KARADENİZ TEKNİK ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
BİLGİSAYAR GRAFİKLERİ LABORATUARI

DirectX ile FPS Oyunu

1. Giriş

Oyunlar, Bilgisayar Grafiklerinin en popüler uygulama alanlarından biridir ve bilgisayarın dışında tabletler ve cep telefonları gibi farklı donanımları destekleyen oyunların da yaygınlaşmasıyla birlikte artan oranda popüler olmaya devam edeceği öngörülmektedir. 3 boyutlu (3D) oyun türleri arasında en yaygın olanlarından biri de oyun ortamının, oyuncunun gözlemlerine göre çizildiği FPS (First Person Shooter) tarzı oyunlardır.

Bu deneyde DirectX 11 API ile tek kullanıcılı basit bir FPS oyununun temel özelliklerinden bahsedilecektir.

2. 3D Cisme Yollanacak Işının Üretilmesi ve Kesişim Testi

Herhangi bir FPS oyununun en temel amaçlarından biri ateş edip rakip oyuncuyu vurmaktır. Ateş edilen rakip oyuncunun vurulup vurulmadığına karar vermek için ona doğru bir ışın yollanır ve bu ışın ile rakip oyuncuyu temsil eden modeli oluşturan üçgenler arasında kesişim testleri yapılır. Bu bölümde, mouse ile ekranda rakip oyuncuya denk gelen piksele tıklanması ile ateş edilip yollanacak mermiyi temsil edecek olan 3D ışının başlangıç noktası ve doğrultusunun nasıl üretildiği ve modelle nasıl kesişim testi yapıldığı anlatılacaktır.

Ekranda rakibe ateş etmek amacıyla tıklanan noktaya ait 2D koordinatlardan 3D ortamdaki cisimle kesişim testinde kullanılacak ışının üretilmesi için bir takım transformasyonlar gerekmektedir.

Bilindiği gibi 3D ortamdaki herhangi bir noktanın ekran koordinatlarına dönüşümü için bu nokta sırasıyla `World`, `View` ve `Projection` matrisleri ile çarpılır. Dolayısıyla ekranda tıklanan 2D noktadan 3D ışını üretmek için matris çarpımlarının tersi yani ters matrisler ile işlemler yapmak gerekir.

Çizim yapılacak pencerenin `800x600` çözünürlükte olduğu varsayıldığında bu pencerenin sol üst köşesinin koordinatları `(0,0)` sağ alt köşesinin de `(799,599)` olur. Bu koordinatlardan pencerenin merkezi `(0,0)`'dan sağa doğru `+X`, sola doğru `-X`; yukarı doğru `+Y` ve aşağı doğru `-Y` olacak şekilde normalize edilmiş yani `(-1,+1)` arası değişen koordinatlara dönüşüm yapan DirectX kodu aşağıdaki gibidir:

```
D3DXVECTOR3 v;
v.x =  ( ( ( 2.0f * ptCursor.x ) / Width ) - 1 ) / pmatProj->_11;
v.y = -( ( ( 2.0f * ptCursor.y ) / Height ) - 1 ) / pmatProj->_22;
v.z = 1.0f;
```

Burada `ptCursor.x` ekranda tıklanan noktanın `(0,799)` arası değişen `x`; `ptCursor.y` de `(0,599)` arası değişen `y` koordinatlarıdır. `pmatProj` de `Projection` matrisidir. İşlemleri ters sırada yaptığımızdan `Projection` matrisinin ilgili diyagonal bileşeni ile çarpmak yerine ona bölüyoruz.

