TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ÖZYİNELEMELİ ÇİZİM TABANLI SINIRSIZ KAPALI ALAN SANAL GERÇEKLİK MOTORU GELİŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Ali Emre GÜLCÜ

Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı: Prof. Dr. Ali Aydın SELÇUK

ARALIK 2022

ÖZET

Yüksek Lisans Tezi

ÖZYİNELEMELİ ÇİZİM TABANLI SINIRSIZ KAPALI ALAN SANAL GERÇEKLİK MOTORU GELİŞTİRİLMESİ

Ali Emre GÜLCÜ

TOBB Ekonomi ve Teknoloji Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı: Prof. Dr. Ali Aydın SELÇUK

Tarih: Aralık 2022

Sanal gerçeklik donanımları geçmişten beri askeri simülasyonlar ve tıbbi uygulamalar gibi gerçeklik gerektiren alanlarda kullanılmaktadır. Gelişen teknoloji ile birlikte maliyetin düşmesiyle sanal gerçeklik donanımları sıradan kullanıcılar tarafından da erişilebilir bir hale geldi. Sanal gerçeklik uygulamalarının simülasyon ve eğitimlerin yanı sıra yaygın olarak oyun ve sinema sektöründe de yer bulmasıyla farklı gereksinimler ortaya çıkmaya başladı.

Tıbbi uygulama ve askeri simülasyonlarda özel alanlar tahsis edilip ek donanımlar kullanılabildiği için rahat ve kullanılabilir bir sanal gerçeklik deneyimi sunulabilmektedir. Ancak kısıtlı bir fiziksel alanda ve standart donanımla sanal gerçeklik uygulamalarını deneyimleyen kullanıcılar için bu her zaman mümkün olmamaktadır. Yaygın sanal gerçeklik donanımları genellikle kullanıcının kapalı bir oda içerisinde kullanacağı varsayımı ile tasarlanır. Hareketin tamamen doğal yürüme ile sağlanamayacağı bu ortamlar için ise farklı hareket yöntemleri geliştirilmiştir.

Kapalı fiziksel ortamlarda büyük sanal ortamların deneyimlenebilmesi için öne çıkan en yaygın yöntemler oyun kolu ile yürüme ve ışınlanma yöntemleridir. Bu yöntemler her ne kadar kısıtlı alan problemini çözseler de, beyin tarafından gözlemlenen hareket ile vücut tarafından hissedilen hareketin uyuşmazlığı sebebiyle mide bulantısı ve baş dönmesi gibi yan etkilere sebep olmaktadır. Bu yan etkileri çözmeyi amaçlayan geçmiş yöntemlerin bazıları ek donanım gerektirirken, bazıları da sanal ortamı

bozmaktadır. Bu çalışmada yan etkileri ek donanıma gerek duymadan aşmak için portal tabanlı, sanal alanı üst üste bindirmeli bir yöntem önerip ortam tasarımı için kurallar belirledik. Sunduğumuzun yöntemin, dikkat dağıtma ve haritalama gibi doğal yürüyüş tabanlı diğer geçmiş yöntemlerle ayrıntılı karşılaştırmasını yaptık.

Hareket yöntemlerine benzer bir şekilde, performans kaybının da sanal gerçeklikte kullanıcı üzerinde yan etkilere sebep olması sebebiyle, bahsi geçen grafik motorunun geliştirilmesinde en yeni teknolojiler kullanılmıştır. 1990'ların grafik mimarisine göre geliştirilmiş olan ve uyumluluk kısıtları sebebiyle yeni donanımlara ayak uydurmakta zorluk yaşayan OpenGL gibi eski bir grafik uygulama arayüzü yerine, modern grafik kartları ve çok çekirdekli işlemci mimarileri göz önünde bulundurularak geliştirilmiş olan Vulkan grafik uygulama geliştirme arayüzü tercih edilmiştir. Portal tabanlı ortamların sunulması için geliştirdiğimiz grafik motoru ise bu arayüzle tam uyumlu olacak şekilde tasarlanmıştır.

Çalışma sonunda OpenGL ve Vulkan kullanılarak ayrı ayrı geliştirilmiş olan grafik motorları arasında ayrıntılı performans kıyaslaması yapılarak yeni teknolojilerin kullanılmasıyla ciddi performans artışları görüldüğü gözlemlendi. Buna ek olarak ortaya çıkan uygulamayı modern başka bir sınırsız alan yöntemi olan ortam haritalama yöntemi ile kıyaslamak için bir kullanıcı testi gerçekleştirildi. Kullanıcı testlerinin sonucunda çalışmamız büyük çoğunluk tarafından tercih edilirken özgünlüğünden dolayı da olumlu geri beslemeler aldı.

Anahtar Kelimeler: Sanal gerçeklik, Hareket yöntemleri, Sınırsız alan, Portallar, Hücre-portal çizgeleri, Ortam eşleme, Çok çekirdekli grafik motoru, Modern grafik motoru, Vulkan, OpenGL

ABSTRACT

Master of Science

RECURSIVE RENDERING BASED INFINITE CLOSED SPACE VIRTUAL REALITY ENGINE DEVELOPMENT

Ali Emre GÜLCÜ

TOBB University of Economics and Technology
Institute of Natural and Applied Sciences
Department of Computer Engineering

Supervisor: Prof. Dr. Ali Aydın SELÇUK

Date: December 2022

Virtual reality hardware has been in use for a while for applications where fidelity matters, like military simulations and medical applications. With the recent leaps in technology, virtual reality hardware costs decreased and the devices became more available to the regular users. In addition to simulation and training applications, virtual reality gained popularity in movie and game industries. This introduced new requirements for virtual reality hardware and applications.

More intuitive military and medical simulations can be achieved with specialized hardware and environment. However, this is not always possible for a regular user with limited space and standard hardware. Mainstream virtual reality hardware is often designed for home usage with the assumption of it being used in regular rooms where the virtual movement cannot be achieved with the natural walking alone. Several virtual movement methods are developed to overcome space limitations.

The most popular virtual locomotion methods are teleportation and shifting using controllers. While these methods solve the movement related problems, they introduce physiological problems such as naussea and dizziness, which are caused by mismatch of the perceived motion with the real movement. Past methods that resolve these side effects either requires specialized hardware or distorts the virtual environment. To solve these limitaions without side effects and requiring extra hardware, we propose a portal based overlapping environment method and defined an environment design

ruleset for infinite spaces. We compared our proposed method to other natural walking based methods such as distraction and mapping.

In addition to locomotion methods, low performance is also known to cause physiological side effects. We used the latest graphical application development interfaces while developing our grahics engine for significant performance improvement. Instead of OpenGL, which was designed with 1990s graphics architecture in mind and comes with a lot of technical debt caused by backward compatibility requirements, we opted to use Vulkan, which is entirely designed for modern graphics cards and multi core processors. Our portal based rendering engine is designed and optimized to be entirely compatible with these modern architechtures.

In order to quantitatively compare performance difference, we also implemented a reference renderer in OpenGL. We observed that significant performance gains are achieved by using modern graphics technologies. In addition to performance comparisons, we conducted a user experience study to compare our portal based method to another modern environment mapping method. Most users preferred our method over the other one and gave positive feedbacks regarding its innovative approach.

Keywords: Virtual Reality, Locomotion methods, Infinite spaces, Portals, Cell-portal graphs, Environment mapping, Multi-threaded graphics engine, Modern graphics engine, Vulkan, OpenGL

İÇİNDEKİLER

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

KISALTMALAR

API : Uygulama Geliştirme Arayüzü (Application Programming Interface)
CPU : Mekezi İşlemci (Central Processing Unit)
FPS : Saniye Başına Kare Sayısı (Frames Per Second)
GPU : Grafik İşlemcisi (Graphics Processing Unit)
VR : Sanal Gerçeklik (Virtual Reality)

1. GİRİŞ

Sanal gerçeklik, sıradan bilgisayar uygulamalarına kıyasla kullanıcıya verdiği gerçeklik hissi sebebiyle spesifik konularda sıradan uygulamaların önüne geçmektedir. Gerçeklik gerektiren simülasyonlar bunun bir numaralı örneğidir. Uçus simülasyonları ve askeri eğitimler gibi kullanımlarda sıradan bilgisayar ekranı yerine tam bir çevre görüşü sağlayan kafalıklar uygulamanın gerçekliğini ciddi derecede artırmaktadır. Bu tür simülasyonlar genellikle büyük enstitüler tarafından sağlandığı için donanımlar çok sayıda kullanıcı tarafından kullanılır. Bu sebeple donanım maliyeti ön planda olmaz ve sanal gerçeklik için özel geniş odalar tasarlanabilir veya kokpit ortamı gibi alanlar birebir benzer şekilde tasarlanabilir. Böylece kullanıcıya çok gerçekçi bir deneyim sunulur.

Sanal gerçekliğin yaygınlaşmasıyla düşük maliyetli donanıma erişim sağlayabilen sıradan kullanıcı için ise bu tür özel tasarımlar her zaman mümkün değildir. Günümüzde sanal gerçeklik kullanıcı kitlesinin büyük çoğunluğunu eğlence amaçlı sanal gerçeklik donanımı edinmiş sıradan kullanıcılar oluşturmaktadır. Ev ortamında kısıtlı bir fiziksel alanda sanal gerçeklik kullanan bu kullanıcı için hareket kabiliyetini kısıtlamayan koşu bandı gibi özel donanımlar yüksek maliyet veya fazla yer kaplamasından dolayı genellikle verimli bir opsiyon değildir.

Sınırlı alanda sanal gerçeklik kullanabilmek için çeşitli yöntemler mevcuttur. Fare ve klavye kullanımı ve sanal gerçeklik oyun kolları yardımıya hareket etmek gibi temel yöntemler, gerçek yürüme hareketi ile desteklenmediği için kullanıcıda fiziksek yan etkilere sebep olabilir. Baş dönmesi, denge kaybı ve mide bulantısı şeklinde ortaya çıkan bu yan etkileri aşmak için çeşitli yöntemler geliştirilmiştir. Yeterli fiziksel alan olduğu takdirde kullanıcının doğal yürümesine bağlı bir sanal hareket yöntemi, bu tür yan etkilere sebep olmadığından genelde tercih edilmesi gereken yöntem olsa da fiziksel alan çoğunlukla kısıtlı olduğundan bu her zaman uygulanamaz. Tıbbi ve askeri uygulamalarda yürüme bandı gibi ek donanımlar kullanılırken sıradan kullanıcı tarafında yeniden yönlendirme ve ortam haritalama gibi yazılım tabanlı çözümler tercih edilir.

1.1 Tezin Amacı ve Katkıları

Tez kapsamında literatürdeki hareket yöntemleri ayrıntılı bir şekilde incelenmiş ve analiz edilmiştir. Bu bilgiler ışığında yeni bir hareket yöntemi önerilerek bu yöntemin denenebileceği sanal gerçeklik uygulaması geliştirilmiştir. Çalışmalar kapsamında ortaya koyulan katkılar şu şekildedir:

- Ek donanım gerektirmeyen, yaygın sanal gerçeklik donanımlarıyla tamamen uyumlu, ortamın lineerliğini bozmayan ve fiziksel yan etkileri en aza indiren özgün portal tabanlı bir sınırsız alan tasarım yöntemi tüm kural ve ayrıntılarıyla sunulmuştur.
- Sunulan bu yöntem ışığında test ortamları tasarlandı ve bu ortamların denenebileceği Vulkan tabanlı, çoklu çekirdek destekli bir grafik motoru geliştirilmiştir.
- Vulkan ile geliştirilen modern grafik motorunun karşılaştırmalı performans testi için aynı algoritmalar kullanılarak OpenGL tabanlı bir grafik motoru da geliştirilip performans analizi yapılmıştır.
- Grafik motoru geliştirilirken grafik kartı mimarisi bazında karşılaşılan bazı eksikler sebebiyle serbest portal çizimleri için özgün ve genelleştirilebilir bir algoritma geliştirilmiştir.
- Ortam haritalama yöntemlerinde (Bölüm 3.5) ayrıntılı incelenecek olan çalışma ile kıyaslamalı kullanıcı testi gerçekleştirildi. Bağımsız kullanıcılara iki çalışma da denetilerek değerlendirmeleri ve önerileri alındı.

Çalışma süresince aşılması gereken birçok zorlukla karşılaşıldı. Bunlardan en önemlileri şu şekilde sıralanabilir:

- Portal çizme yöntemine karar verilmesi. Çalışmanın ilk aşamalarında portal çizimleri için doku resimleri (textures) kullanıldı ancak bunun yol açtığı sıkıntılardan dolayı kalıp arabelleği (stencil buffer) kullanımına geçildi.
- Kalıp arabelleği kullanımında grafik kartlarının mimari kısıtlarından dolayı serbest portal çizimleri için özgün bir algoritma ve veri yapısı geliştirilmesi gerekti.
- Vulkan'ın halen yeni bir teknoloji olması sebebiyle spesifik durumlarda kaynak eksiklikleri ile karşılaşıldı. Bu durumlarda mümkün olduğunca Vulkan ekosistemine katkıda bulunarak çözümler geliştirildi.

- Çok çekirdekli grafik motorunun geliştirilmesinde kapsamlı bir senkronizasyon çalışması yapıldı. Vulkan'ın düşük seviyeli bir API olması ve hem CPU hem GPU ile olan sekronizasyonun geliştiriciye bırakılması sebebiyle bu konuda kapsamlı bir araştırma ve geliştirme yapıldı.
- Patent kısıtları sebebiyle kaynak kodunun bizimle paylaşılmaması sebebiyle kıyaslı kullanıcı testlerinde kullanılacak olan referans grafik motorunun [12] geliştirmesi tarafımızca baştan yapıldı.

Aşılan bu zorluklar, ilgili bölümlerde ayrıntılı olarak tartışılacaktır.

1.2 Tez Organizasyonu

Bu tez altı bölümden oluşmaktadır. Bölüm 2'de grafik API'ları, portallar ve sanal gerçeklik hakkında temel bilgiler verilecektir. Bölüm 3'te sanal gerçeklikte hareket modelleri hakkında yapılan çalışmalar kategorize edilip ayrıntılı bir şekilde incelenecektir. Eski usül yöntemler, ışınlanma, özel donanımlar, yönlendirme ve haritalama yöntemlerinin her biri kendi alt başlığında ele alınacaktır. Bölüm 4'te oyun motoru ayrıntılı olarak açıklanacaktır. Ortam tasarımı, portal çizimler, veri yapıları ve grafik motoru bu bölümde ele alınacaktır. Bölüm 5'te kullanıcı ve performans testlerine yer verilerek, kullanıcıların geri bildirimleri incelenecektir. Bölüm 6'da deney sonuçları yorumlanarak gelecekte yapılabilecek çalışmalar aktarılacaktır.

2. ÖN BİLGİ

Sanal gerçeklik deneyimi, fiziksel etkenlerden ve çizim performansından büyük ölçüde etkilenmektedir. Saniyede üretilen kare sayısının (framerate) belirli bir değerin altında kalması kullanıcı üzerinde baş dönmesi ve mide bulantısı gibi etkilere sebep olmaktadır. Bu sebeple bu çalışmada grafik performansı ayrıntılı bir şekilde incelenecektir.

Elektronik ekran teknolojilerinin ve hareketli medya formatlarının tamamı saniyede belli sayıda gösterilen görseller üzerine kuruludur. 20. yüzyılın ortalarında saniyede 24 kareye kadar algıladığı zannedilen insan gözünün, zaman geçtikçe daha yüksek sayıda kare algılayabildiği ortaya çıkmıştır. Günümüzde standart kabul edilen saniyede 60 kare değerinin bile profesyonel e-spor oyuncuları tarafından yetersiz bulunduğu ve performanslarını etkilediği bilinmektedir. Bu nedenle günümüz ekran teknolojisi standartları saniyede 120 ve 144 kare değerlerine doğru kaymaktadır.

Sanal gerçeklikte ise saniyede üretilen kare sayısının belli bir seviye altında kalması sadece uygulamanın akıcılığını kötü etkilemekle kalmaz ayrıca kullanıcı üzerinde olumsuz fiziksel yan etkilere de sebep olur. Sıradan ekran teknolojilerinde, insan gözünün daha geniş açıda görebildiği gerçek dünya da olduğundan, vücut kare sayısı azlığının sadece ekrandaki içeriğe has olduğunun ayırdına varabilir. Ancak kullanıcıyı tamamen içerisine alan bir sanal gerçeklik deneyiminde referans alınabilecek bir dışarı görüşü mevcut olmadığından düşük kare sayısı mide bulantısı ve baş dönmesine sebep olmaktadır. Bu sebeple, sıradan uygulamalara kıyasla sanal gerçeklik uygulamalarında performansın önemi daha da artmaktadır. Her sanal gerçeklik uygulamasının, kullanıcı memnuniyeti için belli bir kare sayısı üzerinde kalması şarttır. Performansı etkileyen ana unsurlar ve iyileştirme yöntemleri ayrıntılı olarak incelenmiştir.

2.1 Grafik API'ları

Bilgisayar grafiklerinin 90'lı yıllarda hızla gelişmesi ve grafik uygulamalarının yaygınlaşması sonucu standart grafik API'larına ihtiyaç duyulmaya başlandı. Bu ihtiyaç sonrası gerek işletim sistemi özelinde gerekse çoklu platform grafik API'larının geliştirilmesi için yoğun çaba sarfedilerek DirectX ve OpenGL API'ları ortaya çıkarıldı. Bu API'lar geliştiricinin GPU üzerinde kod koşturabilmesi için sunulan yapılardır. CPU'dan farklı bir mimari ve hafızaya sahip GPU'lara standart bir arayüz sağlarlar.

Durum makinası (state machine) mantığıyla çalışan ve sabit boru hattı (fixed pipeline) işlevselliğine sahip olan API'lar standart haline gelerek ortak bir dil sağlamaları sebebiyle grafik uygulaması geliştirme alanında büyük kolaylıklar sağlamıştır. Grafik donanımı mimarisine paralel olarak bu yapılar da geliştirilmeye devam etmiştir. Ancak grafik mimarilerinin çok hızlı değişmesi sebebiyle bu API'larda geriye dönük uyumluluk sağlama isteği yanında bir sürdürülebilirlik yükü de getirmiştir.

Vulkan, daha önceki grafik API'ları gibi, GPU'lar üzerinde çoklu platform bir soyutlama sağlamak üzere tasarlanmıştır. Bu API'ların birçoğunun sorunu, tasarlandıkları çağın gereği olarak sabit bir işlevselliğe sahip olmalarıdır. Şu an yapılandırılabilir (programmable) olsalar da geçmişten gelen bu yük kısıtlayıcı bir etkendir. Eski API'larda programcı verteks verisini standart bir formatta vermeliydi ve ışıklandırma ile gölgelendirme konularında GPU üreticilerinin sağladıklarına bağımlı kalıyordu.

Ekran kartı teknolojileri geliştikçe daha programlanabilir işlevsellikler sağlamaya başladılar. Tüm bunlar var olan API'lara bir şekilde entegre edilmek zorundaydı. Bu da ideal olmayan soyutlamalara sebep olmaya başladı. Sürücü tarafında programcının niyetini anlamaya çalışmak ve bunları modern grafik mimarilerine eşlemek gerekiyordu. Oyun performanslarını artırmak için sıklıkla sürücü güncellemeleri olması ve bunlardan bazılarının performansı büyük ölçüde artırması da bu sebeptendir. Sürücülerin bu denli karmaşık olmasından dolayı geliştiriciler de çoğu zaman farklı üreticiler arasındaki tutarsızlıklarla baş etmek zorunda kalıyordu. Gölgelendiriciler tarafından kabul edilen sözdizimlerinin farklılıkları bu duruma örnek verilebilir. Eklenen yeni özelliklerin yanı sıra, geçtiğimiz yıllarda güçlü grafik donanımlarına sahip mobil cihazların sayısında da hızlı bir artış yaşandı. Bu mobil GPU'lar, enerji tüketimi ve boyut kısıtlarından dolayı farklı mimarilere sahipler ve Vulkan'ın sağladığı özelliklerin büyük ölçüde faydasını görmektedirler [17]. Örneğin parçalı çizim (tiled rendering), programcıya bu özellik üzerinde yeterince kontrol verildiği takdirde performans artışlarına sebep olabilmektedir. Multimedya performans karşılaştırmalarında da Vulkan'ın OpenGL'e kıyasla performans kazancı sağladığı görülmektedir.

