KARADENİZ TEKNİK ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
BİLGİSAYAR GRAFİKLERİ LABORATUARI

DirectX ile Tank Oyunu

1. Giriş

Oyunlar, Bilgisayar Grafikleri'nin en popüler uygulama alanlarından biridir. Bu deneyde DirectX 12 API ile basit bir 3D Tank Oyununun nasıl geliştirilebileceğinden bahsedilecektir.

2. Tank Oyunu Fonksiyonları ve Görevleri

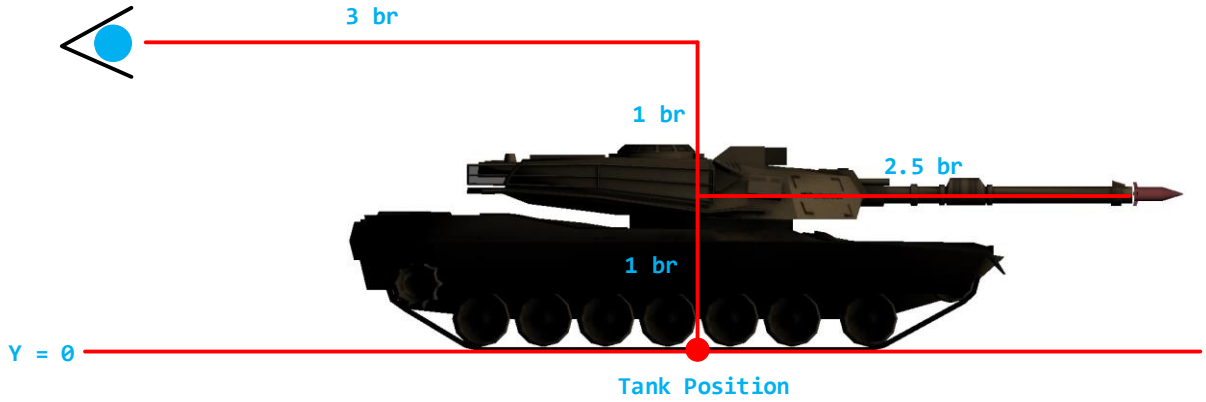
Genel olarak oyunlar 3 temel fonksiyondan oluşurlar:

- **`OnInit()`** : Oyuncular ve oyun ortamı bufferlara (vertex, index, texture) yüklenir. Oyunla ilgili bazı matris ve vektörler ilk değerlerine setlenir.
- **`OnUpdate()`** : Klavye/mouse etkileşimlerine göre oyuncuların konumları güncellenir. Oyuncuların birbirleriyle (veya ortamla) olan etkileşimlerine göre örneğin birbirlerine ateş etmişlerse (veya duvara yaklaşmışlarsa) isabet (kesişim) testleri yapılır.
- **`OnRender()`** : Oyuncular güncel konumlarına çizilir veya vurulan oyuncu artık çizilmez.

`OnInit()`, uygulama çalıştırıldığında ilk koşan ve bir kere koşan fonksiyondur. Dolayısıyla oyuncuların, oyun ortamının yüklenmesi gibi uygulamada bir kez yapılacak işlemlere dair kodlar burada yazılmalıdır. Ayrıca vektörlere/matrislere ilk değer atama da burada yapılır. Tank oyununda oyun ortamını oluşturan zeminin, duvarların, oyuncular olan tankların **`OnInit()`** fonksiyonunda kendilerine ait köşe noktası, doku ve normal koordinatları bilgilerini içeren **`.obj`** formatında model dosyaları okunup ilgili vertex/index/texture bufferlarına yüklenir. **`OnInit()`** fonksiyonunda ayrıca **`Eye`**, `At` ve `Up` vektörleri ilgili vektör değerlerine setlenir bu vektörler kullanılarak **`XMMatrixLookAtLH()`** ile View matrisi setlenir. **`OnInit()`** fonksiyonunda son olarak, görüş açısı (field of view angle), pencerenin yatay/düşey oranı (aspect ratio), yakın ve uzak **`z`**-düzlemlerini (near/far z-planes) parametre olarak alan **`XMMatrixPerspectiveFovLH()`** ile perspektif izdüşüm (projeksiyon) matrisi setlenir.

```
Eye = XMVectorSet(0.0f, 4.0, -30.0, 0.0);
At  = XMVectorSet(0.0f, 0.0,   1.0, 0.0);
Up  = XMVectorSet(0.0f, 1.0,   0.0, 0.0);
g_View = XMMatrixLookAtLH(Eye, At, Up);

g_Projection = XMMatrixPerspectiveFovLH(XM_PIDIV4, 1280/720, 0.01, 1000);
```