Yukarıdaki işlemler ışının üretilmesi için yeterli değildir. `v` vektörü ayrıca `World` ve `View` matrislerinin tersi olan `m` matrisi ile de aşağıda verilen koddaki gibi çarpılmalıdır :

```
// Get the inverse view matrix
const D3DXMATRIX matView = *g_Camera.GetViewMatrix();
const D3DXMATRIX matWorld = *g_Camera.GetWorldMatrix();
D3DXMATRIX mWorldView = matWorld * matView;
D3DXMATRIX m;
D3DXMatrixInverse( &m, NULL, &mWorldView );

// Transform the screen space pick ray into 3D space
vPickRayDir.x = v.x * m._11 + v.y * m._21 + v.z * m._31;
vPickRayDir.y = v.x * m._12 + v.y * m._22 + v.z * m._32;
vPickRayDir.z = v.x * m._13 + v.y * m._23 + v.z * m._33;
vPickRayOrig.x = m._41;
vPickRayOrig.y = m._42;
vPickRayOrig.z = m._43;
```

Böylece 2D ekranda tıklanan noktaya 3D uzayda karşılık gelen ışının başlangıç noktası `vPickRayOrig` ve doğrultusu `vPickRayDir` üretilmiş olur. Bu noktaya kadar verilen kod satırları "`Picking`" örnek programı içindeki `Pick()` isimli fonksiyondan alınmıştır. Yine bu fonksiyonda 3D ortamdaki cismin tam olarak hangi üçgenine tıklandığını belirlemek üzere Tomas Möller'in ışın-üçgen kesişim testi yöntemini kullanan `IntersectTriangle()` fonksiyonu çağrılmaktadır.

.obj formatındaki modeli temsil eden `verticesObjModel[]` dizisinden o anki üçgeni temsil eden `i` değişkeninin `3` katına sırasıyla `0`, `1` ve `2` eklenerek elde edilen indislerdeki köşe noktaları sırasıyla `v0`, `v1` ve `v2` vektörlerine atanmıştır. Işının başlangıç noktası `vPickRayOrig`, doğrultusu `vPickRayDir` ve `v0`, `v1` ve `v2` vektörleri kesişim testini yapacak `IntersectTriangle()` fonksiyonuna parametre olarak yollanmıştır.

`IntersectTriangle()` fonksiyonu kesişimin olup/olmamasına göre boolean bir değer döndürmenin yanında kesişen üçgene olan uzaklığı `fDist` olarak, `(u,v)` doku koordinatlarını veya barisentrik koordinatları da `fBary1` ve `fBary2` olarak hesaplayıp döndürmektedir. Bu değişkenleri içeren `INTERSECTION` türünden structure her bir kesişim için `g_IntersectionArray[]` adlı diziye eklenmektedir. Örneğin o doğrultu boyunca giden ışının kesiştiği üçgenlerden en yakın olanına ait bilgiler `g_IntersectionArray[0]` indisi altındaki değişkenlerde tutulmaktadır. `g_bAllHits` boolean değişkeni bütün kesişimlerin diziye alınıp/alınmayacağını belirler. Bu işlemlerin gerçekleştirildiği kod bloğu aşağıda verilmiştir :

```
for( DWORD i = 0; i < faceCount; i++ )
{
	D3DXVECTOR3 v0 = verticesObjModel[3 * i + 0].Pos;
	D3DXVECTOR3 v1 = verticesObjModel[3 * i + 1].Pos;
	D3DXVECTOR3 v2 = verticesObjModel[3 * i + 2].Pos;

	if( IntersectTriangle(vPickRayOrig,vPickRayDir, v0, v1, v2, &fDist,&fBary1,&fBary2))
	{
		if(g_bAllHits|| g_nNumIntersections==0 || fDist<g_IntersectionArray[0].fDist)
		{
			if( !g_bAllHits ) g_nNumIntersections = 0;
			g_IntersectionArray[g_nNumIntersections].dwFace = i;
			g_IntersectionArray[g_nNumIntersections].fBary1 = fBary1;
			g_IntersectionArray[g_nNumIntersections].fBary2 = fBary2;
			g_IntersectionArray[g_nNumIntersections].fDist = fDist;
			g_nNumIntersections++;
			if( g_nNumIntersections == MAX_INTERSECTIONS )    break;
		}
	}
}
```