Eski mimarilerin yol açtığı bir diğer sorun ise çok çekirdekli işlemci desteğinin kısıtlı olmasıdır. Durum makinesi gibi davranan eski API'larda bu CPU tarafında darboğazlara sebep olmaktadır. Ancak Vulkan'ın komut arabelleği (command buffer) ve işlem kuyruğu (command queue) tabanlı mimarisi bu problemin üstesinden gelmektedir [18]. Eski mimarilerde olduğu gibi yapılacak işleme göre durumları anlık değiştirmek yerine,

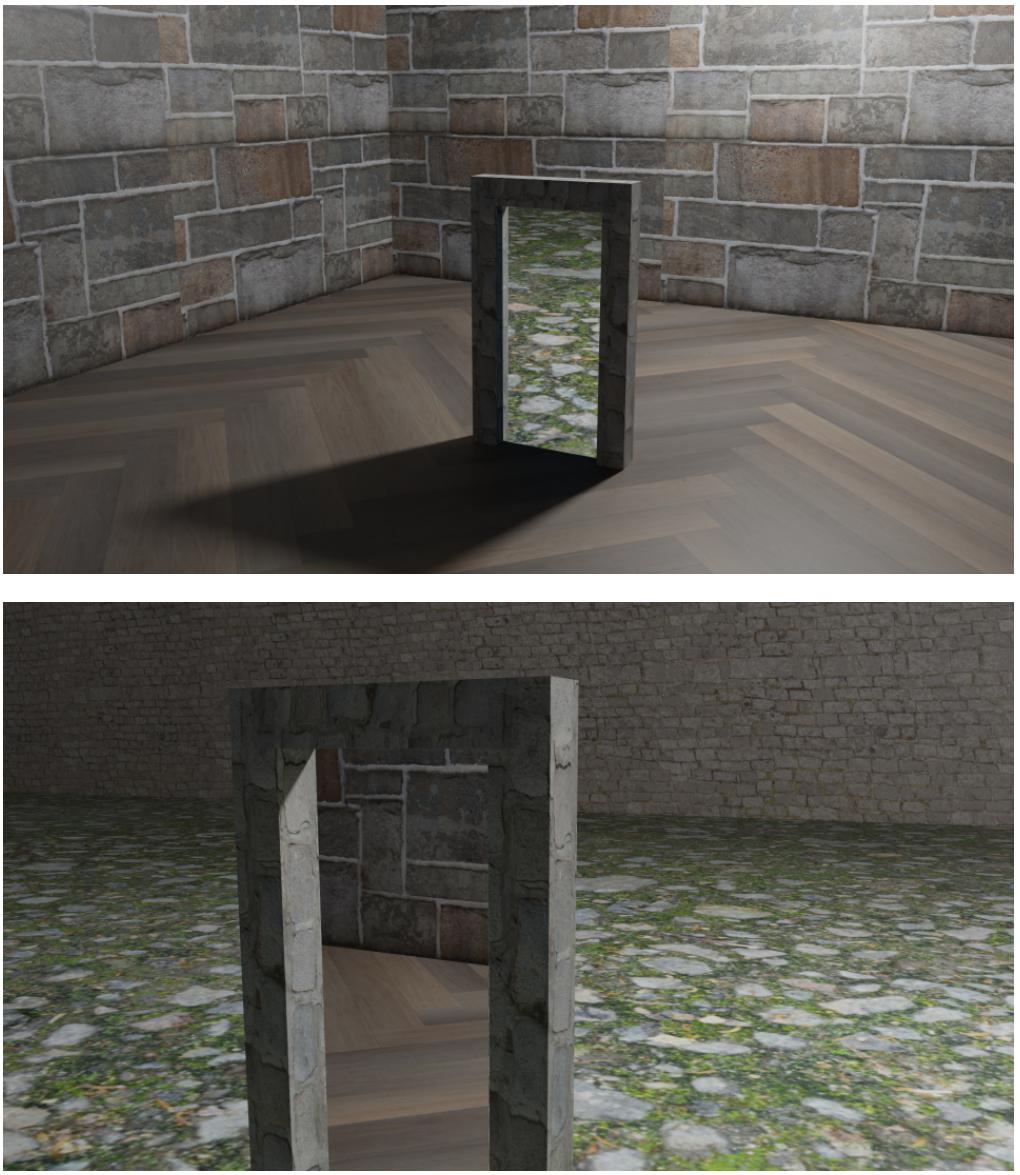
işlemlerin tümü önceden bu yapılara kaydedilerek grafik kartına gönderilir ve grafik kartına ön bir bilgi verildiği için optimizasyonlara olanak sağlanır.

Vulkan bahsi geçen bu problemleri, modern grafik işlemci mimarilerine göre sıfırdan tasarlandığı için çözmektedir. Programcıya daha detaylı bir API sunup niyetini daha ayrıntılı anlatmasına izin vererek, sürücüyü fazlalık yüklerden kurtarmaktadır. Birden fazla iş parçacığının eş zamanlı olarak komut göndermesine olanak sağlar. Standart bir bayt kodu formatı ve derleyicisi getirerek, gölgelendirici derlemelerindeki tutarsızlıklardan kurtarmaktadır. Son olarak, modern grafik kartlarının grafik işlemleri dışındaki kullanımlarını da göz önünde bulundurarak, grafik ve hesaplama işlevlerinin hepsini tek bir çatı altında birleştirmiştir.

2.2 Portal Çizimleri

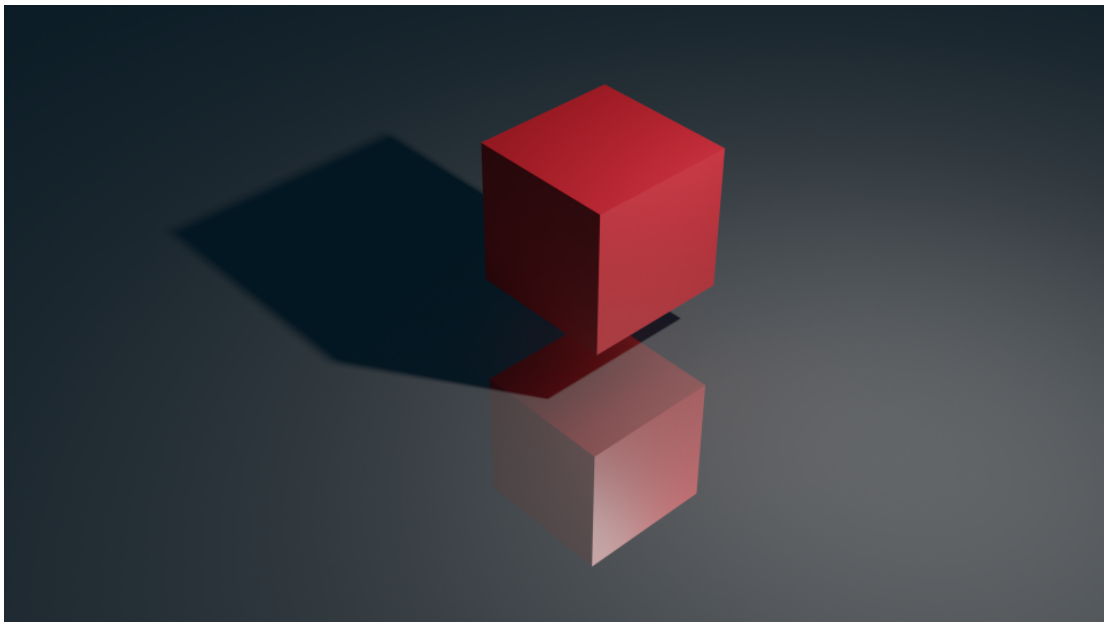
Portallar (Şekil 2.1) çift taraflı, bir taraftan bakınca diğer ucun gerçek zamanlı görülebildiği, içine girildiği takdirde diğer uca anında ışınlayan yapılardır. Bu konsept, çoğu zaman bir bilim kurgu unsuru olarak oyunlarda sıklıkla kullanılır. Bilgisayar grafiklerinde ise sadece iki nokta arasında geçiş için değil, ayna çizimleri veya düzgün diğer yansımalar için de kullanılmaktadır. Çoğu portal çiziminde hem ana kamera açısından çizim yapılıp, hem de kameranın portal uzayına çevrilip bu açıdan çizilerek, bu iki görüntünün doğru bir şekilde birleştirilmesi gerekir. Bu da genellikle 2 geçişli bir grafik boru hattı gerektirir. İlk geçişte portal harici nesneler çizilip portal yerleri işaretlenir, ikinci geçişte ise sadece işaretli noktalara yeni kamera kullanılarak diğer uçtaki sahne çizilir. Geçiş sayısı, ekranda görünen portal geçişleriyle birebir orantılıdır.

Portal ve yansıma çizim tekniklerinin başında resme çizim yapma (Şekil 2.2) vardır. İlk çizim geçişi portal veya yansıma kamerası kullanılarak bir resme yapılır. Bu resim daha sonrasında asıl kamera kullanılarak çizilen sahnede ait olduğu modele doku resmi olarak koyulur (ör. ayna, portal, su birikintisi). Bu yöntem yansımalı nesneler ve geçişli olmayan portallar için kullanılan bir yöntemdir ancak portallar ışınlanmaya elverişli geçişli özellikteyken kullanımı tavsiye edilmez. Bunun nedeni portala geçiş için yakınlaşıldığı takdirde portalın arkasındaki sahnenin, portal resminin çözünürlüğü tarafından kısıtlanmasıdır. Yakın mesafeden alınan çizimde oluşan bu etki, portalın arkasındaki sahnenin, asıl sahneden çok net bir biçimde ayrılmasına sebep olur ve kusursuz geçişler gereken uygulamalarda kullanılmasını engeller.



Şekil 2.1: Geleneksel portal konseptine bir örnek. Kullanıcı bu portallardan herhangi birine girerek diğer odaya ışınlanabilir. Portalın diğer ucu gerçek zamanlı olarak kullanıcıya görünmektedir.

Modern uygulamalarda asıl olarak kullanılan yöntem ise kalıp arabelleklerinin kullanılmasıdır. Asıl sahne kamerası ile çizim yapılırken portal modellerine denk gelen pikseller, kalıp arabelleğinde özel değerler ile işaretlenir. Sahnede görünür olan her portal için ek bir geçiş yapılır, bu geçişlerde portal kamerası kullanılarak yalnızca bu portalın değeri ile işaretlenmiş piksellere çizim yapılır. Böylece herhangi bir resim çözünürlüğü tarafından kısıtlanmayan, her uzaklıkta akıcı geçişleri destekleyen bir portal görüntüsü elde edilir. Bu çalışmada kullanıcıya geçişleri hissettirmemek gerçeklik açısından büyük önem taşıdığı için bu yöntem kullanılacaktır.



Şekil 2.2: Resim üzerine çizime bir örnek. Öncelikle kameranın, parlak düzleme göre simetrisi alınır ve küp bu açıdan düzlem üzerine çizilir, böylece yansıma elde edilmiş olur. Ardından küp normal bir şekidle sahneye çizilir.

Düzgün yüzeyli olmayan portallarda çizim ve geçişler için ek adımlar gerekebilir. Düğüme sahip veya eğik yüzeyleri destekleyen portallar için ayrıca optimizasyonlar mevut olsa da [13–15] bu çalışma bünyesinde yalnızca düzgün yüzeyli portallar kullanılacağından bu tür optimizasyonlara ihtiyaç yoktur.

2.3 Sanal Gerçeklik Optimizasyonları

Sanal gerçeklik deneyiminin performanstan ciddi derecede etkilenmesi, bir takım geliştirme önerilerini ve performans optimizasyonlarını yanında getirir. Facebook tarafından geliştirilen Oculus serisi sanal gerçeklik donanımlarında Asenkron Uzaybükme (Asynchronous Spacewarp) [23] isimli bir teknoloji mevcuttur. Sanal gerçeklik için önerilen alt limit olan saniyede 90 kare değerine, grafik kartı yetersizliği veya uygulamanın ağırlığı sebebiyle ulaşılamadığı takdirde bu teknoloji devreye girer. Saniyedeki kare sayısı 45'e düşürülür ve her iki kare arasına derin öğrenme ile bu iki karenin zamansal ve uzaysal özellikleri kullanılarak sanal bir kare üretilir. Bu performans kayıplarında kullanıcıyı yan etkilerden korumak açısından oldukça verimli bir yöntemdir ancak sanal olarak üretilen kare her zaman istenilen kalitede olmayabilir. Bu sebeple geliştirici hiçbir zaman 90 karenin altına inmemeye gayret göstermelidir.

3. İLGİLİ ÇALIŞMALAR

Sanal gerçekliğin, geleneksel uygulamalar ile karşılaştırıldığındaki en büyük avantajı, sanal ortamla olan etkileşimin çok daha doğal hareketler ile sağlanmasıdır. Fiziksel hareketlerin ve sanal ortamdaki tepkilerin uyumluluğu sonucunda ise kullanıcıyı çok daha içine alan deneyimler ortaya çıkmaktadır. Bu avantajdan en verimli şekilde faydalanabilmek için ise hareket yöntemi olarak doğal yürümenin kullanılması gerektiği geçmişte yapılan çalışmalarda [2] net bir şekilde görülmektedir.

Doğal yürümenin tercih edilmesi için bir diğer sebep ise, beyin tarafından beklenen hareket ile algılanan hareketin birbiriyle uyuşmaması durumunda ortaya çıkan hareket tutmasıdır [1]. Doğal yürüme kullanılarak uyumsuzluk en aza indirilebilmesine rağmen, ne yazık ki her ortam ve uygulama doğal yürüme için uygun değildir. Bu sebeple çoğu sanal gerçeklik uygulaması alternatif yöntemler kullanmak zorunda kalmaktadır [3].

Bu yöntemlerden bazıları tek başına kullanılırken, bazıları ise doğal yürüme veya diğer yöntemlerle birleştirilerek kullanılır. Ek donanım gereksinimler, fiziksel etkileri ve gerçeklik düzeyleri açısından aralarında önemli farklar bulunmaktadır.

3.1 Geleneksel Hareket Yöntemleri

Tahmin edilebileceği gibi, sanal gerçeklikteki en temel hareket yöntemlerinden biri, geleneksel bilgisayar oyunlarından miras kalan klavye, fare veya oyun kolu kullanımıdır. Bu yöntemlerin en büyük avantajı, ek bir tasarım sürecine gerek duymadan, halihazırda mevcut olan uygulamaların, sadece ekrana yansıma biçiminin değiştirilerek sanal gerçeklik platformlarında kullanılabilir hale getirilmesine olanak sağlamalarıdır. Sanal ortam halen aynı hareket mekaniğine sahip olacağından herhangi bir değişikliğe ihtiyaç duyulmaz. Doğal yürüme kullanılmadığı için de fiziksel alan kısıtları herhangi bir sorun teşkil etmemektedir. Bu yöntemde, sanal gerçekliğe dönüşüm için ihtiyaç duyulan tek ek donanım ise başa takılan ekrandır. Tüm bu kolaylıklar geliştiriciler için bu yöntemi, var olan uygulamaların sanal gerçekliğe uyarlanmasında cazip kılmaktadır.

Ancak bilgisayar oyunlarında en çok tercih edilen bu yöntem, tüm bu kolaylıklara rağmen sanal gerçeklik uygulamalarında istenmeyen yan etkilere sebep olmaktadır. Uygulama gereği kişinin ayakta olduğu ve yürüme hareketinin bu yöntemlerle sağlandığı durumlarda, beklenen ve algılanan hareketler birbirleriyle tamamen bir

uyuşmazlık içerisinde olduğundan, kısa süreli kullanımlarda bile ciddi derecede baş dönmesi ve mide bulantısına sebebiyet vermektedir [1]. Uzun süreli kullanıma dahi gerek kalmaksızın, bu yöntemlerden herhangi biriyle sağlanan hareket anında, vücut aynı yönde bir hareket beklediğinden ancak fiziksel olarak buna bir karşılık bulamadığından ötürü anlık denge kaybı da yaşanmaktadır. Bu tür riskler, uygulamanın yalnızca güvenli bir yerde kullanılması zorunluluğunu ortaya çıkarmakla beraber, kullanıcıyı da sık sık ara vermeye mecbur bırakır. Bu sebeple, en temel ve gerçeklenmesi en kolay olan bu yöntemler, sanal gerçeklik uygulamaları tarafından zorunda kalınmadıkça kullanılmamaktadır.

Kullanıcının uygulama içerisinde oturur pozisyonda olduğu durumlarda ise bu yöntemlerin negatif etkileri bir miktar azalmaktadır. Örneğin kullanıcının sürücü koltuğunda oturduğu kara aracı simülasyonlarında, beyni yanıltacak yürüme beklentileri olmadığından baş dönmesi ve mide bulantısı etkisi daha az görülmektedir. Ancak yine de aracın ivmesine karşılık gelen bir fiziksel etki yaratılmadığı takdirde bu etkiler hiçbir zaman sıfıra inmemektedir. Bunun gerçekleştirilebilmesi için ise ek donanıma gerek duyulmaktadır. Bu yöntemler kendilerine ait alt başlıkta ayrıntılı bir biçimde incelenecektir.

3.2 Işınlanma ve Kayarak Yer Değiştirme

Beklenen ve algılanan hareketler arasındaki uyuşmazlığı azaltmanın bir yöntemi, insan beyninin alışık olmadığı ve herhangi bir beklentiye sahip olmadığı sıra dışı sanal hareketler kullanmaktır [4]. Bunun en belirgin örnekleri ışınlanma ve kayarak yer değiştirmedir. Kullanıcı bu yöntemde, genellikle elinde bulunan oyun kolu yardımıyla gideceği yeri işaret ederek, sanal ortamda istediği bölgeye ışınlanır. Maliyet olarak oldukça düşük bir donanım ihtiyacına sahiptir ve gerçeklenmesi oldukça kolaydır. Bu sebeple en sık kullanılan hareket yöntemlerinden biridir.

Fiziksel dünyada herhangi bir karşılığı olmadığından, ışınlanmaya karşı beynin geliştirmiş olduğu herhangi bir fiziksel beklenti bulunmamaktadır. Bu nedenle sanal yürümeye kıyasla, bu yöntem daha az hareket tutmasına sebebiyet vermektedir. Ancak gerçek dünyanın aksine kullanıcılar ortamı devamlı bir şekilde gezmediği için mide bulantısı ve baş dönmesi gibi etkiler yerine yön kaybı gibi sorunlar yaşayabilmektedir [4]. Işınlanma mesafesinin uzun tutulduğu durumlarda beynin iki mekan arasındaki bağlantıyı kurması zorlaşmaktadır. Bu nedenle oyunlarda genelde ışınlanma mesafesi kısa tutulmaktadır.

Ancak bu ışınlanmanın gerçekte bir karşılığı olmaması, bu yöntemin en büyük avantajı olan fiziksel beklenti eksikliğine sebep olurken, bir yandan da uygulamanın gerçekliğini negatif yönde etkilemektedir. Fiziksel dünyada karşılaşılmayan bir fenomen olduğundan, uygulamanın insanı içine çekmesini (immersion) engellemektedir. Işınlanma anında kullanıcı bulunduğu sanal deneyimden koparak durumu yadırgayabilmektedir.

Tüm bu noksanlıklara rağmen, sınırsız bir sanal ortamı kısıtlı bir fiziksel alanda gezmeye olanak sağlayan en basit yöntem olduğu için sanal gerçeklik uygulamalarında en sık kullanılan yöntemdir.

