ANADOLU ÜNİVERSİTESİ BİLİM VE TEKNOLOJİ DERGİSİ –A
Uygulamalı Bilimler ve Mühendislik

ANADOLU UNIVERSITY JOURNAL OF SCIENCE AND TECHNOLOGY –A
Applied Sciences and Engineering
Cilt/Vol.:11-Sayı/No: 1 : 23-33 (2010)

ARAŞTIRMA MAKALESİ /**RESEARCH ARTICLE**

BİLGİSAYAR OYUNLARINDA KALABALIKLARIN HIZLI ÇİZİMİ

Kaya Oğuz [1]

ÖZ

Bilgisayar oyunları, özellikle grafik işlemcilerin de hızlanması ile, görsel olarak sinema endüstrisinin kalitesine yetişmeye başlamıştır. Sinemadaki çevrim dışı hazırlanan sahnelere nazaran, bilgisayar oyunlarında bu işlem akıcı bir şekilde saniyede en az 30 kere yapılmalıdır. Bu yüzden, kalabalık gerektiren sahnelerde çeşitli yöntemlerle işlemci ve bellekten başarım alınmaya çalışılır. Bu makale, güncel grafik işlemcilerin donanımsal bir özelliği olan *örnekleme* tekniği ile, grafik işlemlerinde eskiden beri kullanılan *ayrıntı düzeyi* yöntemlerinin birleştirilmesi hakkındadır. Örnekleme tekniği sayesinde, 15,000 karakter, %10 gibi düşük işlemci kullanımının yanında, saniyede 30 kare hızla ekrana çizdirilmiştir. Uygulama, 40,000 karakteri saniyede 13 karede basabilmektedir.

Anahtar Kelimeler : Kalabalık çizimi, Grafik gölgelendiriciler, Ayrıntı düzeyi, Örnekleme.

FAST CROWD RENDERING IN COMPUTER GAMES

ABSTRACT

Computer games, with the speed advancements of graphical processors, are coming closer to the quality of cinema industry. Contrary to offline rendering of the scenes in a motion picture, computer games should be able to render at 30 frames per second. Therefore, CPU and memory performance are sought by using various techniques. This paper is about using *instancing* feature of contemporary graphical processors along with *level of detail* techniques which has been in use for a very long time. Using instancing, 15,000 instances were successfully rendered at 30 frames per second using a very low %10 CPU usage. The application can render 40,000 instances at 13 frames per second.

Keywords: Crowd rendering, Shaders, Level of detail, Instancing.

1. GİRİŞ

Kalabalık çizimleri, bilgisayar oyunlarında her zaman daha fazla işlemci gücüne ihtiyaç duyan alanlardan biridir. Kalabalıkların oyunlara kattıkları gerçeklik duygusu, birçok oyun türü için onları vazgeçilmez kılmaktadır. Sanal şehirler sokaklarında gezen insanlara, futbol stadyumları da takımları için tezahürat yapan taraftarlara ihtiyaç duyarlar. Elbette, bu kadar çok sanal karakteri yönetmek ve çizmek yüksek işlemci gücü ve bellek gerektirir.

Bu makale, güncel grafik işlemcilerin donanımsal bir özelliği olan *örnekleme* tekniği ile, grafik işlemlerinde eskiden beri kullanılan *ayrıntı düzeyi* yöntemlerinin birleştirilmesi hakkındadır. Örnekleme tekniği sayesinde, 15,000 karakter, %10 gibi düşük işlemci kullanımının yanında, saniyede 30 kare hızla ekrana çizdirilmiştir. Uygulama, 40,000 karakteri saniyede 13 karede basabilmektedir.

Makalenin ilk bölümünde kalabalık çizimleri için kullanılan yöntemlere genel bir bakış ile yapılan diğer çalışmalara değinilmiştir. İkinci bölümde seçilen yöntemler daha detaylı bir şe-

[1] İzmir Ekonomi Üniversitesi, Bilgisayar Bilimleri Fakültesi, Yazılım Mühendisliği Bölümü, Sakarya Cad. No:156, 35330 Balçova, İzmir.
E-posta: kaya.oguz@ieu.edu.tr

Geliş: 24 Ağustos 2009; **Düzeltme:** 30 Kasım 2009; **Kabul:** 13 Nisan 2010

kilde incelenmiştir. Üçüncü bölümde gerçekleme adımları, senaryo seçimi, geliştirme

kütüphaneleri, ortam dosyaları gibi programlama dışı bölümlerle, programlamanın gerçekleştiği gerçekleme konuları anlatılmıştır. Dördüncü bölümde yazılan uygulama ile yapılan testleri ve sonuçlarını işlemci ve grafik işlemci performansını irdelenmiştir. Son bölümde ise sonuçlar ve ilerisi için yapılabilecekler yer almaktadır.

1.1 Yöntemler

Kalabalık çizimi için seçilen tekniğin başarım kalite oranı, kalabalığın davranışlarına ve kişi sayısına bağlıdır. Sanal bir şehrin sokaklarında gezen yüz kişi, kameraya daha yakın ve kullanıcı ile etkileşime girebilecekken, stadyumdaki kalabalıkta onbinlerce kişi kullanıcı ile birebir etkileşime girmeden oyunu izliyor olabilirler. İlk senaryodaki kalabalık karakterleri daha yüksek kalite ile çizilip, daha detaylı bir yapay zekaya sahip iken, ikinci senaryoda, özellikle uzaktaki taraftarların, çok küçük olan görüntülerinde yüksek çözünürlük ve bir yapay zeka gerekmeyebilir.

Ayrıntı Düzeyi (Level of Detail) yöntemi çok eskiden beri grafik uygulamalarında kullanılmaktadır. Bu yöntem, basitçe, çizim yapılırken ekranda az yer kaplayan, önemsiz olan, ya da uzaktaki modeller için daha düşük ayrıntılı bir gösterim kullanılmasını önerir (Clark, 1976). Bu düzeyler genelde belli aralıklara bölünür ve her örnek dahil olduğu ayrıntı düzeyine göre çizilir.

