KARADENİZ TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BIL 3008 BİLGİSAYAR GRAFİKLERİ-I DERS NOTLARI

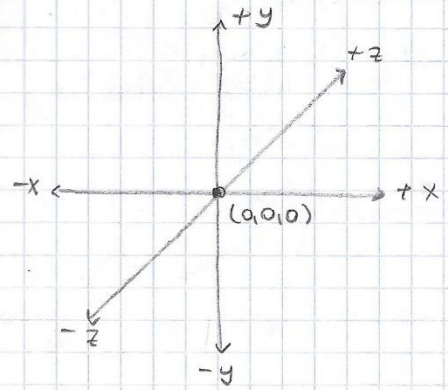
HASRET SUNA DİZDAR

2015-2016 Bahar Dönemi

19.02.2016

RAY TRACING

Vektörel İşlemlerin Temelleri



$$|R1| = \sqrt{x^2+y^2+z^2}$$

* Vektörün boyutunu 1 yapma işlemine normalizasyon denir. Normalize edilmiş vektöre de birim vektör denir.

* Vektörel çarpım *

X → kartezyen çarpım
* → skaler çarpım

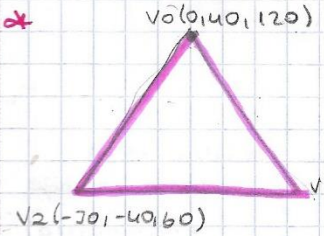
$$R_1 \times R_2 = (R_{1y}R_{2z} - R_{1z}R_{2y},\ R_{1z}R_{2x} - R_{1x}R_{2z},\ R_{1x}R_{2y} - R_{1y}R_{2x})$$

sonucu vektördür

$$R_1 * R_2 = R_{1x}R_{2x} + R_{1y}R_{2y} + R_{1z}R_{2z} = |R1| * |R2| * \cos(\beta)$$

sonucu bir ton sayıdır.

*



$$V1 - V0 = (30, -80, -60)\ \{R1 \quad (R_{1x}\ R_{1y}\ R_{1z})$$

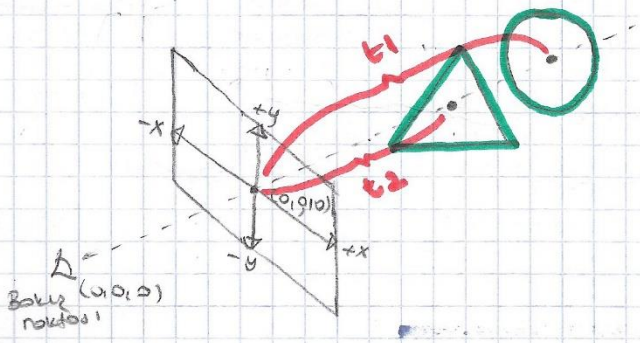
$$V2 - V0 = (-30, -80, -60)\ \{R2 \quad (R_{2x}\ R_{2y}\ R_{2z})$$

$$N = R1 \times R2 = \left((-80)\cdot(-60) - (-60)\cdot(-80),\ (-60)(-30) - (30)\cdot(-60),\ (30)(-80) - (-80)(-30)\right)$$

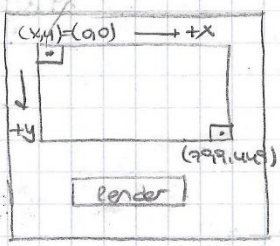
$$N = (0, 3600, -4800) = (0/6000,\ 3600/6000,\ -4800/6000)$$

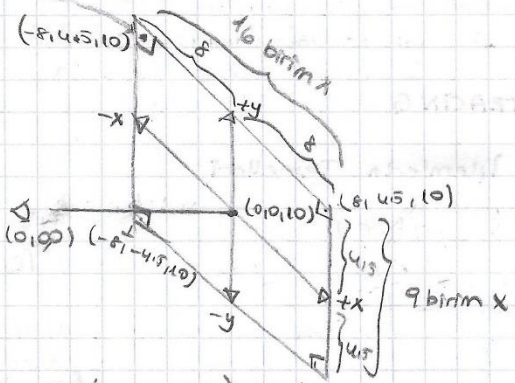
$$N = (0,\ 0.6,\ -0.8)$$

* Bakış noktası (0,0,0) alınır. Bu bakış noktası (z ekseni üzerinde) 'ha belli bir mesafe uzaklıkta 3 boyutlu bir görüntü olduğu var sayılıyor.



* Gönderilen ışın o doğrultuda cisim veya cisimlerle kesişiyorsa, en öndeki cismi görüntüler. En kısa t mesafesini görüntüler.





GD(x,4,7) = (16*x/799 - 8, 4.5 - y*9/449, 10)

3 boyutlu — TKi Boyutlu

* Bu görüntü düzleminde z'ler her yerde 10'dur. Çünkü bakış noktamız bu kadar uzaklıktadır.

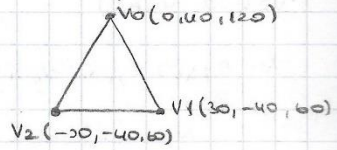
$R = R_o + t R_d$ → orijinden çıktığı yön; t: uzaklık; R_o: orijin

$R_o \rightarrow R_1 \quad (R_1 - R_o)$

Işın–Üçgen Kesişim Testi

Iki aşamadan oluşmaktadır:

- Işın ile üçgenin tanımladığı yüzey arasında kesişim testi.
- Işın yüzey ile kesişiyorsa, kesişim noktasının üçgenin içinde olup olmadığını belirle.



I. Aşama: $N(0, 0.6, -0.8)$ $\quad Ax + By + Cz + D = 0$ (Düzlem)

X Y Z

$0.(30) + (0.6).(-40) + (-0.8).60 + D = 0$

$D = 72$

$A(R_{ox} + tR_{dx}) + B(R_{oy} + tR_{dy}) + C(R_{oz} + tR_{dz}) + D = 0$

$R_x \quad R_y \quad R_z$

$$t = -\frac{AR_{ox} + BR_{oy} + CR_{oz} + D}{AR_{dx} + BR_{dy} + CR_{dz}} = -\frac{N * R_o + D}{N * R_d} \qquad D = -(N * V_0)$$

$t > 0$ ise ışın yüzey ile kesişiyor.
$t < 0$ " " " " kesişmiyor.
$t = \infty$ " yani $N * R_d = 0$ ise ışın yüzeye paralel.

($N * R_d = 0$ ⊥ yolladığımız ışın normale dik ise yüzeye paralel.)

* $R_o = (0,0,0)$ ve $R_d = (0,0,1)$ ise $t = ?$

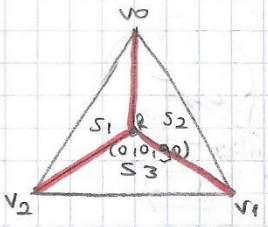
$$t = -\frac{(0, 0.6, -0.8) * (0,0,0) + 72}{(0, 0.6, -0.8) . (0,0,1)} = \frac{72}{-0.8} = 90$$

$R = R_o + tR_d = (0, 0, 90)$ — Noktanın koordinatları

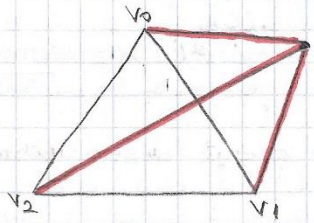
II. aşama

$R = R_o + t R_d$ 'den R bulunur.

$R = (R_{ox}, R_{oy}, R_{oz}) + t(R_{dx}, R_{dy}, R_{dz})$



$S = S_1 + S_2 + S_3$ } Bulunan R koordinatların içindeyse.