3.3 Özel Donanımlar

Gerçekliğin büyük önem arz ettiği simülasyonlarda ve maliyetin öncelik teşkil etmediği özel uygulamalarda, sınırsız doğal yürümeye olanak sağlayan veya gerçekçi fiziksel tepkiler sağlayabilen, projeye özel donanımlar kullanılabilmektedir. Bunun en yaygın örneği havacılıkta kullanılan uçuş simülatörleridir. Gerçeğe yakın hareket ve ivme tepkilerini sağlayabilen donanımlar yardımıyla gerçek eğitimlerde kullanılır. Bu sayede beyin tarafından algılanan ve beklenen tepkiler arasında tam bir uyum sağlanarak herhangi bir yan etki olmaksızın uygulama kullanılabilir.

Bu donanımların bir diğer örneği ise koşu bantlı askeri ve tıbbi simülasyonlardır [5, 6]. Her ne kadar fiziksel alan kısıtlı olsa da serbest hareketli bir koşu bandına sabitlenen kullanıcı, istediği yönde istediği kadar yol alabilir. Böylece doğal yürümenin karşısındaki en büyük engel olan fiziksel alan kısıtı ortadan kalkmış olur. Böylece savaş alanı gibi büyük mekanlar sanal ortama kolaylıkla sığdırılabilir ve verimli bir şekilde hareket sağlanabilir. Uzun süreli kullanımlarda bile oldukça az yan etki ile simülasyon gerçekleştirilebilir.

Ancak bu yöntemin genel olarak tıbbi ve askeri simülasyonlarda kullanılmasının sebebi, sıradan kullanıcı için alan gereksinimi ve maliyetinin yüksek olmasıdır. Sanal gerçekliğin ulaşılabilirliğinin artmasındaki en önemli sebep maliyetin ciddi derecede düşmesidir. Bu nedenle, maliyeti yüksek fiziksel donanımlar kullanıcıların kişisel alanlarında yer bulmak yerine ancak büyük kurumlar tarafından tercih edilebilmektedir.

3.4 Dikkat Dağıtma ve Yeniden Yönlendirme

Ek donanım ile maliyeti artırmadan bu problemi çözen modern yöntemlerin başında dikkat dağıtarak yeniden yönlendirme gelmektedir. Bu yöntemlerin temelinde, fiziksel ortamdaki dönüş ile sanal ortamdaki dönüş miktarı arasında, kullanıcının yön algısını

bozmayacak kadar az derecede bir fark yaratarak, kullanıcıyı müsait olan fiziksel alana tekrar yönlendirmek yatmaktadır [7]. Çözülmesi gereken asıl problem ise kullanıcının hiç dönmeden yürüyebileceği ve fiziksel alanın dışına çıkabileceği senaryoları ortadan kaldırmaktır. Bunun için farklı yöntemler mevcuttur.

Örnek çalışmada [8], fiziksel alana sığmayacak bir sanal ortam ve bu ortamın karşı uç noktasında ulaşılması gereken bir hedef vardır. Bu hedefe doğrudan yürümek, fiziksel alan kısıtından dolayı mümkün olmayacağından, senaryoya dahil edilen bir dikkat dağıtıcı (ejderha), kullanıcı fiziksel alanın sınırlarına her yaklaştığında ortaya çıkarak kullanıcıyı dönmeye zorlar. Ejderha ortamda bulunduğu sürece hareket mümkün değildir. Ejderhadan kurtulmanın yolu ise, bir süre yine senaryoya dahil edilen su tabancası ile ejderhaya su sıkmaktır. Ejderha sürekli olarak kullanıcının etrafından döndüğü için, kullanıcı da kendi etrafında dönmeye zorlanmış olur. Bu süreçte kullanıcının fiziksel dünyadaki dönüşü ile sanal ortamdaki dönüşü arasında belli bir miktar fark yaratılır. Bu süreç öncesinde, sanal hedefe ulaşmak için fiziksel alanın sınırlarına yönelmek gerekirken, süreç sonunda kullanıcı artık hedefe ulaşmak için fiziksel alanın müsait kısmına doğru yönelmelidir. Bu yöntem, kullanıcı fiziksel alanın sınırlarına her yaklaştığında tekrarlanır ve sanal ortamın tamamı gezilebilir kılınır. Bu yöntemin en büyük dezavantajı, fiziksel dönme ile sanal dönme arasındaki fark hesaba katılmasa dahi, kullanıcının belli aralıklarla kendi etrafında dönmek zorunda kalmasıdır. Bu sanal gerçeklik dışında bile baş dönmesi ve mide bulantısına sebebiyet veren bir harekettir. Kullanıcının uzun süre bu yönteme maruz kalması, olumsuz yan etkilere sebep olmaktadır. İlk nesil sanal gerçeklik cihazlarının (ör. Oculus Rift, Valve Index, HTC Vive) başa takılan ekranlarının genellikle kablolu yapıda olması da bu hareketi kısıtlayıcı etki gösterir. Modern cihazlarda (ör. Oculus Quest 2) bu sorun kısmen çözülmüş olsa da fizyolojik etkileri halen devam etmektedir. Bir diğer dezavantaj ise, her senaryonun böyle bir hedef ve dikkat dağıtıcıya uygun olmamasıdır. Eğitim amaçlı uygulamalarda veya sanal resim sergisi gibi durgun ortamlarda bu yöntemin kullanılması olası değildir.

Dikkat dağıtıcı yöntemlerin en yenilikçi yaklaşımlarından biri ise sıçramalı (saccadic) göz hareketleri sırasında oluşan geçici körlüğün kullanıldığı çalışmadır [9]. İnsan gözünün, iki nokta arasında geçiş yapmak için, yavaş ve sürekli bir hareket yerine anlık bir sıçrama hareketi yaptığı bilinmektedir [10]. Bu sıçrama hareketi sırasında ise yaklaşık 20 milisaniyelik bir geçici körlük yaşanmaktadır. Bu geçici körlüğün kişi tarafından hissedilmemesi için beyin, hareketten önceki ve sonraki görüntünün arasını kendisi doldurmaktadır. Bu çalışmada ise, kullanıcıyı yeniden yönlendirmek için bahsi

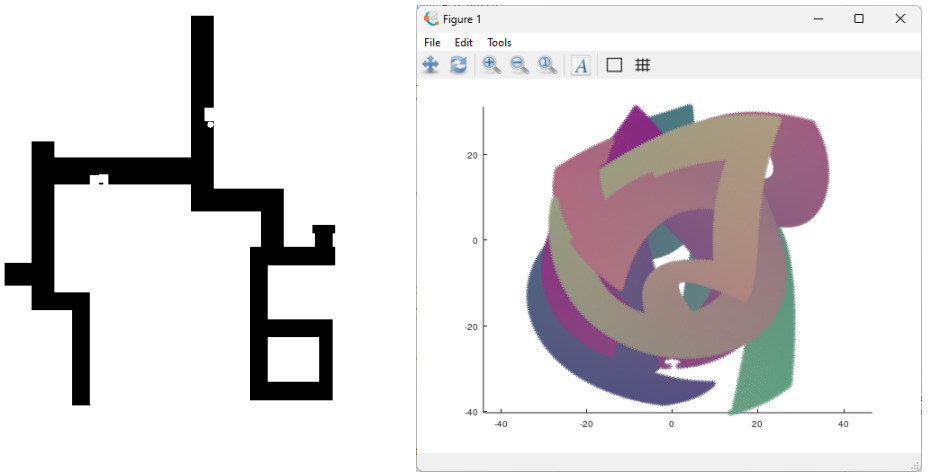
geçen geçici körlük süresi kullanılmaktadır. Dikkat dağıtıcı unsur çalışmasında olduğu gibi, kullanıcı fiziksel alanın sınırlarına yöneldiğinde, nesne uzayında (ör. bir cismin renk değiştirmesi) veya resim uzayında (ör. ekranda ufak bir noktanın parlaması) ortaya çıkan anlık dikkat dağıtıcılar ile kullanıcının bakışı bu yöne kaydırılır. Kullanıcı asıl odak noktasından bu unsurlara bakışını çevirdiğinde ise sıçramalı göz hareketi devreye girer ve kısa süreli körlük oluşur. Tam bu körlük esnasında kullanıcının bakış açısı değişimi ile sanal dünyanın döndürülme oranı arasında küçük bir fark yaratılır, sanal dünya istenilen yönde bir miktar döndürülür. Beynin algılayamayacağı bu küçük fark sayesinde, kullanıcı fiziksel alanın daha müsait olan taraflarına yönlendirilir. Çok kısa süren bu geçici körlüğün değerlendirilebilmesi için yüksek hızlı göz bebeği hareketi tespiti büyük önem taşımaktadır. Bunun için başa geçirilen başlık içerisinde gözü takip eden bir kamera donanımı gereklidir. Yüksek hızlı donanımın yanı sıra, döndürme farkı hesaplama algoritmasının de bu 20 milisaniyelik küçük aralığa sığdırılması gerekmektedir. Bu kısıtlar, sadece hali hazırda mevcut bulunan ana akım donanımlar kullanılarak bu yöntemin uygulanmasını imkansız kılmaktadır ancak yine de diğer özel donanım çözümlerine göre maliyeti ciddi anlamda düşürmektedir. Ayrıca gerçek göz hareketi ile sanal ortamdaki hareketin uyumsuzluğu, doğal bir harekete göre daha fazla baş dönmesi ve denge kaybına sebebiyet verebilmektedir.

3.5 Haritalama ve Bükme Yöntemleri

Ek donanım ve sıfırdan tasarım gerektirmeyen modern yaklaşımların başında haritalama ve bükme yöntemleri gelmektedir [11]. Ana fikir, var olan sanal ortamı belli kurallar dahilinde bükerek, aynı fiziksel alanın birden fazla sanal katman tarafından kullanılmasıdır. Fiziksel alana kıyasla daha büyük olan sanal ortam, fiziksel alanın ölçülerine göre belirlenen bir sertlikte bükülmeye başlanır. Bükülen bu sanal alan katlanarak üst üste binmeye başlar. Bu durum fiziksel alanın aynı noktasının birden fazla kez kullanılarak sanal ortamda farklı mekanlara gidilebilmesini sağlar.

Kullanıcı deneyim testleri için de karşılaştırmalı referans olarak alınan çalışmada [12] bu haritalama ve bükme için çok aşamalı bir yöntem (Şekil 3.1) izlenmektedir. Öncelikle fiziksel alanın ölçüleri belirlenir ve içerisindeki engeller de dahil bir haritası çıkarılır. Ardından ortamdaki gezilebilir bölümlerin haritası çıkarılır. İki harita arasındaki eşleme, Statik İleri Eşleme isimli algoritma kullanılarak çevrimdışı olarak yapılır. Bu yöntemde fiziksel haritanın engel ısı haritasına göre, gezilebilir sanal ortam engellere mümkün olduğunca uzakta kalacak şekilde bir eşleme yapılır. Bu sırada sanal ortam haritasındaki bükülme açıları da en azda tutulmaya çalışılır. Bu süreç sonucunda, sanal ortamın birden fazla noktasının fiziksel alandaki tek bir noktaya

tekabül edebileceği bir eşleme oluşur. Birden fazla sanal noktanın üst üste gelebilmesi hem sanal ortamda dolaşmakta hem de ortamın çizilmesinde sıkıntıya sebep olacaktır. Algoritmanın ikinci aşaması olan Dinamik Ters Eşleme ise bu sıkıntıları çözmek için gerçek zamanlı olarak devreye sokulur. Başlangıç noktası olan fiziksel ve sanal nokta bilinmektedir, bu noktalar kullanılarak her t adımında, t-1 adımındaki sanal konum bilgisi kullanılarak yeni sanal konum hesaplanır. Böylece kullanıcının o anda bulunduğu fiziksel konum birden fazla sanal noktaya tekabül etse bile, yeni konum bir önceki andaki konum bilgisini kullandığı için doğru bir şekilde belirlenir. Kullanıcı bu algoritma yardımıyla, izlediği yola bağlı olarak, aynı fiziksel noktaya varsa bile sanal ortamda farklı bir noktayı gezebilmektedir. Çizim aşamasında üst üste binen noktaların aynı anda ekranda görünmemesi için ise bükme öncesi haritadan alınan derinlik verisi kullanılır. Derinlik verisi yardımıyla, bükülmemiş sanal ortamda daha yakın olan noktalara öncelik verilir ve aynı noktaya denk gelen diğer ortamlar geri planda bırakılır.



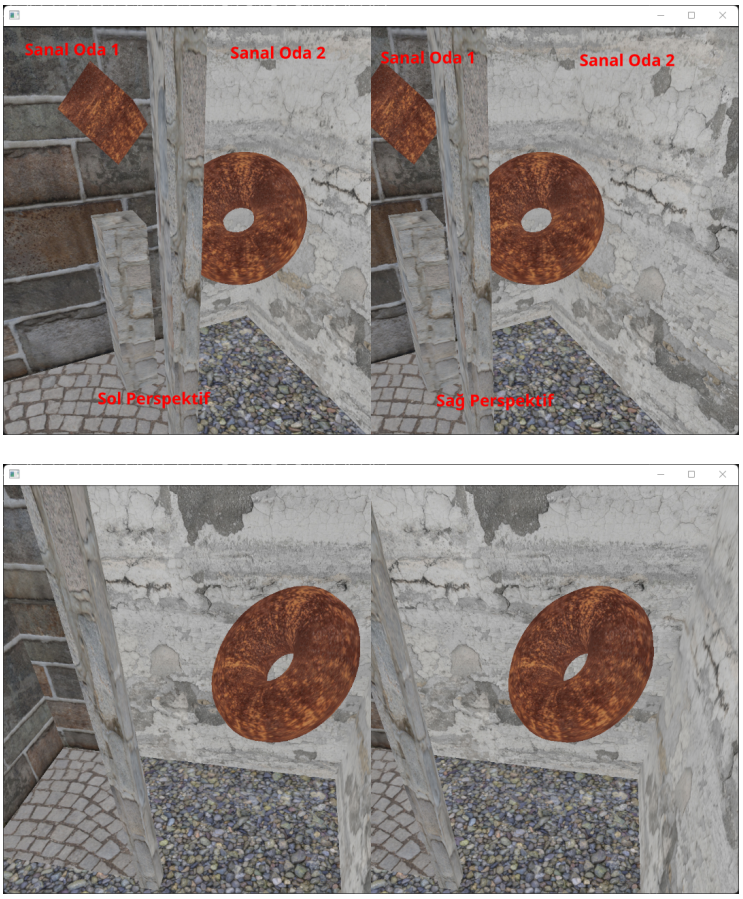
(a) Eşlenecek olan sanal ortam (b) Eşleme sonrası koridorlar

Şekil 3.1: Statik ileri eşleme algoritmasının çalışmasına örnek

Haritalandırma ve bükme yöntemlerinin en büyük avantajı, ek bir donanıma ve ortamın sanal gerçekliğe uygun bir şekilde yeniden tasarımına ihtiyaç duymamasıdır. Ancak hem sanal ortamın hem de fiziksel ortamın önceden haritalandırılmasını ve eşleştirilmesini gerektirmektedir. Eşleme algoritması gereği olarak, tamamıyla açık sanal ortamları da desteklememektedir. Sanal ortamın koridor veya oda bazlı yapıda olması zorunludur. Ayrıca küçük alanlarda ortamın doğrusallığını çok fazla bozduğundan kullanıcılara gerçek dışı ve daraltıcı bir deneyim yaşatabilmektedir. Kullanıcı deneyim testlerinde ayrıntılı bir karşılaştırma yapılacaktır.

4. METOT

Sınırlı fiziksel alanlarda geniş sanal ortamları kullanıcıya herhangi bir yan etkisi olmadan deneyimletebilmek için lineerliğin hiçbir şekilde bozulmadığı ve ışınlanma gibi doğal olmayan yöntemlerin kullanılmadığı bir çözüm sunmaktayız. Farklı odaların portallar aracılığıyla bağlandığı ve aynı fiziksel alanın birden fazla sanal oda için kullanıldığı bu yöntem ortam tasarımı ve grafik motoru gerçeklemesi olarak iki bölümde ele alınacaktır. Örnek bir ortamın, geliştirilen grafik motoru aracılığıyla sunulmasıyla ortaya çıkan sonuç Şekil 4.1'de görülmektedir.



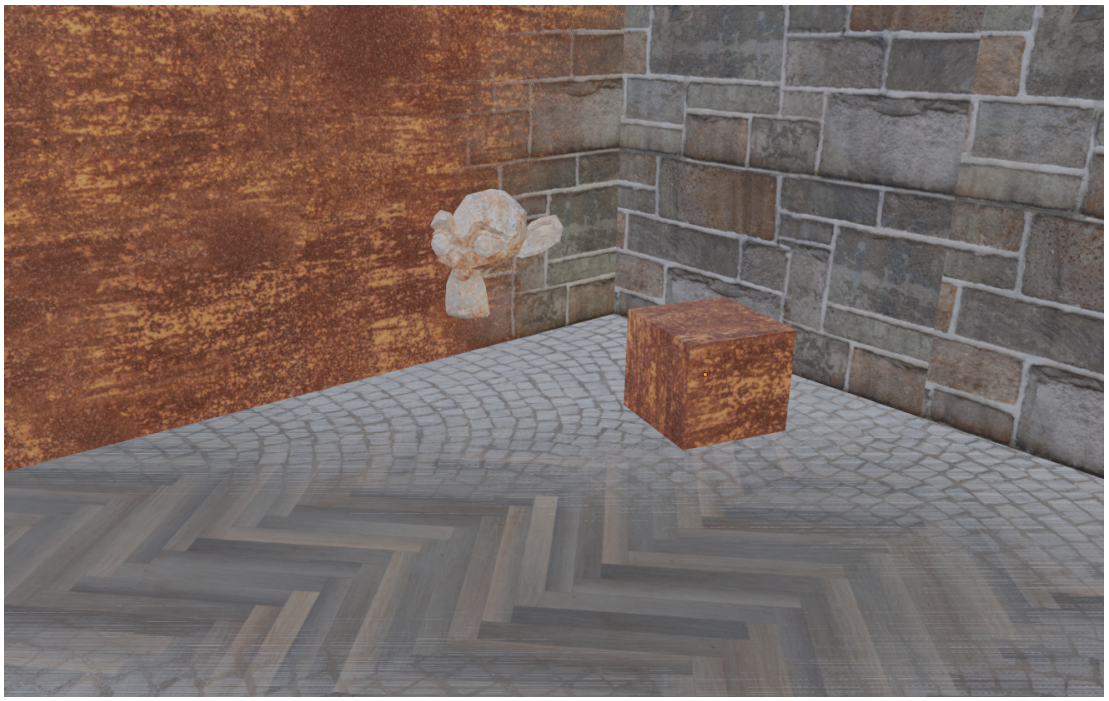
Şekil 4.1: Örnek bir tasarım. İki fotoğrafta bu ortamın farklı açılardan görüntüsü bulunmaktadır. Sanal gerçeklik donanımındaki sağ ve sol perspektifler her iki ekran görüntüsünde de yan yana görülmektedir. Her perspektifte torusun ve küpün bulunduğu iki ayrı oda mevcuttur. Aynı konumdaki aynı alanı kapsayan bu iki oda da portallar aracılığıyla çizildiği için üst üste binme görülmez ve doğal yürüme ile erişilebilirdir.

4.1 Ortam Tasarımı

Bu çalışmanın temelinde yatan fikir, kullanıcıya birbirine portal ile bağlı odalar arasında kusursuz bir geçiş sunmaktır. Üst üste binen sanal odalar kullanılarak, sınırsız alan hedefine ulaşılmaktadır. Bu odaları birbirine bağlayan portallar aracılığıyla serbestçe hareket edilir ve aynı fiziksel ortamda kalınarak farklı sanal odalarda dolaşılır.

