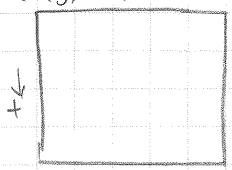
BİLGİSAYAR GRAFİKLERİ

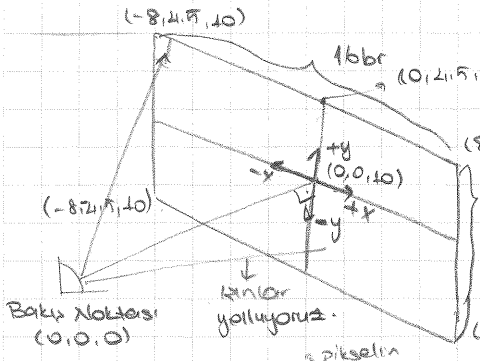
visual c++ , .net framework 26.02.24

RAY TRACING

picture box 2 boyutlu , 3 boyutlu görüntü düzlemi ile aynı çözünürlükte

picture box değerleri hep pozitif





*10 birim yerden bakıyoruz

2 değerleri sbt

*piksel koordinatları genelde float

*pikselin eni yatayda $\frac{16}{800}$

düşeyde $\frac{9}{900}$

*3 boyuttan 2 boyuta dönüşüm yapacağız.

*pikselin ucunu alıyoruz

*pikselin yüksekliği

AD(x,y,z) = (16 * x / 799 - 8, 4.5 - y * 9 / 449, 10) ⇒ x,y koordinatlı picture koordinatından 3 boyutlu koor. bulma

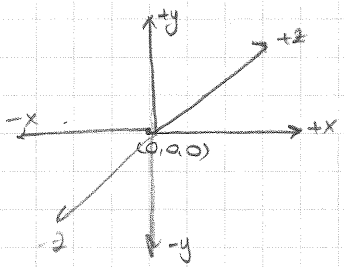
(0-799 arasında değerler için)

Doküman: Işın İzleme (Ray Tracing)

Işın vektörel bir büyüklük

vektörün boyutu (x,y,z) $\Rightarrow \sqrt{x^2+y^2+z^2} = |R|$ (3 boyutlu uzayda)

*OpenGL'de z ekseni ters



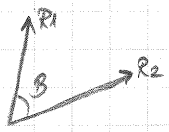
* Vektörün boyunu 1 birim yapılması normalize edilmesi. Her bir ekseni vektörün boyuna bölünmesiyle bulunur.

R(0,6,8) → |R| = 10

R normalize $\left(\frac{0}{10}, \frac{6}{10}, \frac{8}{10}\right)$ → kareleri toplamı 1 yapar

* Skaler çarpım (*) → float bir sayı üretir.

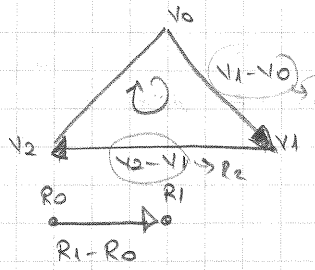
$R_1 * R_2 = R_{1x}R_{2x} + R_{1y}R_{2y} + R_{1z}R_{2z} = |R_1| * |R_2| * \cos(\beta)$



$|R_1| = 1$ ve $|R_2| = 1$ ise $R_1 * R_2 = \cos\beta$

* Vektörel çarpım (x) → 3. bir vektör üretir. Diğer 2 vektöre diktir.

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



Üçgenin normalini hesaplarken (yüzeye dik) vektörel çarpım kullanırız.

$R_1 \times R_2$ çarptırımızda üçgene dik vektörü elde ederiz. Yani üçgenin yüzey normali = $R_1 \times R_2$

Fark vektörlerinin sırası clock wise olduğu için yüzey normuna dik çıkar.

Arka tarafta saatin tersi yönünün tersi olduğu için yüzeye dik çıkmaz.