$S < S_1 + S_2 + S_3$ } Bulunan R koordinatların dışındaysa

$S_1 = |(R - V_0) \times (V_2 - V_0)|$

$S_2 = |(V_1 - V_0) \times (R - V_0)|$

$S_3 = |(V_1 - R) \times (V_2 - R)|$ } $S = |(V_1 - V_0) \times (V_2 - V_0)|$

26.02.2016

```
If(tDistances.size() > 0)
{
    float min_distances = FLT_MAX;
    Int   min_indis   = -1;

    for (int i=0; i<tDistances.size(); i++)
     if (tDistances.at(i) < min_distances)
     {
         min_indis = tIndices[i];
         min_distances = tDistances.at(i);
     }
    return Shapes [min_indis] -> ShapeColor;
}
```

	tDistances		tIndices
T=0	40	→	0
T=1	10	→	2
T=2	20		3

10 ? 40 ✓

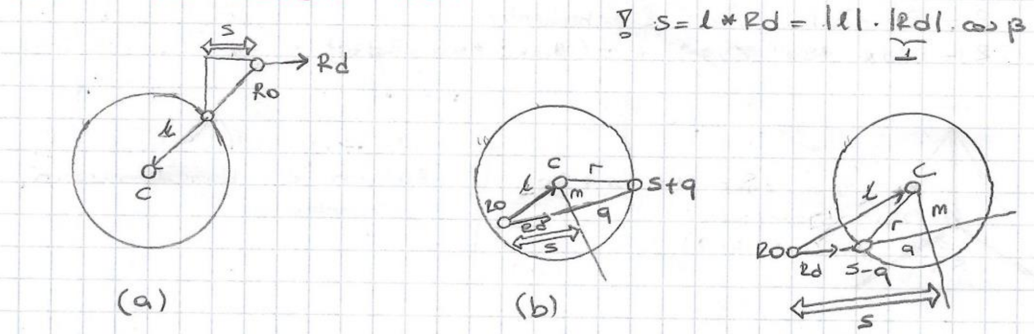
20 ? 10 ✗

min_indis = ~~0~~ 2

min_distances = ~~40~~ 10

Işın - Küre Kesişim Testi

$\nabla \; s = l * Rd = |l| \cdot |Rd| \cos\beta$ (with $|Rd| = 1$)



(c)

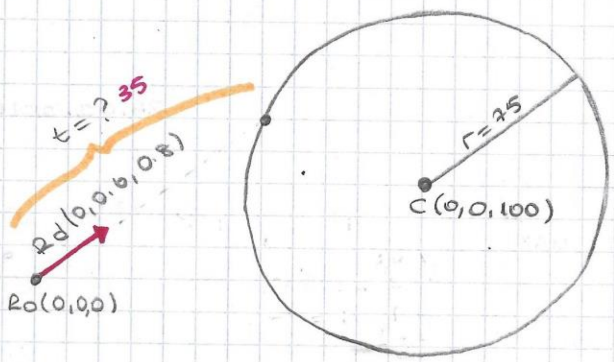
→ Eğer kesişim varsa
$m2 < r2$ 'dir.
$l*l = l2 < r2$ (içten)
$t = s + q \quad (q = \sqrt{r^2 - m^2})$

→ $l*l = l2 > r2$ (dıştan)
$t = s - q \quad (q = \sqrt{r^2 - m^2})$

→ küreyi içerden kesiyorsa $t = s+q$ olur. b'de ışın küreyi içten keser.

→ küreyi dıştan kesiyorsa $t = s-q$ olur. c'de ışın küreyi dıştan keser.

→ $l^2 = m^2 + s^2$



$l = \underset{C}{(0,0,100)} - \underset{Ro}{(0,0,0)} = (0,0,100)$

$s = \underset{l}{(0,0,100)} * \underset{Rd}{(0,0.6,0.8)} = 80$

$l2 = (0,0,100) * (0,0,100) = 10.000 \quad (l^2)$

$r2 = 75 \times 75 = 5625 \quad (r^2)$

$s2 = 80 \times 80 = 6400 \quad (s^2)$

$m2 = \underset{l^2}{10.000} - \underset{s^2}{6400} = 3600 \quad (m^2)$

$q = \sqrt{\underset{r^2}{5625} - \underset{m^2}{3600}} = \sqrt{2025} = 45$

$t = s - q = 80 - 45 = 35$

$R = Ro + t \cdot Rd = (0,0,0) + 35 \cdot (0, 0.6, 0.8)$

$R = (0, 21, 28)$

Phong Boyama Modeli

pixel Color = ka . ambientColor + kd . diffuseColor + ks . specular Color

(R,G,B) (R,G,B) (R,G,B)

0-255 0-255 0-255

kd * cismin rengi ks * ışık kaynak rengi

(255, 0, 0)
(200, 0, 0)
↓
(0, 0, 0) → Siyah
(255, 255, 255) → Beyaz.

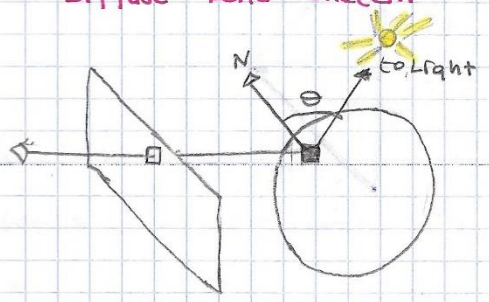
* Phong boyama modeli ışık kaynağına bağlı olarak ambient, diffuse ve specular olmak üzere 3 çeşittir.

$$ka + kd + ks = 1$$

* Ambient için hesaplama yapmamıza gerek yoktur. 0-255 arası sayılar olarak renkler üretir. Diğer ikisi için hesaplama yapılmaktadır.

! (255, 0, 0) → Kırmızı , (0, 255, 0) → Yeşil , (0, 0, 255) → Mavi.
R G B

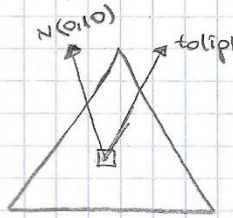
- Diffuse Renk Bileşeni



diffuse katsayı = $\cos\theta$ = N * toLight

$N * toLight = |N| \cdot |toLight| \cdot \cos\theta$
$= \cos\theta$

* diffuse katsayısı 0 – 1 arasındadır.



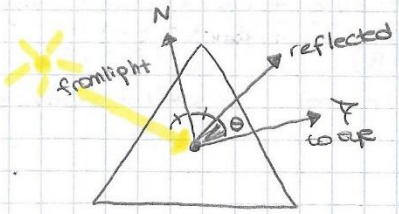
N * toLight = 0.8

diffuse Color = 0.8 . (255, 0, 0)
= (204, 0, 0) → kırmızı

* Bakış noktasının bir önemi yok.

* Işık kaynağı yüzeye tam dik vuruyorsa yani yüzey normali ile aralarındaki açı 0° ise $\cos 0 = 1$ olduğundan max. aydınlatma, açı 90° ise $\cos 90 = 0$ olduğundan minimum aydınlatma söz konusudur.

- Specular Renk Bileşeni



* Bakış noktasının konumu önemlidir.

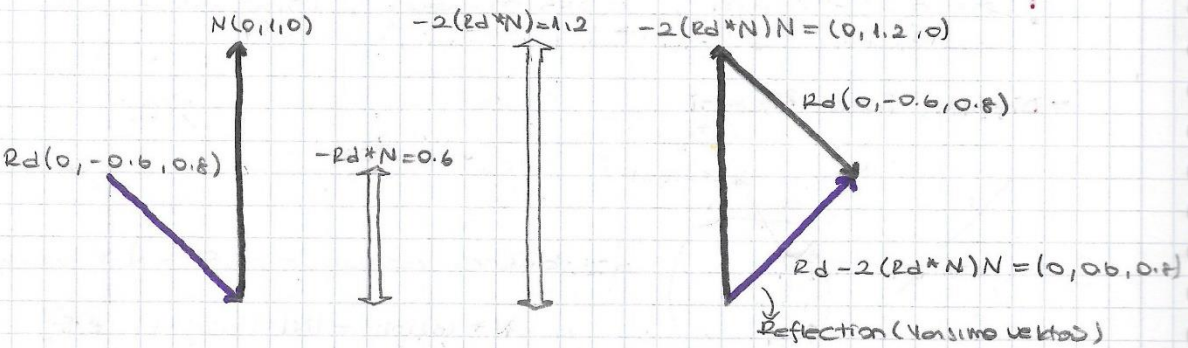
$\cos\theta = (reflected * toEye)^{shininess}$

$[0..1]$

- Ambient Renk Bileşeni

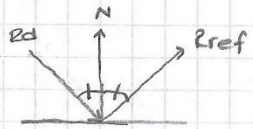
* Cismin rengi 0-1 arası bir katsayı ile direkt çalışır. Sebebi gerçekte diyelim ki bir masa altına ışık kaynağından direkt foton almıyor. Fakat başka eşyalardan yansıyan ışıklardan dolayı direkt siyah görmüyoruz aydınlatılmış oluyor. Fakat Ray Tracing bu durumu modelleyemiyor. Bunun için cismin kendi rengi bir kat sayı ile çarpılıp yolacı bir renk oluşturuyor direkt siyah olmasın diye.