Portallar en sık bilim kurgu veya fantazi unsuru olarak kullanılırlar. Bu tür kullanımlarda fiziksel olarak imkansız geçişlerin kullanıcıdan gizlenmesi gayesi güdülmez. Aksine, yaratılan evrenin sıradışılığına vurgu yapmak için kullanılır. Temadan bağımsız olarak oyunlarda hareket kolaylığı amaçlı da kullanılabilirler. Uzak iki mekan arasındaki mesafeyi kısaltarak kullanıcının zaman kaybetmesinin önüne geçerler. Bunun dışında eski grafik uygulamalarında tüm ortamı hafızaya yüklemek performans problemlerine sebep olduğundan hücre tabanlı bir çözüm için de portallar kullanılmaktaydı. Odalar birbirine birleşik ve kapılarla bağlı tasarlanmak yerine her biri birbirinden bağımsız hücreler şeklinde tasarlanır ve birbirlerine portalla bağlanır. Sadece geçiş yapılması mümkün olan komşu odalar hafızaya yüklenir ve portallar aracılığıyla gerçek zamanlı olarak çizilebilir. Başka bir odaya geçiş yapıldıktan sonra artık komşu olmayan odalar hafızadan silinebilir. Aradaki bağlantıları tutmak için ise sonraki bölümlerde bahsedilecek olan hücre-portal çizgeleri kullanılır.

Portal teknolojisi kullanılmadan tasarlanan klasik oda bazlı yöntemlerde, oda geçişleri de doğal yollardan yapıldığı için kusursuz geçişler zaten sağlanabilmektedir. Ancak bu yöntemlerde üst üste binen alanlar kullanılarak sınırsız alan üretmek mümkün değildir. Üst üste binen alanlar klasik yöntemlerde aynı anda çizilmeye çalışılacağından grafik sorunlarına yol açacaktır. Üst üste binen komşu odaların aynı anda çizilmesi, bu noktalarda iki kez çizim yapılması ve odaların birbirine karışması demektir (Şekil 4.2). Sadece şu an içinde bulunulan oda çizildiği takdirdeyse kapıdan diğer oda görünmeyecek ve gerçekliğin bozulmaması adına geçişler için açılır kapanır kapılar gibi yan unsurlar kullanılmak zorunda kalınacaktır. Bu durumda her ne kadar gerçeklik korunsa da, oda geçişlerinde ek bir adıma gerek duyulduğu için uygulamanın akışını ve sürekliliğini olumsuz etkileyecektir. Portal teknolojisi bu durumu, komşu odaların tamamını değil, sadece portaldan veya kapıdan görünen kısmını çizerek çözüyor. Böylece komşu odalardan herhangi biri, şu an içinde bulunduğumuz oda ile üst üste binse dahi, bu kısım çizilmeyecek ve bu oda sadece kapıdan veya portaldan görünecektir. Böylece tasarımda fiziksel imkanların ötesine geçilebilir ve imkansız ortamlar tasarlanabilir.

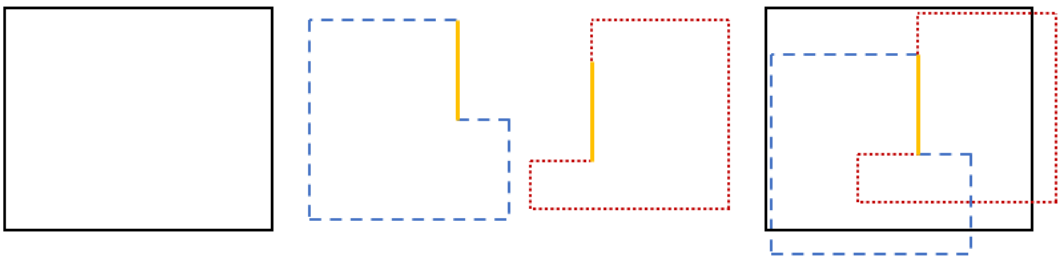


Şekil 4.2: Üst üste binerek grafik problemlerine sebep olmuş iki oda. Komşu odaların yalnızca portallar üzerinden görünebildiği ortamlarda bu tür üst üste binmelerin önüne geçilmiş olur.

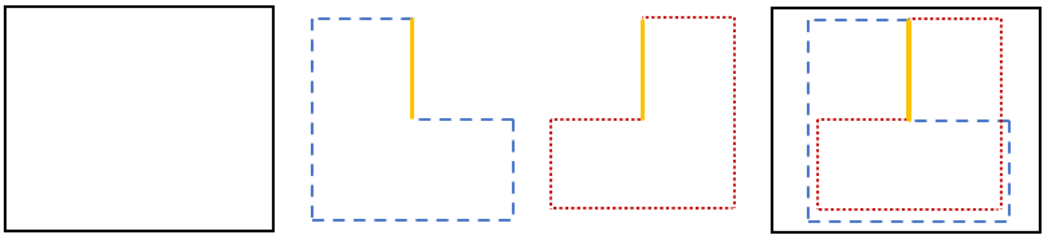
Kullanıcının sınırların dışına çıkmamasını garanti edip aynı zamanda müsait olan fiziksel alandan daha büyük bir sanal ortamda gezebilmesini sağlamak için ortam tasarımı belli kurallar dahilinde yapılmalıdır. Odaların hiçbirinin tek başına boyutu fiziksel alanı aşmamalıdır. Bu durumda kullanıcıyı portal ile müsait alana yönlendirme garanti edilemez ve kullanıcı sanal oda içerisinden hiç çıkmadan fiziksel alanı aşan hareketler yapabilir (Şekil 4.3).

Şekil 4.3: Düz çizgi fiziksel alanı, kesikli çizgi ise sanal ortamı temsil etmektedir. Sanal ortamın boyutu fiziksel alanı geçtiği için bu tasarım uygun değildir.

Sanal odaların her birinin fiziksel alandan küçük olması gereklidir ancak bu da şartları sağlamaya yeterli değildir. Birbirine bağlı iki odanın, portallar pivot alınarak bağlandığında ortak boyutları fiziksel alanı geçmemelidir. Zincir şeklinde birbirine bağlı tüm odalar portallar pivot alınarak birleştirildiğinde oluşan şeklin de fiziksel sınırlar içerisine sığıyor olması gereklidir (Şekil 4.4 ve 4.5).



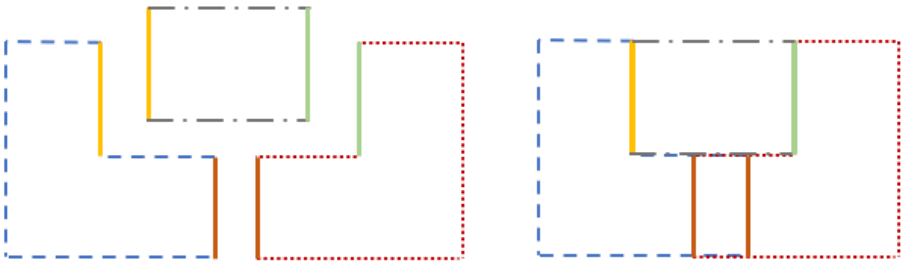
Şekil 4.4: Düz siyah çizgi fiziksel alanı, kesikli çizgiler sanal ortamı, renkli çizgiler ise sanal odaları bağlayan portalları temsil etmektedir. Her sanal odanın boyutu fiziksel alandan küçük olmasına rağmen portallar merkez alınarak birleştirildiklerinde sanal ortam fiziksel alanın sınırlarını aşmaktadır. Bu nedenle bu tasarım da amaca uygun değildir.



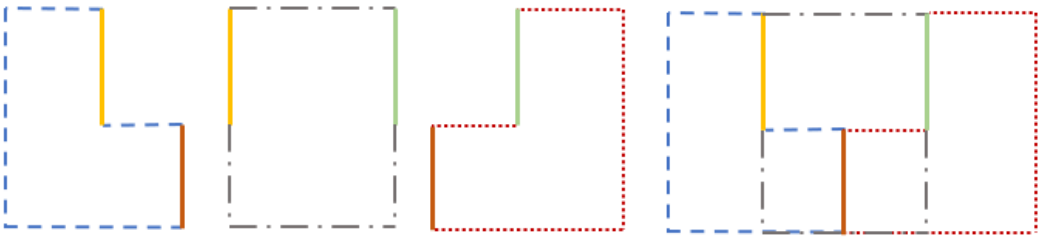
Şekil 4.5: Düz siyah çizgi fiziksel alanı, kesikli çizgiler sanal ortamı, renkli çizgiler ise sanal odaları bağlayan portalları temsil etmektedir. Her sanal odanın boyutu fiziksel alandan küçüktür ve portallar merkez alınarak odalar birleştirildiğinde alanın tamamı halen fiziksel alanın içerisine sığmaktadır. Üst üste binen alanlar da hesaba katıldığında toplam alanın fiziksel alandan çok daha büyük olmasına rağmen bu fiziksel alanda rahatça hareket edilebilmektedir.

Odalar arası döngü olması kabul edilebilir bir durumdur. Kullanıcı aynı odadan birden fazla geçmeyerek başladığı odaya ulaşabilir. Ancak bu durumda sağlanması gereken ek bir tasarım kuralı ortaya çıkar. Odalar sırayla portallar üzerinden birleştirilmeye

başlandığında, döngü tamamlandığı anda döngüyü kapatan portallar da kendiliğinden üst üste binmiş olmalıdır (Şekil 4.7). Bu iki portal arasında bir mesafe bulunması durumunda (Şekil 4.6) kullanıcı bu döngüyü her tamamladığında bu mesafe kadar fiziksel yer değiştirme yapmış olacaktır. Bu da zamanla fiziksel alanın sınırlarının aşılmasına sebep olur.



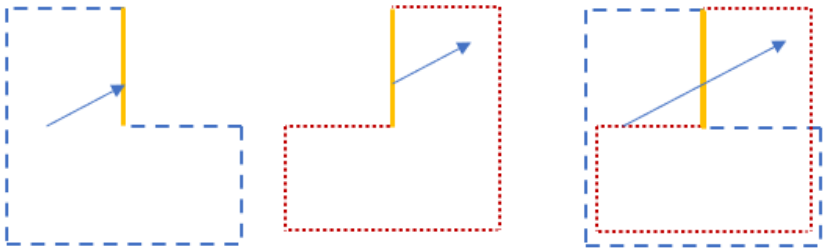
Şekil 4.6: İmkansız bir döngüye örnek. Sarı ve yeşil portallar üzerinden odalar birleştirildiğinde kırmızı portalın iki ucu üst üste binmemektedir. Aradaki boşluk her döngüne kullanıcının fiziksel alanda bir miktar kaymasına sebep olacaktır. Bu nedenle bu tasarım kabul edilemez.



Şekil 4.7: İzin verilen döngüye örnek bir tasarım. Sarı ve yeşil portal çiftleri üst üste bindirilerek odalar birleştirildiğinde kırmızı portalın da iki ucu bir araya gelmektedir. Bu döngü fiziksel alanda herhangi bir kaymaya sebep olmayacağından kabul edilebilir bir tasarımdır.

Bir yan etki olarak bu kural, iki ucu aynı odada olan portalları da engellemektedir. Aynı odaya bağlanan portallar, tek odalık bir döngü oluşması demektir. Bu döngülerde, döngüyü kapatan portalların üst üste gelmesi şartından dolayı aynı odadaki iki portal ancak üst üste binerse bu kural sağlanabilir. Üst üste binmiş bir portal ikilisi, kusursuz geçişler ve çizimler sağlandığından dolayı tamamen etkisiz ve fark edilemez bir unsur olarak var olabilir.

Portal birleştirme işlemlerinde, portalların merkezlerinin çakışması yeterli değildir. Kullanıcının sanal ortamdaki yönünü değiştirmeyecek şekilde birbirlerine paralel ve zıt yönlü durmalıdırlar. Yani bağlı portalların birinin ön yüzü, diğerinin arka yüzüne denk gelecek şekilde birleştirilmelidir, böylece kullanıcı girdiği doğrultu değiştirilmeden diğer odaya geçiş yapabilir (Şekil 4.8).



Şekil 4.8: Portal çiftlerinde istenen giriş ve çıkış yönlerine bir örnek. İki uç birbirine arkası dönük şekilde yerleştirildiğinde, bir uçtan giren kullanıcı, yönü değişmeden diğer uçtan çıkmış olur.

Tüm bu koşullar sağlandığı takdirde istenildiği kadar sanal oda birbirine bağlanıp üst üste bindirilerek sınırsız alan elde edilebilir.

4.2 Oyun Motoru Gerçeklemesi

Ortam tasarım kurallarımız genellikle odaların boyutları ve portalların bulunduğu yerleri kısıtlamaktadır. Portalların sayısı veya hangi odaların ne şekilde bağlanabileceğiyle ilgili durumlar sınırsız alan tasarımını etkilememektedir. Oda bağlantılarının ve portalların bu denli serbest olması da grafik motoru zorluklarını yanında getirmektedir.

Yaygın olarak kullanılan portal uygulamalarının çok büyük çoğunluğu yalnızca birbirine bağlı iki portala izin vermektedir. Işınlanma amaçlı kullanılan bu portallar doku resmine çizme veya kalıp arabelleği yöntemleriyle çok rahat bir şekilde çizilebilmektedir.

Görece daha serbest olan diğer yaygın kullanımda ise yine iki uçlu bir portal çifti bulunurken, bu portal çiftinin özyinelemeli çizimlere sebep olabilecek açılarda yerleştirilmesine de izin verilir. Portalın diğer ucunda aynı portal tekrar görüntülenebilir. Bu yöntemde özyinelemenin ne kadar devam edeceğini bulup çizimi yapmak için basit bir görünürlük testi yeterlidir. Tek bir portal çifti olduğu için karmaşık bir veri yapısına ihtiyaç yoktur.

Bu çalışmada ise aynı odada birden fazla portal tamamen serbest bir şekilde diğer odalara bağlantı sağlayabilmektedir. Portal çiftleri farklı odalarda, ters yönlü ve birbirine paralel olmak zorunda olduğundan direkt özyinelemeye giremezler. Ancak farklı portal çiftlerinin birbirini görmesinin önünde bir engel yoktur ve hatta bu sık karşılaşılacak bir durumdur.

Portal çizimlerinde kullanılacak olan kalıp arabelleği yöntemi, birden fazla geçişli ve sıralı bir algoritmadır. Grafik kartlarının paralel işlemler üzerine özelleşmiş mimarisi sebebiyle, bu bilginin önceden kaydedilerek sıralı bir formatta gönderilmesi gerekir. Veri yapılarını çözmek ve buna bağlı kararlar almak grafik kartının çalışma mantığına aykırı ve performansı kötü etkileyen işlemlerdir. Bağlantılardaki serbestlikten dolayı, hangi portaldan hangi portalların göründüğü ve hangi odaların birbirine bağlı olduğu çözülerek grafik kartının anlayabileceği lineer bir formata dönüştürülmesi ve bu çizim geçişlerinin önceden kaydedilerek grafik kartına gönderilmesi lazımdır. Ayrıca kalıp arabelleğinin çok temel bir yapı olmasından kaynaklı kısıtlar da bu algoritmada daha karmaşık bir veri yapısı seti kullanmayı zorunlu kılmaktadır.

Grafik boru hattı ve kullanılacak olan veri yapılarının ayrıntılarına inmeden önce, bunlara neden ihtiyaç olduğunu daha net görebilmek açısından portalların daha ayrıntılı incelenmesi faydalı olacaktır.

4.2.1 Portal çizim ve ışınlanma tekniği

Portallar genellikle iki uçlu yapılardır ve çift olarak bulunurlar. Bu çalışmadaki tüm portallar da bu tanıma uymaktadır. Herhangi bir ucundan girildiğinde diğer ucundan çıkılır ve çıkılacak olan ucun görüntüsü gerçek zamanlı olarak giriş ucundan görünür. Çiftler arasında bir simetri mevcuttur ve bu nitelikler iki taraflı da geçerlidir.

Bir grafik uygulamasında diğer ucun gerçek zamanlı görülmesinin ve ışınlanmanın mümkün olması için önceden belirlenmiş bir dönüşüm matrisi gereklidir. Kullanıcı portaldan geçtiğinde konum vektörü bu matrisle çarpılarak yeni ortama gönderilir. Ayrıca kamera matrisi de bu matrisle çarpılarak portal içerisindeki sahnenin gerçek zamanlı çizilmesi sağlanır. Eğer portallar bu çalışmadaki gibi paralelse matris çok basit bir şekilde portalın iki ucu arasındaki mesafeden oluşturulan bir çeviri matrisidir ve uygulamanın başında rahat bir şekilde oluşturulabilir.

Portaldan geçişin algılanması ve kullanıcının ışınlanması görece kolay olan işlemdir ve basit bir doğru-düzlem kesişim denklemi çözümü ile gerçekleştirilebilir. Kullanıcının her zaman bir önceki karedeki konumu da tutulur. Geçmiş konum ile şu anki konum arasına çizilen doğru parçası portal düzlemi ile kesişiyorsa kullanıcı iki kare arasında portaldan geçmiştir. Bu durumda kullanıcının konumu, portalın dönüşüm matrisi ile çarpılarak kullanıcı portalın diğer ucundaki mekana aktarılır.

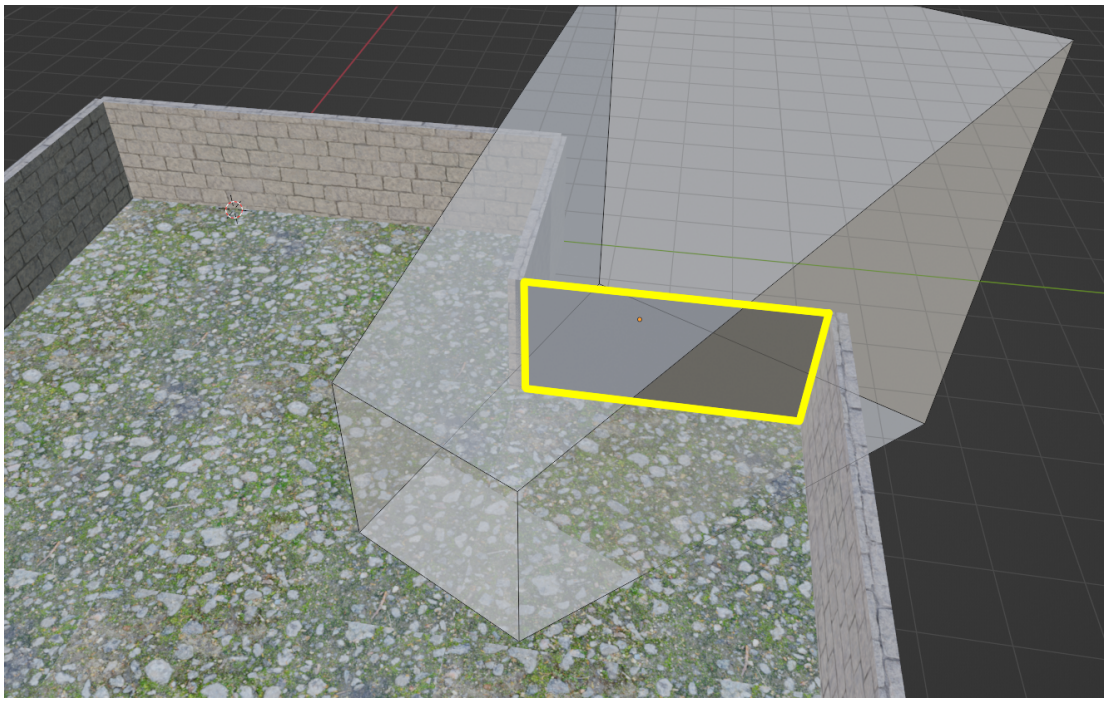
Portalın diğer ucunun gerçek zamanlı çizimi ise grafiksel olarak bizi asıl zorlayan kısımdır. Öncelikle ana kamera matrisi kullanılarak ana sahne çizilir. Ardından her bir portal için, kamera matrisi bu portalın dönüşüm matrisi ile çarpılarak portala tekabül eden yerlere diğer uçtaki sahne çizilir. Portalın göründüğü yerlerin tespit edilmesi ve iç içe portalların nasıl çizileceğinin ayrıntısı sonraki bölümlerde anlatılacaktır.