OnUpdate() fonksiyonunda klavye/mouse etkileşimine göre ilgili değişkenler setlenmektedir. Örneğin **W,A,S,D** tuşları ve mouse ile **Eye, At** vektörleri ve dolayısıyla **g_View** matrisi güncellenmektedir (836. satır). Böylece 3D ortamda mouse doğrultusunda tuşlarla ilerlenmektedir. Tankın konumu (**Tank_Position**) da 3rd Person Shooter tarzı olarak bakış noktasından (**Eye**) hareket doğrultusunca (**At-Eye**) 3 birim ilerisinde ve 2 birim aşağısında olacak şekilde şöyle hesaplanmakdır (847. satır):

```
XMVECTOR Tank_Position = Eye + 3 * (At - Eye) + XMVectorSet(0, -2, 0, 0);
```

Tank düşmana (enemy) **SPACE** tuşu ile ateş etmektedir. Bu tuşa basıldığında namlunun ucundan top mermisi çıkmakta ve düşmana doğru hareket etmektedir. Merminin başlangıç noktası namlunun ucu olacak şekilde World matrisi aşağıdaki kodla setlenir (859-873 arası):

```
Ro_Tank_Missile = Eye + XMVectorSet(0, -1, 0, 0);
Rd_Tank_Missile = XMVector3Normalize(At - Eye);

if (FireTankMissile)
{
	XMVECTOR initialPosition = Ro_Tank_Missile + (3+2.5)*Rd_Tank_Missile;
	XMFLOAT4 initialPosition_F4;
	XMStoreFloat4(&initialPosition_F4, initialPosition);
	g_World_Missile = g_World_Tank;
	XMFLOAT4X4 g_World_Missile_4x4;
	XMStoreFloat4x4(&g_World_Missile_4x4, g_World_Missile);

	g_World_Missile_4x4._41  = initialPosition_F4.x;
	g_World_Missile_4x4._42  = initialPosition_F4.y;
	g_World_Missile_4x4._43  = initialPosition_F4.z;
	g_World_Missile          = XMLoadFloat4x4(&g_World_Missile_4x4);
}
```

initialPosition = Ro_Tank_Missile + (3 + 2.5) * Rd_Tank_Missile ile merminin namlu ucundaki konumu hesaplanır. **Eye + XMVectorSet(0, -1, 0, 0)** ile merminin başlangıç noktası hesaplanır. Namlunun yerden yüksekliği 1'i elde edebilmek için bakış noktasının yüksekliği **2**'den **1** çıkarılmıştır. **(3+2.5)*Rd_Tank_Missile** eklendiğinde başlangıç noktası namlunun ucu olur. Burada **2.5** tankın merkezinden namlu ucuna kadar olan mesafedir. Tank bakış noktasından **3** birim ileride olduğundan ayrıca **3** eklenmiştir. **Rd_Tank_Missile** merminin doğrultusudur.

