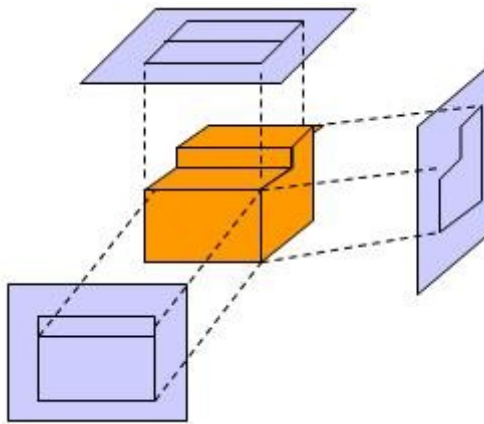


Yukarıdaki şekilde izdüşümün alındığı düzlem bütün eksenler ile kesişmiştir. Bu kesişim sonucunda eksensel izdüşümlerden farklı olarak yansıma alınır.

SORU-37: Eksensel İzdüşüm (Axonometric Projection) hakkında bilgi veriniz.

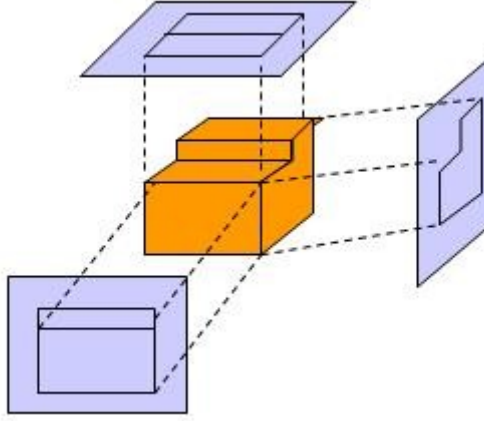
Bilgisayar grafiklerinde kullanılan bir izdüşüm yöntemidir. Paralel izdüşümün bir çeşidi olan dik izdüşümün (orthonogal projection) bir çeşididir. Buna göre bir objenin eksenlere olan yansıması alınır. Yani iz düşümün alınacağı düzlem iki eksen kesen ve bir eksene paralel olarak duran bir düzlemdir.

Şekilde de görüldüğü üzere eksenlerdeki izdüşümleri alınmıştır.



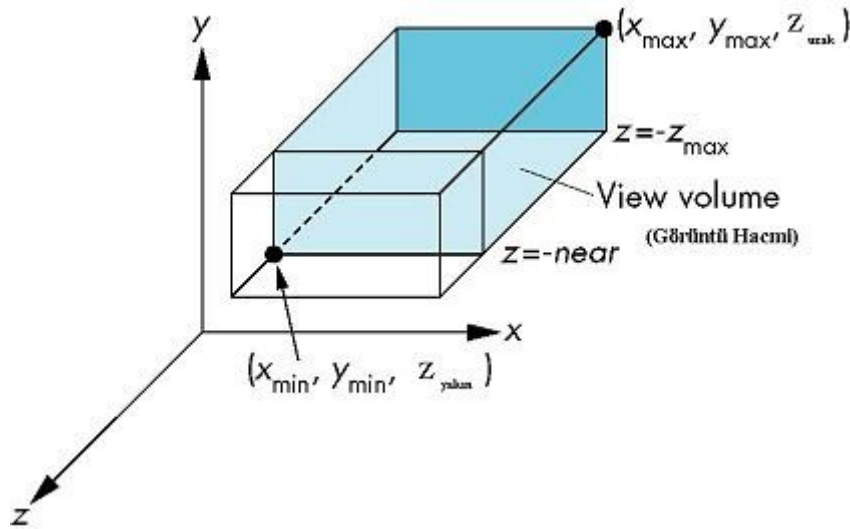
SORU-38: Dik İzdüşüm (Orthogonal, Orthographic Projection) hakkında bilgi veriniz.

Bir düzleme şeklin diklemesine olarak yansıtılmış halidir. Paralel izdüşümün (Parallel Projection) özel bir halidir. Aşağıda bu izdüşüm şekli tasvir edilmiştir:



Yukarıdaki izdüşüm şeklinde de görüldüğü üzere nesnenin eksenlere izdüşümü gösterilmiştir. Bu izdüşüm şekline eksnelere paralel izdüşüm olduğu için, dik izdüşümün bir alt çeşidi olan eksensel izdüşüm (axonometric) ismi de verilmektedir.

Dik izdüşümün OpenGL tarafından hesaplanması için içerisinde hazır bir fonksiyon bulunur. Bu fonksiyonu daha iyi anlayabilmek için aşağıdaki şekli inceleyelim:



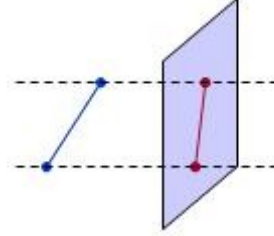
Şekilde görüldüğü üzere iki dikdörtgenin (uzak ve yakın) izdüşümleri bir düzlem üzerinde alınmıştır. İki Z değeri arasında fark olmasına karşılık (yakın ve uzak) izdüşüm aynı olmaktadır. Bunun sebebi dik izdüşümün bir paralel izdüşüm türü olması ve uzaklıkla bir büyüme veya küçülme olmamasıdır.

Bu izdüşüm OpenGL kütüphanesi kullanılarak aşağıdaki şekilde hesaplanabilir:

```
glOrtho(xmin,xmax,ymin,ymax,near,far)
glOrtho(left,right,bottom,top,near,far)
```

SORU-39: Paralel İzdüşüm (Parallel Projection) hakkında bilgi veriniz.

İzdüşüm (Projection) şekillerinden birisidir. Buna göre bir şeklin bir düzlem üzerine izdüşümü bulunurken o şeklin düzleme paralel olan yapısı esas alınır.



Yukarıdaki şekilde bir çizginin bir düzleme izdüşümü gösterilmiştir. Görüldüğü üzere bu çizginin düzlemle paralel olan izdüşümü alınmıştır. Bu düzlemle yapılan açının sıfırlanması ve trigonometrik olarak izdüşümünün alınması olarak da yorumlanabilir.

SORU-40: İzdüşüm (Projection) hakkında bilgi veriniz.

Bilgisayar grafiklerinde bir görüntünün izdüşümünün hesaplanması için kullanılan başlığın adıdır. Kısaca aşağıdaki izdüşümlerden herhangi birisini karşılamak için kullanılıyor olabilir.

SORU-41: çoklu şekil değiştirmeler (multiple transformations) hakkında bilgi veriniz.

OpenGL üzerinde şekil değiştirme işlemleri ilgili yazılarda açıklanmıştır. Bu yazının amacı şekil değiştirme işlemlerinin birden fazla olması durumunda kullanılacak OpenGL fonksiyonlarını açıklamaktır.

Öncelikle OpenGL içerisinde anlık olarak tek bir matris aktiftir. Şekiller basılmadan önce bu masfuf(matrix) ile çarpılarak transform olup ekrana basılırlar.

Herhangi bir anda bu matrisin temizlenmesi için birim matris (Identity Matrix) hafızaya yüklenebilir:

```
glLoadIdentity();
```

Yukarıdaki bu fonksiyon ile birim matris hafızaya yüklenir. Örneğin 4×4 boyutlarındaki birim matris aşağıdaki şekildedir.

```
1000
0100
0010
0001
```

Yukarıda da görüldüğü üzere birim matrisin diyagonu 1'lerden oluşmakta bunun dışındaki değerleri ise 0'dan oluşmaktadır. Dolayısıyla örneğin x,y,z,w 'dan oluşan bir noktanın çarpım sonucu aşağıdaki şekildedir:

```
x
y
z
w
```

$x = x$, $y = y$, $z = z$, $w = w$ sonuçları çıkar ki bu da noktaların orjinal koordinatlarıdır.

OpenGL’de kullanılan diğer bir yapı da [stack \(yığın\)](#) yapısıdır. Bu yapıdan faydalanarak herhangi bir dönüşüm (transformation) hafızadan alınarak yığına (stack) konulabilir. bu işlem:

glPushMatrix() ve glPopMatrix() fonksiyonları kullanılabilir.

Anlık olarak hafızada hazır bekleyen dönüşümü yığına koyar (push) veya yığında en üstte olan matrisi alarak hafızaya yükler (pop).

Ayrıca sıkça gereken matrix çarpımı için de OpenGL’de hazır fonksiyonlar bulunmaktadır. Bu fonksiyonlar sayesinde birden fazla dönüşüm matrisi birbiri ile çarpılarak tek bir matris elde edilebilir. Bunun anlamı iki ayrı dönüşümü elde edilen bu tek matris ile yapmanın mümkün olduğudur.

```
glLoadIdentity ( );  
glMultMatrixf (M2) ;  
glMultMatrixf (M1) ;
```

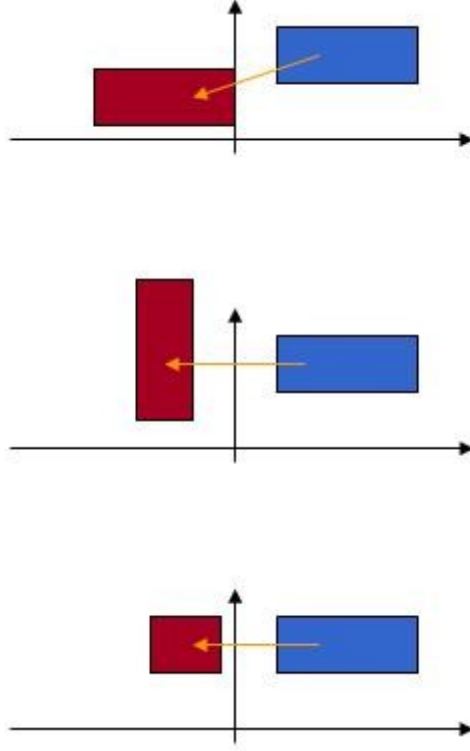
Yukarıda sırasıyla önce birim matris yüklenmiştir. Ardından M2 matrisi ile hafızada yüklü olan matris çarpılmış ve hafızada M2 dönüşüm matrisi geçerli hale gelmiştir.

3. satırda M1 dönüşüm matrisi o anda hafızada bulunan M2 matrisi ile çarpılmış ve hafızada geçerli olan dönüşüm matrisi M1.M2 haline gelmiştir.

Aşağıda dönüşümlerin yapıldığı örnek iki kod bulunmaktadır:

```
glMatrixMode(GL_MODELVIEW);  
// 1. sekildeki mavi kutu  
glColor3f(0.0, 0.0, 1.0);  
Recti(50, 100, 200, 150);  
//1. sekildeki kirmizi kutu  
glColor3f(1.0, 0.0, 0.0);  
glTranslatef(-200.0, -50.0, 0.0);  
Recti(50, 100, 200, 150);  
//2. sekil  
glLoadIdentity ( );  
glRotatef(90.0, 0.0, 0.0, 1.0);  
Recti(50, 100, 200, 150);  
//3. sekil  
glLoadIdentity ( );  
glScalef(-0.5, 1.0, 1.0);  
Recti(50, 100, 200, 150);
```

Yukarıda verilen bu kodun ekran çıktısı aşama aşama aşağıdaki ekranda gösterilmiştir.



Yukarıdaki kod ile aynı sonucu veren aşağıdaki kodu yazmak da mümkündür.

```
glMatrixMode(GL_MODELVIEW);  
//1. sekildeki mavi kutu  
glColor3f(0.0, 0.0, 1.0);  
Recti(50, 100, 200, 150);  
// push matris ile izole bir donusum yapılıyor  
glPushMatrix();  
glColor3f(1.0, 0.0, 0.0);  
glTranslatef(-200.0, -50.0, 0.0);  
Recti(50, 100, 200, 150);  
// pop matrix yapılarak orjinal matrise donuluyor  
glPopMatrix();  
glPushMatrix();  
// ikinci bir donusum (transformasyon) yapılarak  
  
// diger donusumlerden izole ediliyor  
glRotatef(90.0, 0.0, 0.0, 1.0);  
Recti(50, 100, 200, 150);  
// dondurme (rotate) islemi bittikten sonra  
  
// orjinal matris hafızaya geri yukleniyor  
glPopMatrix();  
glScalef(-0.5, 1.0, 1.0);  
Recti(50, 100, 200, 150);
```

Yukarıdaki bu kodun ilk koddan farkı kullanılan stack (Yığın) yapısıdır. Buna göre `glLoadIdentity()` fonksiyonu ile yapılan dönüşümlerin (transformations) temizlenmesi ve

şeklin orjinal halinin basılması mümkün iken, Push ve PopMatrix komutları ile de birbirinden izole dönüşümler elde etmek mümkündür.