4.2.2 Eğik frustum

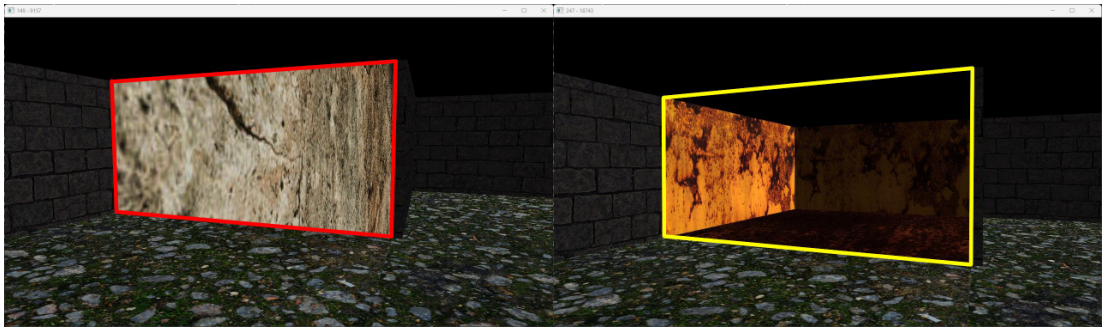
Portal çizimlerinde kalıp arabelleği veya doku resmine çizim tekniklerinden bağımsız olarak, her gerçek zamanlı algoritmada ortak olan konsept, kameranın portala göre taşınmasıdır. Kameranın olduğu konumdan portala doğru bakıldığında portalın içerisinden görünen sahne, portal çiftinin arasındaki mesafe ve rotasyon farkına göre kameranın taşındığı görüntüdür. Kameranın portala uzaklığının ve bakış açısının kaybedilmemesi, gerçek zamanlı çizimlerde portalın arkasının gerçekten de ulaşılabilir ve fiziksel olarak tutarlı bir şekilde çizilmesini sağlar.

Portala göre kamera taşımanın en büyük dezavantajı, portal ile kamera arasına nesneler girebilmesidir (Şekil 4.9). Portallar sadece diğer ucundaki, yani kameraya göre portaldan daha uzaktaki nesneleri göstermelidir. Kamera ile portal arasına giren nesneler ise görüntünün önünü kapatabilmektedir. Bu problem ne yazık ki derinlik arabelleği kullanılarak çözülemez, derinlik arabelleği de sonraki bölümde anlatılacak olan kalıp arabelleği gibi desteklenen işlemlerin basitliği tarafından kısıtlanmaktadır. Belli aralıktaki uzaklığa sahip parçaları derinlik testinden bırakmak gibi bir işlem desteklenmemektedir.

Sıradan perspektif matrisinin yakın ve uzak düzlemleri hesap kolaylığı olması açısından biririne paralel olacak şekilde oluşturulur. Oluşan bu kesik pirmamit şeklindeki yapının içerisinde kalan alan normalize edilmiş cihaz kordinatlarına dönüştürülerek grafik kartının ekranda gösterebileceği bir hale getirilir. Yakın düzlemin daha yakınında ve uzak düzlemin daha uzağında kalan parçalar grafik kartı tarafından atılır.

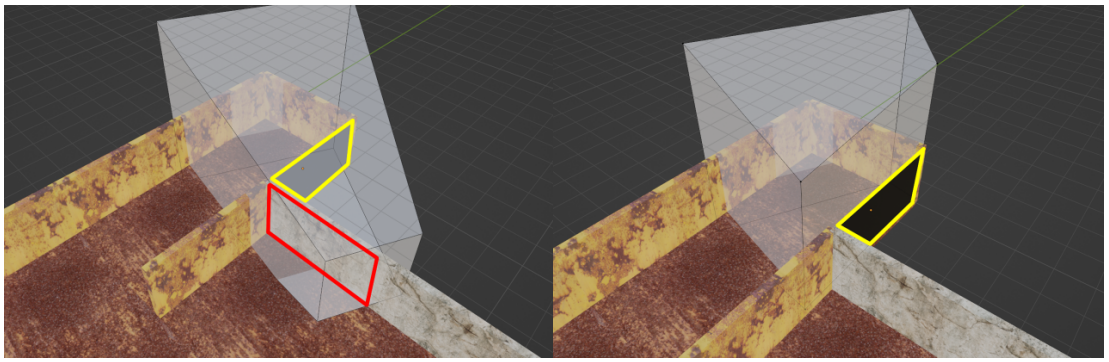


(a) Ana kamera bakış açısı



(b) Düz frustum ile görünen sahne

(c) Eğik frustum ile görünen sahne

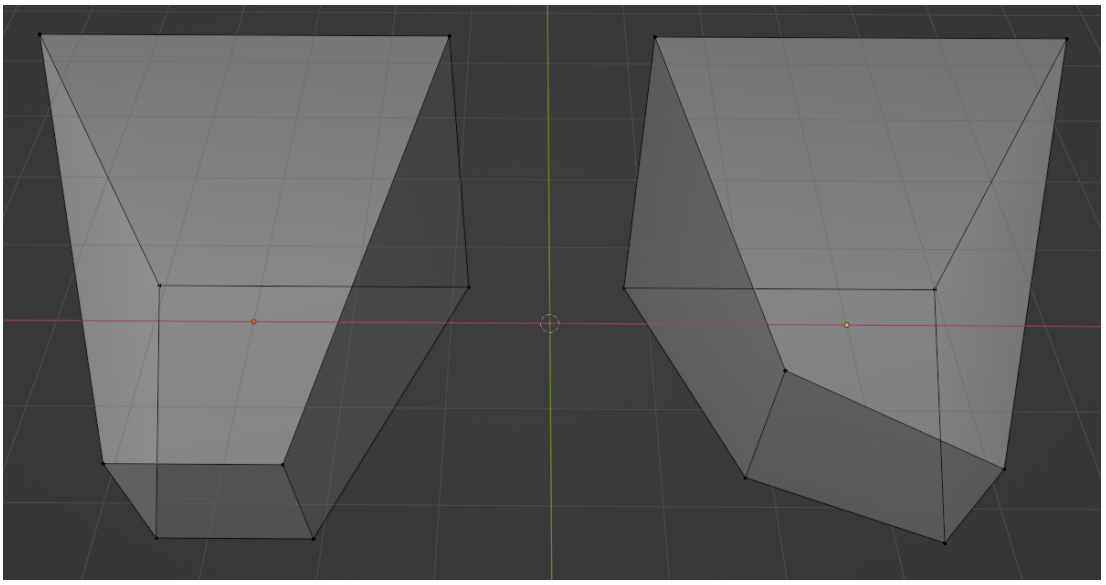


(d) Portala göre taşınmış düz frustum

(e) Portala göre taşınmış eğik frustum

Şekil 4.9: Düz frustumun sebep olduğu sorunlar ve eğik frustum ile çözümü. Şekillerde portallar sarı ile, portalın arkasında kalmasına rağmen görüntüye giren nesneler ise kırmızı ile işaretlenmiştir. Araya nesne girmesini önlemek için frustum yakın düzlemi, portal düzlemi olarak belirlenir.

Grafik boru hattının bu aşamasındaki özellikler kullanılarak bu problem çözülebilir. Yakın düzlem yerine portalın öteki ucunun denk geldiği düzlem koyularak eğik frustum [16] elde edilir. Yakın ve uzak düzlemlerin artık paralel olmadığı bu perspektif matris yapısıyla (Şekil 4.10) gerçekleştirilen uzay dönüşümü sonrasında portalın gerisinde kalan parçacıklar grafik kartı tarafından yok sayılacaktır.



Şekil 4.10: Düz frustum ile eğik frustum örneği. Eğik frustumda yakın düzlem, uzak düzleme paralel olmak zorunda değildir.

4.2.3 Kalıp arabelleği kullanımı

Modern portal çizimlerinde genelde kalıp arabelleği yöntemi seçilir. Grafik kartlarında derinlik arabelleği ile beraber bulunan bu hafıza alanında, çizim geçişlerinde belirli pikseller işaretlenerek sonraki geçişte bu işaretlere göre belirli işlemler yapılabilir.

Portal uygulamalarında öncelikle işaretleme yapılmaksızın ana sahne çizilir. Ardından kalıp arabelleği işaretlemesi açılarak portal çizimleri yapılır, böylece yalnızca portalların görünür olduğu pikseller işaretlenmiş olur. Sonrasında derinlik arabelleği sıfırlanır, böylece portallar üzerine yapılacak çizimlerin derinlik testinden kalması engellenir. Bu aşamadan sonra kamera, portalların çeviri matrisi ile çarpılarak uniform buffer içerisine aktarılır. Yeni kamera matrisi ile yalnızca işaretli olan pikseller üzerine çizim yapıldığında portal üzerinden diğer uçtaki ortamın görüntüsü aktarılmış olur.

Bu yöntem tek portal çiftine sahip ortamlar (Şekil 4.11) için oldukça elverişlidir ancak portal sayıları arttıkça kalıp arabelleği tarafında bazı kısıtlar doğmaktadır. Grafik

boru hattının çok küçük bir kısmı tamamen programlanabilir yapıya sahiptir. Örneğin gölgelendirici aşamasında HLSL ve GLSL gibi yüksek seviyeli programlama dilleri ile karmaşık mantıklar kurulabilse de kalıp arabelleği aşaması malesef grafik boru hattının sabit fonksiyonel kısmına denk gelmektedir. Hem OpenGL'de hem de Vulkan'da kısıtlı bir miktar konfigürasyon dışında grafik boru hattının sabit fonksiyonel kısmında değişiklik yapmak mümkün değildir.

OpenGL'de kalıp arabelleği durumları istenildiği zaman değiştirilebilirken Vulkan'da kalıp arabelleği kısmında yapılabilecek işlemler grafik boru hattı derlemesinden önce ayarlanan belli başlı değerlere bağlıdır. Konfigürasyonun yapılma zamanı değişse de kalıp arabelleği üzerinde değiştirilebilecek değerlerin serbestliği bu iki API'da birebir aynıdır. Derleme sırasında kalıp testi etkinleştirilip arabellek formatına karar verildikten sonra ayarlanabilecek özellikler şu şekildedir:

- compareOp: Testin geçip geçmediğine karar verecek olan karşılaştırma işlemi
- passOp: Testten geçildiği takdirde arabellek üzerinde yapılacak işlem
- failOp: Testten kalındığı takdirde arabellek üzerinde yapılacak işlem
- depthFailOp: Kalıp testinden geçilip derinlik testinden kalındığında yapılacak işlem
- compareMask: Karşılaştırma bit maskesi
- writeMask: Yazma bit maskesi
- reference: Karşılaştırma ve yazmada kullanılacak referans değeri

Belirlenen bu değerler ışığında eğer boru hattında kalıp testi etkinleştirilmişse grafik geçişi sırasında her piksel kalıp testine maruz kalır ve bu piksele tekabül eden kalıp arabelleği değeri güncellenir. Test sırasında öncelikle, bu piksele tekabül eden eski kalıp değerine bakılır. Hem bu değer hem de belirlenen referans değeri, kıyas maskesi ile bit bazında "ve" işleminden geçirilerek son halleri birbirleriyle karşılaştırılır. Karşılaştırma işleminin sonucunu compareOp alanı belirler. Karşılaştırma işlemi şunlar olabilir:

- NEVER: Test, değerlerden bağımsız olarak her zaman başarısız olur.
- LESS: Referans değeri arabellekteki değerden küçükse test başarılı olur.
- EQUAL: Referans değeri ile arabellek değeri aynı ise test başarılı olur.

- LESS_OR_EQUAL: Referans değeri arabellektekinde küçük veya eşit ise test başarılı olur.
- GREATER: Referans değeri arabellek değerinden büyükse test başarılı olur.
- NOT_EQUAL: Referans değeri ile arabellek değeri farklı ise test başarılı olur.
- GREATER_OR_EQUAL: Referans değeri arabellektekinde büyük veya eşit ise test başarılı olur.
- ALWAYS: Değerlere bakılmaksızın test her zaman başarılı olur.

Bu karşılaştırma işleminin sonucuna göre compareOp, failOp veya depthFailOp işlemlerinden biri çalışır. İşlemin alacağı aksiyon da ayarlanabilir kısımdadır ve şu fonksiyonlardan herhangi biri olabilir:

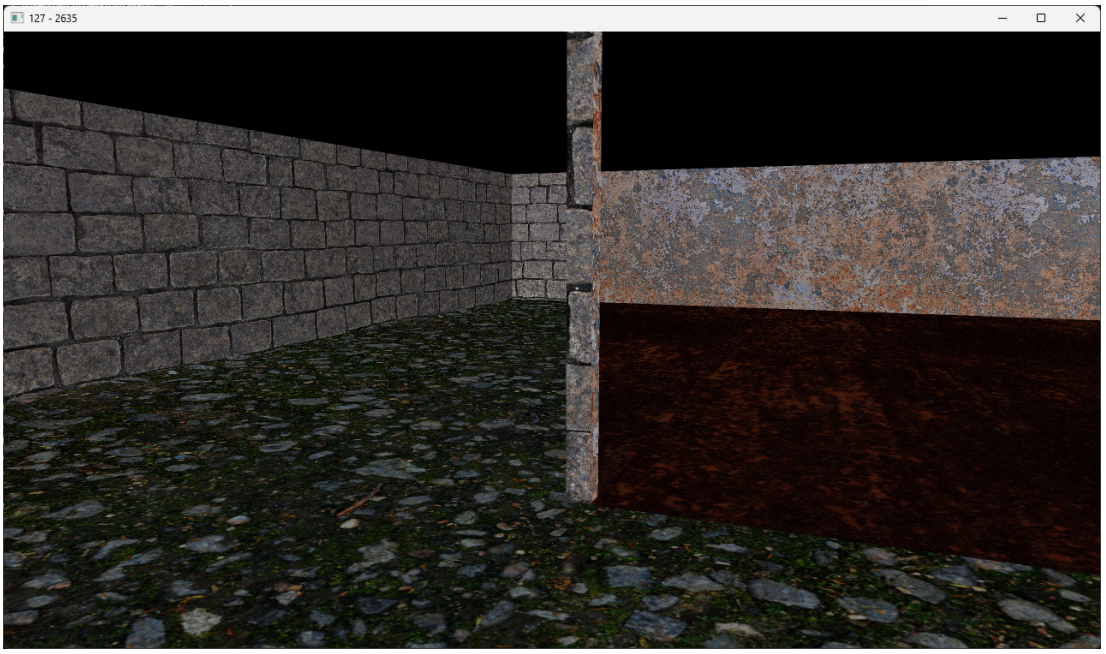
- KEEP: İşlem çalışsa dahi bir değişiklik yapma, arabellekteki değeri koru.
- ZERO: Arabellek değerini sıfır yap.
- REPLACE: Arabellek değerini referans değeri ile güncelle.
- INCREMENT_AND_CLAMP: Arabellekteki değeri bir arttır, eğer zaten arabelleğin tutabildiği en büyük değerde ise değeri olduğu gibi bırak.
- DECREMENT_AND_CLAMP: Arabellekteki değeri bir azalt, eğer zaten arabelleğin tutabildiği en küçük değerde ise değeri olduğu gibi bırak.
- INVERT: Arabellek değerinin bitlerinin tersini al. Sıfırlar bir, birler sıfır olur.
- INCREMENT_AND_WRAP: Arabellekteki değeri bir azalt, eğer zaten arabelleğin tutabildiği en küçük değerde ise değerin tersini alarak en büyük değere sar.
- DECREMENT_AND_WRAP: Arabellekteki değeri bir arttır, eğer zaten arabelleğin tutabildiği en büyük değerde ise değerin tersini alarak en küçük değere sar.

Karşılaştırma sırasında kıyas maskesi ile yapılan bit işlemine benzer bir şekilde arabelleğe yazma sırasında da yalnızca yazma maskesinde bire denk gelen bitlerde değişiklik yapılabilir. Yukarıda belirtilen işlemlerin çoğunda (ters çevirme ve sıfırlama gibi) referans değeri kullanılmaz. Bu durumda yalnızca yazma maskesinin bir olduğu bitlerde işlem yapılır. Referans değerinin kullanıldığı tek işlemde ise (yerine yazma) benzer bir şekilde referans değerinin yalnızca maskelenmiş kısımları arabelleğe yazılır.

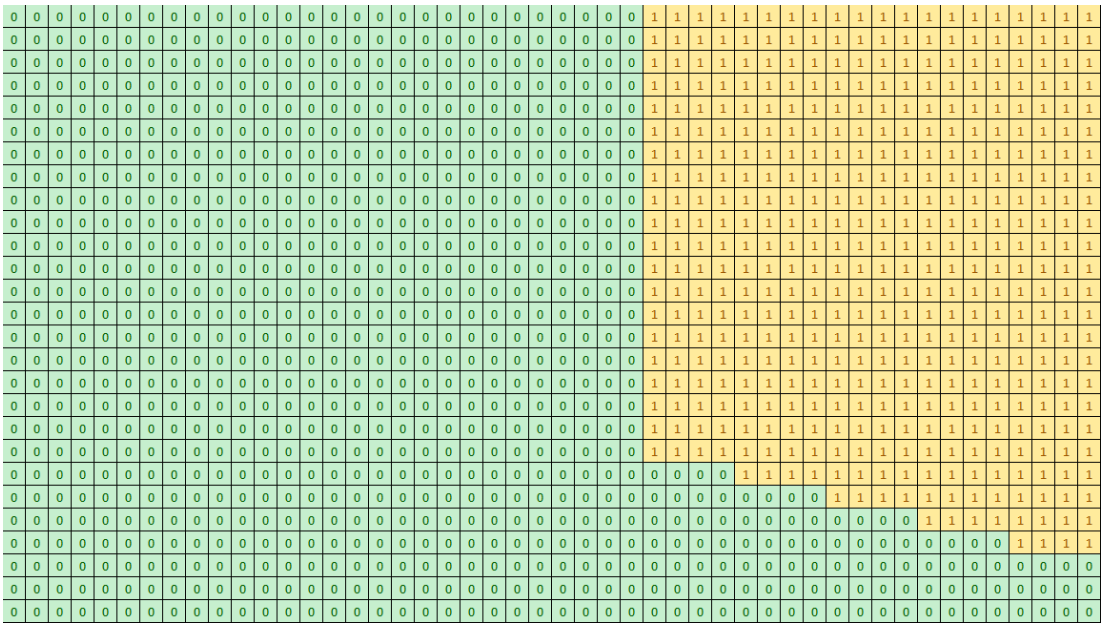
Dikkatli incelendiğinde karşılaştırma ve aksiyon işlemlerindeki basitlikten dolayı ortaya çıkan bir sorun göze çarpacaktır. Karşılaştırma ve yazma işlemleri için ayrı ayrı maskeler olsa da referans değeri tektir. Geçişler sırasında portal içerisine portal çizme işlemi sıklıkla yapılacaktır. Önceki portal tarafından kalıp arabelleğine bırakılmış olan değer yeni portalın değeri ile karşılaştırılarak yeni portalın değeri arabelleğe yazılmalıdır. Ancak yerine yazma için farklı bir değer verilememektedir. Portal çizimleri sırasında karşılaştığımız en büyük engel budur.

Yerine yazma işleminin alternatifleri iç içe portal çizimleri (Şekil 4.12) için uygun değildir. Eğer portal başına birden fazla çizim geçişi göze alınırsa izlenebilecek bir yöntem mevcuttur. Arabellek değerleri sıfırdan değil en yüksek değerden başlatılır. Her portal ilk geçişte denk geldiği piksellerin kalıp değerlerini sıfırlar. Böylece sonraki geçişte eşitlik yerine küçüklük kıyaslaması yapılabilir ve referans değeri kıyaslamadan bağımsız olarak yazma değeri olarak kullanılabilir. Bir sonraki geçiş çizim yapmak yerine yalnızca sıfır değerlerini portalın referans değeriyle değiştirir. Böylece istenilen sonuç elde edilmiş olur ancak ek olarak gelen çizimin belli bir performans götürüsü vardır. Normalde ana sahne hariç her görünür oda için iki geçiş yapılırken (ilki kalıp arabelleğinde portalları işaretleme, ikincisi ise işaretli noktalara bu odanın çizilmesi), bu yöntemde geçiş sayısı üçe çıkmış olur. Performans ise geçiş sayısıyla direkt bağlantılı olduğundan bir kayıp yaşanır.

