Grafik Kartı Üzerinde Paralel Hızlandırılmış Işın İzleme

Mustafa Alper ÇOLAK[1,2], Nadia ERDOĞAN[2]

1. TÜBİTAK UEKAE BTE
alper.colak@bte.tubitak.gov.tr

2. Bilgisayar Mühendisliği Bölümü
İstanbul Teknik Üniversitesi
nerdogan@itu.edu.tr

Öz

Işın izleme, başta ışık olmak üzere yüksek frekanstaki dalgaların benzetiminde kullanılan oldukça popüler bir yöntemdir. Bununla birlikte, izlenen ışın sayısının artması ve üzerinde ışın izleme koşturulan sahnenin büyümesi, hesaplama süresini önemli ölçüde artırmaktadır. Bu işlem esnasında izlenen ışınlar birbirlerinden tamamen bağımsız olduğundan, ışın izleme paralelleştirmeye son derece uygundur. Grafik işlem birimlerinin (GİB)son yıllarda yaşadığı gelişim, yüksek derecede paralel sistemleri masaüstümüze kadar getirmiş, NVIDIA firmasının yayınladığı CUDA platformu ise, bu sistemleri genel maksatlı hesaplama amacıyla kullanabilme imkanını bizlere sunmuştur. Bu çalışmada, CUDA kullanılarak bir ışın izleyici geliştirilmiş ve örnek geometriler üzerinde 5 farklı çözünürlükte koşturulan ışın izleme işlemlerinin başarımları karşılaştırılmıştır. Çalışma sonucunda, GİB üzerinde çalışan paralel kodun, merkezi işlem birimi (MİB) üzerinde çalışan seri koddan 34 ile 85 kat daha hızlı sonuç verdiği gözlemlenmiştir.

1. Giriş

Bu çalışmada, genel maksatlı hesaplama yapabilme yeteneğine sahip bir grafik kartı işlemcisi kullanılarak 3 boyutlu cisimler üzerinde ışın izleme yazılımı geliştirilmiştir. Işın izleme işleminin amacı, bir kaynaktan yola çıkan bir ışının cisim uzayını terk edene kadar takip edilmesi ve cisim üzerinde çarptığı noktaların bulunmasıdır.

Işın izleme, Fermat prensibine göre basit ortamda doğrusal olarak hareket ettiği varsayılan dalgaların benzetiminde kullanılan bir yöntem olup, en yoğun kullanım alanı ışığın modellenerek 2 boyutlu fotogerçekçi görüntülerin üretilmesidir. Bunun dışında yüksek frekans elektromanyetik ve akustik dalgalar da bu yöntemle modellenebilir.

Işın izleme, yüksek maliyetli bir işlem olduğundan özellikle son 15 yıl içerisinde bu işlemi daha verimli hale getirecek ve hızlandıracak yöntemler üzerinde çalışılmıştır. Bu çalışmalar sonucunda hızlı ışın-üçgen kesişim algoritmaları, ışın-kutu kesişim algoritmaları, uzay bölmeleme algoritmaları ile istatistik yaklaşımlar ortaya çıkmıştır. Bu çalışmada bahsi geçen algoritmalardan örnekler kullanılmıştır.

Işın izlemeyi hızlandırıcı bir diğer yöntem yazılımın işlemin paralel olarak yürütülmesidir. Işın izleme, yapısal olarak yüksek derecede paralelliğe uygun bir işlemdir. Yüksek derecede paralel veri işleme, uzun yıllar sadece süper bilgisayarlar ve bilgisayar kümeleri kullanılarak yapılabilecek bir işlem olarak kalmışken, 2007 yılının sonunda grafik kartı üreticisi NVIDIA firmasının yayınladığı CUDA mimarisi sayesinde kişisel bilgisayarlarda da yürütülebilir hale gelmiştir. CUDA, oldukça fazla sayıda ancak basit yapıda işlemcilerden oluşan NVIDIA üretimi grafik kartlarının genel maksatlı hesaplama amaçlı kullanılmasını sağlayan bir uygulama ara yüzüdür.

Bu çalışmada, üçgenlerle modellenmiş geometriler üzerinde yürütülen, Kd-Tree uzay bölmeleme algoritması kullanılarak verimliliği artırılmış, CUDA kullanılarak grafik kartı üzerinde paralel olarak çalışabilen bir ışın izleme yazılımı geliştirilmiş ve geliştirilen bu yazılımla ilgili temel prensipler sunulmuştur.

2. Işın İzleme

2.1. Amaç ve Kullanım Alanları

Işın izleme işlemindeki amaç, belirli bir merkez ve yöne sahip doğrusal bir ışının 3 boyutlu uzayda herhangi bir cisme çarpıp çarpmadığını, çarptıysa çarpma noktasını ve çarptıktan sonra ışının hangi yönde yoluna devam ettiğini ışın cisim uzayından çıkana kadar takip ederek belirlemektir. Işınlar bilgisayarda ışının başladığı merkez noktasının koordinatları (3 adet kayan noktalı sayı) ve ışının yönünü ifade eden bir birim vektörle temsil edilir.

Işın izleme yöntemi, düzlemsel dalga yaklaşımına sahip bütün fiziksel olayların bilgisayardaki benzetiminde kullanılabilir. Dalganın düzlemsel olarak temsil edilip edilemeyeceği, dalganın sahip olduğu dalga boyunun cisme göre büyüklüğüne ve dalganın uzayda kat ettiği mesafeye göre belirlenir. Bu yaklaşıma göre başta ışık olmak üzere, yüksek frekans elektromanyetik dalgalar ve yüksek frekans akustik dalgalar da doğrusal dalga olarak temsil edilebilirler. Bu çalışmada, basit bir ışık benzetimi gerçeklenerek fotogerçekçi görüntü üretimi amaçlanmıştır.