* $V_0(0,10,120)$ $\quad V_1 - V_0 = (30,-80,-60)$

$V_1(30,-10,60)$ $\quad V_2 - V_1 = (-60,0,0)$

$V_2(-30,-10,60)$

$N = \underset{R_1}{(V_1 - V_0)} \times \underset{R_2}{(V_2 - V_1)} = (-80*0 - (-60)*0,\ -60*-60 - 30*0,\ 30*0 - (-80)*(-60))$

$N = (0, 3600, -4800)$

Işının Tanımı

→ Başlangıç noktası ve doğrultuya sahip vektörel bir büyüklük olan R ışını.

$R = R_0 + t \times R_d$

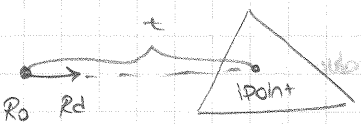
(Ray) ↓ skaler → ışının doğrultusu

ışının başlangıcı

R_d: normalize edildiği varsayılır. $|R_d| = 1$

t: ışının 3D uzayda R_d doğrultusu boyunca kaç birim ilerleyeceğini belirler.

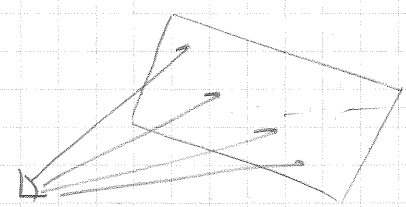
* Yolladığımız ışının üçgenle kesişim testi yapıp t ve u uzaklığını bulurut. i. point $R = R_0 + t \times R_d$ formülü ile bulabiliriz. Kesiştiği nokta yani

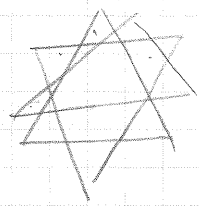


R_d paralellikle birim vektördür.

Ro (0,0,0) için 10 sonraki ışığın konumu nedir?
Rd (0,0,1)

$P = R_o + t * R_d \Rightarrow R = (0,0,0) + 10 * (0,0,1) = (0,0,10)$





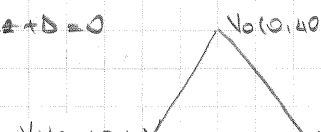
*Gelen ışıklar üçgenlerle kesişebilir. 3 üçgenle de kesişiyorsa t uzaklığı en küçük olanın rengini boyarız. Kesişmiyorsa boyanmaz!

* Hesaplamalar ve üçgenler 3 boyutlu ama form'un üzerindeki picture box iki boyutludur.

Kamera değişken bakış noktası : Ro
Vertex camera = Vertex (0,0,0)
küçük x y'ler picture box pixeli
→ Işığın doğrultusunu hesaplarken pixel ve kamera koordinatlarını alıcaz. Rd
→ TraceRay fonku (kamera (Ro), doğrultuyu (Rd) ve üçgenleri parametre alır.
TraceRay fonku Color türünde değer döndürür ve RGB değerlerine sahiptir.
Her biri 0-255 arasındadır.
TraceRay fonku kesişim testleri yapar ve t uzaklığını hesaplar.
3 üçgen olduğu için for 3 kez koşar.
intersection bir struct (distance + değeri, indis ise üçgen indisi
Intersection.size() > 0 1'den fazla kesişim için minimum t uzaklığını araştırır ve o üçgenin rengini döndürür.
If'e girmezse kesişim yoktur siyah döndürür.
iPoint ışığın üçgen ile kesiştiği noktadır.