Portal dizilimlerindeki serbestlikten ve performanstan taviz vermeden oyun motorunu geliştirebilmek için özgün bir yöntem ortaya koyarak iki parçalı kalıp arabelleği kullanımı geliştirdik. Bu yöntemde referans değerinin yanı sıra, kullanımı pratikte kısıtlı olan kıyas ve yazma maskeleri kullanılarak kalıp arabelleği değerleri ikiye ayrılır. Her bir piksel için kalıp değerinin bir yarısı karşılaştırma için kullanılırken diğer yarısı referans değerini yazmak için kullanılır. Portalın kaçıncı katmanda olduğuna bağlı olarak bu yarılar değişir. Bir portal ana sahnede görünüyorsa kalıp değerinin ilk yarısı karşılaştırma için, son yarısı ise referans değerinin yazılması için kullanılır. Bu portaldan görünür olan ilk katman bir portal için bu sıra değişir ki ilk portalın yazdığı referans değeri bu portal için karşılaştırma değeri olabilsin. Referans değeri ise bir yarısı karşılaştırma için dış portalın değeri, diğer yarısı ise yazma için iç portalın değeri olarak belirlenir.

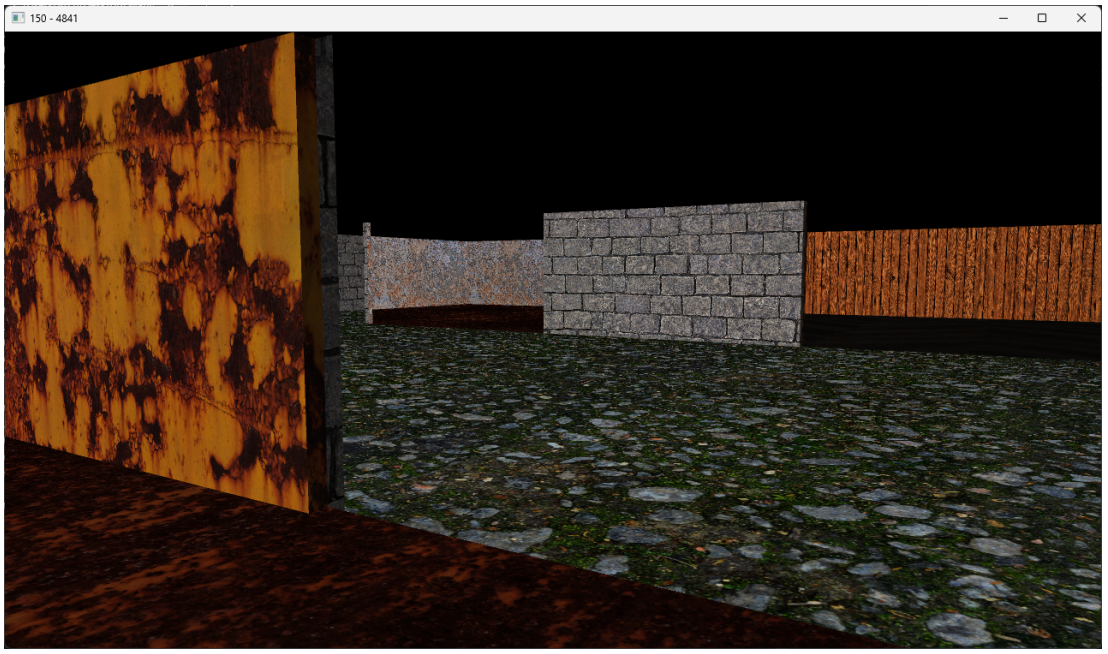


(a) Tek portallı bir sahne

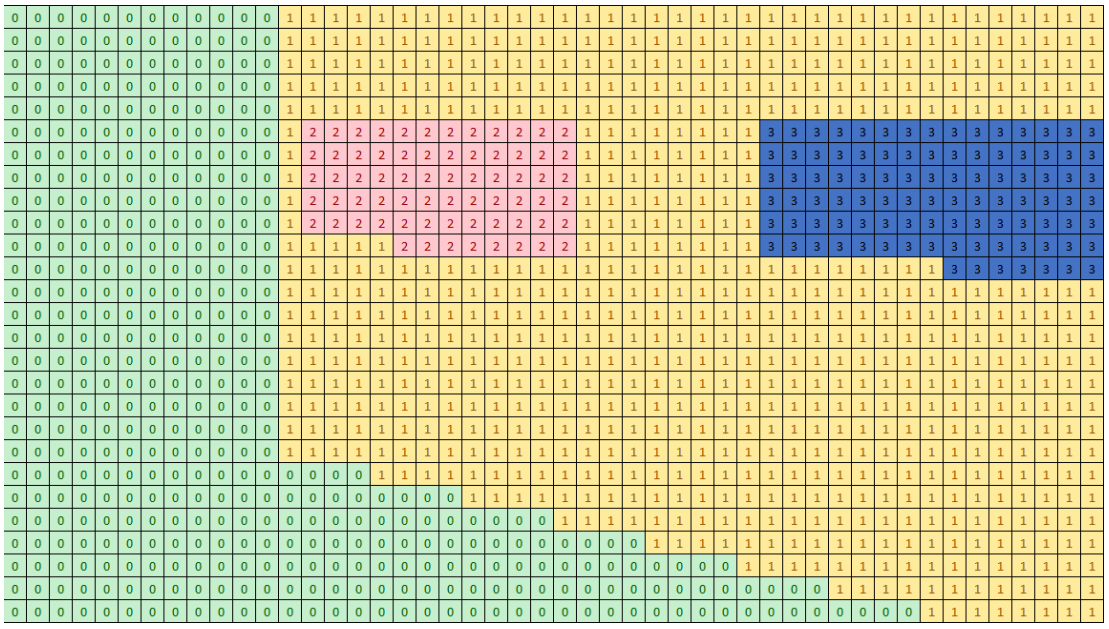


(b) Sahneye tekabül eden kalıp arabelleği

Şekil 4.11: Tek portallı sahnelerde kalıp arabelleği işlemleri yeterlidir. Temiz arabellek ile başlanır ve ana sahne çizilir. Ana sahneye portal çizilirken denk gelen pikseller işaretlenir. Ardından kamera portal çifti arasındaki mesafe kadar kaydırılır ve iç oda yalnızca işaretli piksellere çizilir.



(a) İç içe portallı bir sahne



(b) Sahneye tekabül eden kalıp arabelleği

Şekil 4.12: İç içe portallı bir sahneyi çizmek için basit kalıp arabelleği işlemleri yeterli değildir çünkü "A değeri ile işaretli yere B değeri yaz" işlemi mevcut değildir. Bunun için özel bir yöntem geliştirdik.

Bir portalın (dış portal) içerisinden başka bir portalın (iç portal) görünür olduğu basit bir senaryoda algoritma (Şekil 4.13) şu şekilde işlemektedir:

- Ana sahne portallar haricinde çizilir. Bu sırada kalıp testi kapalıdır ve tüm değerler sıfırdır.

- Ana sahnedeki dış portal çizilir. Bu sırada kalıp testinde yerine yazma işlemi kullanılır, portalın çizildiği pikseller işaretlenir. Kıyas maskesi 0xF0 (11110000) ve yazma maskesi 0x0F (00001111) olarak belirlenir, böylece arabellek değerinin ilk yarısı ile kıyaslanır ve son yarısına yazılır. Kıyaslama ana sahne yani varsayılan değer ile yapılacağından, referans değerinin ilk yarısı 0x0 (0000) olarak belirlenir. Son yarısına ise iç portalın bu senaryodaki referans değeri olan 0x1 (0001) değeri yazılır. Referans değerinin son hali 0x01 (00000001) olmuştur. Belirlenen bu değer ile portal çizildiğinde, portalın çizildiği kısımların kalıp değeri 0x01 olacaktır.

- Dış portalın işaretli olduğu yerlere, portaldan sahnenin odanın çizimi yapılır. Bunun için öncelikle kamera iki portal arasındaki mesafe kadar kaydırılır. İç portal haricinde sahnedeki her şey çizilir. Çizimin sadece portal içerisinde olmasını sağlamak için kalıp testi yapılacaktır. Karşılaştırma, dış portalın yazdığı kısım olan son yarı ile yapılacaktır. Bu sebeple kıyas maskesi 0x0F ve yazma maskesi 0xF0 olarak belirlenir. Referans değerinin ilk yarısı dış portalın değeri olan 0x1 ve son yarısı iç portalın değeri olan 0x2 olacaktır. Ancak sahne çizim aşamasında yerine yazma işlemi aktif değildir, kalıp değerleri olduğu gibi bırakılır.

- Portal içi sahne çizildikten sonra iç portal çizimine geçilir. Bu sefer yerine yazma işlemi aktif edilir ve portalın içinden görünen portal pikselleri işlenir. Önceki aşamadaki değerler olduğu gibi bırakılır, kıyas ve yazma sonrası iç portala denk gelen kalıp değeri 0x21 olmuştur. 0x2 kısmı iç portalın değeri ve 0x1 kısmı önceki portaldan kalan değerdir.

- Ardından tekrar kıyas ve yazma maskeleri değiştirilir ve bu sefer referans değerinin kıyas kısmı 0x2 yapılarak iç portaldan görünen sahne çizilir. Başka portal kalmadığı için yazma kısmı yok sayılabilir.

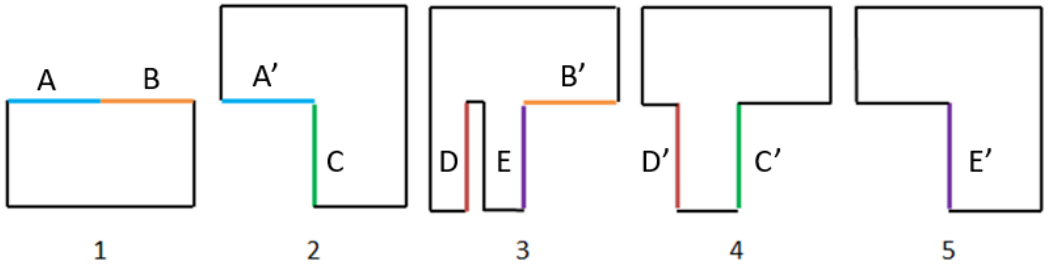
Maskeler ve referans değerleri, bir sonraki bölümde incelenecek olan portal ağacı tarafından belirlenecektir. Bir portalın bu ağaçtaki katmanına göre hangi yarısının karşılaştırma, hangi yarısının yazma kısmı olduğuna karar verilecektir. Algoritmanın tam anlatımı da veri yapıları açıklandıktan sonra yapılacaktır.

Şekil 4.13: Ana sahnede görünen bir portal ve ardından bu portalın içerisinde görünen diğer bir portalın kalıp arabelleği üzerindeki etkileri. Mavi kısımlar kıyas yapılan bölgeleri, turuncu kısımlar ise yazma işlemi yapılan bölgeleri temsil etmektedir. Portalın iç içe görünme katmanının tek veya çift olmasına göre bu bölgeler yer değiştirir.

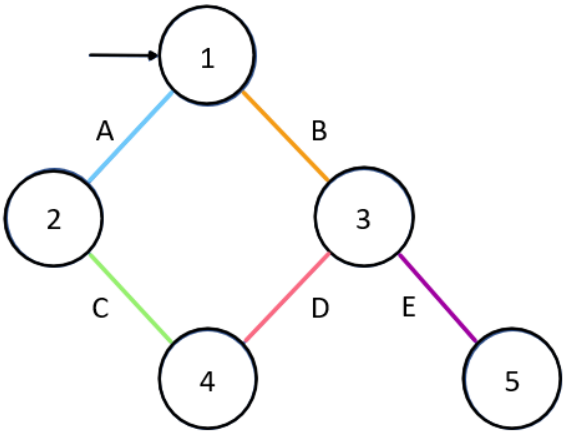
4.2.4 Hücre-portal çizgesi, portal ağacı, portal kuyruğu

Hücre-portal çizgesi (Şekil 4.14) bu tür bağlantıları tutmak için ideal veri yapısıdır. Çizgedeki her bir düğüm bir odayı temsil ederken, aralarındaki bağlantılar da bu iki odayı birbirine bağlayan portal çiftlerini temsil etmektedir. Bu veri yapısı, tasarlanan ortama göre bir kez oluşturulur ve kullanıcı konumu veya kamera açısına göre değişmez. Yapısı gereği ortam tasarımı sonrası çevrimdışı olarak oluşturulup uygulamada kullanılabilir ancak maliyetli bir süreç olmadığından uygulamanın başlangıcında hesaplanarak oluşturulması da herhangi bir sorun teşkil etmez.

Hücre-portal çizgesi, tüm ortamın yapısını statik bir biçimde tutan bir yapı olduğu için çizim anındaki konum ve açı değerlerinin hesaba katılmasını destekleyebilecek bir yapı değildir. Dinamik portal çizimlerinde ise konum ve açı değerlerine göre portalların birbiri içerisinde görünümleri ve buna bağlı olarak odaların çizilme koşulları sürekli değişiklik gösterdiğinden tek başına hücre-portal çizgeleri yeterli olmayacaktır. Grafik boru hattının çizimleri doğru bir şekilde ayarlayabilmesi için çizimlerin kaç geçişli olacağının bilinmesi, kalıp arabelleklerinin değerlerinin belirlenmesi ve tüm bu değerler için portal kamera konumlarının hesaplanması gerekmektedir. Ayrıca grafik kartına gönderilecek kodun çizgeden elde edilebilecek özyinelemelerden de arındırılması gerekmektedir. Tüm bu hesaplamalar için hücre-portal çizgesinden her kare için tekrar türetilecek bir veri yapısına ihtiyaç vardır.



(a) Örnek bir ortam tasarımı



(b) Hücre-Portal Çizgesi

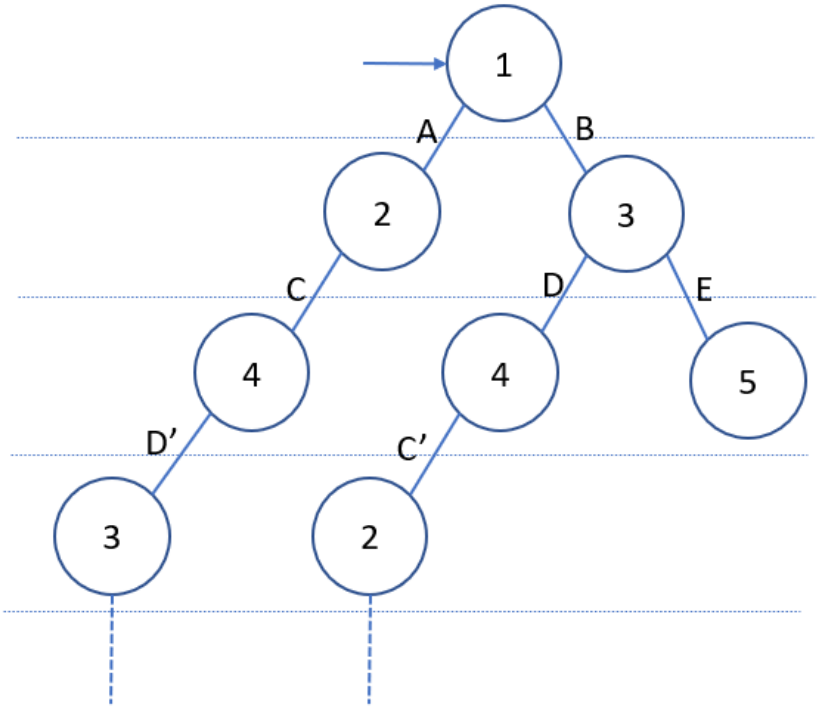
Şekil 4.14: Örnekte verilen ortamadaki oda ve portal bağlantılarına göre üretilmiş bir hücre-portal çizgesi.

Portal ağacı (Şekil 4.15) şeklinde adlandıracağımız bu yapı, her kare için çizimden önce, kullanıcının konumu, bulunduğu oda ve baktığı açıya göre hücre-portal çizgesinden türetilecektir. Portal ağacı, kalıp arabelleği ile portal çizim algoritmasında ayrıntılı olarak açıklanmış sebeplerden ötürü, hücre-portal çizgesindeki belirsizlikleri tamamen ortadan kaldırarak grafik kartının anlayabileceği sonlu ve deterministik bir şekle getirmeyi hedefler. Bunun için öncelikle döngülerden arındırılmalı ve sınırlı bir boyuta indirgenmelidir. Ancak odalar arası döngüler desteklenen bir unsur olduğundan, yok edilmesi değil anlaşılır bir biçime getirilmesi gerekmektedir. Ayrıca bir odanın ağaçtaki varlığına ek olarak bu odanın hangi portal içerisinden görüldüğü de kameranın bu geçiş için ayarlanmasında etkilidir. Bu iki bilgiyi de saklayabilmek için portal ağacına odaların farklı yerlere birden fazla kez eklenmesine izin verilir. Bu hem döngülerde ağaç yapısının korunarak deterministik bir veri üretilmesini yardımcı olur hem de bir odanın her denk gelinişi için ağaçta bu odaya ulaşan portal yolunu da tutarak kamera ayarlarının düzgün bir biçimde yapılmasını sağlar.

Her portal ikilisi, iki ucu arasındaki görsel ve uzamsal geçişi nasıl yapacağının verisini tutmaktadır. Bu veri, kullanıcı kamerası ile çarpıldığında, elde edilen yeni kamera matrisi, portalın diğer ucuna bir bakış sağlayacaktır. Portallar iç içe geçmiş ve birbiri içerisinden görünebilir olduğu için bu veri uç uca eklenebilecek bir matris formatında tutulur. Kullanıcı kamerası, önce A portalının, sonra da B portalının çeviri matrisiyle çarpıldığında kullanıcı B portalı içerisinden A portalının bağlı olduğu odayı görecektir. Portal ağacı oluşturulması aşamasına, hücre-portal çizgesinde kullanıcının bulunduğu odaya denk gelen düğümden başlanır ve bu düğüm portal ağacının kökü olarak belirlenir. Döngülerin önüne geçebilmek için bu kök düğümden komşu düğümlere doğru yayılan bir BFS algoritması koşturulur. Her oda portal ağacına, göründüğü odanın çocuğu olarak eklenir.

Portal ağacının oluşturulma aşaması iki koşuldan birine bağlı olarak son bulur. Eğer hücre-portal çizgesinde herhangi bir döngü yoksa, hücre-portal çizgesindeki tüm düğümler ağaca eklendiğinde algoritma durdurulur. Döngü olmaması durumunda hücre-portal çizgesi zaten bir ağaç niteliği taşır, ortaya çıkan portal ağacı ise bu ağacın yalnızca farklı bir düğüm kök seçilerek yeniden oluşturulmuş hali olur. Hücre-portal çizgesinin döngüler içermesi durumunda ise, BFS algoritması ufak bir değişiklik ile çalıştırılır. Normalde bu algoritmada üzerinden geçilen bir düğüm tekrar kuyruğa eklenmezken, portal ağacında aynı düğümün birden fazla kez bulunması desteklendiği için, zaten üzerinden geçilmiş olma koşuluna bakılmaksızın, kaynak düğüm dışında kalan tüm komşu düğümler kuyruğa eklenir. Bu durumda BFS algoritması döngü içeren çizgelerde hiçbir zaman durmayacağından, portal ağacının oluşturulma aşaması tüm düğümler ağaca eklendiğinde değil, önceden belirlenen bir boyuta ulaşıldığında son bulur.

Portal ağacının limit boyutu, kalıp arabelleğinin desteklediği bit sayısına göre belirlenir. Kalıp arabellekleri genelde her piksel için 8 bitlik bir alan ayırır. Normalde 256 değer alabilen bu alan, kalıp arabelleğini ikiye bölüp katmanlarda ayrı ayrı kullanmamızdan ötürü 4 bitmiş gibi varsayılır ve 16 değerle kısıtlıdır. 0 da bu değerlere dahil olduğundan, portal ağacımızın alabileceği en büyük değer 15'tir. Yani iç içe veya birbirinden bağımsız toplamda en fazla 15 oda/portal çizilebilir.