Başarım kazandıran başka bir yöntem ise *pano*lardır (billboards). Panolar, üç boyutlu bir sahnede bulunan iki boyutlu dörtgenlerdir. Bu dörtgenler devamlı kameraya dönük olurlar ve üzerlerinde yerine geçtikleri modelin bir görüntüsü bulunur. Daha çok, modellemesi zor olan, ateş ve yıldırım gibi olaylar için kullanılırlar. Kalabalık çizimlerinde ise bu karelerde *sahte* karakterler yer alır (impostors) (Aubel vd, 1998). Bu panolar için, karakterlerin canlandırmalarının her karesini belli açılardan içeren görüntü dosyaları kullanılır.

Örnekleme (instancing) grafik işlemcilerin donanım olarak destekledikleri bir yöntemdir (Scott, 2004). Bu yöntem ile grafik kartına iki dizi bilgi gönderilir. Bu dizilerden ilkinde örneklenecek modelin tepe noktaları (vertex) vardır. İkinci dizide ise, bu bilgiler kullanılarak başka yerlere çizilecek her örneğin bilgisi vardır. İkinci dizideki bu bilgiler daha çok dünya yerleşim matrisleri ile o örneğe ait özel bilgilerdir. Grafik işlemci ilk dizideki bilgiyi ikinci dizideki her örnek için kullanır. Bu sayede, grafik kartına

gönderilen tek çizim çağrısı ile birden çok örnek çizilir. Bu yöntem işlemciyi de rahatlatır. İşlemci her örnek için teker teker bilgileri toplayıp bunları grafik kartına göndermek zorunda kalmaz. Grafik kartı da çizimi tek çağrı ile yaptığı için durum değişikliği yapmasına, dolayısı ile zaman kaybetmesine, gerek kalmaz.

1.2 Diğer Çalışmalar

Kalabalık çiziminde kullanılan ayrıntı düzeyi tekniği kesin çizgilerle ayrılmıştır. Belli bir sınırdan sonra 3B model için başka bir gösterim kullanılır. Bu modeller uygulama çalışırken kullanılmak üzere önceden hazırlanır. Fakat *Sürekli Modeller* (Hoppe, 1996) ile bu işlemler uygulamanın çalıştığı sırada yapılır. Direct3D 9 kütüphanesi ile birlikte gelen bu yöntem ile modeller özgün yapılarını kaybetmeden geometrik karmaşıklıklarını arttırıp azaltabilirler. Bu sayede sürekli bir ayrıntı düzeyine sahip olurlar. Elbette, günümüz grafik bağdaştırıcı ve işlemcileri optimize edilmiş, sıralanmış tepe noktaları içeren modellerle daha hızlı çalışırlar. Modelin sürekli değişen geometrik yapısı tekrar optimize edilmesini gerektirmekte, bu da kalabalık çiziminde her birey için daha fazla işlem yükü getirmektedir. Fakat, kameranın çok değişmediği durumlarda Sürekli Modeller ile sürekli bir ayrıntı düzeyi oluşturmak iyi bir çözüm olabilir.

Panolar, 3B uygulamalarda daha çok alev, şimşek gibi modellemesi zor görüntüler için kullanılırlar. Kalabalık için 3B karakterler yerine konan bu *sahte* (impostor) karakterler ile çok düşük geometrik içerik ile çok fazla karakter canlandırılabilir (Aubel vd, 1998). Sahteler çok az geometrik karmaşıklığa sahip olmalarına rağmen, sınırlı sayıda görüş açısına sahiptirler. Bu görüş açılarına ait canlandırmalar, uygulama öncesi kare kare oluştururlar. Genelde büyük bir görüntü dosyası şeklinde olan bu kareler, kameranın açısı ve canlandırmanın ilerlemesine göre seçilir. Görüş açısı ve canlandırmalar arttıkça bu görüntü dosyası da büyür ve bellekte daha fazla yer kaplar. Sahte karakter çizimlerinin gerçekçiliğini arttırmak için yapılan çalışmalar da mevcuttur (Tecchia, 2002). Bu çalışmalar ile, sahte karakterlere gölgeler eklenmiş ve çizim sırasında farklı renkler uygulanıp çeşitlilik arttırılmıştır.

Kalabalık çizimi ile ilgili diğer çalışmalardan öne çıkanlardan bir diğeri ise sahteler ve ayrıntı düzeyi tekniklerini karma bir yöntemde

birleştiren Geopostors tekniğidir (Dobbyn vd, 2005). Bu teknik, kullanılan sahtelerdeki görüntünün belli bir noktadan sonra bozulmaya başlamasından yola çıkıyor. Bu nokta ise, sahte için kullanılan görüntünün kendi boyutlarından daha büyük olarak çizilmeye başladığı an olarak belirlenmiş. Bu noktada görüntünün görüntü elementi ile dokunun doku elementi (teksel) oranı 1:1 değerinden büyük olmaya başlıyor. Görüntü normal boyutundan daha büyük çizilmeye başlandığında yumuşatma devreye giriyor ve görüntü kalitesini kaybetmeye başlıyor. Tam bu noktada, Geopostor uygulaması model için sahte kullanmayı bırakıp üç boyutlu gösterime geçiyor. Bu sayede, daha yakında bulunan modeller üç boyutlu gösterimlerini kullanırken, uzaktakiler panolar ile çiziliyor.

Öne çıkan diğer bir yöntem olan Polypostor yöntemini ise sahte yöntemine farklı bir yaklaşım getirmiştir (Kavan vd., 2008). Polypostor'larda, sahtelerin kullandığı büyük görüntü resimlerindeki birçok karenin aslında birbiri ile çok yakın görüntüler olduğundan yola çıkılarak, canlandırmada farklı bir yöntem izlenmiş. Bir üç boyutlu karakterin model görüntüsünü tamamen bir panoya yerleştirmek yerine, modeli gövde, kollar ve bacaklar olarak beş parçaya ayırıp, bu parçaları iki boyutlu yüzeyler haline getiriyorlar. Canlandırmayı da, görüntüler yerine bu beş parçanın tepe noktalarını yer değiştirerek sağlıyorlar. Canlandırmadaki her karede ise aynı görüntü resmi, bir sonraki kareye yakınlaştırılıyor. Bu şekilde canlandırmada büyük görüntü dosyası boyutları yerine, küçük matris değişimleriyle büyük kazançlar sağlıyorlar. Polypostor'lar, basit yürüme döngülerinde test edilmişler ve başarılı sonuçlar vermişler. Fakat bazı ters açılarda görüntünün bir sonraki kareye yakınlaştırılmasının iyi sonuçlar vermediğini görülmüş.