SORU-42: OpenGL Şekil Değiştirme İşlemleri (Transformations) hakkında bilgi veriniz.

Bu yazının amacı opengl kütüphanesi kullanılarak şekil değiştirme işlemlerinin nasıl yapıldığını açıklamaktır. Temel olarak 3 tip şekil değiştirme işlemi vardır:

- Taşıma (Translate)
- Döndürme(Rotate)
- Ölçekleme (Büyültüp küçültme, Scaling)

Bu işlemlerin hepsi için OpenGL kütüphanesinde hazır fonksiyonlar bulunmaktadır. Aşağıda her işlem için kullanılan fonksiyonlar ve bu fonksiyonların açıklaması bulunmaktadır.

Ayrıca birden fazla şekil değiştirme işlemi arka arkaya yapılacaksa bunun için çoklu şekil değiştirmeler (multiple transformations) başlıklı yazıyı okuyabilirsiniz.

Taşıma

Bu işlem kabaca bir şeklin veya şekli oluşturan noktaların bütün koordinatların belirli bir değer kadar artırılması veya azaltılmasıdır. Bu konuyla ilgili taşıma işlemleri başlıklı yazıyı okuyabilirsiniz. Aşağıda gösterilen örnek kodda glTranslate fonksiyonu kullanılmıştır.

```
glTranslatef(1,0.5,0);
```

Bu fonksiyon float değerler aldığı için ekranı -1,1 aralığında kodlayan ve 0,0 orjin noktası ekranın merkezinde olan koordinat sistemine göre şekli çeyrek ekran yukarı ve yarım ekran sağa kaydıracaktır.

Aynı fonksiyonu double değişken kullanarak çağırmak da mümkündür. Bunun için fonksiyonun sonundaki f yerine d harfi konulur:

```
glTranslated(100,50,0);
```

Bu satırda ise ekranda 100 pixel (imgecik) sağa ve 50 imgecik aşağı kaydırma yapılmıştır. Ekranın boyutları glutInitWindowSize(en, boy); şeklinde main fonksiyonun içerisinde ilk pencere oluşturulurken tanımlı olmalıdır. İşte bu boyutlara göre ekranın sol üst köşesini 0,0 alan koordinat sisteminde kaydırma yapılmaktadır.

Döndürme

Ekrandaki nesneleri istenilen bir eksen etrafında döndürmeye yarayan koddur. Bu işlemin detayı için döndürme başlıklı yazıyı okuyabilirsiniz. Aşağıda örnek bir kod satırı verilmiştir:

```
glRotatef(10, 0, 0, 1);
```

Yukarıdaki kodda, ekranda z ekseni etrafında 10 derece döndürme işlemi yapılacağı anlatılmaktadır. Buna göre ilk parametre döndürme açısını alırken 2. , 3. ve 4. parametrelerde

eksen deęerlerini almaktadır. Örneęin x ve y eksenleri arasında 45 derecelik açý yapan eksen etrafında 10 derece döndürmek isteseydik:

```
glRotatef(10,1,1,0);
```

deęeri verilecekti.

Ölçekleme

Ekrandaki şekillerin verilen oranlarda büyümesi veya küçülmesini sağlayan koddur. Buna göre şeklin orjinal oranlarının bozulmadan büyümesi veya küçülmesi için 3 boyutta da aynı oranda büyüme küçülme olması gerekir. Detaylı bilgi için ölçekleme başlıklı yazıyı okuyabilirsiniz.

Aşağıda örnek bir kod satırı verilmiştir:

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

Yukarıdaki bu koda göre şekil orjinal oranları ile yarı yarıya küçülmüştür. Şekin örneęin x ekseninde genişlemesini (Esnemesini) istiyor olsaydık:

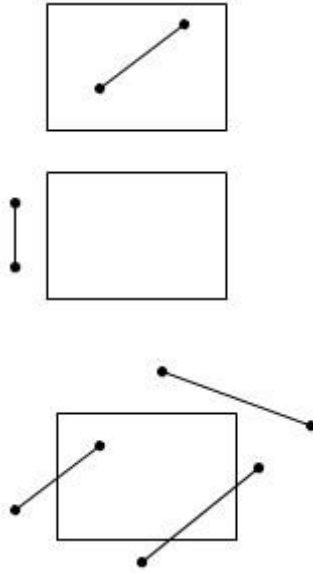
```
glScalef(3.0,1.0,1.0);
```

şeklinde x boyutunu 3 misline çıkarırken y ve z boyutlarını sabit bırakabilirdik.

SORU-43: Cohen-Sutherland Doğru Kesme Algoritması (Line Clipping Algorithm) hakkında bilgi veriniz.

Bu algoritmanın bilgisayar grafiklerindeki kullanımı, görüntülen alan dışına uzanan doğruların kesilmesidir. Yani büyük bir uzayda sadece kısıtlı bir alan gösterilmektedir. Bu algoritma sayesinde gösterilmeyen yerlerde bulunan çizgiler (doğrular) hesaplanmadan sadece gösterilen alan için hesaplama yapılarak performans arttırımı olabilmektedir.

Algoritmanın çalışması bir doğrunun verilen bir düzlem içerisinde olup olmamasını hesaplamaya dayanmaktadır. Buna göre aşağıda bazı doğrular ve bir düzlem verilmiştir. Bu doğrulardan bazıları düzlem üzerindeyken bazıları değildir:



Yukarıda çeşitli durumlarda çizginin düzlem ile olan ilişkisi gösterilmiştir. Bir çizgi düzlemin tamamen içinde olabilir, tamamen dışında olabilir veya kısmen içinde olabilir.

Buna göre Cohen-Sutherland algoritması aslında çok basit bir şekilde 4 kutu hazırlamış ve bu kutulara sırasıyla düzlemin x ve y değerlerinin min ve max değerlerine göre doğrunun başlangıç ve bitiş değerlerinin yazılmasını ister:

1	2	3	4
üst	alt	sol	sağ

Bu kutuların doldurulmuş değerlerine göre bir noktanın düzlemin neresinde yer aldığı aşağıda gösterilmiştir:

1001	1000	1010
0001	0000	0010
0101	0100	0110

Yukarıdaki sayıların anlamı şudur. Örneğin 0000 değerleri çizginin tamamen düzlemin içinde olduğunu gösterir. 0110 değeri ise düzlemin sağ alt köşesinde olduğunu gösterir.

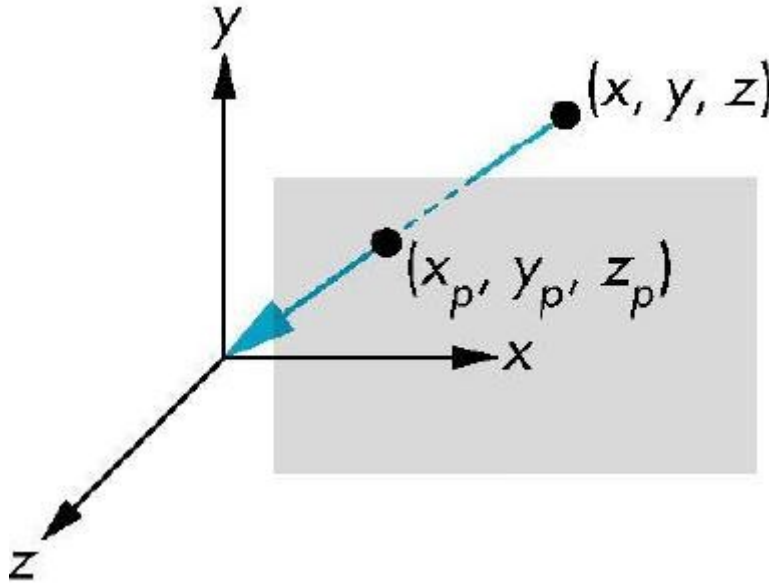
SORU-44: Homojen Koordinatlar (Homogenous Coordinates Form) hakkında bilgi veriniz.

Homojen koordinat formunda kısaca aşağıda gösterilen 4×4 matrisin şekil değiştirmede (transformation) kullanılmasıdır. Bu matris kullanımı sırasında üç boyutlu bir ortam için (x,y,z) koordinatlarından oluşan Kartezyen uzayın her eksenindeki dönüşümünü göstermek mümkündür. Bu

ortak dönüşüm matrisinden ve her eksenin birbirine dönüşümünün kolaylığından dolayı homojen koordinat formu ismin almış olan dönüşüm matrisine aşağıda bir örnek verilmiştir:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

3 boyutlu bir uzay için bu matrisin 4×4'lük bir kare matris olmasının sebebi 4. Kolon ve satırın perspektif ögesi için kullanılmasıdır. Bu durumu aşağıdaki örnekle açıklayalım:



Yukarıdaki şekilde 3 boyutlu Kartezyen uzayda bir noktanın Z eksenine göre ileride olması durumunun 2 boyuta geçirilirken perspektif hissinden dolayı farklı bir nokta olarak kodlanması gösterilmiştir.

Şimdi bu nokta kodlamasının nasıl olacağını aşağıdaki iki örnek için inceleyelim:

Görüldüğü üzere x değeri ile z değeri arasında bir bağlantı vardır:

Yani bir noktanın x eksenindeki değeri z değeri kadar perspektif etkisinde kalmaktadır. İşte homojen koordinat gösteriminde kullanılan bu 4. Kolon ve satır bu işe yaramaktadır.

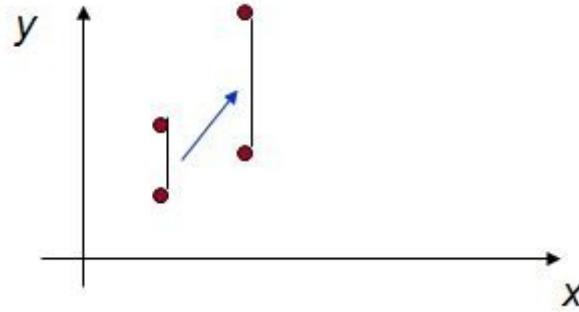
SORU-45: OpenGL Geometrik Nesneler (Geometric Objects) hakkında bilgi veriniz.

OpenGL'in araç kutusu olarak adlandırılabilcek GLUT grafik kütüphanesi araç kutusu (Graphics Library Utility Kit) içinde tanımlı olan geometrik nesneler aşağıda sıralanmış ve redbook üzerindeki bağlantıları verilmiştir:

- 11.1 glutSolidSphere, glutWireSphere
- 11.2 glutSolidCube, glutWireCube
- 11.3 glutSolidCone, glutWireCone
- 11.4 glutSolidTorus, glutWireTorus
- 11.5 glutSolidDodecahedron, glutWireDodecahedron
- 11.6 glutSolidOctahedron, glutWireOctahedron
- 11.7 glutSolidTetrahedron, glutWireTetrahedron
- 11.8 glutSolidIcosahedron, glutWireIcosahedron
- 11.9 glutSolidTeapot, glutWireTeapot

SORU-46: Ölçeklendirme (Scaling) hakkında bilgi veriniz.

Bilgisayar grafiklerinde şekil değiştirme işlemlerinden birisidir. Bu işlemin amacı bir şekli mevcut konumu ve yönü bozulmadan büyültmek ve küçültmektir (Zoom in , Zoom out). Aşağıdaki örnekte gösterilen ölçekleme işleminin formülü verilmiştir:



Yukarıdaki ölçekleme işlemi için :

$$x' = x \cdot sx$$

$$y' = y \cdot sy$$

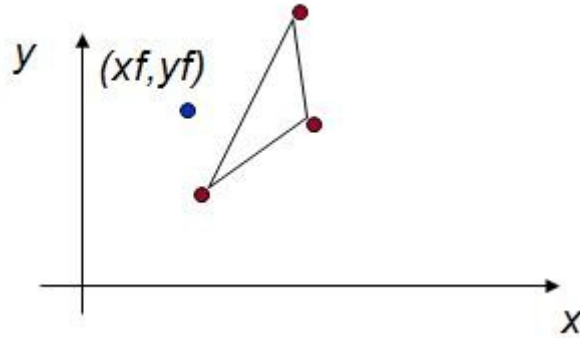
formülleri kullanılabilir. Buradaki sx ve sy değerleri yeni ölçeği belirlemektedir. Yani örneğin şeklin 2 misli büyümesi istenirse $sx = 2$ ve $sy = 2$ değerleri ile şeklin orjinal x ve y değerleri çarpılır.