1. aşamada yakaladığım ışın üçgenin üzerinde olduğu düzlemle kesişiyor mu
2. kesişim noktası üçgenin içinde mi dışında mı

Düzey Denklemi $Ax + By + Cz + D = 0$
N (A, B, C)
x y z



Üçgenin normalini hesaplıyoruz ve A,B,C değerini buluruz. → N (0, 0.6, -0.8)
Köşe noktalarından herhangi biri D'yi hesaplamak için kullanılabilir.
0.6 * 40 - 0.8 *

$\rightarrow Ax + By + Cz + D = 0$
$\rightarrow R = Ro + t * Rd$
$Rx = Rox + t * Rdx$
$Ry = Roy + t * Rdy$
$Rz = Roz + t * Rdz$

yerine yazarım. $\Rightarrow A(Rox + t*Rdx) + B.(Roy + t * Rdy) + C(Roz + t^* Rdz) + D = 0$

$$t = -\frac{N * Ro + D}{N * Rd}$$

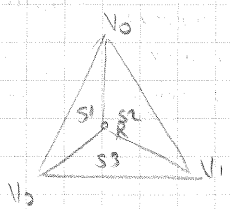
Işın düzleye paralel olursa $Rd = 0$

$t > 0$ ise yüzey ile kesişiyor.

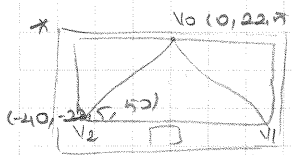
2. Aşama alan testi aşaması

Alt üçgenler belirlenir ve S1, S2, S3 alanları hesaplanır toplamları büyük S'ye eşit olmalı gerekir.

S1, S2 ve S3 değerleri flacing point'dur genelde ve karşılaştırmada çok küçük bir hataya gömülerek gelir.

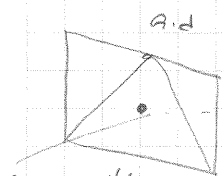


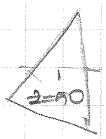
Shpere classda triangle classdan shapeten miras alınır. Çünkü Shape nesne üretiminin içinde farklı classlardan nesneler var ve polimorfizm ile ortak fonksiyon kullanıldığında hangi fonksiyon kullanacağı kendiliğinden belirlenir. Shapein üçgen veya daire olduğunun önemi yoktur.

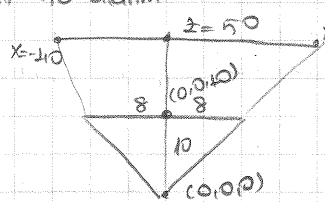


+ çizeceğimiz üçgenin koordinatları pb'in tam köşelerine denk gelsin.

z değerini 50 alalım

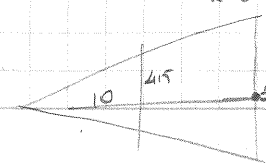






$\frac{40}{50} = \frac{8}{x}$

$x = 40$



$\frac{40}{50} = \frac{41.5}{y}$

$y = 22.7$

Işın İzleme Adımları

1-) Bakış noktasından ışınlar yollanır. 3D Görüntü Düzlemindeki ilgili piksellerden geçen bu ışınlara "Birincil Işınlar" denir.

2-) Birincil ışınlarla 3D uzaydaki cisimler arasında kesişim testleri yapılır. Kesişimler için ışının o cisme olan t uzaklığı hesaplanır.

3-) Eğer herhangi bir birincil ışın 1'den fazla cisimle kesişmişse t uzaklığı en düşük olan cisim belirlenir.

4- t uzaklığı en düşük cismin rengi, kesişim noktasının normali, ışık kaynağının konumu ve rengi gibi parametrelere bağlı olarak birincil ışının geçtiği pikselin rengi hesaplanır ve basılır.

```
class Vertex {   => Bu nesne vektörü temsil ediyor.
    public:
      float x;
      float y;
      float z;

    Vertex Normalize () {                    => Vektörü normalize
      float length = (float) Math::Sqrt (x*x + y*y + z*z);   ettiğimizde x,y,z
      x /= length;                           bileşenlerini vektörün
      y /= length;                           boyuna böldük. Vektörün
      z /= length;                           boyu 1 bir haline
      return *this;                          gelir.
    }
    float Length(){
        return (float) Math::Sqrt (x*x + y*y + z*z)
    }
    Vertex CrossProduct (Vertex p) {   => İki vektörün vektörel çarpımı R1 x R2

        return Vertex (y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);
    }

    Vertex operator+(Vertex p){
        return Vertex( x+p.x , y+p.y , z+p.z);
    }
    Vertex operator-(Vertex p){
        return Vertex(x-p.x , y-p.y , z-p.z);
    }
    float operator*(Vertex p){   => Skaler çarpım R1 * R2
        return x*p.x + y*p.y + z*p.z;   floating point sayı sonucu bu
    }                                   kısım Red *t
```

```
Vertex operator/(float f){
    return Vertex(x/f, y/f, z/f);
}
};

Vertex operator*(float f, Vertex p){          f solda olsada kopar
    return Vertex(f*p.x, f*p.y, f*p.z);       sağda olsada kopar
}
```