Yansıma Vektörü

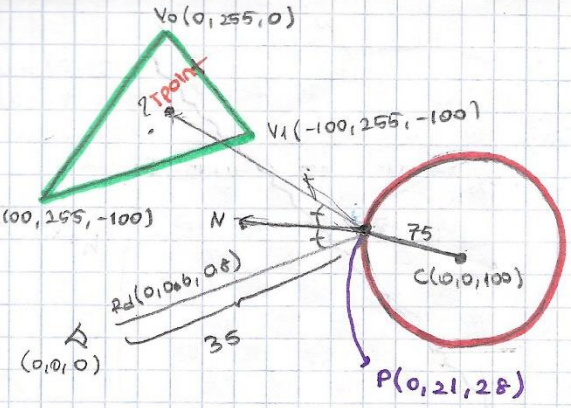


$$R_{ref} = Rd - 2(Rd * N).N$$

*



*



tdistance = 35

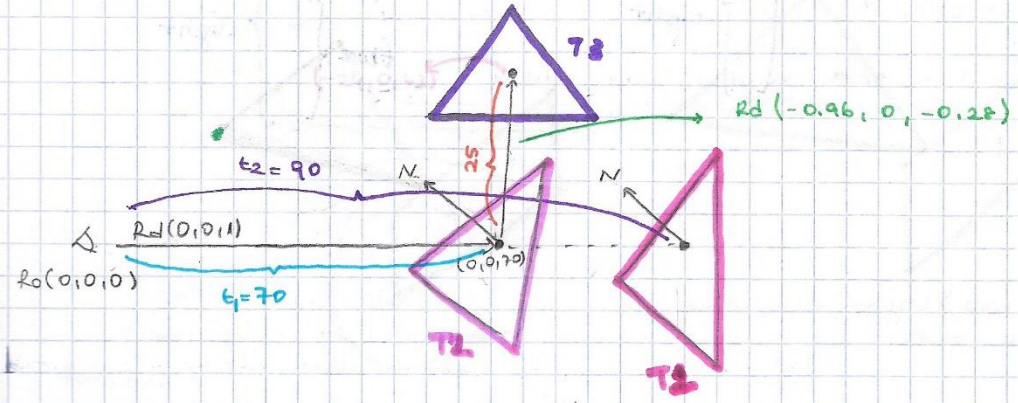
- $P = (0,0,0) + 35(0, 0.6, 0.8) = (0, 21, 28)$ → Intersection Point
- $N_{sphere} = \frac{(0,21,28) - (0,0,100)}{75} = \frac{(0,21,-72)}{75} = (0, 0.28, -0.96)$
- $Reflected_d = R_d - 2(R_d * N)N$

$R_d * N = (0, 0.6, 0.8) * (0, 0.28, -0.96)$
$= 0.168 - 0.768 = -0.6$

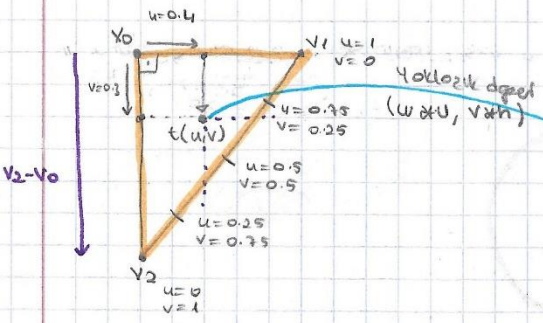
$-2(R_d * N)N = (-2 \cdot -0.6)(0, 0.28, -0.96) = (0, 0.336, -1.152)$

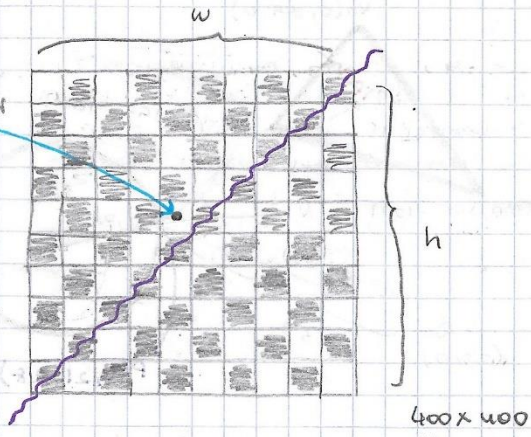
- $R_d * N - 2(R_d * N)N = (0, 0.6, 0.8) + (0, 0.336, -1.152)$
$= (0, 0.936, -0.352)$
- $R_{o\text{-}new} = (0, 21, 28)$
- $R_{d\text{-}new} = (0, 0.936, -0.352)$
- Üçgenin $N = (0, -20000, 0)$ ⟶ Normalize $= (0, -1, 0)$ → R1xR2 formülü
- $t = \frac{N * R_o + D}{N * R_d} = -\frac{-21 + 255}{-0.936} = \frac{234}{0.936} = 250$
- $iPoint = (0,21,28) + 250(0, 0.936, -0.352)$
$= (0, 21, 28) + (0, 234, -88)$
$= (0, 255, -60)$

*



T3'ün OrtoNoktası $= (0,0,70) + 25(-0.96, 0, -0.28) = (-24, 0, 63)$

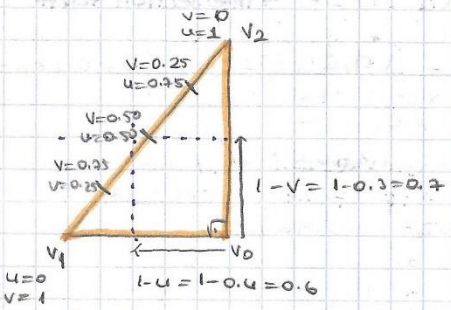




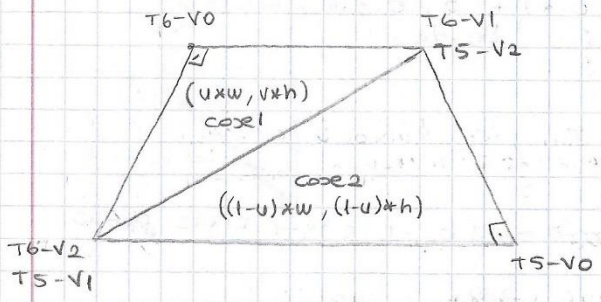
400 x 400
w h
width high

$$t(u,v) = V_0 + u(V_1 - V_0) + v(V_2 - V_0)$$

$u \geq 0$, $v \geq 0$, $u + v \leq 1$



* 3 boyutlu bir obje hali için x ve z, bayrak için x, y, tablo için y, z kullanılır.

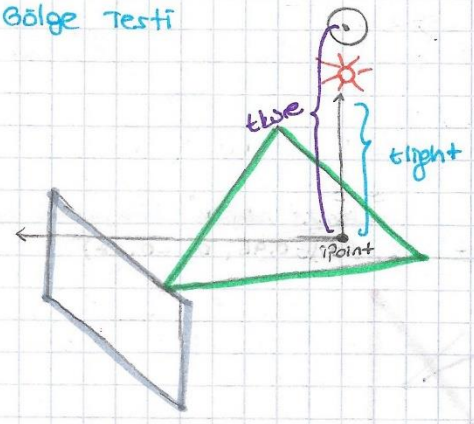


$$u = \frac{TPoint.x - V_0.x}{V_1.x - V_0.x}$$

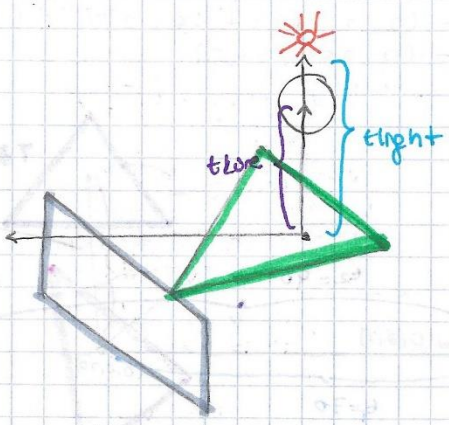
$$v = \frac{TPoint.y - V_0.y}{V_2.y - V_0.y}$$

11.03.2016

Bölge Testi

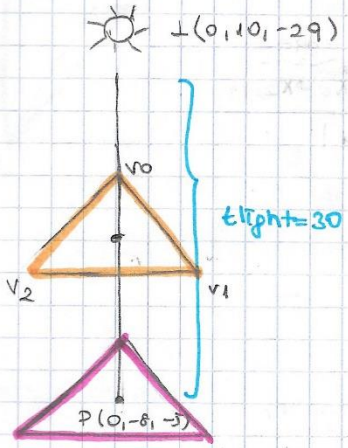


tkore > tlight oldupunda gölge olmaz.



tkore < tlight oldupunda gölge olur.