Yukarıdaki bu çarpım matrisi

$$\begin{matrix} sx & 0 \\ 0 & sy \end{matrix}$$

şeklinde ifade edilebilir ve $P' = P S$ çarpımı ölçekleme (scaling) olmuş olur.

Burada bir yan etki şeklin büyürken aynı zamanda da taşınıyor olmasıdır. Bu problemin çözümü için sabit bir noktayı kerteriz olarak ölçekleme işlemi sonrasında bu noktaya göre şeklin geri taşınması mümkündür:



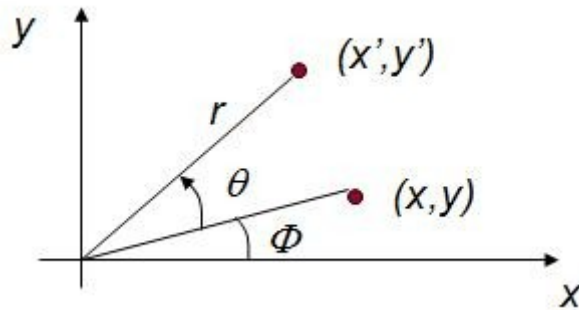
$$x' = x \cdot sx + xf(1-sx)$$

$$y' = y \cdot sy + yf(1-sy)$$

Yukarıdaki ölçekleme işlemi, ölçekleme işleminin tersi kadar taşıma işlemini de içermektedir.

SORU-47: 2 Boyutlu Döndürme (2D Rotation) hakkında bilgi veriniz.

Bilgisayar grafiklerinde şekil değiştirme işlemlerinden birisidir. Bu işlemin amacı bir şekli mevcut konumunu ve şeklini bozulmadan bir eksen etrafında döndürmektir. Aşağıdaki örnekte gösterilen döndürme işleminin formülü verilmiştir:



$$x' = r \cdot \cos(\Phi + \Theta)$$

$$y' = r \cdot \sin(\Phi + \Theta)$$

Yukarıdaki bu formülasyonu aşağıdaki matris ile ifade etmek mümkündür:

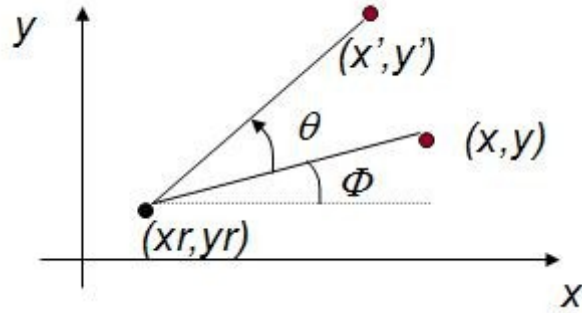
$$\begin{matrix} \cos \Phi & -\sin \Phi \\ \sin \Phi & \cos \Phi \end{matrix}$$

Bu döndürme matrisini çıkardıktan sonra aşağıdaki denklem yazılabilir:

$$P' = R P$$

Buradaki R matrisi döndürme matrisidir. P ve P' matrisleri ise sırasıyla noktanın ilk koordinatları ve son koordinatlarıdır.

Ayrıca döndürme işlemi merkezden kaydırılmış bir şekilde de yapılabilir. Yani merkez etrafında değil, aşağıdaki gibi bir nokta etrafında da yapılabilir:



Bu durumda yukarıdaki formüllerin kaydırılmış hali:

$$x' = x_r + (x - x_r) \cdot \cos \Phi - (y - y_r) \cdot \sin \Phi$$

$$y' = y_r + (x - x_r) \cdot \sin \Phi + (y - y_r) \cdot \cos \Phi$$

SORU-48: 2 Boyutlu Şekil Dönüşümleri (2D Transformations) hakkında bilgi veriniz.

Bilgisayar grafiklerinde temel olarak 3 tip şekil hareketinden bahsedilebilir. Bunun dışındaki bütün şekil değiştirmeler de bu 3 tipin birleşimine indirgenebilir.

Bu 3 tip şekil değişimi:

Taşıma (Translation)

Döndürme (Rotation)

Ölçekleme (Scaling)

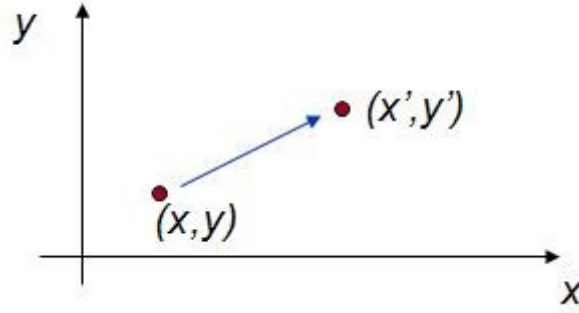
Yukarıdaki bu 3 tip şekil hareketine karşılık şeklin yapısal değişimi de 2 tiptir:

Yansıma (Reflection)

Eğme (Shearing)

SORU-49: 2 boyutlu Taşıma (2D Translation) hakkında bilgi veriniz.

Bu işlemin amacı bir şekli mevcut konumundan bozulmadan farklı bir konuma taşımaktır. 2 boyulu aşağıdaki doğruyu ele alalım:



Yukarıdaki şekilde başlangıç noktası olan noktanın (x,y) koordinatlarından (x',y') koordinatlarına taşınması tasvir edilmiştir.

Bu işlemi basitçe aşağıdaki denklemle yazabiliriz:

$$x' = x + \Delta x$$

$$y' = y + \Delta y$$

Yukarıdaki bu gösterime ilave olarak $P(x,y)$ ve $P'(x',y')$ olarak ifade edilirse taşıma işlemi:

$$P' = P + \Delta$$

matrisleri ile gösterilebilir. Bu durumda matrisler sırasıyla aşağıdaki şekildedir:

$$P = [x, y]$$

$$\Delta = [\Delta x, \Delta y]$$

$$P' = [x', y']$$

SORU-50: OpenGL ile menülerin tasarımı hakkında bilgi veriniz.

OpenGL ve bağlantılı olduğu GLUT kütüphanesi sayesinde ekranda açılır menüler elde etmek mümkündür. Bunun için main fonksiyonu altında menü tanımı yapılmalıdır.

bu işlem için glutCreateMenu fonksiyonu kullanılır. Bu fonksiyonun dönüş değeri int'tir. Aldığı parametre ise bir fonksiyon göstericisidir. Örneğin aşağıda bir menu oluşturulmuş ve menuden bir öğenin seçilmesi sonucunda çağırılacak fonksiyon ismi parametre olarak verilmiştir:

```
int menu1;  
menu1=glutCreateMenu(selectMode);  
glutAddMenuEntry("Sadi",1);  
glutAddMenuEntry("Evren",2);  
glutAddMenuEntry("Seker",3);
```

Yukarıdaki bu satırlar ile menü içerisine 3 adet bilgi girilmiştir. Her bilgi için bir belirleyici sayı atanmıştır (Örneğin Evren seçeneği seçildiğinde 2 sonucu dönecektir). Bu sonuçların

döndüğü selectMode fonksiyonu (ki kendisi glutCreateMenu fonksiyonuna parametre olarak verilmiştir) aşağıdaki şekilde yazılabilir:

```
void selectObject(int obj){  
    switch(obj){  
  
    case 1:  
  
        printf("Secilen Sadi");  
  
        break;  
  
    case 2:  
  
        printf("Secilen Evren");  
  
        break;  
  
    case 3:  
  
        printf("Secilen Seker");  
  
        break;  
  
    }  
  
}
```

Yukarıdaki bu fonksiyon menüden seçilme işlemi yapıldığında seçmi yakalayacak olan fonksiyonu göstermektedir. Gelen parametreye göre hangi işlemin yapılacağı bu şekilde belirlenebilir.

İsteğe göre menülerin altına alt menü de eklenebilir.

```
glutAddSubMenu("İsimler",menu1);
```

veya menü doğrudan bir eyleme bağlanabilir. Örneğin aşağıda sağ fare tıklamasına eklenmiş kod gösterilmektedir:

```
glutAttachMenu(GLUT_RIGHT_BUTTON);
```

Örneğin aşağıdaki menü, yine aşağıda verilen kod ile üretilmiştir:



```
menu3=glutCreateMenu(selectMode);
glutAddMenuEntry("Sadi",1);
glutAddMenuEntry("Evren",2);
glutAddMenuEntry("Seker",3);
menu1= glutCreateMenu(selectObject);
glutAddSubMenu("İsimler",menu3);
glutAttachMenu(GLUT_RIGHT_BUTTON);
```

SORU-51: OpenGL ile kullanıcı ile iletişimi (user interaction) hakkında bilgi veriniz.

Bu yazının amacı OpenGL ortamında kullanıcının klavye ve mouse ile yapmış olduğu etkilerin nasıl algılandığını açıklamaktır.

Yazıya başlamadan önce belirtmelidir ki OpenGL bir çizim ve grafik animasyon geliştirme ortamıdır ve bu ortamın görevleri arasında kullanıcı ilişkileri bulunmamaktadır. Bunun yerine yine OpenGL ile artık özdeşleşmiş olan GLUT (Graphics Library Utility Toolkit) isimli kütüphane kullanılmaktadır.

Bu kütüphaneyi yüklemek için :

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

satırı yazılarak glut.h dosyası projeye dahil edilir (include)

Fare işlemleri (mouse)

Öncelikle OpenGL main fonksiyonunun içerisinde bir mouse fonksiyonu tanımlamak gerekir. Bunun için glut kütüphanesinde bulunan ve bir fonksiyon göstericisi kullanan glutMouseFunc aşağıdaki şekilde bir mouse fonksiyonu tanımlar:

```
glutMouseFunc(mouse);
```

Yukarıdaki bu satır main içerisine yazıldıktan sonra aşağıdakine benzer mouse isminde bir fonksiyon kodlanabilir:

```
void mouse(int button, int state, int x, int y)
{
switch (button) {
case GLUT_LEFT_BUTTON:
if (state == GLUT_DOWN)
printf("x: %d y: %d",x,y);
break;
case GLUT_MIDDLE_BUTTON:
case GLUT_RIGHT_BUTTON:
break;
default:
break;
}
}
```

Yukarıdaki mouse fonksiyonunun ismi farklı olabilmekle beraber prototipi yukarıdaki fonksiyonun aynısı olmalıdır. Buna göre fonksiyon 4 adet int değişken almalıdır.

Alınan bu değişkenlerin anlamları sırasıyla

button: farenin hangi düğmesine basıldığını gösterir (GLUT_LEFT_BUTTON: sol tuş, GLUT_RIGHT_BUTTON: sağ tuş, GLUT_MIDDLE_BUTTON : orta tuş)

state: düğmenin durumunu verir. (GLUT_DOWN: Düğmeye basıldı, GLUT_UP: Düğme bırakıldı)

x: ekrandaki x koordinatı

y: ekrandaki y koordinatı

yukarıdaki bu değişkenlerin değerleri örneğin

button = GLUT_LEFT_BUTTON , state = GLUT_DOWN , x =100, y=250

şeklindeyse, ekranın 100,250 koordinatlarına farenin sol tuşu ile tıklanmış demektir. Bu koordinat sistemi ekranın sol üst köşesini 0,0 olarak kabul eden ve ekranın sağ alt köşesi , main fonksiyonundaki glutInitWindowSize fonksiyonuna parametre olarak verilen değer olan koordinat sistemidir.