Sahte karakter yöntemini kullanan çalışmalarda görülüyor ki ön hazırlık aşaması modellerin sadece belli açılardan görüntülenebilmesini mümkün kılıyor. Aynı zamanda programlama olarak daha fazla emek istiyor. Polypostor örneğinde ise modelleme yapılırken canlandırmanın bu yeni iki boyutlu modellerle her açı için yapılması gerekiyor. Bu yoğun ön hazırlıklar yerine örnekleme tekniği ile, ayrıntı düzeyi için gereken ön hazırlık dışında, bir hazırlık gerekmiyor. Programlama olarak da karmaşık hesaplara gerek kalmıyor. Aynı zamanda da görüntü, belli açılar ile sınırlandırılmamış oluyor.

2. YÖNTEMLER

2.1 Ayrıntı Düzeyi

Ayrıntı düzeyi, daha önceden belirlenmiş uzaklık, ekran boyutu gibi kıstaslara göre, ekranın belli yerlerine çeşitli ayarlar uygulayarak, uygulamanın kalite ve başarım oranını dengelenmesini amaçlar (Clark, 1976). Bu yöntem genelde çizilecek model ile kamera arasındaki uzaklığı esas alır. Model uzaklaştıkça, aynı modelin daha düşük tepe noktası içeren bir gösterimi kullanılır. Fakat bu yöntem, programlama sırasında belirlenen ekrana göre olan uzaklıklarla farklı ekran çözünürlüklerinde aynı oranı vermez. Küçük ekranlarda uzakta olduğu için düşük bir seviye kullanılan bir model, daha büyük çözünürlüklerde daha net gözükebilecekken, düşük model yüzünden kötü bir görüntü sunabilir. Bu görüntü sorununu aşmak için, kameraya olan uzaklık yerine modelin ekranda kapladığı alan hesaplanabilir. Fakat bu işlem biraz karmaşık olduğu için, genelde model bir kare ya da daire içine yerleştirilip bu kare ve dairenin alanı hesaplanır. Bu durumda da bazı şekillerin seçilen kare veya daireye düzgün oturmadığı gözlenebilir.

Ayrıntı düzeyleri genelde ayrık olarak belli aralıklara bölünür. Bir uygulamada gerektiği kadar ayrıntı düzeyi olabilir. Fakat ayrıntı düzeyi arttıkça her düzey için modellere yeni bir gösterim hazırlanması gerekir.

Her ne kadar genelde düşük tepe noktası sayısı olarak düşünülse de, ayrıntı düzeyleri modellerin başka özelliklerine de uygulanabilir. Model için kullanılan görüntü dosyasının boyutu küçültülebilir, düşük düzeydeki modeller için gölgelendirici programcıkları daha basitleştirilebilinir, gölge ve görüntü etkileri azaltılabilir ya da tamamen kaldırılabilir. Bu konu üzerine daha detaylı bilgi almak için (Luebke vd., 2003) kaynağına başvurulabilinir.

2.1.1 Uzaklık Kıstası

Daha önce de belirtildiği gibi, ayrıntı düzeyi için uzaklık seçimi çok sık kullanılır. Bunun nedeni hesaplamasının basit olmasıdır. Tipik bir üç boyutlu ortamda cisim ile kamera arasındaki uzaklık için alınan $P(p_x, p_y, p_z)$ ve $Q(q_x, q_y, q_z)$ noktaları arasındaki Öklid Uzaklığı *d*,

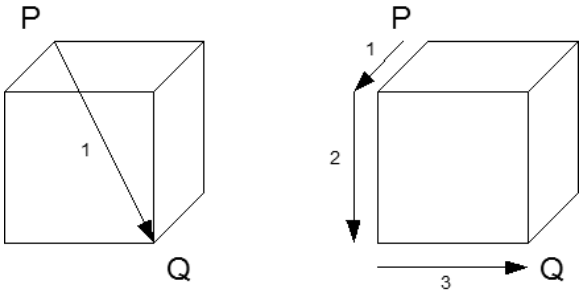
$$d = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2 + (p_z - q_z)^2} \quad (1)$$

ifadesi ile bulunabilir. Her ne kadar basit olsa da, ifade üç kare bir de kare kök alma işlemi içermektedir. Bu işlem, kalabalıktaki her karakter için yapıldığında işlemci için yoğun olacaktır. Yoğun kalabalıklarda, bu ifadeye yakın bir değer verecek olan Manhattan Uzaklığı da he

saplanabilir. Manhattan uzaklığı, Şekil 1 ile görebildiğimiz üzere iki nokta arasındaki mesafeyi dik eksenler üzerinden hesaplar (Paul, 2006).

$$d = |p_x - q_x| + |p_y - q_y| + |p_z - q_z| \qquad (2)$$

Denklem 2 ile de görebileceğimiz üzere, uzaklık daha basit olarak yaklaşık olarak hesaplanabilmektedir.



Şekil 1. Solda Öklid, sağda ise Manhattan uzaklığı gösteriliyor

2.1.2 Zorluklar

Ayrıntı düzeyi ile ilgili akla ilk gelebilecek zorluk, model üzerinde hangi noktanın referans alınacağıdır. Daha kesin bir sonuç için, model üzerinde, kameraya en yakın nokta seçilebilir, ama bu da hesaplamaları arttırmaktadır. Bunun yerine, modelin kendi orijin noktası alınabilir.

Diğer bir sorun ise, bir düzeyden diğerine geçerken oluşan, farkedilebilir bir patlama görüntüsü oluşmasıdır. Bu sorunun üstünden gelmek için geçiş sırasında iki modeli aynı anda kullanıp, birinin alfa değeri, böylelikle de görünebilirliliği, düşürülüp ekrandan kaybolması sağlanırken, diğerinin alfa değeri sıfırdan başlayıp arttırılarak ekranda belirmesi sağlanabilir. Ama bu da tam sınırdaki karakterler için daha fazla veri kullanılmasını sağlar. Diğer bir yöntem ise kesintisiz dönüşümdür. Bu yöntem ile eski modelin tepe noktaları yeni modele dönüştürülür. Bu alfa karışımı tekniğinden daha karmaşık bir işlemdir ama iki model arasında fark edilemeyecek bir geçiş sağlar (Luebke vd., 2003).