2.2. Işın Üçgen Kesişimi

Işın izleme işleminde ilk amaçlanan, ışının üçgenlerle modellenmiş geometrinin hangi üçgenine hangi noktada çarptığını bulmaktır. Bunun için ışın-üçgen kesişimi algoritmaları kullanılır. Bu çalışmada, en çok bilinen uygulamalardan biri olan Möller-Trumbore algoritması kullanılmıştır.[1]

2.3. Başarım Analizi

Işın izleme işlemi, temelde basit olarak görünse de, pratikte oldukça zaman alan bir işlemdir. Bunun sebebi hem ışın hem de üçgen sayılarının oldukça fazla olmasıdır. Üzerinde herhangi bir iyileştirme gerçekleştirilmemiş bir ışın izleyicide 20000 üçgenli bir geometrinin 600*600 piksellik bir görüntüsünü oluşturmak için sadece ışın başlatma

aşamasında 360.000 ışın izlenmeli ve 7.200.000.000 defa ışın üçgen kesişim algoritması yürütülmelidir.
Işın izleme işlemini hızlandırmanın ilk yolu, gereksiz ışın üçgen kesişimi testlerini mümkün olduğunca azaltmaktır. Bunun için uzay bölmeleme algoritmaları kullanılır.
Işın izlemeyi hızlandırıcı bir diğer yol ise işlemi paralel yürütmektir. Işın izleme, birbirinden bağımsız birçok ışın üzerinde işlem yürüttüğünden, bu yöntem paralelleştirmeye oldukça uygundur.

3. Uzay Bölmeleme Algoritmaları

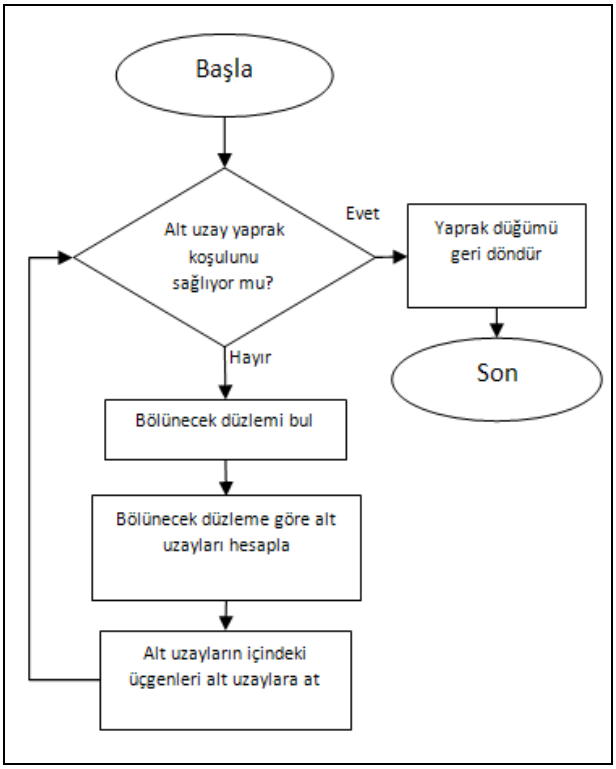
Uzay bölmeleme algoritmaları, esas olarak en yakın komşu problemlerini çözmek üzere geliştirilmişlerdir. Bu algoritmalara göre, cismin içinde bulunduğu uzay farklı derinliklerde alt uzaylara bölünerek, gereksiz işlemlerin yapılması önlenir. Işın izleme işlemi de, ışının merkezine ışının yönü doğrultusundaki en yakın üçgeni bulma işleminden ibaret olduğundan, en yakın komşu problemine indirgenebilir. Dolayısıyla uzay bölmeleme algoritmalarının kullanılması, ışın izleme işleminin verimliliğini önemli ölçüde artırmaktadır. Literatürde en çok kullanılan uzay bölmeleme algoritmaları; "QuadTree", "Octere", "BSP-Tree" ve "Kd-Tree" dir. Bu çalışmada, Kd-Tree uzay bölmeleme algoritması kullanılmıştır.

3.1. Kd-Tree

"Kd-Tree", "K-dimensional tree" ifadesinin kısaltması olup, "K boyutlu ağaç" anlamına gelir. Kd-Tree, BSP Tree'nin özelleştirilmiş bir halidir. Bu yapıda uzay, koordinat eksenlerine paralel bir dikdörtgen prizma olarak düşünülür. Uzay, belirli bir maksimum üçgen sayısına ya da maksimum derinliğe ulaşana kadar yine dikdörtgen prizma şeklindeki alt uzaylara bölünür. Kd-Tree'de uzay, bir kenarının orta noktasından ikiye bölünmek zorunda değildir. Yani bölünen alt uzaylar, geometrik olarak birbirlerine eşit değildir. Bunun yerine Kd-Tree'de uzay alt uzaylara, toplam maliyeti en aza indirecek şekilde bölünür.[2] Bu amaçla çeşitli sezgisel yöntemler geliştirilmiştir.