★ 2012 SINAV S. $L = (0, 10, -29)$ noktasında bir ışık kaynağı olduğu varsayıldığında $P = (0, -8, -5)$ noktasının 2. sorudaki V_0, V_1, V_2 üçgeninin gölgesinde kalıp kalmadığını belirleyiniz.



$$t_{light} = L - P$$
$$= (0, 10, -29) - (0, -8, -5)$$
$$= (0, 18, -24) \longrightarrow 30 \text{ ile bölüp normalize olur}$$
$$= (0, 0.6, -0.8) \checkmark$$
$$|t_{light}| = 30$$

$$t = -\frac{N * R_0 + D}{N * R_d}$$

$$D = -(V_0 * Normal)$$
$$= -(0, 20, -34) * (0, -0.6, 0.8)$$
$$= 39.2$$

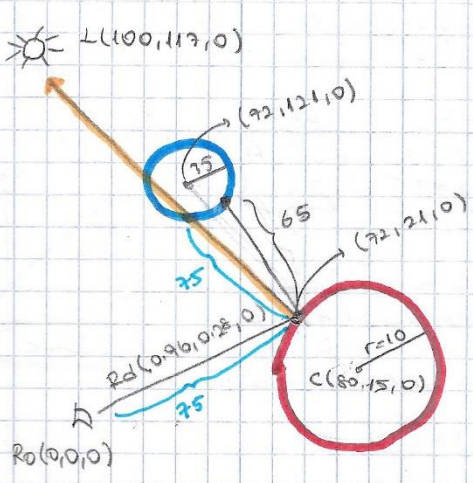
$$t = -\frac{(0, -0.6, 0.8) * (0, -8, -5) + 39.2}{(0, -0.6, 0.8) * (0, 0.6, -0.8)} = 40$$

$t_{light} < t$ olduğu için gölgede kalmaz.

★ 2013 SINAV S. $R_0 = (0, 0, 0)$ başlangıç noktasından, $R_d = (0.96, 0.28, 0)$ doğrultusu boyunca giden bir ışın, $C_x = (80, 15, 0)$ merkez koordinatlarına ve $r_K = 10$ birim yarıçapa sahip kırmızı renkli küreden yansıyıp $C_M = (72, 121, 0)$ merkez koordinatlarına ve $r_M = 35$ birim yarıçapa sahip mavi renkli küre ile kesişiyor.

a) Mavi küre üzerindeki kesişim noktasını bulunuz.

b) $L = (100, 117, 0)$ noktasında bir ışık kaynağı olduğu varsayıldığında kırmızı küre üzerindeki kesişim noktasının mavi küreden gölgesinde kalıp kalmadığını bulunuz.



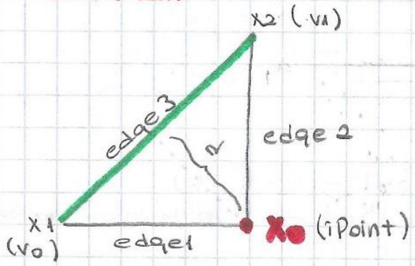
$$t_{light} = (100, 117, 0) - (72, 121, 0)$$
$$= (28, 96, 0)$$
$$= (0.28, 0.96, 0)$$
$$|t_{light}| = 100$$

$t < t_{light}$ olduğu için gölgede kalır.

18.03.2016

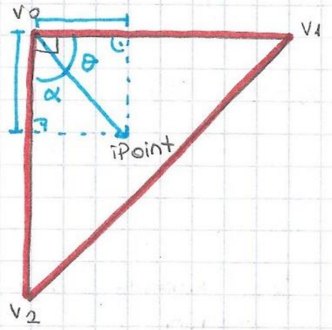
Eğer yüzeye 3 boyutlu olarak farklı şekillerde cisim yerleştirmek istersek;

I. Yöntem



$$d = \frac{\left|\overbrace{(x_0-x_1)}^{edge1} \times \overbrace{(x_0-x_2)}^{edge2}\right|}{|x_2-x_1|}$$

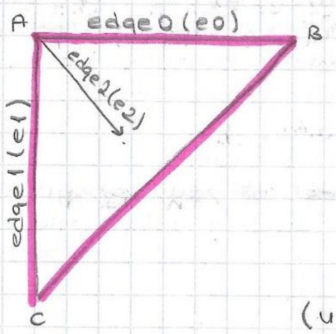
II. Yöntem



$$\left.\frac{\overbrace{(iPoint - v_0)}^{(v_0 - iPoint)} * \overbrace{(v_1 - v_0)}^{(v_0 - v_1)}.Normalize()}{(v_1 - v_0).length()}\right\} v_1 - iPoint$$

$$\left.\frac{(iPoint - v_0) * (v_2 - v_0).Normalize()}{(v_1 - v_0).length()}\right\} v_2 - iPoint$$

III. Yöntem



$$P = A + u(B-A) + v(C-A)$$

$$u\underbrace{(B-A)}_{e0} + v\underbrace{(C-A)}_{e1} = \underbrace{P-A}_{e2}$$

$$u*e0 + v*e1 = e2$$

$$(u*e0 + v*e1)*e0 = e2*e0$$
$$(u*e0 + v*e1)*e1 = e2*e1$$

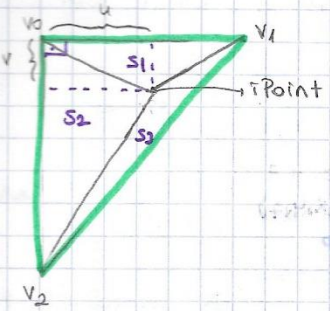
$$d11/ \quad u\overbrace{(e0*e0)}^{d00} + v\overbrace{(e1*e0)}^{d10} = \overbrace{e2*e0}^{d20}$$
$$d10/ \quad u\underbrace{(e0*e1)}_{d10} + v\underbrace{(e1*e1)}_{d11} = \underbrace{e2*e1}_{d21}$$

$$u*d00*d11 + v*d10*d11 = d20*d11$$
$$u*d01*d10 + v*d11*d10 = d21*d10$$

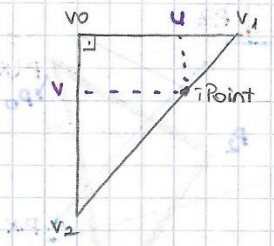
$$u(d00*d11 - d01*d10) = d20*d11 - d21*d10$$

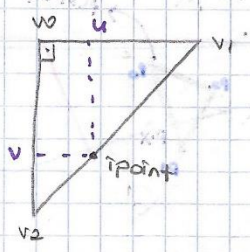
$$u = \frac{d20*d11 - d21*d10}{d00*d11 - d01*d10}$$

IV. Yöntem



$$v = \frac{S1}{S} \qquad u = \frac{S2}{S}$$



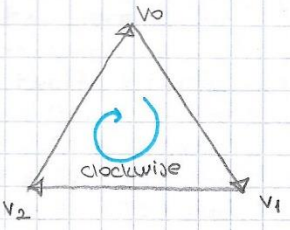


* bu iPoint için v'yi * bu iPoint için u'yu

Hızlandırma Yöntemleri (Acceleration Methods)

1. Backface Culling
2. Axis Aligned Bounding Box (AABB)

1. Backface Culling



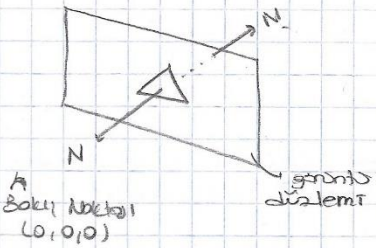
Eğer köşe noktaları saat yönünde tanımlanıyorsa, normali o köselere diktir.

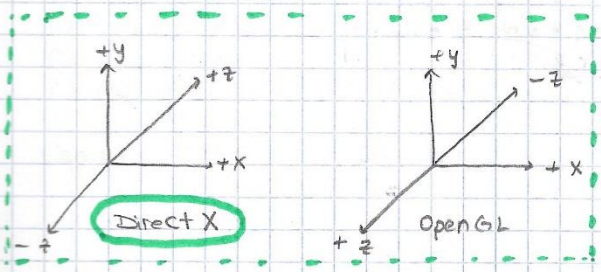
→ Normali baktığımız yönde değilse backface. Bütün üçgenlerin iki yüzü vardır. Bu üçgenlerin, normalinin bize bakanlarını RayTracing ile çizebiliriz. (Render edebiliriz.)

→ 90°'yi geçince negatif olur.