Merminin 4x4 World matrisinin (**g_World_Missile**) öteleme (translation) bileşenleri **_41**, **_42** ve **_43** merminin başlangıç noktasının **x**, **y** ve **z** değerlerine setlenerek başlangıç noktası olarak namlunun ucuna çizilmesi sağlanır. **SPACE** tuşuna basıldığında **FireTankMissile** boolean değişkeni **false** **TraceTankMissile** **true** yapılarak merminin hareket edebilmesi sağlanır. Merminin hareketi için World matrisinin **_41**, **_42** ve **_43** bileşenleri yukarıdakine benzer aşağıdaki kodla setlenir (877-884):

```
if (TraceTankMissile)
{
XMStoreFloat4x4(&g_World_Missile_4x4, g_World_Missile);
g_World_Missile_4x4._41 += 0.5 * Rd_Tank_Missile_Float4.x;
g_World_Missile_4x4._42 += 0.5 * Rd_Tank_Missile_Float4.y;
g_World_Missile_4x4._43 += 0.5 * Rd_Tank_Missile_Float4.z;
g_World_Missile = XMLoadFloat4x4(&g_World_Missile_4x4);
}
```

Merminin düşmana isabet edip/etmediği "*Ray Tracing*" ile test edilir (899-924). **Ro_Tank_Missile** başlangıç noktası **Rd_Tank_Missile** doğrultusuna sahip ışın ile ortamdaki tank, duvarlar ve zemin arasında kesişim testleri yapılır. Mininum **t** uzaklığına sahip cisim düşman tankı ise ve mermi düşman tankına değmişse düşmanın render edilip/edilmeyeceğini belirleyen **RenderEnemy** boolean değişkeni **false** yapılarak düşman öldürülür yani çizilmez. **renderTankMissile** da **false** yapılarak mermi de artık çizilmez. Düşman tankı canlandırmak (tekrar çizmek) için sol mouse butonuna tıklanır.

Merminin düşman tankına değip değmediğinin belirlenmesi için kesişim testlerinden dönen en yakın cismin **nearest.t** uzaklığı ile düşman üzerindeki kesişim noktasının konumu **RedDot_Position** hesaplanır (906). **RedDot_Position** ile merminin o andaki konumu **Missile_Position**'ın fark vektörünün boyu **Missile_RedDot_Distance** çok küçük bir değerin (**0.5**) altına düşünce merminin düşmana isabet ettiği sonucuna varılır. **Missile_RedDot_Distance** şöyle hesaplanır:

```
XMVectorGetX(XMVector3Length(RedDot_Position - Missile_Position))
```

XMVector3Length() aslında skaler bir değer döndürmelidir ama DirectX'in matematik kütüphanesindeki vektörel işlem fonksiyonlarının tamamı vektör döndürmektedir. Dönen bu vektörden örneğin yukarıdaki gibi fark vektörünün boyunu okuyabilmek için **XMVectorGetX()** kullanılmıştır. Bunun yerine **XMVectorGetY()** ya da **XMVectorGetZ()** de olabilirdi. DirectX'in matematik kütüphanesi fonksiyonlarına buradan erişebilirsiniz. Oyunda kullanılan fonksiyonlar "Geometric Functions" ve "Transformation Functions" linklerindedir.

Oyun ortamında ilerlerken duvarların içinden geçilmemesi için de Ray Tracing yöntemi kullanılabilir. **OnUpdate()** fonksiyonunda **W** ve **S** tuşlarının test edildiği **if(){}** bloklarında ışının başlangıç noktası ve doğrultusu **Ro** ve **Rd** vektörleri örneğin W için aşağıdaki gibi setlenip **IntersectTriangle()**'dan dönen **t** uzaklığı belli bir değerin altına inince hareket engellenir. Başlangıç değeri **true** olarak setlenen bool bir değişken **t** belli bir değerin altına inince **false** yapılır. **moveBackForward+=speed** ve **moveBackForward-=speed** güncellemeleri bu değişkene bağlı olarak (**true** ise) yapılır:

```
XMVECTOR Ro = Eye;
XMVECTOR Rd = XMVector3Normalize(At - Eye); // S için : (Eye - At)
```

`OnRender()` fonksiyonunda sırasıyla zemin, duvarlar, oyuncuyu temsil eden tank, o tankın ateş etmesi sonucu yollanacak mermi (tank missile), nişangâh (reddot), düşman tankı (enemy) ve son olarak düşman tankının ateş etmesi sonucu yollanacak mermi (enemy missile) kendileri için `OnUpdate()`'te yapılan `m_constantBufferData.mWorld` constant buffer setlemelerine göre transform edilip çizilir.