3.2. Kd-Tree'nin Oluşturulması

Kd-Tree'nin kullanılması için, öncelikle üçgenlerden oluşan cisim için bir Kd-Tree'nin oluşturulması gerekmektedir. Kd-Tree'nin oluşturulması, ilk bakıldığında ciddi bir hesaplama yükü getirse de, yapı bir kere oluşturulup cismin bütün açılardan görüntüsünü elde etmekte kullanılabileceği ve bu işlemlerde oldukça yüksek bir başarım sağlayacağı için bu hesaplama yükü, göz ardı edilebilecek düzeydedir.
Daha önce bahsedildiği gibi, Kd-Tree oluşturulurken her aşamada uzay, iki alt uzaya bölünür. Uzayın bölüneceği düzlemi belirlemek için çeşitli sezgisel yöntemler kullanılır. Bu yöntemlerin en çok kullanılanları, "Uzaysal Medyan Yaklaşımı" ve "Yüzey Alanı Sezgisel Yaklaşımı"dır.
Kd-Tree oluşturulması konusunda bir diğer önemli nokta, üçgenlerin bölünen her alt uzayın içinde olup olmadığının hesaplanmasıdır. Bunun için de kutu-üçgen kesişim algoritmaları geliştirilmiştir.
Kd-Tree, bu iki ölçüte göre özyinelemeli olarak oluşturulur. Kd-Tree'nin oluşturulma algoritması şu şekildedir: [3]



Şekil 1: Kd-Tree Oluşturma Algoritması

3.3. Bölme Yüzeyinin Belirlenmesi

Kd-Tree'nin oluşturulmasında en önemli işlemlerden biri, bölme yüzeyinin hesaplanmasıdır. Bunun için çeşitli metotlar geliştirilmiştir.

3.3.1. Uzaysal Ortanca Yaklaşımı

Bölme yüzeyinin belirlenmesindeki en basit, bununla birlikte en çok kullanılan yöntem, uzaysal ortanca yaklaşımıdır. Uzaysal ortanca yaklaşımında bölünecek eksen, sıralı olarak belirlenir. Bölme düzlemi ise, sıradaki ekseni dik olarak ortadan ikiye kesen bir düzlemdir.

$$p_k = D(V) \bmod 3$$
$$p_\varepsilon = \frac{V_{min,p_k} + V_{max,p_k}}{2} \quad (1)$$

Bu formülde p_k bölme ekseni, D(V) şu andaki ağaç derinliği, p_ϵ bölme yüzeyi, $V_{min,pk}$ bölme ekseninin en küçük noktası, $V_{max,pk}$ ise bölme ekseninin en büyük noktasıdır.

3.3.2. Yüzey Alanı Sezgisel Yaklaşımı

Daha önce belirtilen ortanca yaklaşımlar; klasik, gerçeklenmesi basit, buna karşın başarımı yüksek olmayan yaklaşımlardır. Kd-Tree'nin, köke en yakın ve en yüksek miktarda boşluklara sahip bir ağaç olarak daha yüksek bir başarıma sahip olduğu tespit edilmiştir. Bununla birlikte, her görüntü için yüksek başarımda bir çözüm bulmak oldukça

zordur. Bu nedenle, önerilen çözümlerin çoğu, belirli tipte bir görüntüye özgü olarak sınırlı kalmıştır.
Bu sorunu çözmek için, her çeşit görüntünün ortak özelliklerine odaklanılmalıdır. Yapılan çalışmaların sonucunda, "Yüzey Alanı Sezgisel Yaklaşımı" (Surface Area Huristic) ön plana çıkmıştır. YAS yaklaşımında; bir V alt uzayının p yüzeyi tarafından bölünen iki alt uzayı olan VL ve VR ile, bu alt uzayların sahip olduğu üçgen sayıları olan NL ve NR için beklenen arama maliyetleri hesaplanır. Bu yaklaşım, şu kabullerle hareket eder:

- Işınlar, düzgün olarak dağılmışlardır; cisim uzayının dışında başlar ve dışında sonlanırlar.
- Her alt uzay için kutu-ışın kesişimi ile her üçgen için ışın-üçgen kesişimi algoritmalarının maliyetleri bilinmektedir. Bunlar KT ve KI olarak temsil edilirler.
- Toplam ışın üçgen kesişimi maliyeti, üçgen sayısı (N) ile lineer olarak artar ve "N KI" ile temsil edilir.

Bu varsayımlara göre, bir V uzayını kestiği bilinen bir ışının V uzayının bir alt uzayı olan Vsub uzayını kesme olasılığı P koşullu olasılığı şu şekilde ifade edilir:

$$P_{[V_{sub}|V]} = \frac{SA(V_{sub})}{SA(V)} \quad (2)$$

Bu formülde SA(V), V uzayının yüzey alanıdır.
Bu durumda p düzleminin beklenen maliyeti $C_V(p)$, ağaç üzerinde bir alt uzaya inmenin maliyeti ile bu alt uzayın sahip olduğu iki çocuk düğümün beklenen kesişim maliyetlerinin toplamıdır.

$$C_V(p) = K_T + P_{[V_l|V]}C(V_l) + P_{[V_r|V]}C(V_r) \quad (3)$$

(3) numaralı denklemi bütün ağaç (T) için genişletirsek:

$$C(T) = \sum_{n\in\text{düğümler}} \frac{SA(V_n)}{SA(V_S)} K_T + \sum_{l\in\text{yapraklar}} \frac{SA(V_l)}{SA(V_S)} K_I \quad (4)$$