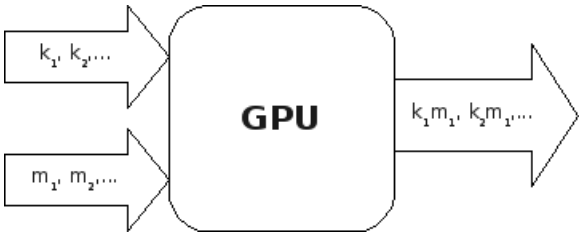
2.2 Örnekleme

Örnekleme, grafik kartlarının donanımsal olarak desteklediği bir özelliktir. Örnekleme sayesinde tek bir çizim çağrısı ile aynı modelin birden fazla örneği çizdirilebilir. Çizim çağrısı, merkez işlemcinin grafik işlemciye bilgi göndermesini sağlar. Bu da sahnede çok model var

sa uygulamada bir dar boğaz oluşturur. Fakat çizim çağrıları değişen dünya matrisleri ve grafik kartının durum değişimleri için gereklidir. Örnekleme bu noktada işlemciyi ve dar boğazı kurtarır (Dudash, 2004).

Örnekleme için iki tampon dizisi kullanılır. Bunlardan ilki örneklenecek modelin, tepe nokta koordinatları gibi, bilgilerini içerir. İkinci tampon dizisi ise örneklerin dünya matrisleri ve gerekli olabilecek diğer bilgileri içerir. Şekil 2 ile de görebildiğimiz üzere ilk dizideki tepe noktaları $k_1, k_2, k_3,...$ şeklinde, ikinci dizideki matrisleri ise $m_1, m_2, m_3,...$ şeklinde göstermek istersek, grafik işlemci öncelikle ilk dizideki k_1 ile ikinci dizideki m_1 matrisini kullanır. Daha sonra k_2 ile m_1 matrisini kullanır ve bu böyle ilk dizideki değerler bitene kadar devam eder (Dudash, 2004).

Her ne kadar bu işlem bedavaya gelmese de, genel durumlarda kazanç sağlamaktadır. Eğer uygulama merkez işlemciye yük getiriyorsa ve aynı modelin bir sahnede birden fazla kopyası varsa, örnekleme kullanmak kazanç sağlayabilir. Elbette, her grafik işlemcinin bir üst sınırı vardır.



Şekil 2. Örnekleme işleminin grafiksel gösterimi

2.3 Gölgelendirici Programcıklar

Sürekli gelişen görüntü bağdaştırıcıları 3B uygulamalardaki gerek duyulan başarımı sağlayabilmek için artık bir grafik işlemci (GPU) içermektedirler. Bu işlemci sayesinde sabit olan 3B grafik çizim hattına müdahele edilebilmektedir. Bu müdaheleler için işlemci üzerinde çalışan minik programcıklara gölgelendirici adı verilmektedir. Bu programcıkların çalıştığı kısımlar programlanabilir grafik çizim hattı bölümleri olarak isimlendirilirler ve en sık kullanılanları

tepe noktası gölgelendiricileri ve benek gölgelendiricileridir.

Tepe noktası gölgelendiricileri, çizim için gerekli olan tepe noktalarının her biri için yürütülür. Grafik çizim hattının bu noktasına müdahele ile 3B modelin yapısına müdahele etmiş oluruz. Bu sayede dalgalanma, şekil değiştirme gibi özel etki içeren işlemler doğrudan GPU üzerinde gerçekleştirilebilir.

Benek gölgelendiricileri ise, modele uygulanan doku üzerindeki her benek için çalışırlar. Genelde ışıklandırma etkileri için kullanılırlar. Bu sayede sabit grafik çizim hattında yer alan çeşitlere bağlı kalınmadan ışıklandırma yapılabilir.

Elbette, benek gölgelendirici programcıkları tepe noktası gölgelendirici programcıklarından çok daha fazla çalışırlar. Bunun nedeni bir modeldeki tepe noktası sayısının benek sayısından çoğu zaman daha az olmasıdır.

DirectX'in yeni sürümleri ile geometri gölgelendiricisi de grafik çizim hattına eklenmiştir. Tepe noktası gölgelendiricilerine benzeyen bu gölgelendirici, tek bir tepe noktası yerine bir grup tepe noktası üzerinde çalışır ve çıktı olarak tepe noktalarının sayısının arttırabilir. Bu sayede modele yeni tepe noktaları eklenebilir.

3. GERÇEKLEME

3.1 Uygulama Senaryosu

Kalabalığın dahil olabileceği birçok senaryo mevcuttur. Kullanılabilecek en yüksek kişi sayısına ulaşmak için stadyum kalabalıkları bu uygulama için uygun olacaktır. Bu tip kalabalıklar genelde kullanıcı ile etkileşime girmezler ve genel olarak bir yapay zekaları yoktur. Gerçekçiliği arttırmak adına, arka planı süsleyen bu kalabalıklara küçük etkileşimler eklenebilir. Bunun için kalabalığın ortada oynanacak spor olayını takip etmesi sağlanabilir. Bu uygulamada bir futbol ya da başka bir karmaşık oyun programlamak yerine, sahada bir Pong oyunu canlandırılacaktır. Kalabalık ise, yüzlerini her zaman diskin olduğu yere çevirecekler, bu şekilde daha gerçekçi bir şekilde oyunu izliyormuş görünümü verebilirler. Pong oyunundaki yapay zeka oyuncular ise her zaman topu karşılayacak şekilde programlanmış oldukları için, karşılaşma sürekli devam edecektir.

Stadyum kalabalıklarının bir diğer özelliği ise yürümek yerine, sabit bir şekilde oyunu izlemeleridir. Bu, elbette, hiç hareket etmedikleri anlamına gelmez. Kendi senaryomuzda, taraftarlar aşağı yukarı hareket ederek bir zıplama izlenimi verecekler. Bunun dışında, az önce de belirtildiği gibi oyunu izlemek için diski takip edecekler ve kendi *y* eksenleri etraflarında diske doğru döneceklerdir.