Klavye işlemleri

Fare işlemine benzer olarak main fonksiyonunun içerisinde:

glutKeyboardFunc(keyboard);

fonksiyonu yazılarak bir fonksiyon gösterecisi parametre verilir. Aşağıda örnek bir klavye fonksiyon kodlaması bulunmaktadır:

```
void keyboard(unsigned char key, int x, int y)
{
    switch (key) {
        case 'r':
        case 'R':
            printf ("Klavyeden r tuşuna basıldın");

            break;
        case 'z':
            zoomFactor += 0.5;
            if (zoomFactor >= 3.0)
                zoomFactor = 3.0;
            printf ("zoomFactor is now %4.1f\n", zoomFactor);
            break;
        case 'Z':
            zoomFactor -= 0.5;
            if (zoomFactor <= 0.5)
```

```
zoomFactor = 0.5;
printf ("zoomFactor is now %4.1fn", zoomFactor);
break;
case 27:
exit(0);
break;
default:
break;
}
}
```

Yukarıdaki kodda da görülebileceği üzere keyboard fonksiyonumuz 3 parametre almıştır bunlar:

unsigned char key : basılan tuşun ascii kod karşılığı

int x: tuşa basıldığı sırada farenin x koordinatı

int y: tuşa basıldığı sıradaki farenin y koordinatı

dolayısıyla bu fonksiyonu tetikleyen olay klavyeden bir tuşa basılmasıdır. Bu basılma sırasında farenin konumu da fonksiyon tarafından yakalanmaktadır. Ayrıca basılan tuşun ascii kodu da ilk parametre olarak yakalandığı için basit bir switch / case yapısı içerisinde bu bilgi algılanabilir.

Yukarıdaki kodda son case için 27 kodu verilmiştir. Bu ascii olarak escape (esc.) tuşunun kodudur. Dolayısıyla escape tuşuna basıldığında exit(0) ile programdan çıkılması kodlanmıştır. Yine büyük ve küçük harflerin ascii kodları farklı olduğu için ayrı ayrı kodlanması gerekmektedir.

Özel Klavye işlemleri

GLUT kütüphanesi kullanılarak özel klavye işlemlerini de algılamak mümkündür. Temel olarak yön tuşları Fonksiyon tuşları ctrl, alt, shift gibi tuşlar bu türden tuşlardır. Bu tuşların karşılığı olan GLUT değişkenleri bulunur. Öncelikle özel klavye fonksiyonu tanımı aşağıdaki şekilde yapılmalıdır:

```
glutSpecialFunc(special_key);
```

Yukarıdaki bu tanımdan sonra aşağıdaki fonksiyon tanımı mümkündür:

```
GLvoid special_key(int key, int x, int y)
{
switch (key) {
case KEY_RIGHT:
printf("sağ oka basıldı");

break;
case KEY_LEFT:
```

```

printf("sol oka basıldı");

break;
case KEY_UP:

printf("yukarı okula basıldı");

break;

case KEY_DOWN:
printf("aşağı oka basıldı");

break;

default:
printf(" %d karakteri tanımlı değildir.n", key);
break;
}
}

```

Yukarıdaki örnek kodda daha önce de açıklanan glutKeyboardFunc ile aynı mantık izlenir ve tıklanan tuşun kodu ile birlikte tıklanma sırasındaki farenin konumu da alınır.

Özel klavye kodları için aşağıda bir tablo verilmiştir:

```

GLUT_KEY_F1
    F1 fonksiyon tuşu.
GLUT_KEY_F2
    F2 fonksiyon tuşu.
GLUT_KEY_F3
    F3 fonksiyon tuşu.
GLUT_KEY_F4
    F4 fonksiyon tuşu.
GLUT_KEY_F5
    F5 fonksiyon tuşu.
GLUT_KEY_F6
    F6 fonksiyon tuşu.
GLUT_KEY_F7
    F7 fonksiyon tuşu.
GLUT_KEY_F8
    F8 fonksiyon tuşu.
GLUT_KEY_F9
    F9 fonksiyon tuşu.
GLUT_KEY_F10
    F10 fonksiyon tuşu.
GLUT_KEY_F11
    F11 fonksiyon tuşu.
GLUT_KEY_F12
    F12 fonksiyon tuşu.
GLUT_KEY_LEFT
    Sol yön tuşu.
GLUT_KEY_UP
    Yukarı yön tuşu.
GLUT_KEY_RIGHT
    Sağ yön tuşu.
GLUT_KEY_DOWN

```

```
Aşağı yön tuşu.  
GLUT_KEY_PAGE_UP  
Page up yön tuşu.  
GLUT_KEY_PAGE_DOWN  
Page down yön tuşu.  
GLUT_KEY_HOME  
Home yön tuşu.  
GLUT_KEY_END  
End yön tuşu.  
GLUT_KEY_INSERT  
Inset yön tuşu.
```

SORU-52: DEV CPP ile OpenGL Derleme hakkında bilgi veriniz.

DEV-CPP geliştirme ortamında gelen OpenGL kütüphanesi ne yazık ki standart OpenGL ve GLUT fonksiyonlarını desteklememektedir. Bu fonksiyonların yüklenmesi için ilave glut kütüphanesinin kurulması gerekir.

Bu işlem için öncelikle aşağıda bulunan bağlantıdan bu kütüphaneyi indiriniz.

www.sadievrenseker.com/graf/glut-MingW-DEV-C++.zip

Ardından dosyanın içinden çıkan

```
glut.h – C:Dev-CppincludeGL altına  
glut32.def – C:Dev-Cpplib altına  
glut32.dll – C:WINDOWSsystem32 altına
```

kopyalayın (kurulumunuzun standart olduğu varsayılmıştır şayet Dev-CPP farklı yerdeyse c:dev-cpp yerine bu kurulum dizinini, windows c:windows dışında bir yerdeyse c:windows yerine bu kurulum yerini yazınız)

Ardından yeni bir proje oluşturup

Project -> Project Options ->Parameter->Linker altındaki yazıyı (şayet varsa) temizleyip aşağıdaki linker özelliklerini ekliyoruz:

```
-lopengl32 -lglu32 -lglut32
```

Bu işlemden sonra projenizin başarıyla çalışması gerekir.

SORU-53: Çokgenler ve OpenGL hakkında bilgi veriniz.

Bu yazının amacı bilgisayar grafiklerinin en temel konularından birisi olan çokgenlerin OpenGL ile nasıl çizilebildiklerini açıklamaktır.

Öncelikle OpenGL kütüphanesindeki fonksiyon isimlerini hatırlayalım:

glRect*(x1, y1, x2, y2);

```
■ i (integer)  
■ s (short)  
■ f (float)  
■ d (double)  
■ v (vector)
```

Yukarıdaki fonksiyonda * ile gösterilen yere aşağıdaki değerler gelebilmektedir. Örneğin glRectiv şeklinde bir kullanımın anlamı glRect fonksiyonunun parametreleri integer (tam sayı) olacak ve vector (yöney) içerecek demektir.

Buna göre aşağıdaki tam sayı kullanımı:

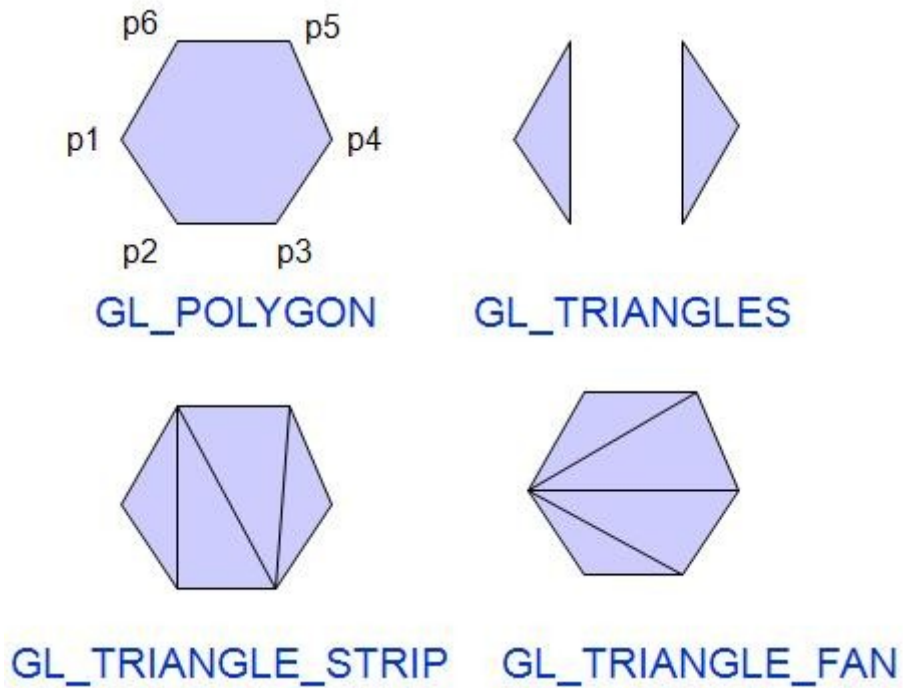
```
glRecti(200, 100, 50, 250);
```

ile aşağıdaki tamsayılı yöney kullanımı:

```
int v1[ ]={200, 100};  
int v1[ ]={50, 250};  
glRectiv(v1, v2);
```

Aynı anlama gelmektedir.

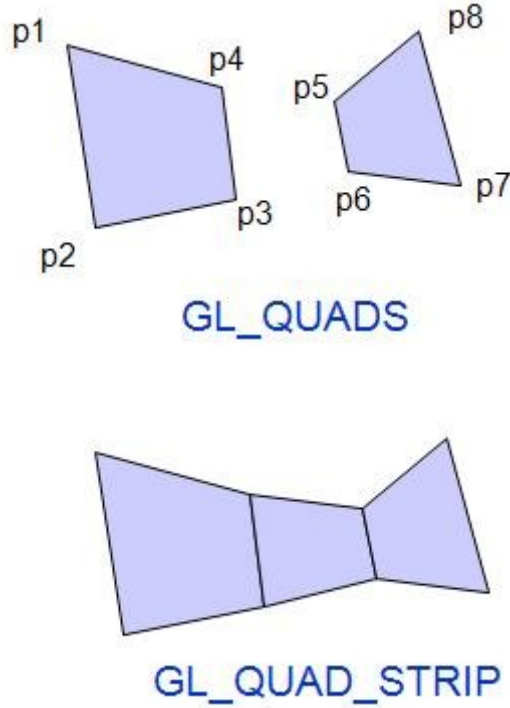
OpenGL üzerinde ayrıca 2 boyutlu düzlemleri ifade etmek için kullanılabilecek çeşitli alternatifler bulunmaktadır. Örneğin aşağıdaki şekilleri ve çizimleri için kullanılan OpenGL geometrik objelerinin (geometric primitives) farklı durumları ele alınmıştır:



yukarıdaki her şeklin farklı bir geometrik obje kullanılarak çizilmesi mümkündür. Örneğin GL_POLYGON objesi aşağıdaki kod ile çizilebilir:

```
glBegin (GL_POLYGON);  
glVertex2iv(p1);  
glVertex2iv(p2);  
glVertex2iv(p3);  
glVertex2iv(p4);  
glVertex2iv(p5);  
glVertex2iv(p6);  
glEnd();
```

Yukarıdaki kodda örnek olarak verilen şekildeki altıgenin her kenarını ayrı ayrı çizen bir geometrik obje gösterilmektedir.



Yukarıdaki bu şekil için de örneğin aşağıdaki kod kullanılabilir:

```
glBegin (GL_QUADS);  
    glVertex2iv(p1);  
    glVertex2iv(p2);  
    glVertex2iv(p3);  
    glVertex2iv(p4);  
    glVertex2iv(p5);  
    glVertex2iv(p6);  
    glVertex2iv(p7);  
    glVertex2iv(p8);  
glEnd();
```

SORU-54: Taşıma Algoritması (Flood Filling Algorithm) hakkında bilgi veriniz.