Denklemi bulunur. Burada V_S, bütün sahneyi içeren S uzayının sınırlarını kapsayan hacimdir. Bu durumda S sahnesi için en iyi T ağacı, C(T) değerinin en düşük olduğu ağaçtır. İşte YAS yaklaşımı, C(T) değerini en küçüğe indirgemeyi amaçlar. Bununla birlikte, sahne büyüdükçe C(T) maliyetinin en düşük olabileceği muhtemel ağaçların sayısı da hızla artar. Dolayısıyla çok basit sahneler haricinde uygulanabilir bir ağaç bulmak imkansız hale gelir.
Bu nedenle genel bir ideal çözüm yerine, yerel bir yaklaşım uygulanabilir. Bu yaklaşıma göre, her V uzayının sahip olduğu alt V_L ve V_R alt uzaylarının yaprak olduğu varsayılır. Dolayısıyla bu varsayım, açgözlü bir çözüm olarak nitelendirilebilir. Bu durumda p düzleminin beklenen maliyeti $C_V(p)$ şu şekilde hesaplanır: [3]

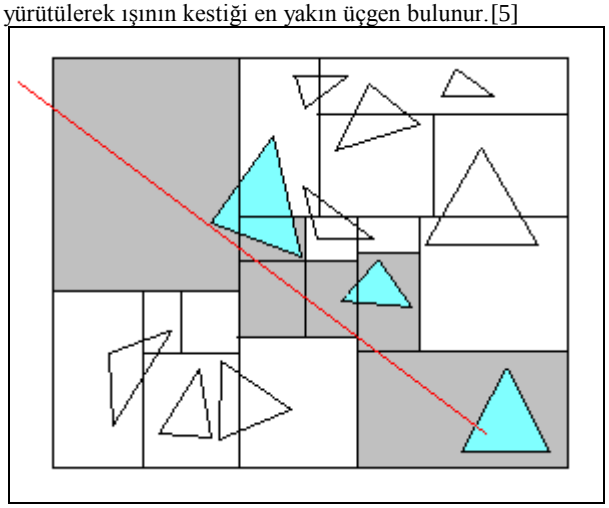
$$C_V(p) = K_T + K_I\left(\frac{SA(V_L)}{SA(V)}|T_L| + \frac{SA(V_R)}{SA(V)}|T_R|\right) \quad (5)$$

3.4. Kutu Üçgen Kesişim Algoritması

Kd-Tree oluşturulurken, her adımda o alt uzayın içerdiği üçgenler belirlenir. Alt uzaylar birer dikdörtgen prizma (kutu) olarak temsil edildiğinden, bir üçgenin bir alt uzayın içinde olup olmadığını ya da herhangi bir yüzeyden alt uzayı kesip kesmediğini belirlemek için bir kutu-üçgen kesişim algoritması kullanılmalıdır. Bu çalışmada, Möller'in kutu üçgen kesişim algoritması kullanılmıştır.[4]

3.5. Kd-Tree Üzerinde Arama

Kd-Tree için özyinelemeli arama algoritması, esasen genel BSP-Tree'ler için Jansen tarafından 1986 yılında geliştirilmiş ve yayınlanmıştır.
Bu algoritmada her derinlikte ışının kestiği alt uzay ya da uzaylar belirlenir. Işın, herhangi bir alt uzayı kesmiyorsa, alt uzayların içerdiği üçgenleri de kesmiyor demektir. Işın, alt uzaylardan birini kesiyorsa, arama alt uzayın sahip olduğu çocuk alt uzaylar için tekrarlanır. Eğer ışın bir derinlikteki iki alt uzayı birden kesiyorsa, aramaya ışının kaynağına en yakın alt uzaydan başlanır. Bu şekilde özyinelemeli olarak bir yaprak düğüme ulaşıldığında, o yaprak düğümün içerdiği bütün üçgenler için ışın-üçgen kesişim algoritması yürütülerek ışının kestiği en yakın üçgen bulunur.[5]



Şekil 2: Kd-Tree üzerinde ışın takibi

Kd-Tree üzerinde örnek bir ışının takibi Şekil 2'de gösterilmiştir. İlgili şekilde ışın kırmızı renkli bir çizgi ile aranan alt uzaylar gri renk ile kontrol edilen üçgenler mavi renk ile gösterilmiştir. Şekilde gösterilen senaryo, Kd-Tree üzerinde bir üçgene çarpan bir ışın için en kötü olasılıklı senaryodur. Bu durumda bile, uzayın içerdiği 11 üçgenden yalnızca 3 tanesi kontrol edilmiş ve görüldüğü gibi gereksiz kontrollerden kurtulunmuştur. Görüldüğü gibi Kd-Tree, örnekteki temsili senaryoda bile yüksek bir verime sahiptir.

3.6. Işın Kutu Kesişim Algoritması

Kd-Tree üzerinde arama yapılırken öncelikle ışının alt uzayları kesip kesmediği belirlenir. Kd-Tree yapısında alt uzaylar birer kutu (dikdörtgen prizma) ile temsil edilmiştir. Dolayısıyla ışının herhangi bir alt uzayı kesip kesmediğinin belirlenmesi için bir ışın-kutu kesişim algoritması kullanılmalıdır. Bu amaçla Smits etkin bir ışın kutu kesişim algoritması önermiştir.[6] Tek ışın ve üçgen için oldukça verimli çalışan bu algoritma, daha sonra birçok defa tekrarlanan ışın kutu kesişimleri için iyileştirilmiştir.[7]