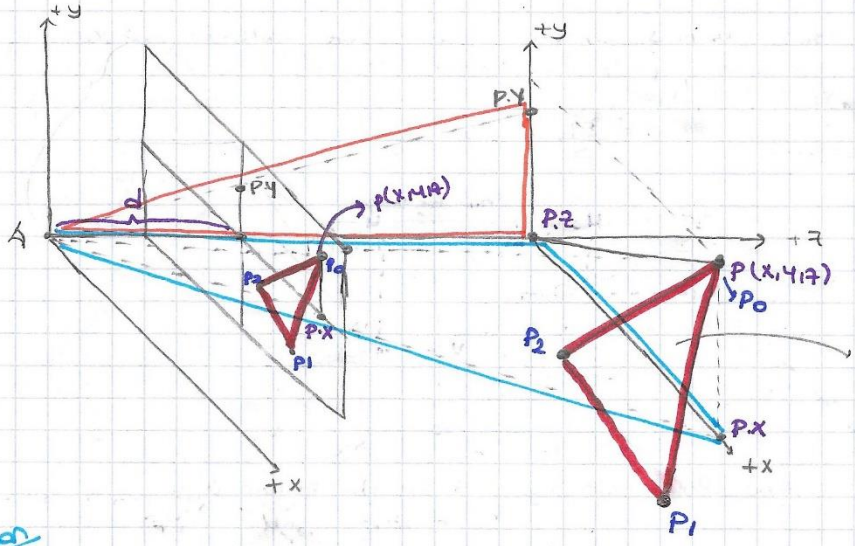
→ Normali baktığımız yönde ise frontface'dir.

→ Üçgenin noktaları saat yönünde yerleştirilmişse ve biz noktaların yerine değiştirirsek backface olur yani hiçbir şey göremeyiz.





* z bileşeni negatifse frontface, z bileşeni pozitifse backface.



Taban

$$\frac{d}{P.z} = \frac{P.x}{P.X}$$

$$P.x = \frac{d * P.X}{P.z}$$

Yan

$$\frac{P.y}{P.Y} = \frac{d}{P.z}$$

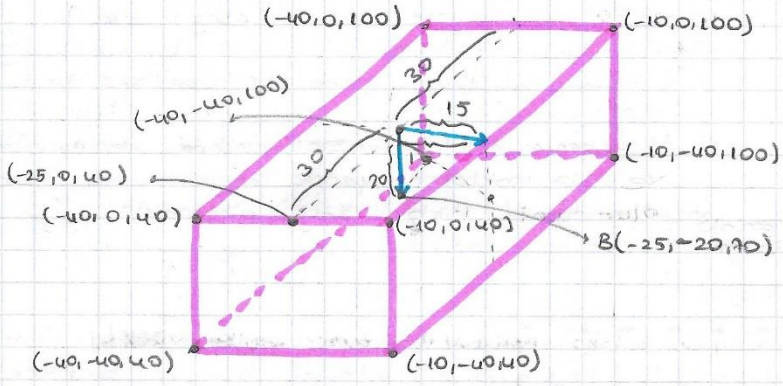
$$P.y = \frac{d * P.Y}{P.z}$$

25.03.2016

2. Axis Aligned Bounding Box

float AABB(Vertex R0, Vertex Rd, Vertex B, float u, float v, float w);

- R0: gelen ışının başlangıç noktası
- Rd: gelen ışının doğrultusu
- B: dikdörtgenin metresindeki vektör

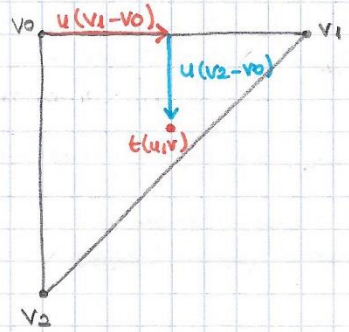


AABB(-25, -20, 70, 15, 20, 30)

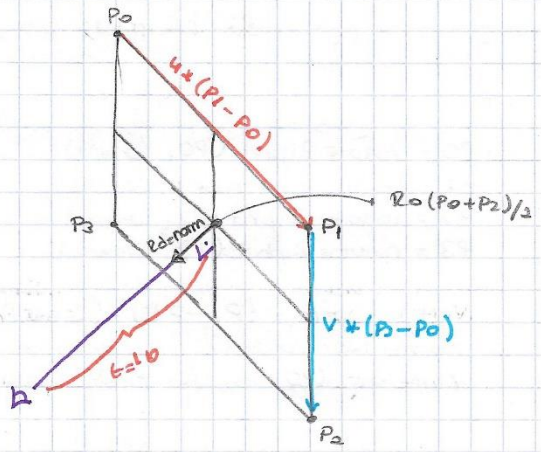
u v w

* Program kodunda; W A S D , Z X : oda içinde hareket sağlar.
(yukarı, aşağı, sola, sağa, z, yukarıdan bakış, x, o>aşağıdan bakış)

**



$t(u,v) = V_0 + u(V_1 - V_0) + v(V_2 - V_0)$
(piksel koordinatlarını hesaplarken)



W

```
Vertex norm = (P1 - P0) . Cross Product (P3 - P0);
norm . Normalize();

P0 = P0 - 10 * norm;
P1 = P1 - 10 * norm;
P2 = P2 - 10 * norm;
P3 = P3 - 10 * norm;

Camera = (P0 + P2) / 2 + 10 * norm;
```

-10 olma sebebi normalin ters yönünde hareket etmektir. 10 da 10 birim değişmiş anlamına gelir.

S

```
Vertex norm = (P1 - P0) . Cross Product (P3 - P0);
norm . Normalize();

P0 = P0 + 10 * norm;
P1 = P1 + 10 * norm;
P2 = P2 + 10 * norm;
P3 = P3 + 10 * norm;

Camera = (P0 + P2) / 2 + 10 * norm;
```

+10 çünkü normal ile aynı yönde gidilir.

a

```
P0 = rotateLeft (P0, camera);
P1 = rotateLeft (P1, Camera);
P2 = rotateLeft (P2, Camera);
P3 = rotateLeft (P3, Camera);

Vertex norm = (P1-P0) . (CrossProduct (P3-P0));
norm. Normalize ();
Camera = (P0+P2) / 2 +10* norm;
```

RotateLeft saat yönünün tersine 15° dönme yapar (sola)

d

```
P0 = rotate Right (P0, Camera);
P1 = rotate Right (P1, Camera);
P2 = rotate Right (P2, Camera);
P3 = rotate Right (P3, Camera);

Vertex norm = (P1-P0) . (CrossProduct (P3-P0));
norm. Normalize ();
Camera = (P0+P2) / 2 +10* norm;
```

RotateRight saat yönünde 15° dönme yapar. (sağa)

```
Vertex rotateLeft (Vertex P, Vertex camera)
{
    P = P-camera;   3 boyutlu ortam başka nokta etrafında döner
    float tmpX = P.X;

    P.X = P.X * 0.966F - P.Z * 0.259F;
    P.Z = tmpX * 0.259F + P.Z * 0.966F;

    P = P+ camera;
    return P;
}
```

$$\begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}$$

```
Vertex rotateRight (Vertex P, Vertex camera)
{
    P = P-camera;
    float tmpX = P.X;

    P.X = P.X * 0.966F + P.Z * 0.259F;
    P.Z = tmpX * 0.259F - P.Z * 0.966F;

    P= P+ camera;
    return P;
}
```

$$\begin{bmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{bmatrix}$$

* Odanın içini bakış noktası etrafında döndürmek istiyoruz. Eğer p-camera yapınca cameraya orijine taşıyoruz. Rotasyon matrisiyle çarptık. Daha sonra eski haline getirdik. Yani 3 boyutlu ortamın başka bir nokta etrafında dönmesini istiyorsak p-camera yapmalıyız. P'yi direkt rotasyon matrisiyle çarparsak 3 boyutlu ortam orijin etrafında döner.

DIRECTX 11

* Hızlı grafik uygulamaları yazabilmek için geliştirilmiş donanım destekli yazılımlardan biridir.

* Basit bir windows uygulaması için 3 tane fonksiyon kullanılır;

  - InitWindow()
  - WinMain() → mesaj döngüsü var.
  - WndProc() → CallBack fonksiyon. Çağrıyo çünkü yok olay gerçekleşince kendi kendine tetikleniyor.

* WinMain() fonksiyonu içinde kullanılan 2 fonksiyon vardır;

  - InitDevice() → Çizim için ayarlamaları yapan. (initialization işlemleri)
  - Render() → Çizim fonk.

InitDevice() içerisinde 3 tane nesne oluşturuluyor.

- Device → Buffer oluşturur.
- Immediate Context → Çizim işlemlerinden sorumlu. } Device (DIRECTX 10)
- SwapChain → Çizim yapılan bellek alanını monitöre yollayan nesne.

2 tane buffer vardır: backbuffer, frontbuffer. Ekranda görüntülenen görüntü frontbuffer'dır. Görüntülenecek nesnenin başka bir durumu backbuffer'a yazılır sonraki görüntü backbuffer'dan görüntülenir. Bu durumda swapChain çalışmış olur. (Görüntülenen frontbuffer, sonraki görüntü yazılan backbuffer.)

* D3D11CreateDeviceAndSwapChain() fonksiyonu ile 3 tane nesne oluşturulur.