→ Rd piksel koordinatları - bakış noktasından bulunur. Birincil ışınların doğrultusu Rd'dir.

```
struct intersections
{
    float distance;
    int   indice;
} intersection;
```

```
Triangle T1(Vertex(0,30,40), Vertex(40,-30,120), Vertex(-40,-30,120), Color::Red);
Triangle T2(Vertex(-50,30,124), Vertex(50,30,124), Vertex(0,-30,124), Color::Green);
Triangle T3(Vertex(-30,0,37), Vertex(30,40,117), Vertex(30,-40,117), Color::Blue);

Shape* shapesT3T = {&T1, &T2, &T3};
Vertex camera = Vertex(0,0,0)   → Bakış noktasını temsil eder.

for(int y=0; y<480; y++){
    for(int x=0; x<800; x++){
        Vertex pixel = Vertex(16*x/799 - 8, 4.5 - y*9/449.0f, 40);
        Vertex Rd = (pixel - camera).Normalize()
        Color c = TraceRay(camera, Rd, Shapes)
        surface->SetPixel(x,y,c)
    }
}
```

2 boyutlu picture boxtan 3D görüntü düzlemine geçtik

Işın - Üçgen Kesişim Testi

1-) Işın ile üçgenin üzerinde olduğu düzlemsel yüzey arasında kesişim testi
2-) Işın yüzeyle kesişiyorsa kesişim noktasının üçgenin içinde olup olmadığını belirleme

Birinci aşama ışın üçgenin tanımladığı yüzeyin denklemini çıkarmak gerekmektedir.
Yüzey denklemi $Ax+By+Cz+D=0$ 'dır. Burada (A,B,C) yüzey normalidir.
Yüzeyin üzerinde olduğu için yüzey denklemini sağlayacağından üçgenin köşe noktalarından herhangi biri D'nin hesabı için kullanılabilir.

$N = (0, 0.6, -0.8)$ ve $V_0 = (0, 40, 120)$ alalım.
A B C

$0*0 + 0.6*40 - 0.8*120 + D = 0 \Rightarrow 24 - 96 + D = 0 \Rightarrow D = 72$

Yüzey Denklemi $0.6y - 0.8z + 72 = 0$ ✓

* Işın yüzey ile kesişiyorsa üçgenin köşe noktaları gibi ışının yüzey üzerindeki koordinatları da yüzey denklemini sağlamalıdır.

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

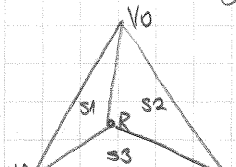
$$t = -\frac{AR_{ox} + BR_{oy} + CR_{oz} + D}{AR_{dx} + BR_{dy} + CR_{dz}} = -\frac{N * R_o + D}{N * R_d}$$
(skaler çarpım)

$t > 0$ ise ışın yüzey ile kesişiyor
$t < 0$ ise ışın yüzey ile kesişmiyor
$t = \infty$ ise $N * R_d = 0$ ise ışın yüzeye paralel demektir
(skaler çarpım)

Örnek: $R_o = (0,0,0)$, $R_d = (0,0,1)$ ve $N = (0, 0.6, -0.8)$ için R ışınının bu yüzey ile kesişip kesişmediğini t hesabı ile belirleyelim

$$t = -\frac{N * R_o + D}{N * R_d} = -\frac{72}{-0.8} = 90$$

Işının yüzey üzerindeki koordinatları $R = R_o + t \cdot R_d = (0,0,0) + 90*(0,0,1) = (0,0,90)$ olarak bulunur. Böylece kesişim testinin 1. aşaması tamamlanmış olur.
2. aşamada noktanın üçgenin içinde olup olmadığına karar verilmelidir. Bunun için alan testi yöntemi kullanılacaktır.