4. Grafik Kartı Üzerinde Paralel Veri İşleme

Işın izleme işlemi, birbirinden bağımsız çalışan ışınlar göz önüne alındığında, muazzam derecede paralelleştirilebilir bir işlemdir. "Muazzam derecede paralel bilgisayar" kavramı, bu güne kadar vektör makineler ya da çok sayıda yüksek performanslı bilgisayarın bir araya getirilmesiyle oluşmuş bilgisayar kümeleri için kullanılmıştır. "Süper bilgisayar" olarak adlandırılan bu bilgisayarlar, maliyetleri yüz binlerce dolardan başlayıp milyonlarca dolara çıkan sistemlerdir. Dolayısıyla bu bilgisayarlar, dünyada az sayıda bulunmakta ve çoğunlukla üniversiteler, enstitüler, savunma sanayisindeki büyük şirketler tarafından kullanılmaktadır.
Süper bilgisayarlar ile birlikte, özellikle son 10 yılda kişisel bilgisayarlar üzerinde de muazzam derecede paralel sistemler gelişmeye başlamıştır. Fakat bu sistemler, kişisel bilgisayarın merkezi işlem birimlerini değil de grafik işlem birimlerini oluşturmaktadır. Muazzam derecede paralel veri işleme kapasitesine sahip bu sistemler, kullanıcıların doğrudan grafik kart üzerlerinde program yürütmelerini sağlayan bir yapı olmadığından, uzun yıllar sadece grafik kartı üreticilerinin yayınladığı sürücüler üzerinden grafik üretmek için kullanılmıştır.

4.1. CUDA Teknolojisi

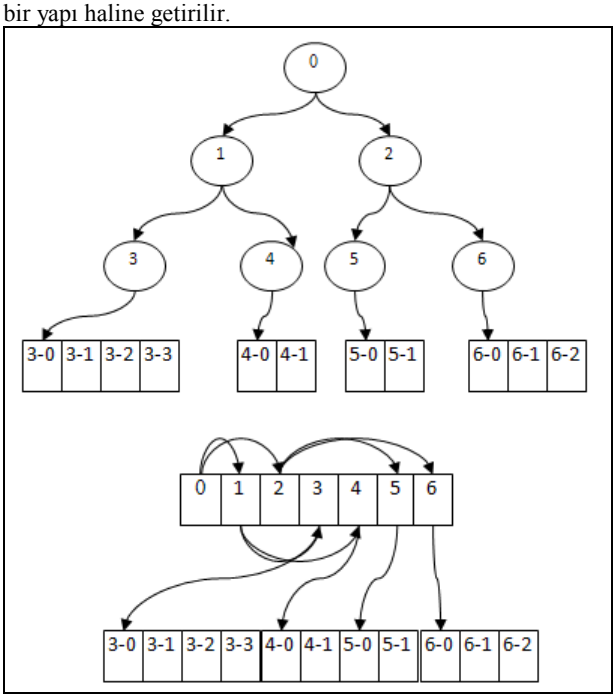
Özellikle son yıllarda, bilgisayar oyunları ve grafik tasarım programlarının gelişmesi, "Grafik İşlem Birimi (GİB)" adı verilen ve grafik kartlarının yerel merkezi işlem birimleri olarak nitelendirebileceğimiz bileşenlerin oldukça yüksek bir ivmeyle evrimleşmesine neden olmuştur. Zaman içerisinde GİB'ler, yüksek işlem kapasitesine sahip, muazzam derecede paralel, aynı anda birden fazla iplik işleyebilen, çok çekirdekli bir işlemciye dönüşmüştür.
CUDA™ (Compute Unified Device Architecture), grafik kartı üreticisi NVIDIA® tarafından 2006 yılının sonunda yayınlanmış genel maksatlı bir paralel veri işlemem mimarisidir. CUDA, yeni bir programlama modeli ve komut seti mimarisi ile NVIDIA üretimi GİB'lere sahip grafik kartlar üzerinde paralel hesaplama yapılabilmesine olanak sağlar. CUDA; C, Fortran, OpenCL gibi farklı dilleri, belirli ek komut ve kısıtlamalarla desteklemektedir.
CUDA'nın getirdiği en önemli kısıtlamalar:

- Nesneye yönelik programlama
- Özyineleme
- GİB üzerinde çalışan programda dinamik bellek ayrımı

işlemlerini desteklememesidir.
CUDA destekleyen grafik işlem birimleri, çoğunlukla 8'er çekirdeğe sahip çok çekirdekli birden fazla işlemcinin bir araya gelmesiyle oluşur. Çekirdeklerin 32KB'lik saklayıcıları dışında 16KB'lik çok hızlı çalışan ve sadece blok içerisindeki çekirdeklerin erişebileceği bir paylaşılan bellek ve bir de aygıt belleğine sahiptir. Aygıt belleği üzerinde genel bellek, değişmez bellek, yerel bellek ve doku belleği alanları bulunur. Aygıt belleği genel olarak yavaş çalışmakla birlikte, önbelleğe sahip olan değişmez bellek diğer alanlara göre daha yüksek bir hıza sahiptir.[8]