Bilgisayar grafiklerinde kullanılan ve çokgenlerin içlerinin doldurulması için kullanılan algoritmadır. Basitçe bir [çokgenin \(polygon\)](#) için belirli bir renge boyanacağını düşünelim. Bu durumda çokgenin içrisinin belirlenmesi ve belirlenen bu alanın daha önceden seçilmiş renk ile doldurulması gerekir.

```
void floodFill4 (int x, int y, int fillColor, int  
    interiorColor)  
{  
    int color;  
    /* Set current color to fillColor, then perform  
       following operations. */  
    getPixel (x, y, color);  
    if (color == interiorColor) {
```



```

setPixel (x, y); // Set pixel color to fillColor
floodFill4 (x + 1, y, fillColor, interiorColor);
floodFill4 (x - 1, y, fillColor, interiorColor);
floodFill4 (x, y + 1, fillColor, interiorColor);
floodFill4 (x, y - 1, fillColor, interiorColor)
}

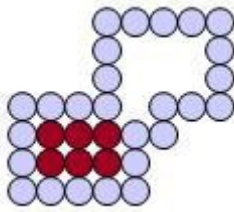
```

Yukarıdaki koddan da anlaşılacağı üzere algoritma, çokgenin içerisindeki rastgele bir noktadan başlayarak, bu başlangıç noktasının komşusu olan noktaları (4 yöne doğru) teker teker boyamaya çalışmaktadır. Bu işlem sırasında boyanacak olan piksellerin renk kodları kontrol edilerek, zaten boyanmış veya kenar olup olmadığına bakılmaktadır.

Bu algoritmanın kenar doldurma (boundary filling) algoritmasından en belirgin farkı, bu algortmada herhangi bir kenar rengi bulunmamasıdır. Dolayısıyla bu algoritma ile doldurulan çokgenlerin kenarları da dolgu renginde olmaktadır.

SORU-55: Sınır Doldurma Algoritması (Boundary Filling Algorithm) hakkında bilgi veriniz.

Bilgisayar grafiklerinde kullanılan ve çokgenlerin içlerinin doldurulması için kullanılan algortmadır. Basitçe bir çokgenin (polygon) içinin belirli bir renge boyanacağını düşünelim. Bu durumda çokgenin içisinin belirlenmesi ve belirlenen bu alanın daha önceden seçilmiş renk ile doldurulması gerekir.



Örneğin yukarıda verilen çokgenin bir alanına bu algortma uygulanmıştır. Bu algortmanın kodu aşağıdaki şekildedir:

```

void boundaryFill4 (int x, int y, int fillColor, int
    borderColor)
{
    int interiorColor;
    /* Doldurulacak olan renk olarak ayarlama yapılıyor */
    getPixel (x, y, interiorColor);
    if ((interiorColor != borderColor) &&
        (interiorColor != fillColor)) {
        setPixel (x, y); // Set pixel color to fillColor
        boundaryFill4 (x + 1, y , fillColor, borderColor);
        boundaryFill4 (x - 1, y , fillColor, borderColor);
        boundaryFill4 (x , y + 1, fillColor, borderColor);
        boundaryFill4 (x , y - 1, fillColor, borderColor)
    }
}

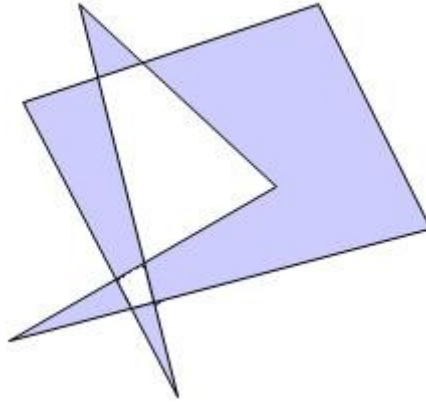
```

Yukarıdaki koddan da anlaşılacağı üzere algoritma, çokgenin içerisindeki rastgele bir noktadan başlayarak, bu başlangıç noktasının komşusu olan noktaları (4 yöne doğru) teker teker boyamaya çalışmaktadır. Bu işlem sırasında boyanacak olan piksellerin renk kodları kontrol edilerek, zaten boyanmış veya kenar olup olmadığına bakılmaktadır.

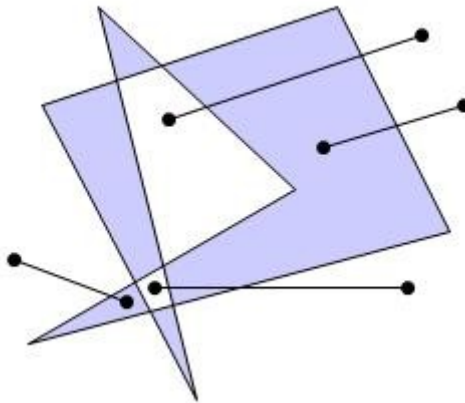
SORU-56: Çokgenlerin Doldurulması (Filling Polygons) hakkında bilgi veriniz.

Bu yazının amacı bilgisayar grafiklerinde kullanılan çokgenlerin sınırları içerisinde kalınarak nasıl doldurulduğunu açıklamaktır. Bir çokgeni doldurmak en basit anlamda çokgenlerin boyanması sırasında kullanılabilir.

Boyama işleminden önce bir poligonun alt poligonlardan oluşması ve basitleştirilmesi durumunu inceleyelim. Örneğin aşağıdaki çokgende birden fazla iç alan bulunmaktadır ve bu alanların farklı renk veya dolgular ile doldurulması istenebilir.



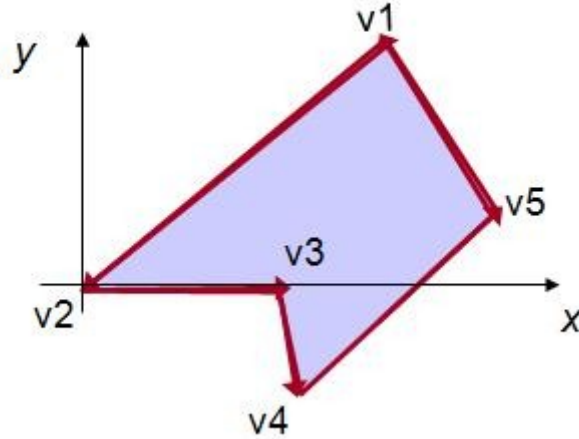
Bu durumda çokgeni oluşturan alt çokgenlerin bulunması için tek-çift kontrolü yapılabilir. Bu işlem aslında oldukça basittir. Herhangi bir alandan rastgele bir nokta alınır. Bu noktadan poligonun dışına doğru (istenilen yönde) bir doğru çizilir. Doğrunun çokgenin kaç kenarı ile kesiştiği sayılır. Şayet kesişim sayısı tek ise alan tek alan, çift ise alan çift alan olarak kabul edilir. Buna göre çokgenin tek ve çift olarak alt alanlara bölünmesi mümkün olur.



SORU-57: Çokgenlerin Üçgene Çevrimi (Splitting Polygons to Triangles) hakkında bilgi veriniz.

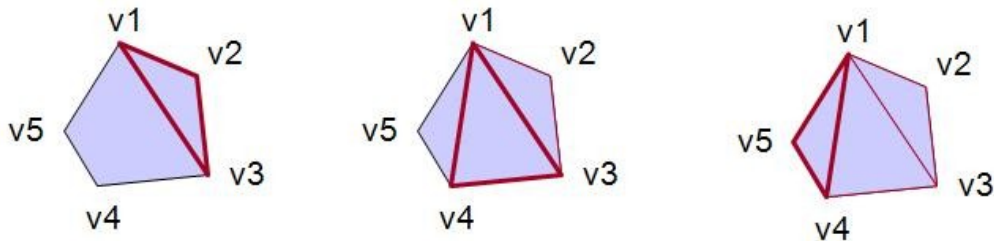
Bilgisayar grafiklerinde 2 boyutlu şekilleri tasvir etmek için kullanılan çokgenler (polygon) aslında bir grup üçgen ile ifade edilebilir. Bu konuyu aslında 2 farklı başlık altında inceleyebiliriz. Basitçe çokgenler içbükey (mütekavvis) ve dışbükey (konkav) olarak iki grupta incelenebilir.

Aşağıda bir içbükey (mütekavvis) çokgenin basit bir adımla nasıl bir üçgen ve bir dışbükey çokgene dönüştürüldüğü gösterilmektedir.



Bu şekilde de görüldüğü üzere çokgeni oluşturan yöneylerden (vector) bir tanesi eksen üzerine getirilir ve eksenin altı ve üstü şeklinde şekil 2'ye bölünür. Eksenin altında bir üçgen ve eksenin üzerinde ise bir çokgen elde edilmiş olur.

Bu aşamadan sonra her iki çokgen şeklide dışbükey olarak düşünülebilir. Bir dışbükey çokgenin ise üçgenlere bölünmesi basitçe ortak komşusu bulunna iki noktanın birleştirilmesi ile elde edilir. Farklı bir ifade ile çokgeni oluşturan yöneylerin (vektör) bileşkesinin alınması ile çokgen bir üçgen ve bir çokgen (bu yeni çokgen de bir üçgen olabilir) şeklinde ikiye bölünmüş olur.

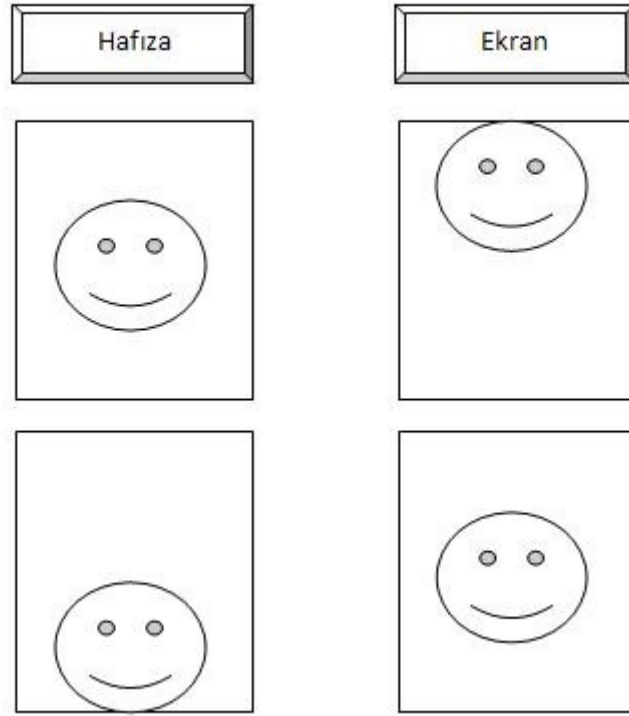


Yukarıdaki örnekte bu üçgenlerin nasıl elde edildiği adım adım gösterilmiştir.

SORU-58: Çift Tamponlama (Double Buffering, Çift Arabellek) hakkında bilgi veriniz.

Bilgisayar grafiklerinde kullanılan bu terime göre ekranda görüntülenecek olan bilginin iki kere hafızalanmasıdır. Buna göre görüntünün iki ayrı kopyası hafızada tutulur. Grafiklerden birisi gösterilirken diğeri hazırlanır. Böylece ekranda sadece bir tanesi görüntülenirken diğeri hafızada bir sonraki görüntülenme için hazırlanır.

Hafızada ayrıca hazırlanılmasının sebebi, bu hazırlama işleminin vakit alması ve bütün işlemin çok hızlı bir şekilde bitmemesidir. Yani bir görüntüde çok sayıda nesne olduğunu düşünürsek bu nesnelerin hepsi farklı zamanlarda sırayla çizilmektedir. İlk çizilen nesne ile son çizilen nesne arasında zaman farkı olmaktadır. İşte izleyicinin bu farkı görmemesi için hafızada çizim işlemi yapıp bitmiş ve hazır görüntü kullanıcıya gösterilir.



Yukarıdaki şekilde iki hafızada görüntülenmiştir. Sağdaki kolonda hafıza, soldaki kolonda ise ekran görüntülenmiştir. Yukarıda gösterilen animasyona göre gülen yüz yukarıdan aşağıya kaymaktadır. 3 kareden oluşan bu animasyonun 2 ayrı durumu yukarıda görülmektedir. İlk satırda ekranda gülen yüz yukarıdayken hafızada bir sonraki kare olan ortadaki hali hazırlanır. İkinci satırda ise daha önce ortalanmış olan gülen yüz ortalanırken hafızada gülen yüzün alttaki hali hazırlanır. Bu şekilde animasyon bir sonraki görüntüyü hafızada hazırlarken bir önce hazırladığı görüntüyü ekrana basar.

SORU-59: Uzaysal Çözünürlük (Spatial Resolution) hakkında bilgi veriniz.