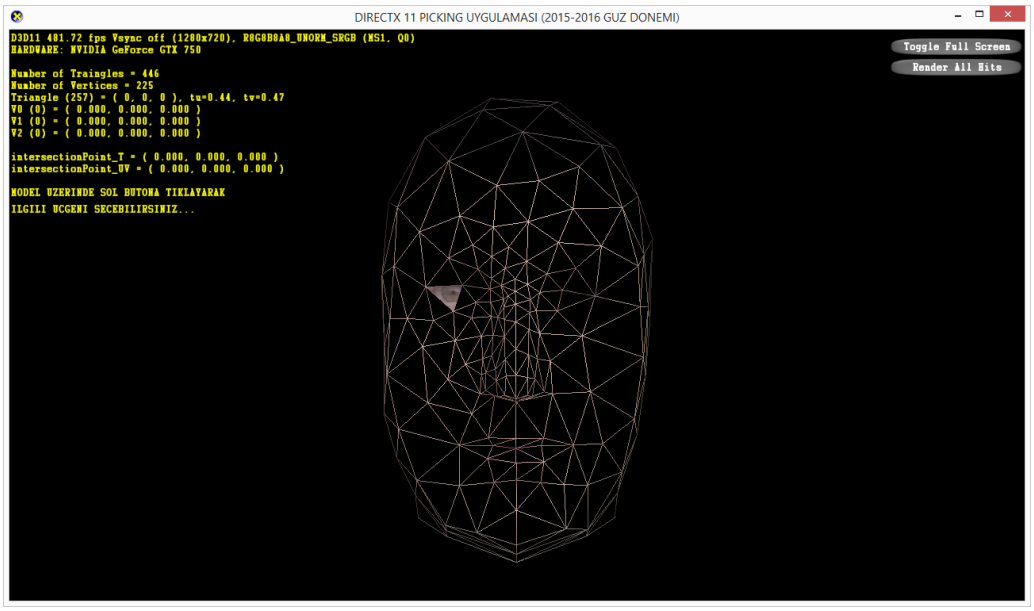
Model çizilirken yalnızca vertex buffer kullanılmıştır. Başka bir deyişle `verticesObjModel[]` dizisindeki köşe noktaları sırasıyla 3'erli gruplar halinde modelin üçgenlerini temsil etmektedir.

3. Kesişen Üçgenlerin Çizilmesi

Kesişen üçgenlerin çizimi `OnD3D11FrameRender()` fonksiyonunda aşağıdaki kod ile yapılır. Bu üçgenlerin köşe noktaları `PickedTriangle` isimli vertex bufferda tutulmaktadır. Buffera köşe noktalarını kopyalama işlemleri için `Map()` ve `Unmap()` emirleri kullanılmıştır (870. ve 894. satırlar). Çizim modu `RSSetState(g_pRasterizerStateSolid)` ile setlendikten sonra `Draw()` emri ile kesişen üçgen(ler)in çizimi gerçekleştirildikten sonra çizim modu `RSSetState(g_pRasterizerStateWireframe)` olarak setlenerek kafa modeline ait diğer üçgenler wireframe (sadece kenarlar) çizilmektedir. Böylece kesişen üçgen(ler) daha belirgin hale getirilmiştir.

Ekranda tıklanan noktadan yollanan ışın, kafa modeli ile önden ve arkadan olmak üzere iki noktada kesişmektedir. Bunlardan sadece en yakın olan öndekinin mi yoksa ikisinin birden mi Render edileceği "`Render All Hits`" butonu ile setlenir. Şekil-1'de programdan bir ekran görüntüsü verilmiştir :