4.2. Kd-Tree'nin Grafik Kartı Belleği Üzerinde Gerçeklenmesi

MİB ve GİB'in kullandığı bellekler, birbirinden tamamen farklı olduğundan ve bu bellekler arasında doğrudan bir iletişim kurulamadığından, oluşturulan Kd-Tree'nin CUDA kullanılarak GİB üzerinde işlenebilmesi için öncelikle kullanılan ağaç yapısının grafik kartı belleğine aktarılması gerekir.
Oluşturulan ağaç yapısının sahip olduğu işaretçiler nedeniyle yapının olduğu gibi grafik kartı belleğine aktarılması mümkün değildir. Bununla birlikte dizilerin grafik kartı belleğine aktarılması mümkündür. Bu nedenle yapı, grafik kartı belleğine aktarılmaya uygun hale getirilmelidir.
Ağaç yapısı iki çeşit işaretçi içermektedir. Birinci çeşit, her düğümün çocuklarına işaret ettiği işaretçilerdir. Bu işaretçi yoğunluğundan kurtulmak için öncelikle ağaç, dizi üzerinde gerçeklenir. Dizi üzerinde gerçekleştirilmiş bir ağaçta, n'inci düğümün çocukları 2n+1'inci ve 2n+2'inci düğümlerdir.
İkinci çeşit işaretçi ise, yaprakların sahip olduğu üçgen dizilerine işaret eden işaretçilerdir. Üçgen dizilerinin sahip olduğu değişken boyutlar yüzünden bu işaretçilerden kurtulmak mümkün değildir. Bununla birlikte bu işaretçilerin gösterdiği adresleri bulmak kolaylaştırılabilir. Bunun için öncelikle diziler bir bellek havuzunda birleştirilir. Dizinin ilk elemanının adresi alınarak ilk yaprağa atılır. Diğer yaprakların sahip olduğu işaretçilerin gösterdiği adres değerleri, ilk adres üzerinden dizilerin her birinin sahip olduğu üçgen sayılarına eşit olan öteleme değerleri kullanılarak hesaplanır. Bu şekilde ağaç, iki diziden oluşan bir yapı haline getirilir.



*Şekil 3:*Ağaç yapısının diziler üzerinde gerçeklenmesi

4.3. İpliklerin yaratılması

Bir CUDA programında yaratılacak iplik sayısı için iki parametre önemlidir. Bunlardan birincisi GİB üzerindeki

çekirdek bloklarından kaç tanesinin kullanılacağını, ikincisi ise bu blokların her birinde kaçar iplik kullanılacağını belirtir. Çoğunlukla, bloklarda kullanılacak iplik sayısı (blok boyutu) sabit bir sayı seçilerek, kullanılacak blok sayısı hesaplanacak toplam eleman sayısına göre değişken olarak hesaplanır. Daha sonra, CUDA çekirdek fonksiyonu çağrılırken bu değerler fonksiyona özel parametreler şeklinde aktarılır. Aktarılan bu parametrelere göre iplikler, CUDA tarafından otomatik olarak oluşturulur.

4.4. Işınların İpliklere Paylaştırılması

CUDA programlarında, işlenecek verinin her bir birimi bir iplik tarafından işlenir. Burada önemli olan, işlenecek bu veri birimini doğru olarak belirlemektir.
Bu çalışmadaki veri birimi, oluşturulacak görüntünün her bir pikseli olarak seçilmiştir. Bu durumda toplam iplik sayısı, toplam piksel sayısına, yani toplam ışın sayısına eşittir. Blok boyutu sabit bir sayı seçildiğinde; blok sayısı toplam sütun sayısının blok boyutuna bölünmesiyle bulunur.

4.5. Işınların İpliklerde İşlenmesi

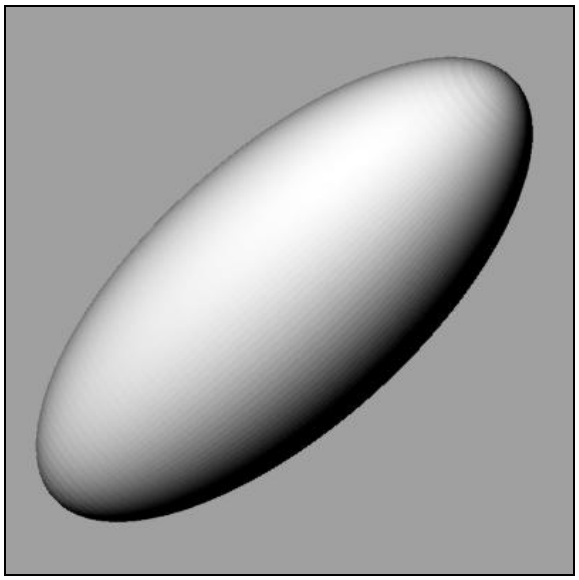
Işınların ipliklerde işlenebilmesi için, öncelikle her ipliğin sorumlu olduğu ışınları bilmesi gerekir. Bunun için, her ipliğe ayrı ayrı bilgi gönderilmesine gerek yoktur.
CUDA'da çalışan her iplikte, ipliğin kaç numaralı bloğun kaçıncı ipliği olduğunu saklayan değişkenler bulunur. "blokckIdx" değişkeni blok numarasını, "blockDim" değişkeni toplam blok sayısını, "threadIdx" değişkeni ise blok içerisindeki iplik numarasını saklar.
İşlenecek sütun numarası belirlendiğine göre, bu sütunun her satırı için Kd-Tree arama algoritması yürütülür. Fakat daha önce belirtilen kısıtlamalar nedeniyle özyinelemeli olan bu fonksiyon, GİB üzerinde doğrudan yürütülemez.
Kd-Tree üzerinde arama fonksiyonunun GİB üzerinde yürütülebilmesi için, öncelikle özyinelemeden kurtarılması gerekmektedir. Bunun için bir yığın benzetimi yapısı kullanılabilir. Bu yapıda, her düğüm işlendiğinde çocuk düğümler için aynı fonksiyonun özyinelemeli olarak çağrılması yerine, her adımda işlenilmesi istenen düğüm indeksleri bir yığına atılır. Bir sonraki adımda bu indeks yığından çekilerek, işlem yığından çekilen indekse sahip düğüm için tekrarlanır. Bu işlem yığın boşalıncaya kadar devam eder.