Hesaplanan kesişim noktasından üçgenin köşelerine doğrular çizilerek 3 alt üçgen oluşturulur. Alt üçgenlerin toplamı üçgenin alanına eşitse kesişim noktası üçgenin içinde demektir.
V_0, V_1, V_2 üçgeninin alanı $|(V_1 - V_0) \times (V_2 - V_1)|$ ile hesaplanabilir.
(vektörel çarpım) (vektör uzunluğu)

Triangle'ın Intersect Fonku ⇒ Kesişim uzaklığını hesaplar

```
float Intersect ( Vertex Ro, Vertex Rd)
{ Vertex normal;
  Vertex R
  float S, s1, s2, s3;

  normal = (V1-V0).CrossProduct(V2-V1);
  float D = - (normal * V0);
  float t = - (normal * Ro+D)/( normal * Rd);
```

} Için yüzey kesişim testi

```
  if(t >0){
    R = Ro + t* Rd ;
    S = (V1-V0).CrossProduct(V2-V1).Length()
    s1 = (R- V0).CrossProduct(V2-V1).Length()
    s2 = (V1-V0).CrossProduct(R-V1).Length()
    s3 = (V1-R).CrossProduct(V2-V1).Length()

    float fark = (float) Math::Abs(S - (s1+s2+s3));
    float epsilon =0.005F;
    if (fark <= epsilon) return t; else return 0;
  }
  else return 0;
}
```

Belirli bir harta payına bağlı olarak üçgen alan testi yaparak için üçgen kesişim testi yaptık.

* t uzaklığına bağlı olarak hangi üçgenin önde olduğunu veya zemin olduğunu alpilayarak renk döndüren TraceRay fonku

```
Color TraceRay(Vertex Ro, Vertex Rd, Shape * Shapes[7]) {
    vector<intersections> Intersections;

    for(int i=0; i<3; i++){
      float t= Shapes[7]-> Intersect(Ro,Rd);
      if (t > 0.0F){
          intersections.distance = t;
          intersections.indice = i;
      }
    if(Intersections.size() > 0){
      float min_distance =FLT_MAX;
      int min_indis =-1
      for( int i=0; i<Intersections.size(); i++){
```

⇒ 3 üçgen içinde 3D düzlemin içinde mi diye test edildi eğer içindeyse vektörün içine üçgenin indisi ve uzaklığı atıldı.

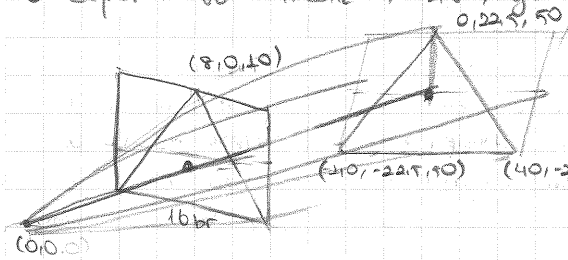
⇒

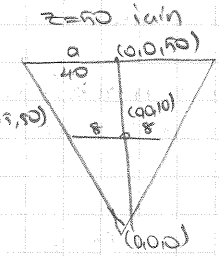
```
if (Intersections[i].distance < min_distance){
    min_distance = Intersection[i].distance;
    min_indice = Intersection[i].indice;
}
}
return Shapes[min_indice]->shapeColor;
}
return Color::Black;
```

⇒ Işın doğrultusundaki bu üçgen pixellerinden hangisi daha önde ise yani t'si küçük olanın rengi pixele basılır.