Şekil 4.15: Kullanıcının konumuna ve açısına göre her çizimden önce çıkarılacak olan portal ağacına bir örnek. Kök, kullanıcının şu an bulunduğu odadır, portallardan görünen odalar alt düğüm olarak eklenir. Direkt olmayan döngüler deterministik bir şekilde açılır ve farklı düğümler olarak ağaca eklenir.

Portal ağacı oluşturulduğunda, elimizde kullanıcının bulunduğu odadan başlayarak tüm odaların hangi portal aracılığıyla hangi odadan göründüğü bilgisi net bir şekilde belirmiş olur. Portal kameralarının hesaplanması için gerekli alt-üst düğüm bilgisi elimizde bulunsa da bu hesaplar henüz yapılmamıştır ve grafik kartı çizim geçişleri için bizden halen bir işlem kuyruğu beklemektedir. *Portal kuyruğu* (Şekil 4.16) bu verileri grafik kartından direk olarak kullanılabilecek şekilde sağlamak için portal ağacından türetilecektir.

Portal kuyruğu, ağacı tam olarak çizim geçişlerinde kullanılacak sıraya koymalı ve kamera matrislerini direkt olarak uniform buffer ile uyumlu formata çevirmelidir. Portal çizimi için kalıp arabelleğinde kullanılacak olan özgün değerler de bu kuyruğa eklenme sırasına bağlı olarak belirlenecektir. Portal kuyruğu oluşturma işlemi, portal ağacı oluşturma işlemine benzer bir şekilde, bir önceki veri yapısına BFS uygulanarak yapılmaktadır. Portal ağacının kökünden başlanarak BFS uygulanır ve algoritmanın geçtiği sırayla düğümler portal kuyruğuna eklenir.

Portal kuyruğuna ekleme sırasında her düğümün kamera matrisi, üst düğümün kamera matrisiyle çarpılarak kaydedilir. Üst düğüm zaten özyinelemeli olarak kendi üstündeki düğümler ile çarpılmış olacağından odaların portallar içerisinde görüneceği pozisyonlar ağaçta aşağı doğru aktarılmış olur. Kök düğüm kullanıcının mevcut olarak bulunduğu odayı gösterdiğinden hesaplamaların temelini bu düğümde bulunan kullanıcı kamerası vardır. Aynı odayı belirten düğümlerin kamera matrisleri, bu düğüme ulaşılan yola bağlı olarak birbirinden farklı olabilir. Çünkü bu son üretilen veri, yalnızca portalın iki ucu arasındaki geçiş değil, bu odaya bağlanana kadar içerisinden geçilen tüm portal ikililerin çeviri matrislerinin kullanıcı kamerası ile çarpımıdır. Böylece aynı odanın birden fazla portal aracılığıyla farklı pozisyonlardan görünmesi sağlanabilir.

Portal kuyruğunun tuttuğu diğer bir veri ise çizim sırasında kalıp arabelleğinde portallara tekabül eden pikselleri belirten özgün değerlerdir. Kalıp arabelleğinin temiz değeri sıfırdır ve bu değer ana sahne için ayrılmıştır. Ana sahneden başlanarak çizim geçişleri yapıldığında portallara portal kuyruğundaki sırasına göre bu değerler atanır. Odalarda olduğu gibi aynı portalın farklı görünümleri kuyrukta birden fazla defa geçebilir ve farklı değerler alabilir.

Tüm bu verilerin yanı sıra kalıp arabelleği bölümünde ele alınan kısıtlardan dolayı bu düğümün ağaçta hangi seviyeye denk geldiği de tutulmalıdır. Bu değer tek ise kalıp arabelleğinin ilk yarısı, çift ise kalıp arabelleğinin son yarısı kullanılacaktır. Ağacın kökünü oluşturan ana sahne düğümü hariç tüm düğümler kendisini bir üst katmandaki odanın düğümüne bağlayan portal bağlantısı aracılığıyla görünür. Yani her düğüm kendi çizimini, kalıp arabelleğinde yalnızca üst düğümünün referans değerinin yazılı olduğu kısma yapabilecektir çünkü bu düğümün göründüğü aralık bu portalın çizildiği kısımdır. Bu nedenle üst düğüm değeri ile karşılaştırma gerekirken, ayrıca kendi içerisinde portallar var ise bunların değerlerini de güncellemesi gerekmektedir. Her katman, kendinden önceki katmanın yazdığı değer ile karşılaştırma yaparak, kalıp arabelleğinin diğer yarısını da kendi değerini yazmak için kullanacaktır. Bu nedenle katmanın çift veya tek olması bilgisi de kuyruk düğümlerine eklenir.

Portal kuyruğu oluşturulması, çizimden önceki son aşamadır. Bundan sonra portal kuyruğundaki veriler kullanılarak seçilen API aracılığıyla grafik kartına çizim geçişleri gönderilir.

Şekil 4.16: Portal ağacından türetilen portal kuyruğu. Kırmızı sayılar kuyruktaki sıraya göre belirlenen kalıp arabelleği referans değerleridir. Siyah numaralı düğümler ağaçta çift katmanlara, yeşil numaralı düğümler ise tek katmanlara tekabül eder. Kök düğüm yalnızca kullanıcının kamera matrisini (M_{PC}) tutarken diğer düğümler ağaçtaki alt-üstlük ilişkisine göre üst düğümlerinin kamera matrisiyle çarpılarak tutulur.

4.2.5 Çok çekirdekli Vulkan grafik motoru

Algoritma geliştirilip gerekli veri yapıları oluşturulduktan sonra geriye sadece çizimleri yapmak kalır. Modern CPU ve GPU mimarilerinin performans potansiyellerinden yararlanabilmek için grafik API olarak Vulkan seçilmiştir. Motorun birden fazla parçaya bölünüp paralel olarak koşturulduğu bu mimaride en büyük zorluk senkronizasyondur.

Vulkan geliştiriciye iki farklı senkronizasyon nesnesi sunmaktadır. Bunlardan ilki semafor (semaphore) yapısıdır. Vulkan semaforları birden fazla grafik komutu arasındaki senkronizasyonu sağlamak için kullanılır. Senkronizasyon gerektiren komutlar grafik kartına gönderilirken parametre olarak iki tane de semafor istenir. Bu semaforlardan biri bekleme semaforu (waitSemaphore), diğeri ise sinyal semaforudur (signalSemaphore). Bu komut grafik kartına gönderildiğinde başlaması için bekleme semaforuna başka bir komut tarafından sinyal verilmesi beklenir. Bu sinyal geldiğinde gönderilen komut çalıştırılır. Komut tamamlandığında ise kendi sinyal semaforuna sinyal verir ve böylece bu semaforu bekleyen başka bir işlem varsa o aktif edilir.

Vulkan'ın geliştiriciye sunduğu bir diğer senkronizasyon nesnesi ise çitlerdir (fence). Bu yapı GPU-CPU arası senkronizasyondan sorumludur. GPU'da koşturulabilen belli başlı komutlara opsiyonel olarak bir çit parametresi gönderilebilmektedir. Semaforun aksine tek başına bu parametrenin herhangi bir işlevi yoktur. Bu senkronizasyon nesnesini kullanabilmek için yine işlemci tarafında vkWaitFence (çiti bekle) ve vkResetFence (çiti sıfırla) fonksiyonları kullanılmalıdır. Bekleme fonksiyonu CPU'daki işlemi, GPU'da çitin parametre olarak gönderildiği işlem tamamlanana kadar bloklayacaktır. Böylece ortak kullanılan bir kaynak varsa işlemci tarafından buna dokunulmasının önüne geçilir. İşlem bittikten sonra çit tekrar kullanılacaksa çit sıfırlama fonksiyonu kullanılarak tekrar hazır hale getirilir.

Bu yapıların yanı sıra grafik motorumuz birden fazla iş parçacığı içerdiğinden, işletim sistemi tarafından sağlanan klasik senkronizasyon nesneleri de kullanılacaktır. Grafik motoru için dil tercihimiz, performans ve uyumluluk sebebiyle C++ olduğundan, standartlardaki ikili semafor (binary_semaphore), sayılı semafor (counting_semaphore) ve muteks/karşılıklı dışlama (mutex) yapıları tercih edilecektir ve CPU-CPU senkronizasyonu bu üç nesne ile sağlanacaktır.

Grafik motorumuzda sürekli çalışan üç farklı tür iş parçacığı yazılmıştır. Öncelikle veri yapılarını oluşturmayı, çizim komutlarını kaydetmeyi ve bunları sürekli en güncel halde tutmayı sağlayan kayıt iş parçacıkları (record threads), ardından bu iş parçacıkları tarafından kaydedilen komutları grafik kartına gönderip ekrana sunmayı sağlayan çizim iş parçacıkları (render threads) ve son olarak paylaşılan kaynaklardan, hareketlerden ve veri yapılarını üretmekten sorumlu ana iş parçacığı (main thread). Çizim ve kayıt iş parçacıklarının sayısı, Vulkan takas zincirinin (swapchain) resim sayısı ve boru hattındaki aktif resim sayısına göre belirlenecektir.

Takas zinciri, Vulkan'ın çizimleri ekrana sunma sisteminin temelidir. Belirli bir sayıda resim içeren ve pencere yüzeyiyle direkt iletişimde olan bu sistem, ekrana gönderilecek resimlerin sırasını belirler ve bu sıraya göre geliştiriciye çizim yapılacak bir sonraki resmi verir. Dört farklı sunum modunu desteklemektedir:

- Anlık (Immediate): Takas zincirine gönderilen resim direkt olarak ekrana basılır. Resimlerde kırılma olabilir.
- İlk giren ilk çıkar (FIFO): Takas zinciri birden fazla resim içeriri ve bunlar kuyruk mantığı ile sunuma gönderilir. Çizim arka plandaki bir resme yapılır ve çizim tamamlandığında öne atılır. OpenGL'in çift arabellekleme (double buffering) mantığına benzerdir. Resimlerde kırılma olmaz.
- Rahatlatılmış ilk giren ilk çıkar (FIFO relaxed): Önceki moddan tek farklı, eğer arka plandaki çizim tamamlanmamışsa bile ön plana atılabilir. Resimlerde kırılma olabilir.
- Posta kutusu (Mailbox): İkinci moddaki kuyruk yapısına benzerdir ancak resim takasları yalnızca ekran ile kuyruk arasında değil, kuyruk içerisinde de olur. Kuyruğa yeni resim eklendiğinde sırada bekleyen resim ile yer değiştirilir, böylece hep en güncel ve tamamlanmış resim ekrana sunulur. Üçlü arabellekleme (triple buffering) için kullanılabilir.

Masaüstü uygulamalarında genellikle 3 resimli posta kutusu modu kullanılırken, Oculus SDK de ilk giren ilk çıkar modeline çok benzeyen 2 resimli bir takas zinciri sunmaktadır. Çizimin nerede yapıldığına bağlı olarak 2 veya 3 resim idealdir. Buna takas zinciri resim sayısı diyeceğiz ve kayıt iş parçacıklarının sayısı belirleyen değerdir. Her takas zinciri resmi için bir tane kayıt iş parçacığı çalışır çünkü kayıt sırasında çizimin hangi takas zinciri resmine çizileceği bilgisi gereklidir.

Takas zinciri resimlerine ek olarak, boru hattı farklı aşamalarda aynı anda çalışabilen bir yapı olduğundan karar verilen bir diğer değer de boru hattındaki aktif resim sayısıdır. Takas zincirinden bağımsız olarak genellikle boru hattına 2 resim alınır. Bir resim tamamlanırken diğer resim çizilmeye başlanır ve böylece boru hattı boş beklemez. Çizim iş parçacıkları sayısı da bu değer tarafından belirlenmektedir.

Ana iş parçacığı her zaman bir tanedir. Fare/klavyeden veya sanal gerçeklik sensörlerinden gelen hareketleri işler. Portaldan geçiş olup olmadığına bakıp hangi konumda ne yöne bakıldığını belirler. Her hareket sonrası portal ağacı ve portal kuyruğunu oluşturur. Veri yapılarına yazma işlemi sırasında bir muteks ile kilit alır, tamamlandığında kilidi geri sisteme döndürür.

Kayıt iş parçacıkları, portal kuyruğundaki Vulkan komutlarını komut arabelleğine (command buffer) kaydeder. Ana sahneden başlanır, portallar dışında her şey çizilir ve kalıp testi açılarak portallar işaretlenir. Portalların işaretlediği alanlara, kuyrukta tekabül eden düğümler çizilir ve kuyruğun sonuna gelene kadar bu işlemler tekrarlanır.

Kalıp arabelleğindeki değer değişimler, dinamik boru hattı kullanılarak yapılır. Normalde Vulkan boru hatlarının önceden derlenmesini ve gerektikçe diğer boru hatlarına geçiş yapılmasını şart koşarken, son yıllardaki gelişmeler sonucu dinamik boru hattı değişkenleri desteği getirilmiştir, referans değerler gibi basit verilerin değiştirilebilmesi sağlanmıştır. Testler sonucu önceden derlenmiş boru hatları ile dinamik boru hattı arasındaki performans farkının yok sayılabilecek kadar düşük olduğu görülmüş ve dinamik boru hattı seçilmiştir.

Her kayıt iş parçacığı başına 3 komut arabelleği tutulur. Biri grafik kartında işlemde olabilen arabellektir, diğeri şu an bu iş parçacığı tarafından kayıtta olabilen arabellektir. Sonuncusu ise eğer daha güncel bir portal kuyruğu oluştuysa, hazır kayıtlı arabellekten daha güncelini oluşturmak için kayda başlanan arabellektir. Eğer kayıtlı olan bellek GPU'ya gönderilmeden önce bu kayıt tamamlanırsa, ilki geçersiz kılınır.

Takas zincirinin resimleri hangi sırayla vereceğinin, Vulkan spesifikasyonlarına göre garantisi yoktur. Çizim iş parçacıkları, sorgu sonrası Vulkan'ın verdiği, sıradaki takas zinciri resim indisini alarak, bu indise tekabül eden daha önce kayıt iş parçacığı tarafından oluşturulmuş en güncel kayıt arabelleğini alarak komut kuyruğuna yollar. Komut kuyruğu GPU'daki işlemleri sıraya dizip koşturan yapıdır. Komut kuyruğuna gönderilen komut arabelleği çalıştırılmayı bekler ve eğer bir semafor tarafından bloklanmadıysa sırası geldiği anda çalıştırılır. Komut arabelleği çalışıp bittiği anda artık takas zincirindeki resim hazırdır ve yine çizim iş parçacığı tarafından sunuma için işletim sistemine gönderilir. Paralel çalışan iş parçacıkları tarafından performans maksimumda tutulurken gecikme de en azda tutulur.

5. DENEYLER

Literatüre geçmiş çalışmalara bakıldığında ek donanım gerektirmeyen modern çalışmaların genellikle kullanıcıyı yeniden yönlendirmek için bir dikkat dağıtıcı kullanarak döndürdüğü veya sanal ortamı bükerek fiziksel ortama eşlediği görünmektedir. Bu iki yöntem de kısıtlı alan problemini çözmektedir ancak geleneksel hareket yöntemleri kadar olmasa da kullanıcının üzerinde fiziksel etkileri olacaktır.

Bu çalışmada lineerliğin hiç bozulmamasından ve hareketlerin tamamen tutarlı olmasından ötürü doğal yürümeye kıyasla herhangi ek bir fiziksel rahatsızlık olmayacağını düşünmekteyiz. Hem grafik motorunun pratikteki etkilerini gözlemleyebilmek için hem de haritalama yöntemleri ile kıyaslayabilmek için karşılaştırmalı bir kullanıcı deneyi ihtiyacı oluşmuştur.

Kullanıcı deneylerine ek olarak Vulkan tabanlı ve çok çekirdekli sistemleri destekleyen modern grafik motoruna ek olarak OpenGL ile de benzer bir grafik motoru geliştirilmiş ve bu ikisinin performans karşılaştırması yapılarak sonuçlar sunulmuştur.

5.1 Kullanıcı Deneyi

TOBB Ekonomi ve Teknoloji Üniversitesi İnsan Araştırmaları Değerlendirme Kurulu'nun 10.06.2022 tarihli ve 2022-26 numaralı izni ile "Mapping Virtual and Physical Reality" [12] isimli çalışma referans seçilerek karşılaştırmalı bir deney düzeni oluşturulmuş ve 12 gönüllü ile deney gerçekleştirilmiştir.

5.1.1 Referans çalışma

Referans çalışmamız, ilgili çalışmalar kısmında da anlatıldığı gibi, sanal ortamı fiziksel ortama eşleme çalışmasıdır. Fiziksel alan ile sanal ortamın haritalarını ve boyutlarını alarak, öncelikle Statik İleri Eşleme isimli algoritmayı çalıştır. Bu işlemin her fiziksel ve sanal ortam çifti için bir kez yapılması yeterlidir.

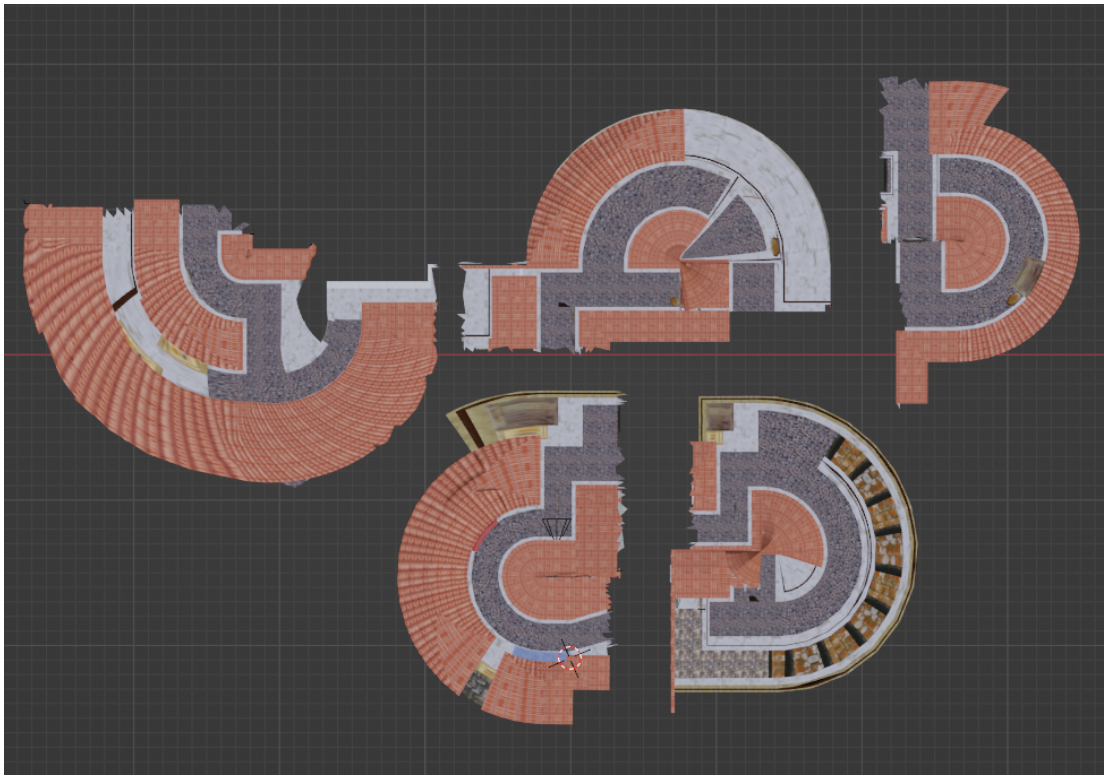
Sanal ortam genellikle fiziksel ortamdan daha büyük olacağından, birden fazla sanal ortamın aynı fiziksel ortama denk gelmesi muhtemeldir. Uygulamadaki konumu tutabilmek ve grafiksel olarak üst üste binmeleri engelleyebilmek Dinamik Ters

Eşleme algoritması kullanılır. Bu algoritma kullanıcının bir önceki konumu ve güncel hareketini kullanarak yeni konumu hesaplar. Sanal ortamdaki başlangıç noktası önceden belirlendiği için ve her karedeki konum bir önceki kareye göre hesaplandığı için, kullanıcının fizikseldeki konumu birden fazla sanal ortama denk geldiğinde bile herhangi bir belirsizlik ortaya çıkmaz ve kullanıcı sanal ortamda doğru konuma yerleştirilir.