4.6. Bellek Kullanımının İyileştirilmesi

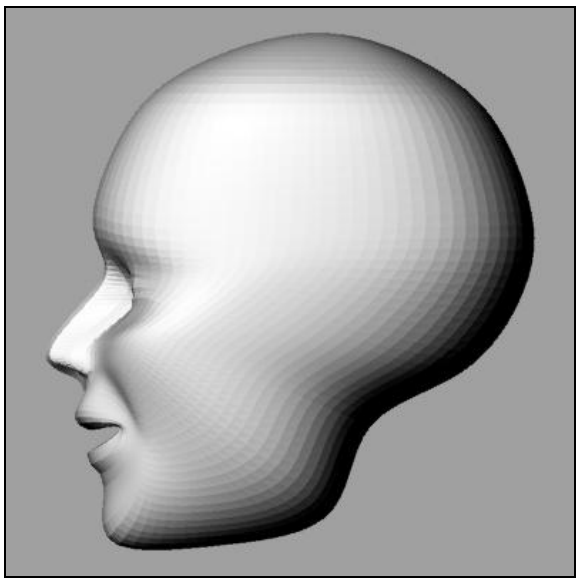
Bilgisayar belleğinden grafik kartı belleğine aktarılan değişkenler, aygıt belleği üzerinde bulunan genel bellekte saklanır. Yürütülen bütün iplikler bu belleğe erişebilirler ancak hem genel belleğin yavaş çalışması, hem de binlerce ipliğin aynı anda aynı bellek gözlerine ulaşmaya çalışmasından ileri gelen yoğun trafik nedeniyle bellek kullanımında bir darboğaz oluşur. Bu darboğaz, paralelleştirmeden istenen verimin alınmasını engeller.
CUDA üzerinde çalışan programlarda en çok dikkat edilmesi gereken noktalardan biri, genel bellek kullanımının en aza indirilerek mümkün olduğunca saklayıcıların ve paylaşılan belleğin kullanılmasıdır. Özellikle fazla üçgene sahip geometrilerde, üçgenlerin kapladığı büyük alan nedeniyle yapının tamamının paylaşılan belleğe aktarılması imkansızdır. Bununla birlikte üçgenler hariç olmak üzere sadece ağacı oluşturan düğümlerin paylaşılan belleğe aktarılması da, uygulamanın başarısını önemli ölçüde artırmaktadır.

5. Uygulama Sonuçları

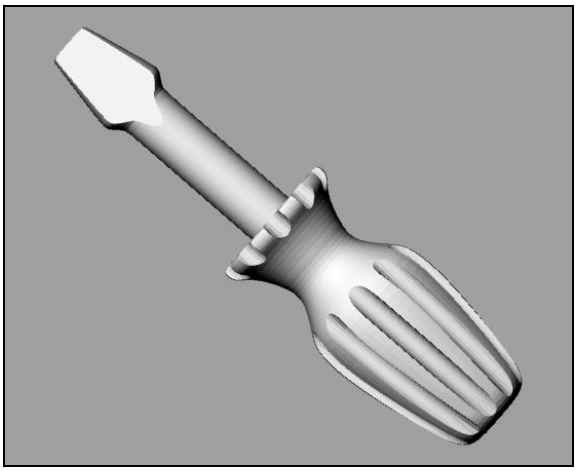
Bu bölümde, GİB üzerinde çalışan ışın izleme uygulaması, aynı uygulamanın MİB üzerinde çalışan sürümü ile karşılaştırılmış ve hızlanma oranları hesaplanmıştır. Ayrıca GİB üzerinde çalışan bu uygulamanın başarımı, ticari bir BDT yazılımı olan ve bir ışın izleme aracına sahip "Rhinoceros"un başarımı ile karşılaştırılmıştır.[9] Karşılaştırmalar, üç ayrı geometri ve beş ayrı çözünürlük değeri kullanılarak yapılmıştır. Kullanılan örnek geometriler, 21182 üçgene sahip bir elipsoid modeli, 14360 üçgene sahip bir insan kafası modeli ve 16400 üçgene sahip bir tornavida modelidir. Her sonuç, 20 koşumun ortalama değerini belirtir.



Şekil 4: Örnek elipsoid modeli



Şekil 5: Örnek insan kafası modeli



Şekil 6: Örnek tornavida modeli

MİB üzerinde çalışan ardışıl programla GİB üzerinde çalışan paralel program arasındaki karşılaştırma sonuçları şunlardır:

Tablo 1: Elipsoid modeli için MİB-GİB karşılaştırması

Çözünürlük	200 x 200	400 x 400	800 x 800	1600 x 1600	2400 x 2400
MİB süresi (saniye)	3,20	12,672	50,875	210,531	469,467
GİB süresi (saniye)	0,054	0,203	0,750	2,922	6,469
Hızlanma oranı	59,3	62,42	67,45	72,05	72,57

Tablo 2: İnsan kafası modeli için MİB-GİB karşılaştırması

Çözünürlük	200 x 200	400 x 400	800 x 800	1600 x 1600	2400 x 2400
MİB süresi (saniye)	2,11	8,422	34,016	140,063	313,844
GİB süresi (saniye)	0,06	0,140	0,485	2,002	4,226
Hızlanma oranı	34,0	60,16	70,14	69,96	74,26

Tablo 3: Tornavida modeli için MİB-GİB karşılaştırması

Çözünürlük	200 x 200	400 x 400	800 x 800	1600 x 1600	2400 x 2400
MİB süresi (saniye)	2,219	8,812	35,422	146,781	329,765
GİB süresi (saniye)	0,035	0,140	0,453	1,859	3,875
Hızlanma oranı	63,4	62,94	78,19	78,96	85,10