Yine bir stadyumda, her kişinin aynı olması gibi bir durum olamaz, herkes farklı bir şekilde gözükür, giyinir. Kendi senaryomuzda bunu sağlayabilmek için benek gölgelendiricilerini kullanacağız.

3.2 Geliştirme Kütüphaneleri

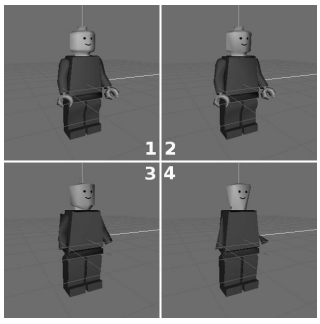
Uygulama, Direct3D 10 ile beraberinde gelen DXUT uygulama çatısını kullanarak hazırlanmıştır. Direct3D'nin bu son sürümü sabit grafik boru hatları yerine programlanabilir grafik boru hattı yapısını zorunlu kılmaktadır. Eski sürümlerde sunulan tepe noktası ve benek gölgelendirici programcıklarına ise geometrik gölgelendirici aşamasını da eklemiştir. Bu sürüm ile, örnekleme yaparken kullanabileceğimiz *SV_InstanceID* gibi yeni sistem değişkenleri de eklenmiştir.

Bu özellikler kalabalık çizimine yardımcı olsa da, kütüphane hala geliştirilme aşamasında olduğu için eksikleri de uygulamaya zorluklar çıkarmaktadır. Önceki sürümlerinde kendi içinde fonksiyonlarla da desteklediği X dosya biçimi yerine SDKMesh adı verilen yeni bir ikili dosya biçimini desteklemeye başlamıştır. Ne yazık ki bu biçim de tam olmadığı için canlandırmalı modellerde sorun yaşanmaktadır.

3.3 Ortam Dosyaları

Daha önce de belirtildiği gibi Direct3D 10'un şimdilik canlandırmalı model desteği olmadığı için, bu uygulama için sabit model kullanılacaktır. Kullanılacak bu model, aynı zamanda kalabalıkta çeşitlilik yaratmaya uygun olabilmesi için bir Lego modeli olarak seçilmiştir.* Uygulamadaki diğer modeller Milkshape ve GIMP uygulamaları kullanılarak yapılmıştır.

Lego adamın ayrıntı seviyelerini ayarlamak için Milkshape programı ile poligon sayısı azaltılmıştır. Şekil 3 ile görebileceğimiz üzere dört ayrı düzey belirlenmiştir. Seviyelerin poligon sayıları Tablo 1 ile gösterilmiştir.



Şekil 3. Lego adamın dört ayrıntı düzeyindeki gösterimi

* Dosya bu adresten ücretsiz olarak edinilebilir: http://www.turbosquid.com/3d-models/lego-3d-model/422994

Tablo 1. Lego adamın farklı düzeylerdeki tepe noktası ve üçgen sayısı

Düzey	**Tepe Noktası Sayısı**	**Üçgen Sayısı**
1	725	992
2	613	798
3	248	220
4	202	150

3.4 Uygulama Kontrolleri

Örneklemenin ne kadar işe yarayıp yaramadığını öğrenmek için uygulamaya, arayüz kontrolleri sayesinde bazı seçenekler eklenmiştir. Bu seçenekler, örneklemeyi açıp kapatma, Lego adamların ortadaki diski takip etmesini durdurma ve Öklid uzaklığı yerine Manhattan uzaklığı kullanma seçenekleridir. Aynı zamanda, stadyumdaki Lego adam sayısı kontrol edilebilmektedir. Stadyum, 40,000 kişiye kadar Lego adamlarla doldurulabilir. Uygulama arayüzünde, grafik kartın modeli, saniyedeki kare sayısı ve her ayrıntı düzeyine düşen Lego adam sayısı da gösterilmektedir.

3.5 Gerçekleme

Uygulamanın genel akışı şu şekilde gerçekleşmektedir. Öncelikle Direct3D ve DXUT kütüphaneleri ilklendirilir. Daha sonra Lego adamlar ilklendirilmeye başlanır. Oyun sahasının etrafına orantılı bir şekilde yerleştirilmeleri için her birinin ilk koordinatları hesaplanır. Bu koordinatlar hesaplanırken, her Lego adam farklı renkte gözükmesi için rasgele bir renk değişkeni, hareketlerinin farklı olması için rasgele belirlenen bir hız değişkeni ve ne kadar hareket edebileceğini belirten başka bir değişken verilir. Ayrıca diskin olduğu noktaya doğru ilk yönleri belirlenir. Pong oyunun diğer parçacıkları da ilklendirildikten sonra oyun döngüsüne geçilir.

Oyun döngüsünde ilk olarak Pong parçacıkları güncellenir. Daha sonra Lego adamların her biri için ayrıntı düzeyi belirlenir. Lego adam her ayrıntı düzeyi için açılmış diziye eklenir. Hareketine göre öteleme ve diske göre döndürme matrisleri belirlenir. Sonra Lego adam ilgili ayrıntı düzeyi listesine eklenir. Bu işlemler bittikten sonra her ayrıntı düzeyi için, o ayrıntı düzeyine karşılık gelen model bu listelerdeki bilgiler ile çizilir. Arkasından Pong parçacıkları ve arayüz bileşenleri çizilir.

3.5.1 Tepe Noktası Yerleşimleri ve Gölgelendirici Programcıkları

Daha önce de belirtildiği gibi örnekleme işlemi için iki tane tampon dizisi gerekmektedir. Bunlardan ilkinde örneklenecek modelin bilgileri, ikinci dizide de modelden üretilecek her örneğe ait bilgiler yer alır. Bu bilgiler dünya yerleşim matrisleri ve örneğe özel bilgiler olabilir. Bu iki dizinin uygun olarak kullanılabilmesi için, Direct3D'nin akımlardan bilgi sağlayan Input Assembler bileşenine dizilerin yapıları detaylı olarak belirtilmelidir. Bu da D3D10_INPUT_ELEMENT_DESC yapısından bir dizinin uygun olarak doldurulmasıyla oluşturulur. Uygulamamızda bu dizide yer alan bileşenlerin tipleri Tablo 2'de gösterilmiştir. Dikkatli incelenecek olursa, bu yapı içerisinde hangi değişkenlerin hangi dizide kullanılacağı da belirtilmiştir. Buna göre konum (POSITION), normal (NORMAL) ve doku koordinatları (TEXCOORD) ilk dizide yer almaktadır. İkinci diziye ise dünya matrislerini içeren *mTransform* ve her Lego adam için farklı olan renk, *color*, değişkeni düşmektedir.