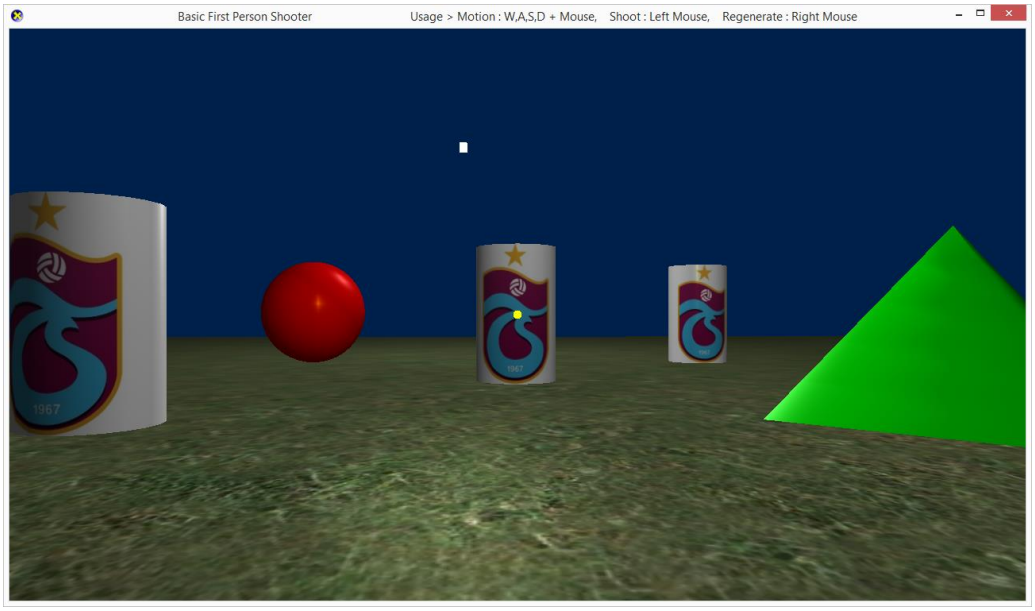
```
if( g_nNumIntersections > 0 )
{
	UINT Strides[1];                          UINT Offsets[1];
	Strides[0] = sizeof(SimpleVertex);        Offsets[0] = 0;

	pd3dImmediateContext->IASetVertexBuffers(0, 1, &PickedTriangle, Strides, Offsets);
	pd3dImmediateContext->IASetPrimitiveTopology(D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST);
	pd3dImmediateContext->PSSetShaderResources(0, 1, &g_pTextureObjModel);
	pd3dImmediateContext->PSSetSamplers( 0, 1, &g_pSamLinear );
	pd3dImmediateContext->Draw( 3*g_nNumIntersections, 0 );

	// Set render mode to Wireframe
	pd3dImmediateContext->RSSetState( g_pRasterizerStateWireframe );
}
```



Şekil-1: DirectX Picking Uygulaması

4. Basit bir FPS Oyunu

Bu bölümde basit bir FPS oyunu olan "`FPS_Game`" adlı DirectX 11 uygulamasından bahsedilecektir. Uygulamada ateş edildiğinde rakibin vurulup/vurulmadığı önceki bölümde anlatıldığı gibi ışın-üçgen kesişim testleriyle yapılmaktadır. Bunun için `testIntersection()` adlı fonksiyon yazılmıştır. Fonksiyonun başındaki `for()` döngüsü içinde test edilecek rakip oyuncuyu temsil eden modelin üçgenlerinin köşe noktaları 3'er 3'er ışın-üçgen kesişim testine tabi tutulur. Eğer kesişim varsa `TRUE` döndürülür. Nişan alınan modele mouse'un sol butonuna tıklanarak ateş edilir ve yollanan ışın modelle keşisirse modelin çizilip/çizilmeyeceğini temsil eden boolean değişken `FALSE` olarak setlenerek çizimi engellenir (öldürülür). Boolean değişken `TRUE` yapılarak tekrar çizilmesi (canlandırılması) için mouse'un sağ butonuna tıklanır. Uygulamanın ekran görüntüsü Şekil-2'de verilmiştir:



Şekil-2: DirectX FPS_Game Uygulaması

`testIntersection()` adlı fonksiyon kesişimin olup/olmamasına bağlı olarak boolean değer döndürmenin yanında modele olan `t` uzaklığını da döndürmektedir. Bu uzaklık belli bir değerin altına düştüğünde hareket durdurularak duvarların içinden geçilmesi engellenmiştir. `testIntersection()` fonksiyonu basılan tuş veya mouse butonu ile ilgili kodları barındıran `DetectInput()` fonksiyonu içinde çağırılmıştır.