Yapılan karşılaştırmalar sonucunda en düşük hızlanma değeri 34,02 ile 200x200 çözünürlüğe sahip insan kafası görüntüsünün oluşturulmasında, en yüksek hızlanma değeri 85,10 ile 2400x2400 çözünürlüklü tornavida görüntüsünün oluşturulmasında gözlemlenmiştir. Ortalama hızlanma değeri 67,40 olup, hızlanma değerleri ışın sayısıyla birlikte artan bir eğilim göstermiştir. Ayrıca, koşum süresinin cismin modellendiği üçgen sayısıyla beraber arttığı gözlemlenmiştir. GİB üzerinde çalışan program ile Rhinoceros yazılımının başarımlarının karşılaştırılması şu şekildedir:

Tablo 4: Elipsoid modeli için GİB-Rhino karşılaştırması

Çözünürlük	200 x 200	400 x 400	800 x 800	1600 x 1600	2400 x 2400
Rhino süresi (saniye)	0,26	0,708	2,323	11,551	22,952
GİB süresi (saniye)	0,05	0,203	0,750	2,922	6,469

Tablo 5: İnsan kafası modeli için GİB-Rhino karşılaştırması

Çözünürlük	200 x 200	400 x 400	800 x 800	1600 x 1600	2400 x 2400
Rhino süresi (saniye)	0,20	0,572	1,928	7,635	17,659
GİB süresi (saniye)	0,06	0,140	0,485	2,002	4,226

Tablo 5: Tornavida modeli için GİB-Rhino karşılaştırması

Çözünürlük	200 x 200	400 x 400	800 x 800	1600 x 1600	2400 x 2400
Rhino süresi (saniye)	0.21	0.536	1.679	6.599	15.277
GİB süresi (saniye)	0,035	0,140	0,453	1,859	3,875

6. Sonuç ve Öneriler

Bu çalışmada, grafik kartı üzerinde çalışan bir ışın izleme uygulaması geliştirilmiştir. Uygulamanın amacı, 3 boyutlu cisimlerden mümkün olduğunca hızlı ve verimli bir şekilde fotogerçekçi görüntüler üretmektir.

Bu çalışmada ilk adım olarak 3 boyutlu cisimler üçgenlerle modellenerek bilgisayara aktarılmıştır. Ardından Kd-Tree uzay bölmeleme algoritmasıyla ışın izleme işleminin verimli olarak yürütülebilmesi için altyapı hazırlanmıştır. Hazırlanan bu yapı, CUDA kullanılarak grafik kartı belleğine aktarılmıştır. Ardından GİB üzerinde ışınları temsil eden iplikler oluşturulmuştur. Her ipliğin, sorumlu olduğu ışını Kd-Tree yapısı üzerinde izleyerek ilgili piksellere ait renk değerlerini bulmaları sağlanmıştır. Son olarak hesaplanan renk değerleri, bit eşlem resmi olarak görüntülenmiştir.

Çalışmanın sonucunda elde edilen uygulamanın başarımı MİB üzerinde ardışıl olarak çalışan sürümüyle ve ticari bir ışın izleme uygulamasıyla karşılaştırılmış ve oldukça yüksek hızlanma değerleri elde edilmiştir.

Bu çalışmada geliştirilen ışın izleme uygulaması, grafik kartı üzerinde paralel olarak yürütülebilecek benzer uygulamalar için bir örnek teşkil eder. Bu uygulama, daha gelişmiş ışık benzetimi yöntemleri kullanılarak ve GİB ile grafik kartı belleğinin daha verimli bir şekilde kullanılması sağlanarak mükemmelleştirilebilir.

Bunun dışında bu uygulama, gerekli değişiklikler yapılarak ışın izleme yönteminden yararlanan yüksek frekans elektromanyetik ve akustik dalgaların benzetiminde kullanılabilir.

7. Kaynakça

[1] Möller, T. and Trumbore, B., 2005: Fast, Minimum Storage Ray/Triangle Intersection, *International Conference on Computer Graphics and Interactive Technique*, Los Angeles, California, USA, July 2005.

[2] Sherrod, A., 2007: *Data Structures and Algorithms for Game Developers*, pp. 39-48, Charles River Media, Rockland, MA, USA.

[3] Wald, I., and Havran, V., 2006: On Building Fast Kd-Trees for Ray Tracing, and On Doing That In O(N log N), *RT'06: IEEE Symposium on Interactive Ray Tracing*, Salt Lake City, UT, USA, September 2006.

[4] Möller, T., 2005: Fast 3D Triangle-Box Overlap Testing, *International Conference on Computer Graphics and Interactive Techniques*, Los Angeles, California, USA, July 2005.

[5] Jansen, F. W., 1986: Data Structures for Ray Tracing, *Data structures for Raster Graphics, Proceedings Workshop*, Berlin, Germany, June 1986.

[6] Smits, B., 2002: Efficient Bounding Box Intersection, *Ray Tracing News*, Vol. 15, no. 1,pp. 7-9.

[7] Williams, A., Barrus, S., Morley, R. K. and Shirley, P., 2005: An Efficient and Robust Ray-Box Intersection Algorithm, *International Conference on Computer Graphics and Interactive Technique*, Los Angeles, California, USA, July 2005.

[8] CUDA C Programming Guide, <*http://developer.nvidia.com/object /cuda_3_0_downloads.html*>, alındığı tarih 03.03.2010.

[9] <*http://www.rhino3d.com/*>, alındığı tarih 01.03.2010.