Dizilerin yapıları bu şekilde tanıtıldıktan sonra, bu yapılara uyacak diziler hazırlanmalıdır. Bu dizilerden ilki, DXUT kütüphanesinin SDKMesh dosyasını yükleyen sınıfıyla kolayca elde edilebilir. Model sınıfın *Create* metodu ile yüklendikten sonra, *GetVB10* metodu ile tepe noktası tampon dizisi alınabilir.

Diğer dizi ise, her örnek için özel olarak doldurulmalıdır. Bunu yapmak için programın ilk başladığı anda dizilerde her örnek için bir konum ve renk değişkenleri tutulmaktadır. Oyun döngüsünde, bu bilgiler *InstanceData* adı verilmiş bir diziye kopyalanırlar.

İki dizi de hazır olduğunda, bu diziler *IASetVertexBuffers* ve *IASetIndexBuffer* komutları ile çizim için tanıtılmış olurlar.

Tablo 2. Tepe Noktası Yerleşimleri

Anlamsal	**Dizi**	**Değişken**
POSITION	0	Vector3f
NORMAL	0	Vector3f
TEXCOORD	0	Vector2f
mTransform	1	Vector4f
COLOR	1	Vector1i

Bundan sonraki adım ise, Direct3D aygıtına ait normal çizim komutu olan *DrawIndexed* yerine *DrawIndexedInstanced* komutunu çağırmaktır. Elbette bu adım sırasında tepe noktası gölgelendirici ve benek gölgelendirici programcıkları devreye girer.

Gölgelendirici programcıklarına geldiğimizde ise, Tablo 2'de görülen anlamsal ifadeler, buradaki değişkenlere eşleştirilir. Tepe noktası gölgelendirici programında her örneğin matrisleri hesaplanır. Benek gölgelendirici programında ise ışıklandırma ve gölgeleme işlemleri yapılır.

3.5.2 Çeşitlilik Sağlanması

Uygulamadaki örneklenen Lego adamların tek tip olması yerine farklı özelliklere sahip olması inandırıcılığı arttıran özelliklerden biridir. Bunun için ilklendirme anında her örneğin x ve y koordinatlarına rasgele değerler eklenerek her örneğin aynı hizada olması engellenir. Lego adamlar aşağı yukarı bir zıplama hareketi yapacağından, her örneğin hızının farklı olması çeşitliliği arttıracaktır. Bu yüzden, yine ilklendirme kısmında her örneğe rasgele bir hız değişkeni atanır.

Son olarak, ilklendirme anında her örneğe bir renk değişkeni atanır. Bu renk değişkeni benek gölgelendirmesi sırasında Lego adama farklı bir kazak rengi verilmesini sağlar. Lego adamın tek bir doku görüntüsü mevcuttur. Bu doku görüntüsünde yer alan kırmızı rengi, benek gölgelendiricisi programcığında, burada verilen rasgele renk ile tekrar renklendirilir ve o şekilde ışıklandırılması yapılır.

3.5.3 Ayrıntı Düzeyleri

Ayrıntı düzeyleri için hazırlanan dört ayrı model bulunmaktadır. Asıl model yaklaşık bin tane üçgenden oluşurken, en düşük model 150 üçgene kadar düşmektedir. Bu da bir model için altı kat daha az üçgen kullanılması anlamına gelmektedir.

Gerçekleme kısmında, daha önce örnekleme yapılırken kullanılan *InstanceData* dizisi her ayrıntı düzeyi seviyesi için bir tane olmak üzere yaratılmıştır. Bu sayede, her örnek güncellenirken kamera ile olan uzaklığı hesaplanır ve dahil olduğu ayrıntı düzeyi dizisine eklenir.

Uzaklığı hesaplamak için DXUT çatısında yer alan kamera sınıfının *GetEyePt* metodu kullanılır. Uzaklık bu metod ile gelen noktayla, modelin kendi uzayındaki orijin noktası arasında, aksi belirtilmedikçe Öklid uzaklığı olarak DirectX'in *D3DXVec3Length* fonksiyonu ile hesaplanır. Aynı uzaklık Manhattan olarak x, y ve z koordinat farklarının mutlak değeri olarak elde edilebilir.

Uzaklık bulunduktan sonra model daha önceden belirlenmiş ayrıntı düzeyi grubuna eklenir. Bu gruplar deneme yanılma yöntemi ile bulunmuştur. Farklı ekran çözünürlüklerinde bu değerleri değiştirmek gerekebilir.

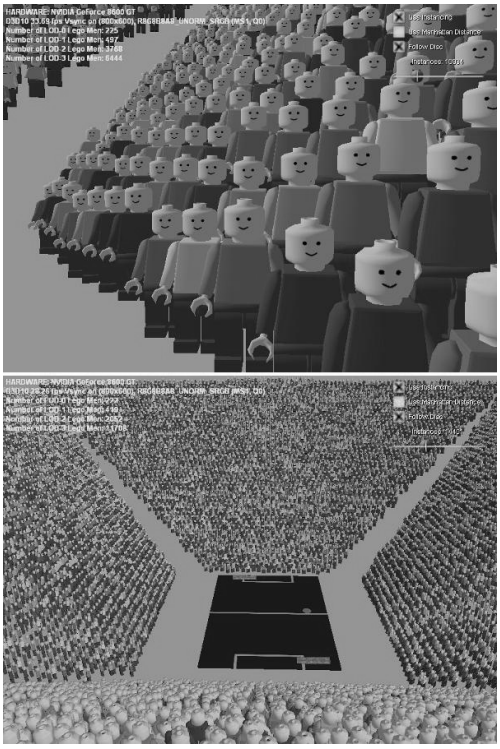
3.6 Çalıştırma

Şekil 4 ile görebileceğimiz üzere, uygulama başarıyla onbinlerce Lego adamı etkileşimli kare sayılarıyla çizebilmektedir. Şekildeki ilk karede diski izleyen Lego adamları görebiliyoruz. İkinci karede ise bir sıranın en arkalarından oyuna ve stadyum kalabalığına bakıyoruz. Bu noktadan sonra uygulamayı test aşamasına geçirebiliriz