5. Deney Hazırlığı

❖ Seçilen üçgene ait köşe noktalarının indisleri (`index_v#`) ve koordinatları (`_v#`) başlangıç değeri olarak `0`'a setlenmiştir. Doğru değerler ekrana yazılacak şekilde programda `861.` satırdan itibaren gerekli güncellemeleri yapınız. Hesaplanan değerlerin ekrana yazılması ile ilgili kodlar `RenderText()` adlı fonksiyonda vardır.

❖ Seçilen üçgende mouse'un hangi koordinatlara işaret ettiği 2 şekilde hesaplanabilir:

`fDist` uzaklık değerini kullanarak → `intersectionPoint_t`
`fBary1`,`fBary2` barisentrik koordinatlar ile → `intersectionPoint_uv`

`intersectionPoint_t` ve `intersectionPoint_uv` değişkenlerini hesaplamak için gerekli kodları yine programa `861.` satırdan itibaren ekleyiniz. Yazdığınız kodları (dosya boyutunu azaltmak üzere `.sdf` uzantılı dosya, `ipch` ve `Debug` klasörlerini sildikten sonra projenin tamamını .rarlayıp en geç deney saatine kadar) hem mustafayazici61@gmail.com adresine grup adına e-mail ile gönderiniz hem de USB bellekte deneye getiriniz. Programın tamamlanmış haline ait ekran görüntüsü `PickScreen.wmv` isimli video olarak kaynak kodların olduğu klasördedir.

6. Deney Tasarımı ve Uygulaması

❖ `IntersectTriangle()` isimli ışın-üçgen kesişim testi fonksiyonu yerine "Alan Testi" yöntemini gerçekleyecek şekilde `IntersectTriangleAlan()` fonksiyonuna gerekli kodları ekleyiniz. Kodları yazarken şunlara dikkat ediniz :

Skaler ve vektörel çarpımlar için sırasıyla `D3DXVec3Dot()` ve `D3DXVec3Cross()` emirlerini; vektör boyunu hesaplamak için de `D3DXVec3Length()` emrini kullanacaksınız.

Fonksiyon, kesişim varsa `true` döndürmenin yanı sıra `intersectionPoint_t` hesabında `fDist` olarak kullanılacak `t` değişkenini gerektiği gibi setlemelidir.

Kod tamamlandıktan sonra `Pick()` fonksiyonu içindeki `IntersectTriangle()` satırını kapatıp `IntersectTriangleAlan()` satırını açınız (`840.` ve `841.` satırlar).

❖ FPS_Game uygulamasında aynı doğrultudaki modellere ateş edildiğinde hepsi ölmektedir. Yalnızca en öndekinin ölmesi için gerekli kod güncellemelerini yapınız.

❖ FPS_Game uygulamasında `Render()` fonksiyonu içindeki `if(renderTSilindir2){}` kod bloğundaki kapalı (comment) satırlar açıldığında silindirlerden biri 5 birim yarıçapla dönmeye başlamaktadır. `testIntersection()` fonksiyonunda gerekli güncellemeleri yaparak hareket halindeki nesnelerin de vurulabilmesini sağlayınız.

7. Deney Raporu

Deney Raporunu, **`Rapor.docx`** adlı şablon belgeye göre grup adına hazırlayıp **Deneye Hazırlık** bölümünde istenilen kodlarla birlikte **en geç deney saatine kadar** `mustafayazici61@gmail.com` adresine gönderiniz. Mailin başlığına grubunuzu ve deneyin ismini yazınız. Rapor ayrıca printer çıktısı olarak **deneye de getirilecektir**.