En basit anlamda bir uzayda bulunan örneklerin birbirine olan uzaklığını belirtir. Yani bir uzaydan (örneğin 3 boyutlu bir ortamdan) bir örnek alındığında (örneğin bir kalem bilgisi alındığında) bu bilgiler arası mesafe, örneğin çözünürlüğünü belirler. Buna göre mesafe kısaltıldıkça alınan örnek sayısı artar ve dolayısıyla kalem daha çok detayı hakkında bilgi sahibi olunur ve çözünürlük artmış olur. Mesafeler uzadıkça ise örnek sayısı azalırken kalem hakkında daha az bilgi sahibi olunur.

Örneğin aşağıda orijinali verilen resmi ele alalım:



Sapancada ay ierken

Bu resimin uzaysal özünürlüğü 400×300 pikseldir ve azaltılarak aynı boyutlarda bir resim 40×30 piksele indirgenirse (yani 10 imgeciğe (pixel) 1 imgecik denk düşecek şekilde indirgenirse) aşığıdaki halde görülür:



Sapancada ay ierken resmi 10'a 1 oranında imgecik (pixel) azaltmasına gidilmiş

Görüldüğü üzere resmin ana hatları aynı kalmakta ancak resmi oluşturan imgecikler (pixel) birer kare olarak düşünülürse bu karelerin boyutu artmaktadır.

Bu sayede ikinci resim 1. resimden ok daha az yer kaplamakta ancak daha az veri iermektedir.

SORU-60: OpenGL'in alıřma Sırası hakkında bilgi veriniz.

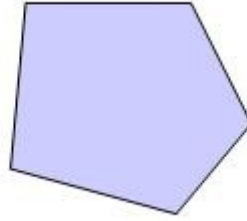
Bilindiği üzere OpenGL ortamında yazılan bir programın, çalışma sırasında yaptığı bir dizi işlem sırası bulunmaktadır. Bu işlemler şu şekilde sıralanabilir:

1. Sanal ortamın toparlanması (assemble)
2. Sanal olarak kameranın ortama yerleştirilmesi ve görüntü açısının belirlenmesi
3. Görüntünün bir düzlem üzerine iz düşümünün alınması. (3 boyutlu olan OpenGL ortamı sonuçta iki boyutlu ekranlarda görüntülenmektedir ve bu yüzden görüntü bir düzlem üzerine indirgenir)
4. Uzaysal örnekleme (Spatial Sampling) ile resmin sayısallaştırılması (digitisation). (Bu aşamada ızgaralama (rastering (yani görüntüyü dizilere ve veri ünitelerine ayırma)) ve kırpma(clipping (yani görüntünün görünmeyen kısımlarının kırılarak hız kazanılması)) gibi işlemler yapılmaktadır.

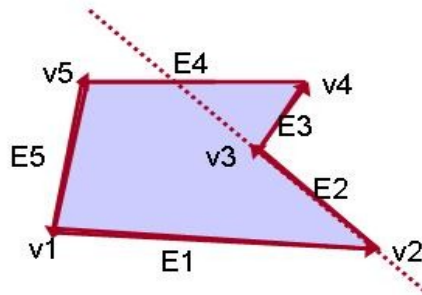
Kullanıcıyla iletişim kurmak gibi sebeplerle gerçek OpenGL ortamının dışında bir takım ilave yardımcı fonksiyonlara ihtiyaç duyulmaktadır. Bu noktada GLUT (GL Utility Toolkit) adı verilen ilave paket kullanılır ve bu fonksiyonların öneki “glut”tur.

SORU-61: Çokgen (Poligon, Polygon) hakkında bilgi veriniz.

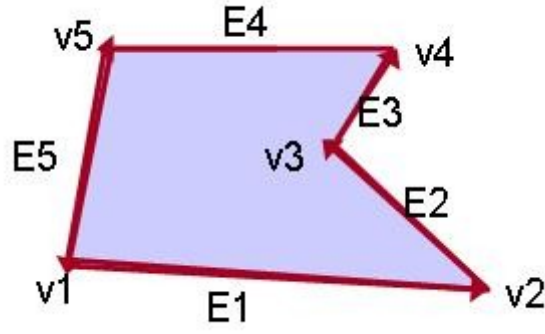
Bir düzlemde (2 boyutlu), çok sayıdaki (n adet) noktanın belirlediği kapalı alanın adıdır. Örneğin aşağıdaki bir çokgen resmedilmiştir:



Bu şekildeki her köşe bir noktadır ve koordinatlarının bilinmesi yukarıdaki şekli çizmek için yeterlidir. Yukarıdaki çokgenin ayrıca bir özelliği de dışbükey (concave) olmasıdır.

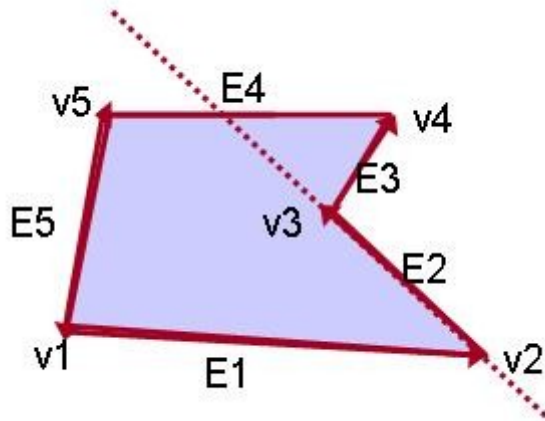


Örneğin yukarıdaki ikinci çokgen tasvirinde ise ilkinden farklı olarak mütekavvis (iç bükey, convex) yapıdaki bir çokgen resmedilmiştir.



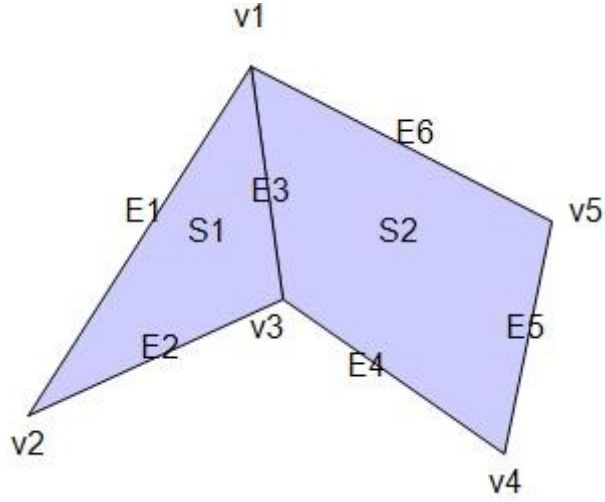
Dolayısıyla bir çokgenin yukarıdaki gibi köşelerini belirten birer vertex (nirengi , üç boyutlu nokta) ve aynı zamanda her kenarı belirten birer vektör (yöney) ile gösterilmesi mümkündür.

Bu vektörler (yöney) bir çokgenin içbükey (mütekavvis) veya dışbükey olup olmadığını anlamak için kullanılabilir. Buna göre şayet yöneylerden birisi kendi doğrultusunda veya eksi doğrultusunda başka bir yöneyi keserse bu durumda şekil mğtekavvistir denilebilir. Örneğin aşağıdaki şekilde böyle bir yöney bulunmaktadır:



Görüldüğü üzere E2 yöneyi E4 yöneyini kesmektedir.

Çokgenleri bilgisayarlar üzerinde ifade edebilmek için çeşitli tablolardan istifade edilebilir. Örneğin aşağıdaki çokgenin tabloları yine tasvirin altında verilmiştir:



Vertex Tablosu

v1: x1, y1, z1
v2: x2, y2, z2
v3: x3, y3, z3
v4: x4, y4, z4
v5: x5, y5, z5

Kenar Tablosu

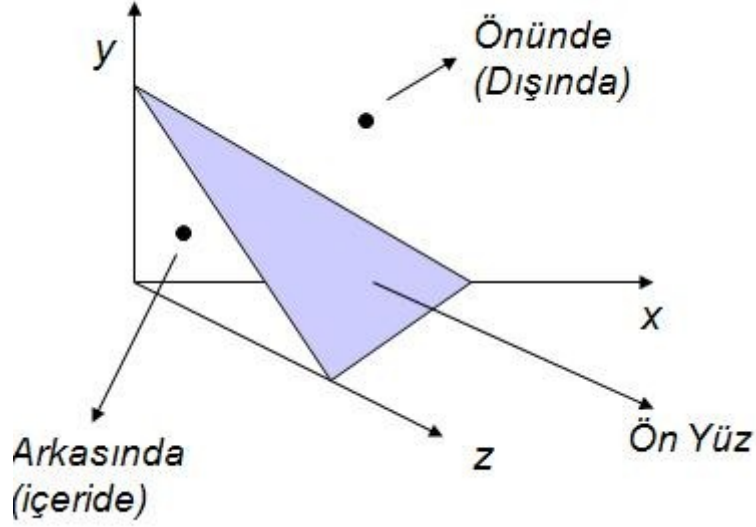
E1: v1, v2
E2: v2, v3
E3: v1, v3
E4: v3, v4
E5: v4, v5
E6: v5, v1

Yüzey Tablosu

S1: E1, E2, E3
S2: E3, E4, E5, E6

Yukarıda da görüldüğü üzere çokgen 2 yüzeyden (Surface) oluşmakta olup, çokgenin toplam 6 kenarı ve 5 noktası bulunmaktadır.

Bilindiği üzere bir çokgen 2 boyutludur ve 2 boyutlu bir nesnenin ön ve arka yüzleri bulunmaktadır. Buna göre bir noktanın çokgenin dışında, önünde veya arkasında olduğu aşağıdaki şekilde belirlenir:



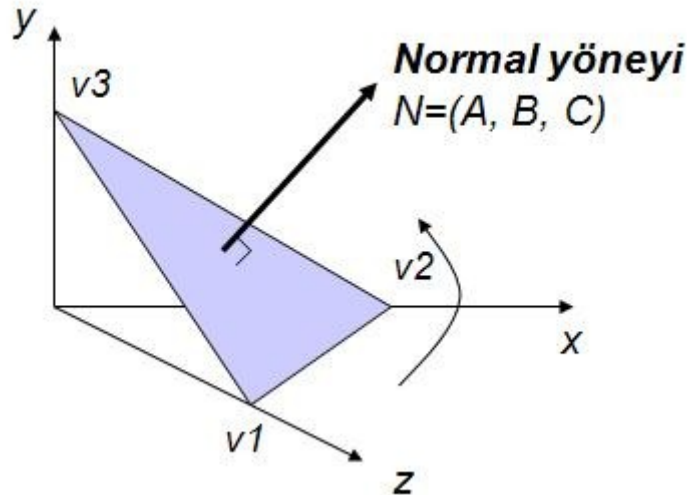
Yukarıdaki resimde de tasvir edildiği üzere bir nokta bir çokgenin önünde veya arkasında bulunabilir. Buna karar vermek için aşağıdaki matematiksel bağlantılardan faydalanılır:

$A.x + B.y + C.z + D = 0$ ise çokgenimizin üzerinde

$A.x + B.y + C.z + D < 0$ ise çokgenin arkasında

$A.x + B.y + C.z + D > 0$ ise çokgenin önünde demektir

Yukarıdaki örnekte bir noktanın, çokgenin neresinde bulunduğu gösterilmiştir ancak bir de çokgenin ön ve arka yüzlerine karar verilmesi sorunu bulunmaktadır. Bu sorunun çözümü için normal yöneyinden (normal vector) faydalanılır:



yukarıdaki resimde de tasvir edildiği üzere çokgeni teşkil eden yöneylerin bileşkesi alınmış ve normal yöneyi bulunmuştur. (elektronikteki manyetik akımın bulunması için kullanılan sağ el kuralına benzetilebilir). Bu işlemin sayısal karşılığı:

$$N = (v_2 - v_1) \times (v_3 - v_1)$$

$$N.P = -D$$

SORU-62: Open GL hakkında bilgi veriniz.

Open GL, bilgisayar grafikleri için kullanılan bir 3 boyutlu geliştirme platformudur. Platform çeşitli dillerde kod yazılmasına izin verir (örneğin JAVA, C, C++ gibi) ve platformun dillerden bağımsız olarak kendisine özgü bir fonksiyon kütüphanesi bulunur.