3. Deney Hazırlığı

- Sağ mouse butonu tıklanılmış olarak tutulurken zoom yapılacak ve mousedan el çekildiğinde zoom öncesine dönülecek şekilde 800 ve 805. satırlardaki `if(){}` bloklarına gerekli kodları yazınız.
  İpucu → Projeksiyon matrisinin görüş açısı (`FovAngleY`) daha küçük bir değere (örneğin `XM_PI/12`) setlenerek zoom yapılabilir.

- Duvar arkasından ateş edildiğinde mermi duvarı delip geçmeyecek şekilde kodu güncelleyiniz.

- Oyun ortamında ilerlerken duvarların içinden geçilmemesi için önceki sayfanın son paragrafında anlatılan yönteme göre 742 ve 747. satırlardaki `if(){}` bloklarına gerekli kodları yazınız.

- Tank düşmana ateş ettiğinde mermi dönerek ilerleyecek şekilde kodu güncelleyiniz.

- Tank düşmana ateş ettiğinde mermi düşmana ulaşana dek namluyu oynatmamak gerekir. Eğer oynatılırsa merminin yönü değişir ve kesişim testleri merminin (namlunun) o anki doğrultusuna göre yapılır. Programdaki bu hatayı düzeltiniz. Yani ateş ettikten sonra namlu oynatılsa bile mermi ateş edilen doğrultuda ilerlesin.

4. Deney Tasarımı ve Uygulaması

Düşmanı temsil eden tank (enemy) bizim kontrol ettiğimiz tanka (tank) ateş edip vuracak şekilde kodu güncelleyiniz. Basitten zora doğru adım adım ilerlemek için öncelikle namluyu döndürmeksizin sanki dönmüş ve hedefe kilitlenmiş gibi varsayarak ateş ettirebilirsiniz. Bunun için `OnUpdate()` fonksiyonunda 945. satırdan itibaren kod yazacaksınız. Ekleyeceğiniz kodlar 859-924 arasındaki kodlara benzer olacaktır. Yani biz düşmana ateş ettiğimizde gerçekleşen olayların benzerleri düşman bize ateş ettiğinde yaşanacaktır. Öncelikle **`if(FireEnemyMissile){}`** ve **`if(TraceEnemyMissile){}`** şeklinde iki kod bloğu ekleyip bu bloklara sırasıyla merminin başlangıç konumunun setlenmesi ve ilerlemesi için gerekli kodlar yazılır:

`if(FireEnemyMissile){}` içinde merminin başlangıç noktası namlunun ucu olarak setlenirken namlunun doğrultusu olan **`(0,0,1)`** vektörü namlunu boyu **`2.5`** ile çarpılıp düşman mermisinin 4x4 World matrisinin (**`g_World_Enemy_Missile`**) öteleme (translation) bileşenleri **`_41`**, **`_42`** ve **`_43`** setlenir. Bizim kontrol ettiğimiz tank için ayrıca **`Ro_Tank_Missile`** değişkenini de kullanmıştık. Çünkü biz 3D ortamda hareket ediyoruz. Düşman tankı ise sabit olduğundan düşmanın konumu **`(0,0,0)`** olarak varsayılabilir ve dolayısıyla merminin konumu hesaplanırken **`Ro`**'ya gerek yoktur.