4. TESTLER VE SONUÇLARI

Test için, orta düzey bir masaüstü bilgisayar kullanılmıştır. Bilgisayarın özellikleri Tablo 3'de verilmiştir. Bu özellikleri kullanarak iki farklı noktada farklı testler yapılmıştır. Bu testler ile saniyedeki kare sayısı (FPS) ve işlemci kullanımı (CPU) ölçülmüştür.

İlk nokta, uygulamanın ilk başladığı nokta olarak seçildi. Bu nokta, alanın biraz üzerinde, kalabalığa doğru bakıyor ve kalabalığa fazla yakın değil, o yüzden ilk iki ayrıntı düzeyi listesi genelde boş kalıyor.



Şekil 4. Uygulamadan görüntüler

Tablo 3. Test Bilgisayarı Özellikleri

Parça	**Model**
İşlemci	AMD Athlon 64 X2 2.61Ghz
Bellek	2GB
Grafik Kartı	NVidia GeForce 8600 GT
Grafik Belleği	512MB
Ekran Çözünürlüğü	1440x900*
İşletim Sistemi	Microsoft Vista Business N (SP1)
3B Kütüphane	DirectX (Kasım 2008)

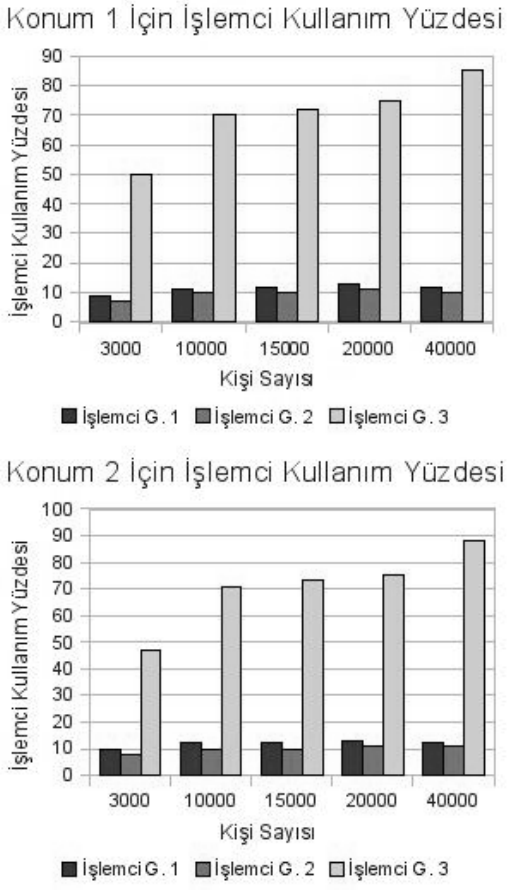
***** Uygulama 800x600 çözünürlükte pencere olarak çalıştırılmıştır.

İkinci nokta ise, alana ve kalabalığa yakın olarak seçildi. Bu yüzden bütün ayrıntı düzeylerinde Lego adamlar içeriyor. Ama, kalabalık arttırıldıkça yeni Lego adamlar, kameradan uzakta belirdikleri için, üç ve dördünce ayrıntı düzeylerine dahil oluyorlar.

Bu iki noktada farklı üç seçenek belirlenerek işlemci kullanımı ve saniyedeki kare sayısı kaydedildi. Bu seçeneklerin ilkinde, örnekleme ve diski takip etme özellikleri açıkken, uzaklık hesabı Öklid olarak hesaplanıyor. İkinci grupta, örnekleme yine açık, ama diski takip etme özelliği kapalı ve uzaklık Manhattan olarak hesaplanıyor. Son olarak üçüncü grupta ise örnekleme ve diski takip etme kapalı, uzaklık ise Manhattan olarak hesaplanıyor.

4.1 İşlemci Performansı

Örneklemenin işlemciyi rahatlattığını daha önce belirtmiştik. Bu yüzden ilk iki grupta işlemci kullanımının, her iki konum için de çok düşük olduğunu görebiliyoruz. İkinci grupta, işlemci kullanımı daha da düşüyor, bunun nedeni uzaklığın Manhattan olarak hesaplanması ve diski takip etme özelliğinin kapalı olması. Bu iki seçenek ile işlemci biraz daha serbest kalıyor. Son grupta ise, örnekleme kullanılmazsa, işlemci kullanımı %50 gibi bir değer ile başlıyor. Kişi sayısı arttıkça %80 değerlerini buluyor. Bu sonuçları Şekil 5 ile de görebiliriz.