15.04.2016

* VIEWPORT(): Üreteceğin grafik pencerenin tamamına mı çizecek bir kısmına mı çizecek bunun için kullanılır. (Arabo yarışı oyuyan iki oyuncu için 2 pencereye ihtiyaç vardır.)

Player I ← 1. oyuncunun tuşlarına göre güncellenir.
Player II ← 2. oyuncunun tuşlarına göre güncellenir.

* void Render()
{
  float ClearColor[4] = {0.0f, 0.125f, 0.6f, 1.0f}; ← α bileşeni (saydamlık)

  g_pd3dDevice → ClearRenderTargetView(g_pRenderTargetView, ClearColor);

  g_pSwapChain → present(0.0);
}

// ClearColor ile backbuffer'ı o renge boyar.
// Device kullanılması yerine çizim fonk. olan immediate content kullanılmalı.

Üçgen Çizimi: İki tane dosya vardır. Bunlar;
- .cpp uzantılı dosya. CPU'da koşulur.
- .fx(.hlsl) uzantılı dosya.

* SimpleVertex(): Tanımlanan köşe noktası ekran kartına bu yolla gönderilir.

```
struct SimpleVertex
{
    XMFLOAT3 Pos; // Köşe noktası için sadece position tut-
};                // mayacağız. Color, normal vs. tutulacak.
```

* Birden fazla vertexlayout tanımlamak için vertex'lere ihtiyaç etmeli.

IASetInputLayout(): Hangi buffer'ı kullanacağını belirliyorsun ve onun için işlem yapılıyor.

* VertexBuffer(); Üçgenlerin köşe noktasını tanımladığımız bufferdır. XMFLOAT 3 tanedir. Çünkü üçgenin 3 köşesi vardır.

```
SimpleVertex Vertices[]
{
    XMFLOAT3(0.5f, 0.5f, 0.5f);
    XMFLOAT3(0.5f, 0.5f, 05f);
    XMFLOAT3(0.5f, 0.5f, 0.5f);
};
```

Buffer'ın boyutu = sizeof(SimpleVertex) * 3;

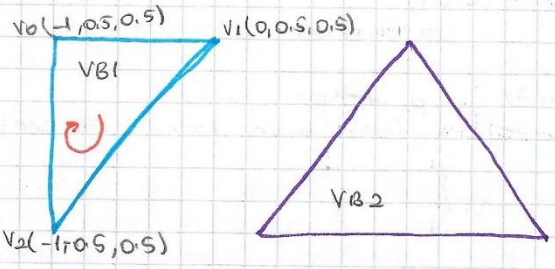
Primitive Topology:

* TRIANGLESTRIP: ABCD → 2 tane üçgen çizer bu fonksiyon XMFLOAT'dan 4 tane varsa 2 tane üçgen çizer.

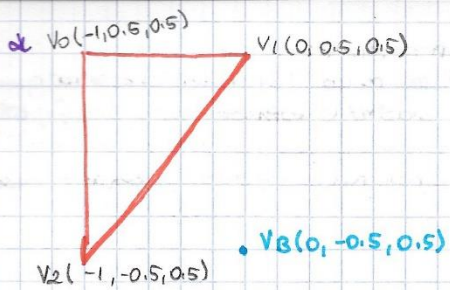
* VertexShader (VS): Köşe noktası üzerinde koşuyor. World, View, Projection matrisleriyle çarpılacak. Üçgenlerin ekrana izdüşümünü alıyor.
  - World: Rotation, Translation, Scaling
  - View: Gözlemci nasıl görüyor.
  - Projection: Ekrana izdüşüm.

PS: İki boyutlu üçgenin rengini boyuyor. Renk döndürüyor. Bakış noktası yüzey normalini hiç dikkate almıyor. Her bir noktayı aynı renge boyuyor.

* V0(-1,0.5,0.5) V1(0,0.5,0.5)



Eğer V1 ve V2'nin yerleri değişirse backface olarak yorumlar ve ekrana çizmez.



- Iki tane buffer tanımlarda ... 4 nokta tanımlı ..... 6 nokta tanımlı. Aynı şekli çizer çünkü primitive Topology-leri aynı.

TRIANGELSTRIP, yani XMFLOAT 4 tane nokta olacak bu 4 noktadan 2 tane üçgen çizecek. (ABCD için ABC bir üçgen, BCD diğer üçgen) TRIANGLELIST'de kare çizmek için 6 tane noktaya ihtiyaç vardır.

Kareler in renklerin farklı olmasını istiyorsak yeni bir PS yazarak rengini farklı yapabiliriz.

22.04.2016

Object Space

3 boyutlu sistemde çizdiğimiz üçgenlerin, şekillerin çizildiği ortamdaki koordinatlara denir. Vertex buffer'da ilk setlenen koordinatlardır. Object space'e ait bir matris yoktur.

World Space

DirectX'deki uzaya denir. World Space'e ait bir matris vardır: World Matrix. Bu uzayda döndürme gibi işlemler yapabilmek için bu matris kullanılır. Scaling, rotation ve translation işlemleri için birer matris vardır. Bunlar çarpılarak şekillere farklı transformasyonlar yapılır.

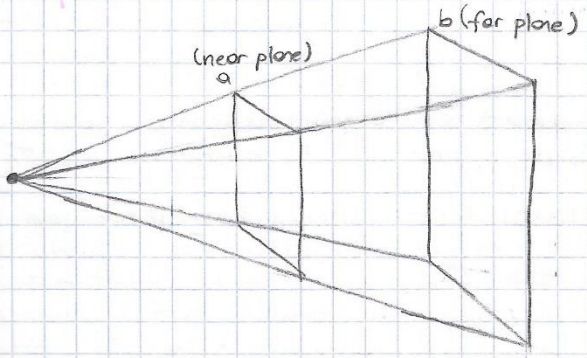
View Space

3 boyutlu ortamı nasıl gördüğümüzle ilgilenir. View space'e ait bir matris vardır: View Matris. View matris 3 alt vektörden oluşuyor. Bunlar: Eye, At, Up. Eye vektörü bakış noktamız, At vektörü hangi doğrultu boyunca baktığımızı gösterir, Up vektörü kafa hareketine göre belirlenir.

```
XMVECTOR Eye = XMVectorSet(0.0f, 3.0f, -10.0f, 0.0f);
XMVECTOR At  = XMVectorSet(0.0f, 0.0f, 1.0f, 0.0f);
XMVECTOR Up  = XMVectorSet(0.0f, 1.0f, 0.0f, 0.0f);
```

Ortamda gezinmiyorsak view matrisini sabit tanımlar değiştirmeyiz. Fakat ortamda geziniyorsak her bir Render'da view matrisi değişir.

Projection Space



Projection View'e ait matris vardır: Projection Matrix.

Matrix PerspectiveFovLH(XM_PIDIV4, width/(FLOAT) height, 0.01f, 100.0f)

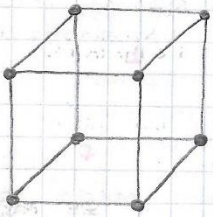
LH: Left Handed
Fov Angle: Field Of View
PIDIV4 → 45° (Pi'yi 4'e böl)

Küp Çizimi

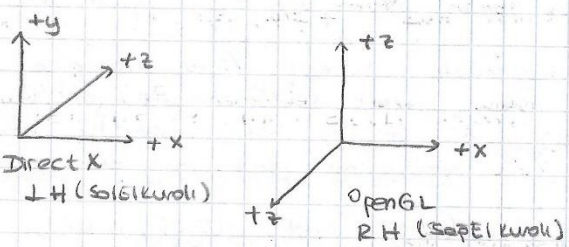
* Küp çizebilmek için 36 tane nokta bellekte tutulur. Bellek gereksiz yere kullanılmaktadır. (Vertex Buffer tek kullanıldığında)

* Index ve Vertex birlikte kullanıldığında köşe noktaları vertex buffer'da bir kez tanımlanıyor. Index buffer, vertex buffer'la birlikte kullanılır.

* Vertex buffer'da köşe noktaları tanımlandıktan sonra, index buffer tanımlanır.



* Index buffer köşe noktaların 3'erli şekilde grupluyor. Index buffer gruplamada sol el kuralını kullanıyor.




```
WORD Indices[]
{
    3,1,0,  } Her bir üçgeni temsil ederler.
    2,1,3,
    0,5,4,
    1,5,0,
    3,4,7,
    0,4,3,
    1,6,5,
    2,6,1,
    2,7,6,
    3,7,2,
    6,4,5,
    7,4,6,
};
```

* Herhangi bir üçgeni backface yapmak için, 3,1,0 yerine 3,0,1 kullanılır.