`if(TraceEnemyMissile){}` içinde mermi ilerletilmek üzere hareket doğrultusu belirlenmelidir. Bizim kullandığımız tankta merminin yönü **`XMVector3Normalize(At-Eye)`** ile belirleniyordu. Bu sefer mermi yönü bizim kullandığımız tankın konumundan merminin konumu çıkarılarak bulunacaktır. İlgili World matrislerinin **`_41`**, **`_42`** ve **`_43`** bileşenlerinin farkları **`XMVectorSet()`** ile bir vektöre setlenip **`XMVector3Normalize()`** ile normalize edilerek yön belirlenebilir. Herhangi bir tuşa (örneğin '**`F`**') basıldığında **`FireEnemyMissile`** boolean değişkeni **`false`** **`TraceEnemyMissile`** **`true`** yapılarak mermi ilerletilir.

Düşman tankından ateşlenen merminin bizi vurup/vurmadığı da benzer şekilde test edilbilir. Merminin başlangıç noktası **`if(FireEnemyMissile){}`** içinde, doğrultusu da **`if(FireEnemyMissile){}`** içinde hesaplanmıştı. **`testIntersections()`** fonksiyonuna bu vektörler **`Ro, Rd`** değerleri olarak verilir. 3. parametre de bizim tankın World matrisidir.

`testIntersections()` fonksiyonu **`IntersectTriangle()`** fonksiyonunu zemin, duvarlar ve tank için koşar ve (varsa) kesişimleri **`intersect`** türünden structlar olarak `intersections` adlı STL vektöre ekler. **`intersect`** struct'ı `t` uzaklığına ek olarak `isWall` ve `isEnemy` olmak üzere iki tane boolean değişken tutar. **`testIntersections()`** fonksiyondaki kesişim testlerine göre duvarla kesişim varsa `isWall`, tankla kesişim varsa da `isEnemy` **`true`** yapılır. **`nearestObject()`** fonksiyonu en yakın kesişimi **`intersect`** olarak döndürür. Eğer **`isEnemy=true`** ise (burada enemy bizim kontrol ettiğimiz tanktır) ve nişan alınan nokta (**`RedDot_Position`**) ile merminin o anki konumu arasındaki uzaklık çok küçük bir değerin altına düşmüşse **`renderTank`** ve **`renderEnemyMissile`** **`false`** yapılır yani artık render edilmezler.

Deney uygulamasının bu noktaya kadarına ait videoyu buradan izleyebilirsiniz.

Şimdi sıra geldi düşmanın dönerek hedefe (bize) kilitlenip ateş etmesine. Düşman tankını döndürmek üzere **g_World_Enemy** matrisi **0.02** gibi küçük bir açı ile y-ekseninde dönme yapacak şekilde **XMMatrixRotationY(0.02)** matrisi ile çarpılarak güncellenir. **(0,0,1)** başlangıç değerine setlenmiş namlu doğrultu vektörü, **g_World_Enemy** matrisine göre **XMVector3TransformNormal()** fonksiyonu ile transform edilerek güncellenir. Yukarıda da anlatıldığı gibi World matrislerinin **_41**, **_42** ve **_43** bileşenlerinin farkları **XMVectorSet()** ile bir vektöre setlenip **XMVector3Normalize()** ile normalize edilerek düşmanı temsil eden tanktan bizim tanka doğru olan vektör hesaplanır. Bu vektör ile **XMVector3TransformNormal()** ile transform edilen vektör arasındaki açının kosinüsü **XMVector3Dot()** ile hesaplanır. Bu değer **1** olana kadar (veya **>0.9999**) yani transform edilen namlu doğrultu vektörü ile düşmandan bize doğru olan vektör arasındaki açı **0** derece olana (bize kilitlenene) kadar bu paragraftaki işlemler tekrarlanır.

Deney uygulamasının bu noktaya kadarına ait videoyu buradan izleyebilirsiniz.

5. Deney Raporu

Kaynak kodlardaki **Rapor.docx** adlı şablon belge **deney saatine kadar** takım adına doldurulup **Deney Hazırlığı** olarak istenen güncellemeleri içeren **D3D12TankOyunu.cpp** kod dosyası ile birlikte (takım adına biriniz) dersin Moodle Sayfasına yükleyiniz.