Şekil 5. Konum 1 ve 2 için işlemci performansı

4.2 Grafik İşlemci Performansı

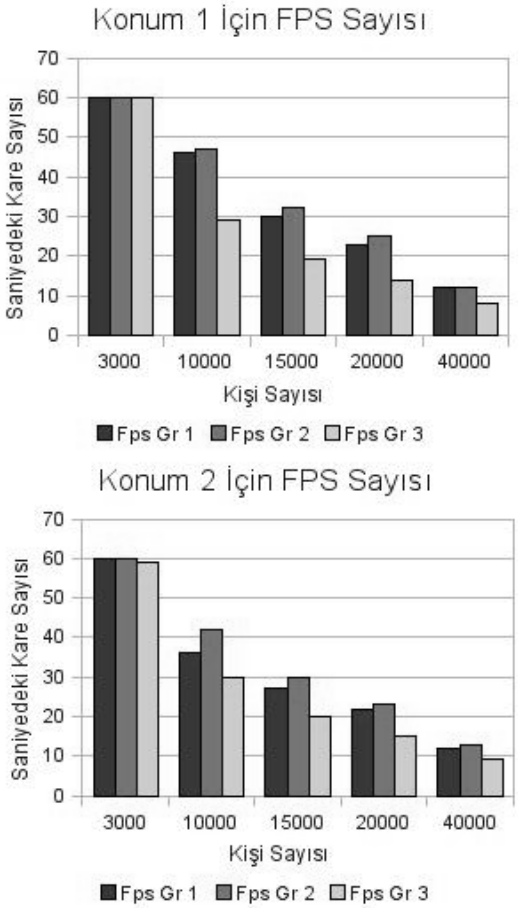
Örneklemenin işlemcinin yanında grafik başarımını da arttırdığını Şekil 6 ile de görebiliyoruz. Kişi sayısı arttıkça, saniyedeki kare sayısı düşüyor. İşlemci başarımı çizelgesi ile karşılaştırılırsa, 15000 kişi civarında grafik işlemcinin en iyi başarımı verdiğini görebiliyoruz. Burada işlemci düşük seviyede kalırken, saniyede 30 kare civarında kabul edilebilir bir çizim sağlıyor. Örnekleme kullanmayan son grupta yoğun işlemci kullanımı ile birlikte grafik işlemci başarımının da örneklemeden düşük olduğunu görebiliyoruz. Her üç grup için de 3000 kişilik ilk durumda saniyede 60 kare olmasının nedeni, DXUT uygulama çatısının burada uygulamayı sabit kare sayısında tutmasıdır.

5. SONUÇLAR VE DEĞERLENDİRME

Her ne kadar sahte karakter kullanımı içeren yöntemler ile daha fazla karakter çizilse de, uygulama öncesi hazırlıklar, yüksek görüntü boyutu ve programlama açısından zorluklar içermektedirler. Onlara nazaran, örnekleme yöntemi işlemciye daha az yük getirmesi, uygulama öncesi özel bir hazırlık gerektirmemesi, görüş açısı sorunları olmaması ve gerçeklemesi programlama açısından çok daha kolay olmasıyla öne çıkıyor.

Örneklemenin kalabalık içeren uygulamalarda başarım kazandıracağı uygulamaya uygulanan testlerle açık bir şekilde görülebilmektedir. Sayısal başarım olarak, 15,000 kişi saniyede 30, 18,000 kişi ise saniyede 24 karede çizdirilmiştir. Bu kare sayılarına ise çok düşük işlemci kullanımı, %10 civarı, eşlik etmektedir. Uygulama 40,000 kişilik tüm bireyleri ise saniyede 13 karede çizmiştir. Sonuçlar örneklemenin hem işlemci hem de grafik işlemci için kazançlı olduğunu gösteriyor. Görüntü kalitesi ise sahnedeki poligon sayısıyla ters orantılıdır. Her ne kadar grafik işlemcileri zorlasak da, her işlemcinin bir sınırı vardır. Uygulamanın görüntü kalitesi daha iyi 3B modellerle arttırılabilir. Bunun sayısal başarıma etkisi elbette daha düşük kare sayısı ve daha fazla işlemci kullanmak olacaktır.

Bu makalenin hazırlanması esnasında Direct3D 11 henüz geliştirilme aşamasındadır ve yeni gelecek bu sürümünde öne çıkan özelliklerinden biri çoklu iş parçacıklı çizimidir.



Şekil 6. Konum 1 ve 2 için grafik işlemci performansı

Bu sayede yaygın olan çoklu işlemcileri kullanıp, çizdirme işlemini bu işlemcilere yayılmasıyla performans artışı sağlanacaktır. Ayrıca, yüksek kaliteli dokuları daha iyi paketleyebilecek sıkıştırma algoritmaları kullanacak bir özelliği olacaktır. Son olarak, örnekleme işlemi geometri gölgelendiricileri ile yapılabilecektir. Bu sayede, kalabalık çizimleri geometri gölgelendiricisi ile yeni tepe noktalarını doğrudan grafik işlemci üzerinde üretebilir.

KAYNAKLAR

Aubel, A., Boulic, R. and Thalmann, D. Animated impostors for real-time display of numerous virtual humans. *Proceedings of the First International Conference on Virtual Worlds*. Springer-Verlag, pp. 14-28. 1998.

Clark, J.H. (1976). Hierarchical geometric models for visible surface algorithms. *Commun. ACM*. 19(10). 547-554.

Dobbyn, S., Hamill, J., O'Connor, K. ve O'Sullivan, C. (2005). Geopostors: a real-time geometry / impostor crowd rendering system. *Symposium on Interactive 3D Graphics.* Proceedings of the 2005 symposium on Interactive 3D graphics and games, 95-102.

Dudash, B. (2004). Mesh instancing. *Technical Report, NVIDIA Corporation*.

Kavan, L., Dobbyn, S., Collins, S., Zara, J. ve O'Sullivan, C. (2008). Polypostors: 2D Polygonal Impostors for 3D Crowds. *Symposium on Interactive 3D Graphics.* Proceedings of the 2008 symposium on Interactive 3D graphics and games, 149-155.

Luebke, D., Reddy, M., Cohen, J., Varshney, A., Watson, B. ve Huebner, R. (2003). *Level of detail for 3D graphics*. San Francisco: Morgan Kaufmann Publishers.

Paul, E.B. (2006). Manhattan distance. *Dictionary of Algorithms and Data Structures*. U.S. National Institute of Standards and Technology.

Scott, P. (2004). Shader model 3.0, best practices. *Technical Report, NVIDIA Corporation*.

Tecchia, F., Loscos, C., Chrysanthou, Y. Image-Based Crowd Rendering. IEEE Computer Graphics and Applications. 22(2) 36-43.

Kaya Oğuz, 2007 yılında İzmir Ekonomi Üniversitesi Bilgisayar Bilimleri Fakültesi'nden Yazılım Mühendisi olarak mezun olmuştur. Ardından, İskoçya'da, University of Abertay Dundee'de Bilgisayar Oyunları Teknolojileri dalında yüksek lisansını tamamlamıştır. Şu an İzmir Ekonomi Üniversitesi'nde araştırma görevlisi olarak çalışmalarına devam etmektedir. Bilgisayar grafikleri dışında yapay sinir ağları üzerine çalışmaktadır.