* ID3D11 Buffer türünden hem Index hem Vertex.

```
g-pImmediateContext → VSSetShader(g-pVertexShader, NULL, 0);
g-pImmediateContext → VSSetConstantBuffers(0, 1, &g-pConstantBuffer);
g-pImmediateContext → PSSetShader(g-pPixelShader, NULL, 0);
g-pImmediateContext → DrawIndexed(36, 0, 0);
                                  └ index buffer vars
```

Rotation

```
gWorld = XMMatrixRotation(t);
                          └ birim matris
```

Programdaki 2 küpte aynı vertex ve index buffer'ı kullanıyor. Fakat ikisini de görebiliyoruz. Görebilme sebebimiz SwapChain'dir. İki küpte çizilir. Sonra sonra çalışır.

```
g-pSwapChain → Present(0,0); → Back buffer içeriğini ekrana yansıtır.

XMMATRIX mRotate = XMMatrixRotation(-t * 2.0f);
XMMATRIX mTranslate = XMMatrixTranslation(4.0f, 0.0f, 0.0f);
XMMATRIX mScale = XMMatrixScaling(0.3f, 1.0f, 0.3f);
                                  └ 4 ekseninde bir scaling olmaz.
} küçük küp çizimi

gWorld = mScale * mTranslate * mRotate;
         ↓ İlk uygulanan      ↓ son uygulanan
```

gWorld'ü bulurken önce ilk uygulanandan sona doğru çarpılır.

- Translate, Rotate'ten önce yapılırsa belli bir yarıçap etrafında döner.
- Rotate, Translate'den önce yapılırsa küp kendi etrafında döner
- Translate → Scale → Rotate sırada olursa scale işleminde cismi de cismin orijine olan uzaklığını da küçültür. (4*0.3=1.2 oluyor orijine uzaklık)
- Translate → Rotate → Scale sırada olursa yukarıdaki ile aynı şey olur.
- Rotate → Scale → Translate sırada olursa 2. ile aynı olur.
- Rotate → Translate → Scale sırada olursa kendi ekseni etrafında döner. Scale sayesinde yarıçap oluşur.

29.04.2016

Lighting

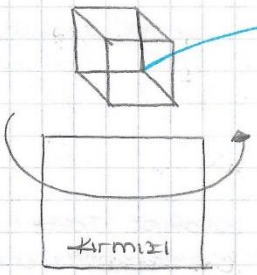
Pixel Shader içeriği şekillenir. Vertex Shader renk hesabı yapmıyor.

```
struct ConstantBuffer
{
    XMMATRIX    mWorld;
    XMMATRIX    mView;
    XMMATRIX    mProjection;
    XMFLOAT4    vLightPos[2];
    XMFLOAT4    vLightColor[2];
    XMFLOAT4    vMeshColor;
    XMFLOAT4    vEyePos;
} cBuffer
```

} 2 tane ışık kaynağını küp olarak çiziyor. (sarı ve beyaz)

Pixel Shader'da vLightPosition[2] tutuluyor. Pixel Shader'a da Constant Buffer'dan gönderilir. Constant Buffer'da ışık kaynakları için 2 bilgi tutulur. Bunlar; konumu ve rengi. Pixel Shader tarafında ambient ve specular bileşenler hesaplanır.

Işık kaynakları 2 şekilde teslim edilir
- Işık kaynaklarını temsil eden küpde tanımlamak lazım.
- Arkaplanda diffuse ve specular bileşenler için gerekli olan küpün merkez koordinatları gereklidir.



→ Küpün merkezi için vLightPos[2] değişkeni tutulacak. vLightPos[2] hem diffuse hem de specular bileşende gereklidir.

Işık kaynakları hareketli olduğu için Vektörü transformasyona tabi tutmak lazım. Bulunduğu veya bulunmadığı yerleri değişik anda aydınlatması için.

vLightPos[2]; ⟶ transformasyona uğrayan dizi

```
{
  XMFLOAT4 (0.0f, 0.0f, 0.0f, 0.1f)
  XMFLOAT4 (0.0f, 0.0f, 0.0f, 0.1f)
};
```

vLightColor[2]; ⟶ ışık kaynağının rengi

```
{
  XMFLOAT4 (1.0f, 1.0f, 1.0f, 1.0f)
  XMFLOAT4 (1.0f, 1.0f, 0.0f, 1.0f)
};
```

Phong boyama modelini uygulayacağız. Ambient, diffuse ve specular değişkenler;

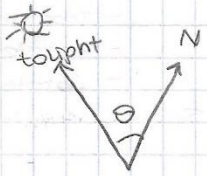
```
for(int i=0; i<2; i++)
{
  //DIFFUSE COLOR
  toLight = normalize(vLightPos[i] - input.Posw);
  float dotEyeNorm = dot(toLight, input.Norm);

  diffuseColor = max(dotEyeNorm * vMeshColor, 0);

  //SPECULAR COLOR
  fromLight = normalize(input.Posw - vLightPos[i]);

  toEye = normalize(vEyePos - input.Posw);

  float3 reflected = fromLight + -2 * dot(fromLight, input.Norm) * input.Norm;
  float dotEyereflected = dot(toEye, reflected);
  specularColor = max(pow(dotEyereflected, 32.0f) * vLightColor[i], 0);
```



↳ fromLight

```
finalColor.rgb += min(0.1F * vMeshColor + 0.4F * diffuseColor + 0.3F * specularColor, 1);

}

finalColor.a = 1;
return finalColor;
```

Işık kaynağı için diffuse ve specular değer hesaplamaya gerek yoktur. Çünkü ışık kaynağının rengi değişmez, belli bir renktir. Yüzeydeki yansımalar ışığın normaline göre farklılık gösterir ve değer hesaplaması yapılmalıdır.

Sadece ambient bileşeni görmek istersek vMeshColor dizisindekilerin katsayılarını 0 yapmalıyız. Tabanın rengi çok fazla görülmez.

Diffuse bileşen dizisindekilerin katsayıları 0 olursa ışık kaynağı hareket ederken dik vurduğu yerler daha parlak görünür.

Specular bileşen dizisindekilerin katsayıları 0 olursa ışık kaynağının yüzey üzerindeki parlamaları görülür. (Taban rengi görülmez.)

* `XMMATRIX mScale = XMMATRIX Scaling(0.3f, 0.3f, 0.3f);`
  `XMMATRIX mTranslate[2];`
  `XMMATRIX mRotate[2];`
  ↳ Transformasyon vardır. Vektörün transformasyonu.

* Vektör matrisle değildir çarpılmaz, transform komutuyla çarpılır.

* `cBuffer world = XMMatrixTranspose(mScale * mTranslate[m] * mRotate[m]);`
  ↑ Işık kaynaklarının Translate ve Rotate'leri farklıdır. Çünkü iki ışık kaynağı farklı yörüngede farklı yönlerde dönüyor.

* `struct simpleVertex_PN`
  `{`
  `XMFLOAT3 Pos;`
  `XMFLOAT4 Normal;`
  `};`
  Zemin için Pos ve Normale ihtiyaç vardır. Çünkü zeminin konumu değiştiğinde normalide değişecektir. Normal değişkeni olmasaydı zeminin konumu değiştiğinde normal değişmeyecekti bu sefer yanlış normal değerine göre zeminde aydınlatma yapılacaktı.

  `struct simpleVertex_P`
  `{`
  `XMFLOAT3 Pos;`
  `};`
  Işık kaynakları tek bir renge sahiptir. Bu yüzden diffuse ve specular bileşenleri hesaplamaya gerek yoktur.

* `g_pVertexLayout_P` dersek küpler çizilir.
  `g_pVertexLayout_PN` dersek zemin çizilir.

* `struct VS_INPUT_PN` ⟶ Zemin için tanımlanan
  `{`
  `float4 Pos : POSITION;` (World matrisi 4x4'lük matris olduğu için)
  `float3 Norm: NORMAL;`
  `};`

  `struct VS_INPUT_P` ⟶ Küpler için tanımlanan
  `{`
  `float4 Pos: POSITION;`
  `}`

```
* struct    PS_INPUT_PN
  {
    float4  POSH : SV_POSITION;
    float3  POSW : POSITION;
    float3  Norm : NORMAL;
  };

  struct    PS_INPUT_P
  {
    float4  POS : SV_POSITION;
  };
```

⇒ Vertex Shader'ın output'u Pixel Shader'ın input'u

* g_p PixelShader Phong → zemini boyarken
  g_p PixelShader Solid → küpleri boyarken.

```
* PS_INPUT_PN  VS_PositionNormal(VS_INPUT_PN input)
  {
    Output.POSH = mul(input.POSH, world);
    Output.POSH = mul(output.POSH, View);
    Output.POSH = mul(Output.POSH, Projection);
    Output.Norm = mul(input.Norm, world);   → normali değişirse
    Output.Norm = Input.Norm;               → normali değişmezse
    Output.POSW = mul(input.POSH, world);
  } return output;
```

⇒ Zemin koordinatları değişmezse world matrisiyle çarpmaya gerek yok.