Statik ileri eşleme algoritması paylaşılmışken dinamik ters eşleme ve çizim programı tarafımızla patent kısıtları sebebiyle paylaşılamamıştır. Ancak çalışma incelendiğinde yöntem farklı olsa dahi portal tabanlı çalışmamız kullanılarak referans çalışmanın dinamik ters eşleme kısmının benzetilebileceği görülmüştür. Statik ileri eşleme sonrası ortaya çıkan harita ışığında ortam Blender programı yardımıyla aynı şekilde bükülerek üst üste binmelerin önüne geçecek parçalara ayrılmıştır (Şekil 5.2). Bu parçalar birer oda olarak düşünülüp birbirlerine portallar aracılığıyla bağlandığında dinamik ters eşleme kullanılmadan da üst üste binmelerin önüne geçilerek referans çalışmadaki ile birebir aynı harita (Şekil 5.1) sunulabilir. Kullanıcı deneylerinde statik ileri eşleme ile üretilip portal tabanlı grafik motoru kullanılarak gösterilen olan bu sahne kullanılacaktır.

Şekil 5.1: Referans çalışmada kullanılan ve kendi oyun motorumuza adapte edilmiş olan eşlenmiş Italy haritasından bir kesit.

(a) Orijinal



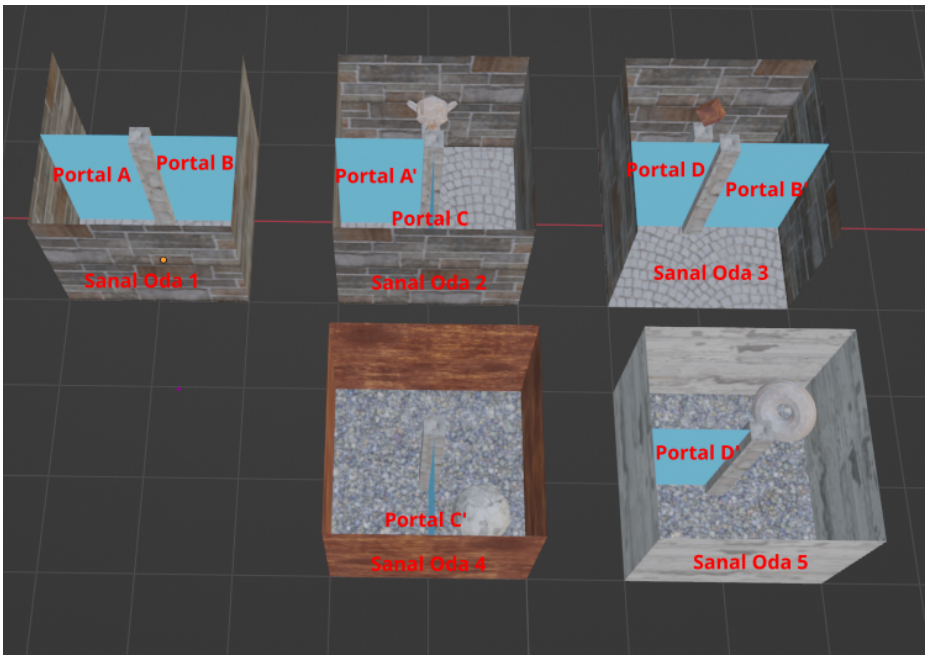
(b) Eşlenmiş

Şekil 5.2: Orijinal ve eşlenmiş Italy haritası. Eşleme, statik ileri eşleme algoritması kullanılarak yapılmıştır. Daha sonra üst üste binmelerin önüne geçmek için parçalanarak portallarla bağlanmıştır.

5.1.2 Deney düzeni

Kullanıcı testlerinin yapılması için Oculus Rift sanal gerçeklik donanımı seçilmiştir. En yaygın donanımlardan biri olan Oculus Rift, ayrıca kablolu olmasından ötürü alan ve mesafe kısıtlarının da gerçek hayatta rahatlıkla görüldüğü bir donanımdır.

Referans çalışmada kullanılan sahne, kendi çalışmalarında kullandıkları asıl sahne olan Counter Strike: Italy sahnesidir. Blender ile statik ileri eşleme ışığında bölünerek portal tabanlı motora uyarlanmıştır. Kendi çalışmamızda kullandığımız sahne ise ayrıntılı olarak anlattığımız tasarım kurallarına uygun olarak oluşturduğumuz özgün bir sahnedir (Şekil 5.3).



Şekil 5.3: Özgün olarak tasarladığımız portal tabanlı sahne. Portal çiftleri ile birbirlerine bağlı ve aynı fiziksel alanı kullanan toplam 5 odadan oluşmaktadır.

Testler yaklaşık 3 metreye 3 metrelik kare bir alanda gerçekleştirilmiştir. Kullanıcılara öncelikle 5 dakika kadar referans çalışma denetilmiştir. Ardından kısa bir ara verdirilerek yine 5 dakika kadar portal tabanlı çalışma denetilmiştir.

Test bitiminde kullanıcılara aşağıdaki sorular sorularak fikirleri alınmıştır:

- Daha önce hiç sanal gerçeklik deneyiminiz oldu mu?
- Eğer olduysa hangi donanım ile ne kadar tecrübeniz oldu?

- Az önce denediğiniz testler sizde herhangi bir rahatsızlığa sebep oldu mu?
- Testlerden hangisi size daha rahat ve doğal hissettirdi?
- Hangi test size daha çok o ortamın içindeymişsiniz gibi hissettirdi?
- Testlerin hangisinde yolunuzu bulmanız ve daha önce geçtiğiniz yerleri hatırlamanız daha kolay oldu?
- Bu grafik motorlarından herhangi birini tamamlanmış bir sanal gerçeklik oyununa çevirmek isteseniz bu hangisi olurdu?

5.1.3 Deney sonuçları

Farklı yaş aralıkları ve sanal gerçeklik tecrübelerine sahip toplam 12 kişi (Çizelge 5.1) gönüllü olarak bu teste katılım sağlamıştır.

Çizelge 5.1: Yaşlarına ve sanal gerçeklik deneyimlerine göre kullanıcı sayıları

	Hiç	**2 saatten az**	**2-8 saat arası**	**8 saatten fazla**
18-40 yaş	3	1	4	2
40-60 yaş	0	2	0	0

Sanal gerçeklik tecrübeleri:
Kullanıcılara bakıldığında 3 kişinin daha önce hiç sanal gerçeklik kullanmadığı görünüyor. 2 kişi düzenli kullanıcı iken geri kalanlar kullanıcılar en az bir kez kısa süreliğini sanal gerçeklik kullanmış kişiler.

Düzenli kullanıcıların ikisi de Oculus Rift ve Oculus Quest 2 kullanmışken, bunlardan biri ek olarak Oculus Rift Dev kit ile deneyimi olduğunu da belirtmiştir. Az miktard sanal gerçeklik deneyimi olan kullanıcıların tamamı yalnızca Oculus Rift kullandığını belirtmiştir. Bu da deney için seçilen donanımın yaygınlığını desteklemektedir.

Rahatsızlık Hissi:
Testlerden herhangi birinin rahatsızlığa sebep olup olmadığıyla ilgili olan soruya kullanıcıların çok büyük çoğunluğu hayır cevabı vermiştir. Bir kullanıcı anlık performans kaybından dolayı yalpalama yaşadığını belirtirken bu uzun süreli bir yan etkiye sebep olmamıştır. 3 kullanıcı testlerde hafif seviyede baş dönmesi yaşadığını belirtmiştir, bunlardan 1 tanesi yalnızca referans çalışma özelindedir. Düzenli

kullanıcılardan biri ise 10 dakikalık bu testin yan etki görmeye yetecek kadar uzun olmadığını belirtmiştir.

Kullanıcıların ikisi hariç tamamı portal tabanlı çalışmada daha rahat ve doğal hissettiklerini belirtti. Bir kullanıcı ikisinde de rahat hissederken, son kullanıcı referans çalışmada gökyüzünü izlemenin sürreal bir deneyim olduğunu belirtti ancak koridorların dar olmasından rahatsızlık duyduğunu belirtti.

İçindelik Hissi:
Kullanıcılardan biri referans çalışmanın daha çok içinde (immersive) hissettirdiğini belirtirken iki kullanıcı arada fark olmadığını belirtti. Kullanıcıların geri kalanı ise bu konuda portal tabanlı çalışmayı tercih etti.

Yol Bulma:
Üç kullanıcı referans çalışmada yollarını bulmanın ve geçtikleri yerleri hatırlamanın daha kolay olduğunu belirtti. Bir kullanıcı arada fark olmadığını belirtti. Düzenli kullanıcılardan biri portal tabanlı çalışmada daha kolay yön bulduğunu belirtti ancak bunun ortam tasarımının daha sade olmasından kaynaklı olabileceğini belirtti. Geri kalan kullanıcılar portal tabanlı tasarımı tercih etti.

Oyuna Çevrilme Tercihi:
Hangi grafik motorunun tam bir oyuna çevrilmesini tercih ettiklerini sorduğumuzda iki kişi herhangi bir tercihte bulunmadı. Bir kişi eşleme tabanlıyı seçerken kullanıcıların geri kalanı ise portal tabanlıyı seçti. Portal tabanlıyı seçenlerden biri eşlemedeki koridor tabanlı tasarıma göre daha ferah ve alanın daha kullanışlı olduğunu belirtti. Bir diğer kullanıcı ise portal tabanlı çalışmanın yaratıcı değerinin daha çok olduğunu belirtti.

5.2 Performans Analizi

Performans testleri için Intel i7 8750H CPU and Nvidia GTX 1050 Ti GPU'ya sahip bir bilgisayar kullanılmıştır. Ortalama düzeyde ve popüler olan bu donanım kombinasyonu sıradan bir kullanıcı sistemini yansıtmak için seçilmiştir.

2-3 görünür portalın bulunduğu ortalama bir durumda OpenGL tabanlı tek çekirdek destekli grafik motoru saniyede yaklaşık olarak 1600 kare üretebilmektedir. Bu kare başına yaklaşık 0.625 milisaniye harcanması demektir. Vulkan tabanlı çoklu çekirdek destekli grafik motoru ise saniyede yaklaşık 2000 kare üretebilmektedir. Bu da kare başına yaklaşık 0.5 milisaniye harcanmasına tekabül eder.

OpenGL motorunun saniyede ürettiği minimum ve maksimum kare sayıları da hesaba katıldığında, bu değerlerin saniyede yaklaşık 1500 ile 1700 arası değiştiği görülmektedir. Vulkan motorunda ise bu değerler 1900 ile 2400 arası değişiklik göstermektedir. Her ne kadar Vulkan motoru tutarlı bir şekilde OpenGL motorundan daha çok kare üretebilse de, OpenGL motorunun daha stabil çalıştığı görünmektedir. Vulkan motoru çoklu çekirdek desteğinden ve senkronizasyon bağlılıklarından dolayı daha dengesiz bir kare üretim sayısına sahiptir.

Sonuç olarak Vulkan tabanlı grafik motoru bize OpenGL tabanlı grafik motoruna kıyasla %20 ile %40 arası bir artış sağlamaktadır. Her ne kadar iki grafik motoru da maksimum büyüklükteki portal ağacına sahip test sahnelerimizde bile 1200 FPS sağlayabilmekteyken, performans daha karmaşık sahnelerde kolaylıkla istenilen değer olan 90 FPS altına düşebilir. Bu durumlarda Vulkan tabanlı grafik motorunun sağladığı performans artışı daha da öne çıkacaktır. Ancak nadiren de olsa daha stabil bir sayıda kare üretiminin tercih edileceği durumlar ortaya çıkabilir.

6. DEĞERLENDİRME

Deney sonucunda kullanıcılardan gelen yorumlara bakıldığında bu çalışmanın doğal yürümeyi kullanması ve evrenin lineerliğini bozmaması sebebiyle diğer referans çalışmaya kıyasla mide bulantısı ve baş dönmesi gibi etkilere daha az sebebiyet verdiği görünmektedir. Referans çalışmanın da bu konularda çok başarılı olduğu göz önüne alınırsa portal tabanlı sınırsız alan uygulamamızın büyük bir potansiyeli olduğunu görünmektedir.

Fiziksel yan etkilerin azlığına ek olarak bu çalışmanın yaratıcılık yönünün de başarılı bulunmuş olması, ileride özgün çalışmalarda kullanılabileceğini göstermektedir. Bazı kullanıcılar tarafından sıradışı olarak nitelendirilen bu çalışmanın ayrıca dar ve tuhaf değil ferah ve algılanabilir ortamlar yaratabilmesi, amacımız olan sınırsız alan problemini özgün ve kullanıcıyı rahatsız etmeyen bir yöntemle çözebildiğimizi göstermektedir.

Sadece yaratıcı uygulamalarda değil, resim ve heykel gezileri gibi gerçekliğin önemli olduğu ve dikkat dağıtıcıların koyulamayacağı durağan uygulamalarda kullanılabileceği ve kullanıcının herhangi bir yan etkiyle karşılaşmadan uzun süreler geçirebileceği ortamlar tasarlanabileceği görülmektedir.

Performans değerlerine bakıldığı takdirde eski usül mimarilere göre büyük ölçüce performans kazancı elde edildiği de net bir şekilde görülmektedir. Ayrıca bu çalışma kapsamında geliştirilen veri yapısı ve portal algoritması sadece sanal gerçeklik ve sınırsız alan uygulamalarıyla kısıtlı kalmayıp her türlü portal uygulamasına serbestçe uygulanabilir. Bu konuda çalışmamızın literatüre kazandırdıkları değerlidir.

6.1 Kısıtlar ve Gelecek Çalışmalar

Geliştirmiş olduğumuz özgün kalıp arabelleği algoritmamız arabellek değerlerini ikiye bölerek kullandığı için arabellekte kullanılabilecek değerler büyük ölçüde azalmaktadır, bu da toplamda çizilebilecek portal sayısını 8 bitlik kalıp arabelleklerinde en fazla 15 portal değerine düşürmektedir. İleride Vulkan'ın kalıp arabelleği operasyonları genişletilip çift referans değeri desteği getirilirse bu kısıt çözülmüş olur.

Portal tabanlı yöntemimizinin, haritalama ve eşleme algoritmalarına kıyasla bir eksiği de uygulama için ortamın baştan tasarlanması gerekliliğidir. Bahsi geçen çalışmalarda fiziksel ve sanal ortamın haritaları çıkarılıp birbirine eşlenerek tekrar kullanılabilirken portal tabanlı yöntemde, bahsettiğimiz kurallara uygun bir biçimde sıfırdan bir tasarım yapılması gerekmektedir.

Portal tabanlı yöntemin odalı ve koridorlu yapıya çok müsait olması sebebiyle ileride fiziksel ortamın boyutlarına göre kurallara uygun otomatik sanal ortam üretimi üzerine çalışılabilir. Labirent benzeri ortamlar sınırsız alanı destekleyecek şekilde bu yöntem ile geliştirilmeye açıktır.

Vulkan henüz yeni bir teknoloji olduğundan büyük bir hızla gelişmektedir. Hem sanal gerçeklik hem de çoklu çekirdek desteği ve senkronizasyon konularında yeni çözümler sunulmaktadır. Sanal gerçeklikte uygulamalarında kullanılmak üzere, belli bir noktaya kadar boru hattının paylaşılmasını sağlayan VK_KHR_multiview [20] (çoklu görünüm) uzantısı, görünürlük testi için VK_KHR_conditional_rendering [21] (koşula bağlı çizim) uzantısı ve senkronizasyonu basitleştirmek için VK_KHR_timeline_semaphore [22] (zaman çizelgesi semaforları) uzantıları kullanılarak performansta daha büyük artışlar sağlanabilir.

KAYNAKLAR

[1] **C. Zhang**, (2020). "Investigation on Motion Sickness in Virtual Reality Environment from the Perspective of User Experience", In: IEEE ICISCAE

[2] **Williams** et al., (2007). "Exploring Large Virtual Environments with an HMD when Physical Space is Limited", In: SIGGRAPH APGV

[3] **Usoh** et al., (1999). "Walking, Walking-in-Place, Flying, in Virtual Environments", In: SIGGRAPH

[4] **E. Bozgeyikli, A. Raij, S. Katkoori, R. Dubey,** (2016). "Point and Teleport Locomotion Technique for Virtual Reality", In: ACM SIGCHI CHI PLAY

[5] **E. Pelosin** et al., (2022). "Motor–Cognitive Treadmill Training With Virtual Reality in Parkinson's Disease: The Effect of Training Duration", In: Frontiers in Aging Neuroscience

[6] **D. Solanki, S. Kumar, P. Raj, U. Lahiri,** (2019). "Body Weight Support Assisted Virtual Reality based Treadmill Walk with Gait Characterization", In: ICCCNT

[7] **E.A. Suma, G. Bruder, F. Steinicke, D.M. Krum, M. Bolas,** (2012). "A taxonomy for deploying redirection techniques in immersive virtual environments", In: IEEE VRW

[8] **H. Chen, H. Fuchs,** (2017). "Supporting free walking in a large virtual environment: imperceptible redirected walking with an immersive distractor", In: CGI

[9] **Sun** et al., (2018). "Towards Virtual Reality Infinite Walking: Dynamic Saccadic Redirection", In: ACM TOG

[10] **J. J. Hopp and A. F. Fuchs,** (2004). "The characteristics and neuronal substrate of saccadic eye movement plasticity", In: Progress in Neurobiology

[11] **Dong** et al., (2017). "Smooth Assembled Mappings for Large-Scale Real Walking", In: ACM TOG

[12] **Sun** et al., (2016). "Mapping Virtual and Physical Reality", In: ACM TOG

[13] **M. L. Sümmermann,** (2021). "Knotted Portals in Virtual Reality", In: The Mathematical Intelligencer

[14] **N. Lowe, A. Datta,** (2005). "A New Technique for Rendering Complex Portals", In: IEEE Trans. Vis. Comput. Graph.

[15] **N. Lowe, A. Datta,** (n.d.). "A fragment culling technique for rendering arbitrary portals", Lecture Notes in Computer Science, School of Computer Science and Software Engineering, University of Western Australia

[16] **E. Lengyel,** (2005), "Oblique View Frustum Depth Projection and Clipping", In: Journal of Game Development, Vol. 1, No. 2, Charles River Media, pp. 5–16.

[17] **M. Gambhir, S. Panda, S. J. Basha,** (2018). "Vulkan rendering framework for mobile multimedia", In: SIGGRAPH Asia

[18] **A. Blackert,** (2016). "Evaluation of Multi-Threading in Vulkan", Linköping University, Department of Electrical Engineering, Information Coding

[19] **Khronos Group**, (n.d.). OpenGL AMD Stencil Operation Extended Extension, URL: `https://registry.khronos.org/OpenGL/extensions/AMD/AMD_stencil_operation_extended.txt`

[20] **Khronos Group**, (n.d.). Vulkan Multiview Extension, URL: `https://www.khronos.org /registry/vulkan/specs/1.3-extensions/man/html/VK_KHR_multiview.html`

[21] **Khronos Group**, (n.d.). Vulkan Conditional Rendering Extension, URL: https://www.khronos.org/registry/vulkan/specs/1.3-extensions/man/html/VK_EXT_conditional_rendering.html

[22] **Khronos Group**, (n.d.). Vulkan Timeline Semaphore Extension, URL: https://registry.khronos.org/vulkan/specs/1.3-extensions/man/html/VK_KHR_timeline_semaphore.html

[23] **Meta Platforms, Inc.**, (n.d.). Asynchronous Spacewarp, URL: https://developer.oculus.com/blog/asynchronous-spacewarp