Açık kaynak kodlu olan platformun farklı ortamlarda rahatça kullanılabilirliği geniş bir kullanım alanına sahip olmasını sağlamıştır. Dillerden bağımsız olarak pek çok işletim sistemi ve geliştirme ortamında da kod yazılabilmektedir. Platformun çalışabilmesi için bilgisayarda bulunan ekran kartının desteklenmesi gerekmesi bazı kurulumlarda problem yaratmaktadır.

Windows için geliştirme ortamlarından birisi olan DEV C++ bilgisayara başarıyla kurulduktan sonra basit Open GL uygulamaları hızlıca geliştirilebilir. Aşağıda DEV-C++ ile yazılmış basit bir uygulama örneği açıklanmıştır:

```
#include <windows.h>
#include "glutil.h"
```

windows üzerinde glutil kütüphanesi ile kod yazılacaktır. Open GL fonksiyonları bu kütüphanede bulunduğu için include edilir.

```
OpenGL *opengl;
```

OepnGL sınıfından (class) bir gösterici (pointer) tanımlanmıştır. Daha ileride bu gösterici kullanılacaktır.

```
bool init()
{
    glClearColor(0.93f, 0.93f, 0.93f, 0.0f);
    glColor3f(0.0f, 0.0f, 0.0f);

    return true;
}
```

Kodun init fonksiyonudur. ve kod ilk çalıştırıldığında ekranı ve renk sistemini temizlemek için kullanılan fonksiyonları çağırır.

```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINE_STRIP);
        glVertex3f(-1.0f, 1.0f, 0.0f);
        glVertex3f(-0.5f, 0.5f, 0.0f);
        glVertex3f(-0.7f, 0.5f, 0.0f);
    glEnd();
    glBegin(GL_TRIANGLES);
        glVertex3f(-0.5f, -0.5f, 0.0f);
        glVertex3f(0.5f, -0.5f, 0.0f);
        glVertex3f(0.0f, 0.5f, 0.0f);
    glEnd();
}
```

```

glBegin(GL_QUADS);
    glVertex3f(-1.0f, -1.0f, 0.0f);
    glVertex3f(-0.5f, -1.0f, 0.0f);
    glVertex3f(-0.5f, -0.5f, 0.0f);
    glVertex3f(-1.0f, -0.5f, 0.0f);
glEnd();
glBegin(GL_POLYGON);
    glVertex3f( 0.8f, 0.6f, 0.0f);
    glVertex3f( 0.9f, 0.8f, 0.0f);
    glVertex3f( 0.5f, 0.9f, 0.0f);
    glVertex3f( 0.6f, 0.5f, 0.0f);
glEnd();
glFlush();
}

```

Yukarıdaki kod display fonksiyonudur. Bu kodun içerisinde basit bazı varlıklar çizilmiştir. Sırasıyla her glBegin ve glEnd arasında kalan grupta bir doğru, üçgen, dörtgen ve çokgen (poligonI) çizilmiş ve bu çizme işlemi için geçerli olacak köşe noktalarının koordinatları verilmiştir.

```

void idle()
{
}

```

Open GL için animasyonlarda anlamlı olan fonksiyon, bu uygulama için boş bırakılmıştır.

```

int WINAPI WinMain(HINSTANCE hInstance,
HINSTANCE hPrevInstance, LPSTR lpCmdLine,
int nShowCmd)
{
    opengl = OpenGL::create();
    opengl->setInitFunc(init);
    opengl->setDisplayFunc(display);
    opengl->setIdleFunc(idle);
    if (!opengl->initGL("05 - Primitives", 500, 500, false, 32))
        return 1;
    opengl->runMessageLoop();
    return 0;
}

```

windows.h dosyası include edilerek oluşturulan uygulamamızda klasik c, c++ programlamasından farklı olarak main() fonksiyonu yerine windows'un main fonksiyonu olan WinMain fonksiyonu kodlanmıştır. Kodun başında tanımlanmış olan ve OpenGL sınıfından bir gösterici olan opengl değişkeni burada değer kazanmış ve OpenGL kütüphanesinden create() fonksiyonunu çağırarak çalışma ortamını hazırlamıştır.

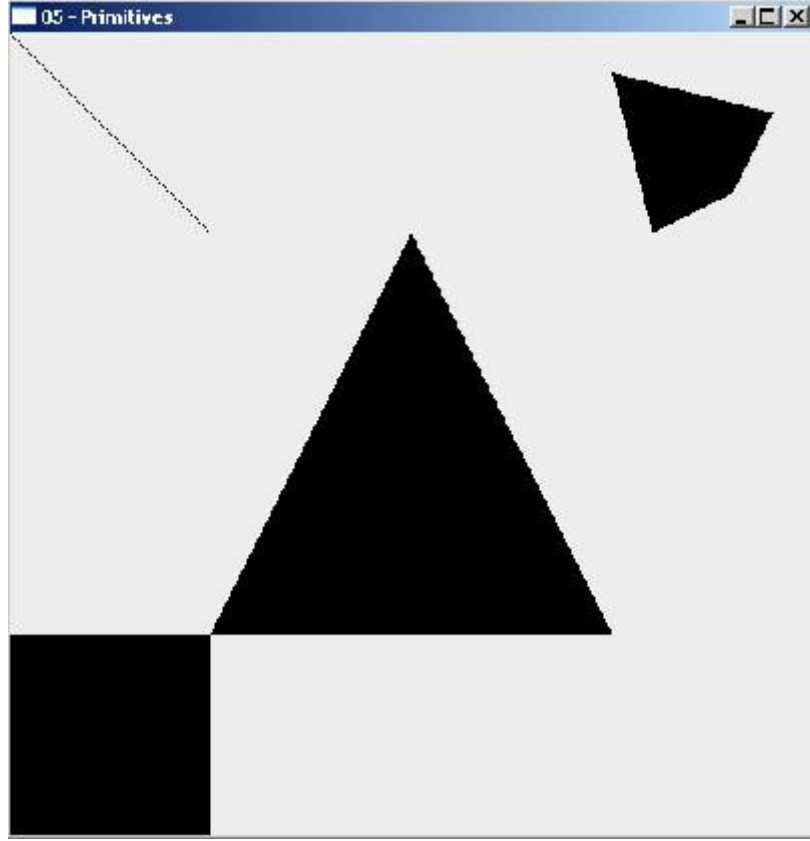
Daha sonra kullanılacak olan özel fonksiyonlar belirlenmiştir. Buna göre OpenGL ortamının init, display ve idle fonksiyonlarının kodumuzdaki hangi fonksiyonlara karşılık geldiği belirtilmiştir.

Bu 3 fonksiyon dışında da özel fonksiyonlar bulunmakta ve bunlar da benzer tanımlamalar ile koddaki başka fonksiyonlara bağlanabilmektedir. (örneğin klavye işlemlerinin hangi fonksiyon tarafından algılanacağını belirlemek gibi)

if kontrolü içerisinde bir opengl penceresi açılmış pencerenin başlığına “05 – Primitives” yazısı yazılmış, pencerenin boyutu (eni ve boyu) 500’e 500 olarak belirlenmiştir.

son olarak opengl göstericisinin runMessageLoop fonksiyonu çağırılarak opengl’in çalıştığı ana döngü tetiklenmiş ve return 0 ile de kod bitmiştir.

Çalışan kod aşağıdaki şekildedir:



SORU-63: Yumuşatma (Antialiasing) hakkında bilgi veriniz.

Bilgisayar grafiği ve görüntü işleme konularında sıkça kullanılan ve görüntünün daha yumuşak hatlar ile görülmesini sağlayan yöntemdir. Aslında basit bir interpolasyon yöntemidir ve işlem yapılan ortamdaki renk farklılıklarını yumuşatmak için renk geçişi çok yüksek olan imgecikler (pixel) arasında orta noktalara orta renkleri koyarak yumuşatma işlemi yapar.

Aşağıdaki iki resimde bu durum görülmektedir:



Kahirede, sfenks

Yukarıdaki resmin yumuşatılmış hali aşağıdadır:



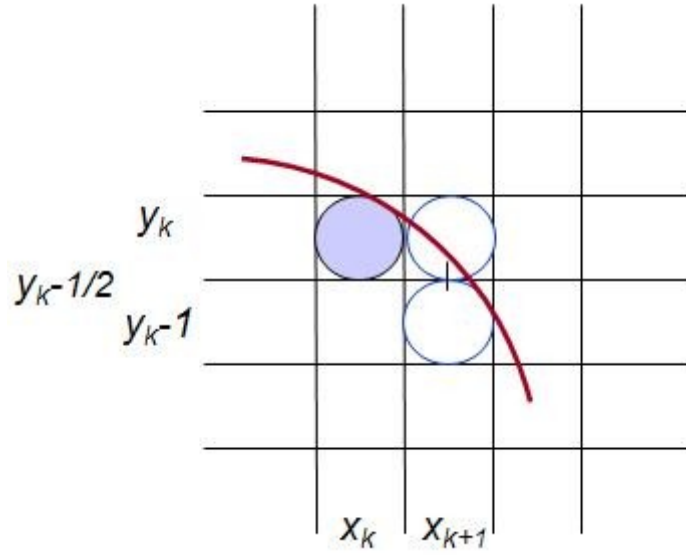
Kahire, sfenks önünde çekilmiş ve yumuşatılmış resim

Yukarıdaki resme dikkat edilirse, düz hatlar yumuşatılmış ve komşu olan imgecikler ile ortak noktaları alınmış yani bu imgeciklerin renk değerine yaklaştırılmıştır. İlk resimde ise düz hatların arka plandan daha net ayrılması söz konusudur.

SORU-64: Ortanokta Çember Çizimi (Circle Drawing with Midpoint Algorithm) hakkında bilgi veriniz.

Bilgisayar grafiği konusunda çember çizmek için pisagor yöntemi ve kutup koordinat yöntemi ile birlikte kullanılan yöntemlerden birisidir. Buna göre merkezi ve yarı çapı belirli bir

çemberi çizmek için hangi koordinatlardaki imgeciklerin (pixel) işaretleneceği (veya imgeciklerin hangi koordinatlara konulacağını) belirler.



Yukarıda bir çember ve bu çemberin geçtiği imgecikler tasvir edilmiştir. Bu algoritma bir çember çizilirken hangi imgeciklerin işaretleneceğini aşağıdaki matematiksel yöntem ile belirler:

Öncelikle çemberdeki (x,y) değerlerini veren fonksiyonu hatırlayalım:

$$f(x,y) = x^2 + y^2 - r^2$$

Bu fonksiyonun değeri aşağıdaki 3 ihtimalden birisi olabilir ve anlamı da yanında verilmiştir:

- Şayet $f(x,y) < 0$ ise değer çemberin içindedir
- Şayet $f(x,y) = 0$ ise nokta çemberin üzerindedir
- Şayet $f(x,y) > 0$ ise nokta çemberin dışındadır.

Bu denklemde $x > y$ durumu söz konusuysa simetriden faydalanılabilir.

Bu durumda çember çizilmesi sırasında yukarıdaki denkleme göre bir imgecik hesaplandıktan sonra bu imgeciğin x,y değerinin yakınındaki imgeciklerin merkezine olan mesafesine bakılır ve şayet yakınsa bu imgecik çember üzerinde olarak işaretlenir.

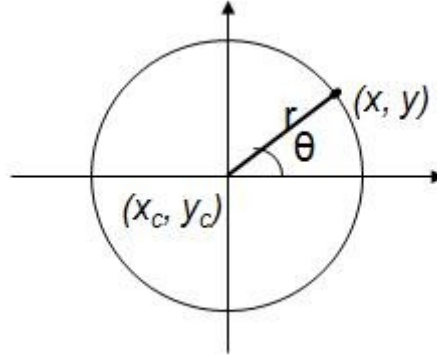
SORU-65: Kutup Koordintaları ile Çember Çizimi (Circle Drawing with Polar Coordinates) hakkında bilgi veriniz.

Bilgisayar grafikleri konusunda çember çizmek için kullanılan pisagor yöntemi ve orta nokta yöntemi ile birlikte alternatif algoritmalarından birisidir.