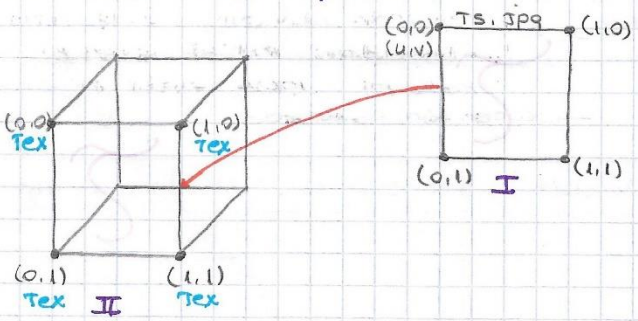
⇒ Zeminin normalinide transform etmeliyiz ki ışık kaynağının aydınlattığı yerlerde değişsin.

Texture Mapping (Doku Kaplama)

Doku kaplama işlemi Pixel Shader'ın içinde yapılır. Doku kaplarken dokudaki renk geçişlerinin daha yumuşak olması için g_pSamplerLinear kullanılır.

```
* struct SimpleVertex
  {
    XMFLOAT3 Pos;
    XMFLOAT2 Tex;   → Float sayılar u ve v parametreleri. Sadece köşe noktaları olan u ve v'yi setlemek yeterli olur.
  };
```

Küpün Üzerine Doku Kaplama

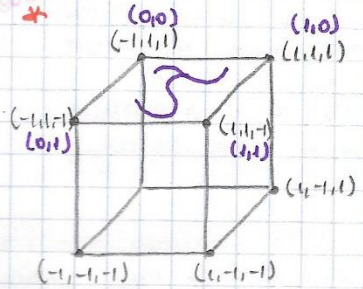


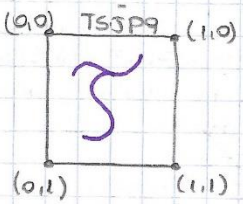
(u-v: 0-1 aralığında)

Ön yüzüne doku kaplayacağız. Resim dosyası üzerine default olarak u ve v değerleri yazılır. Sonra resim küp üzerine kaplanır, köşe noktaları aynen küpün üzerine geçer. Doku kaplama yapılırken yapılacaklar;

1. Resim dosyasındaki u-v değerlerini belirle.
2. Küpün üzerinde nasıl durur. Buna bak

```
SimpleVertex vertices[] =
{
    //FRONT (z=-1)
    { XMFLOAT3(1.0f, 1.0f, -1.0f), XMFLOAT2(1.0f, 0.0f) };
    { XMFLOAT3(1.0f, -1.0f, -1.0f), XMFLOAT2(1.0f, 1.0f) };
    { XMFLOAT3(-1.0f, -1.0f, -1.0f), XMFLOAT2(0.0f, 1.0f) };
    { XMFLOAT3(-1.0f, 1.0f, -1.0f), XMFLOAT2(0.0f, 0.0f) };
};
```





Küpün üst yüzeyini kaplayan XMFLOAT2'leri doldurun.

```
SimpleVertex vertices[] =
{
    { XMFLOAT3(-1.0f, 1.0f, -1.0f), XMFLOAT2(0.0f, 1.0f) },
    { XMFLOAT3(-1.0f, 1.0f, 1.0f), XMFLOAT2(0.0f, 0.0f) },
    { XMFLOAT3(1.0f, 1.0f, 1.0f), XMFLOAT2(1.0f, 0.0f) },
    { XMFLOAT3(1.0f, 1.0f, -1.0f), XMFLOAT2(1.0f, 1.0f) },
};
```

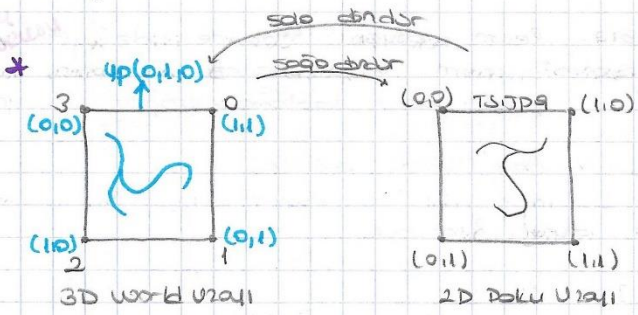
- t → texture buffer, temsil eder.
  s → sample

- cBuffer obVeve Changes? register(b0)
  {
  Matrix View;
  };

  } Doku kaplarken View matrisi değişmez.

⇒ Doku kaplamada meshColor değişmez.

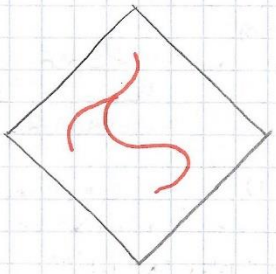
⇒ SamplerState samLinear : register(s0);
  → resim dosyasını renkle kaplamak için
  → Dokudaki renkleri okurken filtrelemek mi yoksa direkt mi kaplayacağımızı söyler.



. 2 cm 90° dönmeli (3D → 2D)

⇒ up vektörü (0,0,1) olduğunda küp kenarda olduğu için hiçbir şey görülmez.

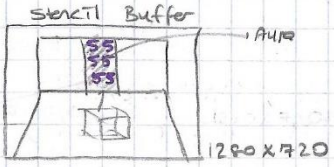
⇒ up vektörü (0.7f, 0.7f, 0.7f) olduğunda z ekseninde 45° dönmüş olur.



The Stencil Test

06.05.2016

Depth buffer ve stencil buffer olmak üzere iki tane vardır. Stencil buffer üretilecek veri ile aynı çözünürlükte olan bir bufferdır.



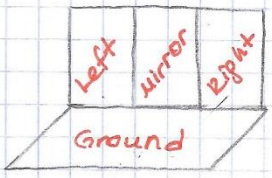
Stencil bufferdaki pixelleri belli sayılara setleyerek aynalılığı belirleriz.

2 tane struct tanımlayacağız. Birincisi setlemeyi yapacak ikincisi de setleme yapılan bölgeye resmi çizme yapacak. Bunlar; MarkMirrorDSS, DrawReflectionDSS.

```
//MARK MIRROR DSS
mirrorDesc.FrontFace.StencilPassOp = D3D11_STENCIL_OP_REPLACE;
mirrorDesc.FrontFace.StencilFunc = D3D11_COMPARISON_ALWAYS;

//DRAW REFLECTION DSS
drawReflectionDesc.FrontFace.StencilPassOp = D3D11_STENCIL_OP_KEEP;
drawReflectionDesc.FrontFace.StencilFunc = D3D11_COMPARISON_EQUAL;
```

Duvar tek bir Vertex Buffer'la ifade ediliyor. Duvarın sağ', sol tarafı, zemin, küp normal olarak back buffer'a çiziliyor. Daha sonra orta kısmı çiziliyor.



* Left, Right Wall ve Ground için aynı Vertex Buffer kullanılıyor.

```
// Ayna Çizimi
g_pImmediateContext->OMSetDepthStencilState(MarkMirrorDSS, 1);
cb.vMeshColor = XMFLOAT4(0.0f, 0.125f, 0.3f, 1.0f);
g_pImmediateContext->UpdateSubresource(g_pCBChangesEveryFrame, 0, NULL,

g_pImmediateContext->Draw(6, 12);
```

```
//Yansıma Hesabı
XMVECTOR mirrorPlane = XMVectorSet(0.0f, 0.0f, 1.0f, -6.0f);
                                   A     B     C     D

XMMATRIX R = XMMatrixReflect(mirrorPlane);
g_World-Box = g_World-Box * R;
```

* Küpün yansıması alındıktan sonra saat yönün tersine çevriliyor. Back buffer olarak saat yönde olanı alın daha bu kod kullanılır:

```
g_pImmediateContext->RSSetState(CullClockwiseRS);
```

* Zemin Identity ile çarpılıyor çünkü zemin sabit. (XMMatrixIdentity() * R)

```
// Küpün Gölgesinin Çizimi
XMVECTOR shadowPlane = XMVectorSet(0.0f, 1.0f, 0.0f, 2.0f);
XMFLOAT4 lightDir = XMFLOAT4(0.6f, -0.8f, 0.0f, 0.0f);
XMVECTOR toMainLight = -XMLoadFloat4(&lightDir);
XMMATRIX S = XMMatrixShadow(shadowPlane, toMainLight);
XMMATRIX shadowOffsetY = XMMatrixTranslation(0.0f, 0.005f, 0.0f);
```