Bu algoritmaya göre verilen bir merkez koordinatlarında verilen yarıçapta bir çember çizmek için yapılması gereken yöntem incelenmektedir. Bilgisayar garfiği konusunda bir şekli çizmek için veya verilen bir noktanın bu şekil üzerinde olduğunu anlamak için kullanılan yöntem

aynıdır. Buna göre bir döngü yapısı içerisinde belirli bir matematiksel yöntem ile çemberi oluşturan noktalar teker teker kodlanır veya bir noktanın bu formülü sağlayıp sağlamadığı tecrübe edilebilir.

Bir çemberi ve bu çember üzerindeki bir noktayı kutupsal koordinat (polar coordinates) yardımı ile aşağıdaki şekilde gösterebiliriz:



Bu gösterimde merkezi ifade eden bir nokta ve bu noktanın koordinatları, çemberin yarı çapı ve çember üstündeki noktamızın X eksenini ile yaptığı açı verilmiştir. Bu bilgiler ile noktamızın koordinatlarını aşağıdaki matematiksel yöntem ile elde edebiliriz:

Noktamızın x koordinatını veren denklem için:

$$x = x_c + r \cdot \cos \theta$$

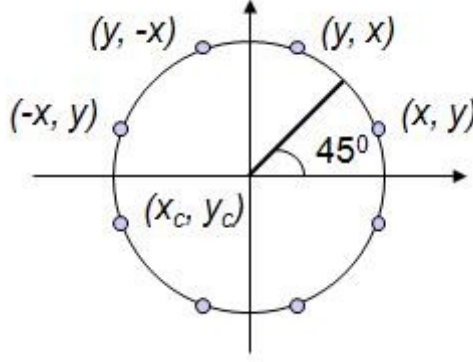
Noktamızın y koordinatını veren denklem için:

$$y = y_c + r \cdot \sin \theta$$

Buradaki (x, y) koordinatları verilen θ değeri için bulunur. Şayet θ değerini arttıran bir döngü ile noktalarımızın (x, y) koordinatlarını hesaplarsak bu döngü aslında çemberimizi de çizen döngü olmuş olur.

Bu döngüdeki θ artım miktarı olarak $1/r$ değeri kullanılması tavsiye edilmektedir.

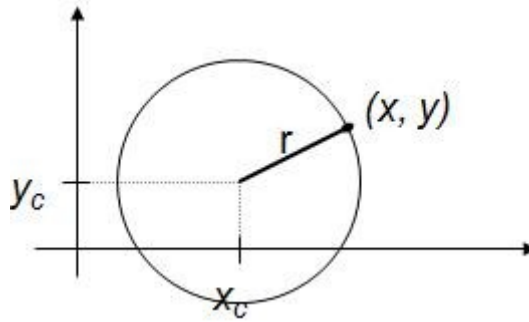
Ayrıca aşağıdaki tasvirde gösterilen bütün noktalar birbirinin simetriği olmaktadır. Bu durumda çizmek istediğimiz bölgenin tespit edilmesi ve bu bölgeye göre bir simterik hesaplama yapılması gerekmektedir.



SORU-66: Pisagor Yöntemi İle Çember (Pythagorean Theorem in Circle Drawing) hakkında bilgi veriniz.

Bilgisayar grafiklerinde çember çizmek için kullanılan yöntemlerden birisidir. Diğer çok kullanılan yöntemlerden birisi de kutup koordinatları (polar coordinates) kullanmaktır.

Bu yöntemde, bir çemberin formülü alındıktan sonra bu çember üzerindeki imgeciklerin (pixel) nasıl bulunacağı çözüme ulaştırılmıştır. Örneğin aşağıdaki gibi bir çemberi ele alalım:



Yukarıdaki bu resimde merkezi belirli olan (x_c, y_c) ve r yarıçapındaki bir çemberin üzerinde bulunan herhangi bir noktanın (x, y) koordinatları tasvir edilmiştir. Bu (x, y) imgeciğini hesaplamak için aşağıdaki pisagor bağlantısı üzerine inşa edilmiş matematiksel yöntem kullanılabilir:

Öncelikle çemberin formülünü hatırlayalım:

$$x^2 + y^2 = r^2$$

Bu formülden yola çıkarak şayet çember x_c ve y_c kadar kaydırılmışsa aşağıdaki formül elde edilir:

$$(x-x_c)^2 + (y-y_c)^2 = r^2$$

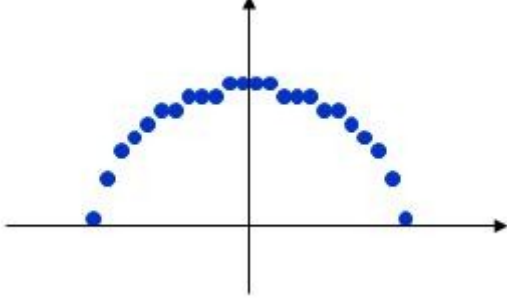
yukarıdaki formülümüzden y değeri çekilirse aşağıdaki sonuç elde edilir:

$$y = y_c \pm \sqrt{r^2 - (x-x_c)^2}$$

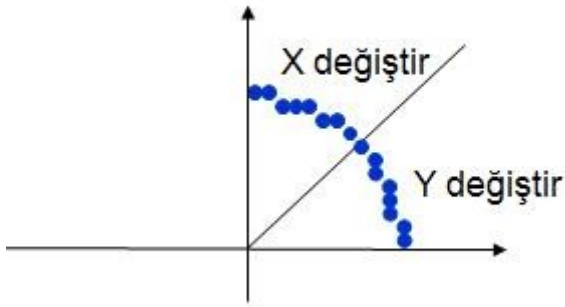
Ayrıca şekilden de anlaşılacağı üzere çember üzerindeki bir x noktası hem merkeze r eklenmiş hem de r çıkarılmış halidir, bunu da aşağıdaki şekilde gösterebiliriz:

$$(x_c-r) \leq x \leq (x_c+r)$$

Bu matematiksel bağlantıları kullanarak verilen bir imgecik koordinatlarının çember üzerinde olup olmadığı bulunabileceği gibi çember üzerindeki imgecikler de hesaplanabilir. Bu hesaplamalar sonucunda da aşağıdakine benzer bir çember elde edilir.



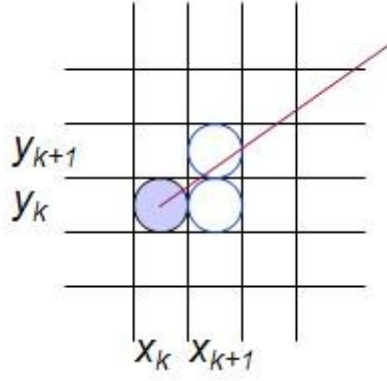
Bu çemberde dikkat edilirse doğrudan çizim algoritması (direct drawing algorithm) benzer bir problem ortaya çıkmaktadır. Bu problem de çember dikleştikçe kodlanan imgecik sayısının azalmasıdır. Problem bahsi geçen algoritmadakine benzer şekilde eğim değişince algoritmanın farklı davranması sağlanarak çözülebilir.



Bu çözüm sayesinde yukarıdaki gibi daha yoğun imgecikler elde edilebilmektedir.

SORU-67: Bresenham Doğru Çizim Algoritması (Bresenham's Algorithm) hakkında bilgi veriniz.

Bilgisayar grafiği konusunda kullanılan bir doğru çizme algoritmasıdır. İki veya üç boyutlu ortamlarda kullanılabilir. Buna göre başlangıç ve bitiş noktalarının koordinatları belirli bir doğruyu çizmek için nasıl bir yol izleneceğini belirler ve bu doğrunun geçtiği imgeciklerin (pixel) hesaplanmasında kullanılır.



Örneğin yukarıdaki tasvirde bir doğrunun geçtiği imgeciklerin belirlenmesinde nasıl bir yol izlendiği gösterilmiştir. Buna göre bir doğrunun hangi imgeciklerden geçtiğini bulmak için öncelikle doğrunun eğimi hesaplanır. Ardından bu eğim miktarı başlangıç imgeciğinin koordinatlarına eklenerek sonuç imgeciğine kadar olan yolda bulunan bütün imgeciklerin koordinatları hesaplanır. Burada dikkat edilecek olan nokta eğim miktarının tam sayı olmaması ve imgeciklerin küsurat değerlerini göstermemesidir. Bu sorunun çözümü olarak değer en yakın tam sayıya yuvarlanır. Bu tam sayıların imgeciklerin merkez noktaları olduğu düşünülürse, aslında brehensam algoritması doğrunun geçtiği yoldaki imgeciklerin merkezlerine en yakınlarını işaretlemiş olur.

Yukarıda anlatılan bu hesaplama yöntemi için öncelikle eğim bulunmalıdır. Eğim için

$$m = (y_{\text{son}} - y_{\text{başla}}) / (x_{\text{son}} - x_{\text{başla}})$$

Formülü kullanılabilir.

Şimdi hesaplamak istediğimiz ve doğru üzerinde olan herhangi bir imgeciğin x ve y koordinatlarını bulalım. Bu imgeciğin aşağıdaki formülü sağlaması gerektiği açıktır:

$$y - y_{\text{başla}} = ((y_{\text{son}} - y_{\text{başla}}) / (x_{\text{son}} - x_{\text{başla}})) (x - x_{\text{başla}})$$

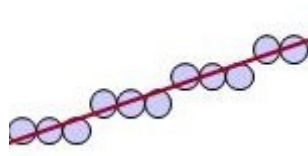
Bu formülden örneğin imgeciğin y değeri çekilirse :

$$y = ((y_{\text{son}} - y_{\text{başla}}) / (x_{\text{son}} - x_{\text{başla}})) (x - x_{\text{başla}}) + y_{\text{başla}}$$

olarak hesaplanır. İşte bu yöntem kullanılarak imgecik koordinatları hesaplanmakta ve nihayetinde bir doğrunun çizilmesi mümkün olmaktadır.

SORU-68: Doğrudan Çizim Algoritması (Direct Draw Algorithm , DDA) hakkında bilgi veriniz.

Bilgisayar grafiği konusunda bir doğru çizmek için kullanılan algoritmalarındandır. Algoritma 2 veya 3 boyutlu olarak uygulanabilir. Algoritma basitçe bir doğrunun geçeceği yatay hatları işaretleyerek çalışır. Örneğin aşağıdaki tasvirde bir doğru gösterilmiş ve bu doğrunun hangi imgecikleri (pixel) işaretleyeceği gösterilmiştir.



Yukarıda da görüldüğü üzere bir doğru, yatay eksene paralel veya yatay eksene yakın bir eğimdeyse net olarak görülebilmektedir. Ancak dikey eksene paralel bir doğru ancak iki noktadan (başlangıç ve bitiş noktaları) oluşmaktadır. Bu durumda DDA algoritması çok başarılı olamamaktadır. Bu durumun iyileştirilmesi için de doğrunun eğimi hesaplanmakta ve eğim 45 dereceden yüksekse bu defa işaretlenen imgecikler dikey eksene paralel olarak alınmaktadır.

Bu algoritmanın çalışmasını matematiksel olarak şöyle anlatabiliriz. Basitçe bir doğruyu veren formül iki boyutlu düzlem için aşağıdaki şekildedir:

$$y = m x + b$$

Buradaki y ve x değerleri koordinat sistemindeki değişken eksenler, m eğim ve b ise kayma miktarıdır. Örneğin eğim için:

$$m = (y_{\text{son}} - y_{\text{başla}}) / (x_{\text{son}} - x_{\text{başla}})$$

şeklinde hesaplanabilir. Dolayısıyla eğimin aslında yatay eksenle yapılan açının tanjantı olduğunu hatırlayalım. DDA algoritması bu eğim değerini işaretleyeceği imgecikleri (pixel) belirlerken kullanır.

Şayet eğim 1'den küçükse yani $|m| < 1$ ise:

$$x_{k+1} = x_k + 1$$

$$y_{k+1} = y_k + m$$

Şayet eğim 1'den büyükse yani $|m| > 1$ ise:

$$y_{k+1} = y_k + 1$$

$$x_{k+1} = x_k + 1/m$$

olarak bir sonraki imgeciğin (piksel) hesaplanması mümkün olmaktadır. Buna göre DDA algoritmasına başlangıç ve bitiş koordinatları verilen iki nokta arasında doğru çizmesi istendiğinde bu iki nokta arasındaki eğimi ve bu eğime göre doğrunun geçtiği her imgeciğin (pixel) koordinatlarını hesaplayarak çizmektedir.