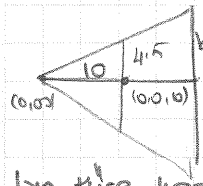
⇒ vektöre hiç şekil atılmamışsa yani bir üçgenle kesişmiyorsa o pixel siyah yapıldı

z değerini 60 alırsak x = 21.5 , y = 27 olur





$\frac{1}{5} = \frac{8}{a}$ $a = 40$



$\frac{1}{5} = \frac{4.5}{b}$ $b = 22.5$

Işın küre kesişim Testi

Kesişim testine ışının başlangıç noktasından (Ro) kürenin merkezine (c) $l = c - Ro$ vektörü çizilerek başlanır

Sonra üç tane skaler değer $s = l * Rd$, $l2 = l * l$ ve $r2 = r * r$ şeklinde hesaplanır

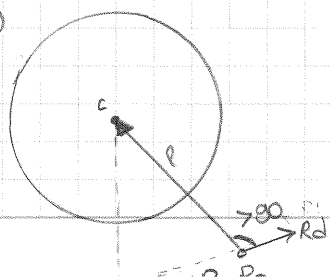
Burada s, l vektörünün boyunun Rd üzerine izdüşümüdür. Çünkü Rd birim vektör olduğundan $l * Rd$ aynı zamanda $|l| * \cos\beta$ demektir

$s = |l| * |Rd| * \cos\beta = |l| * \cos\beta$

l2 l vektörünün, r2 de yarıçapın boyunun karesidir

$l2 = l * l = |l| * |l| * \cos 0$

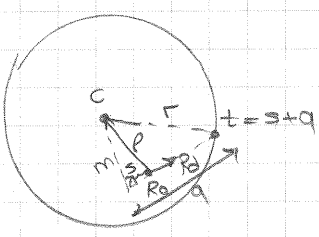
a.)



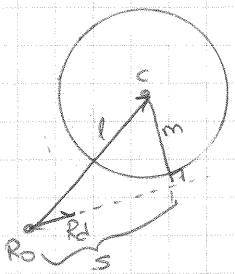
Cosβ açı 90'dan büyük olduğunda negatif değer aldığı için s değeride negatif olur ($s < 0$)

* l vektörünün boyu yarıçaptan büyükse ışının başlangıç noktası kürenin dışındadır, küçükse içindedir.
Eğer (s<0 && l2>r2) şartı sağlanıyorsa a'daki gibi kesişim yok demektir.

* s<0 olsa bile l2<r2 ise yani ışının başlangıç noktası kürenin içinde ise her zaman kesişim vardır.

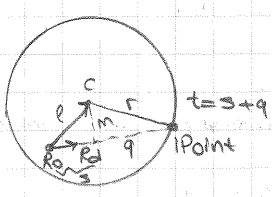


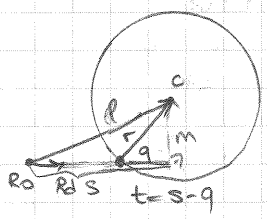
* s>0 iken hangi durumlarda kesişim olduğunu belirlemek üzere m2=l2-s2 hesaplanır. m2>r2 ise kesişim yoktur



m2 = l2 - s2
m2 > r2 (kesişim yok.)

* m2<r2 ise kesişim vardır ve l2<r2 ise ışın içten (d), l2>r2 ise dıştan(e) küreyle kesişir.

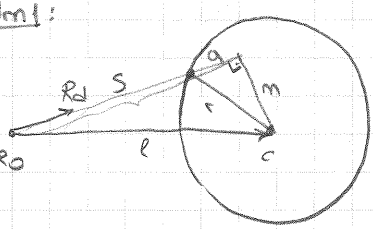




q= sqrt(r2 - m2) ile hesaplanır.
Dıştan kesiştiğinde ışının küreye uzaklığı t=s-q ile içten kesiştiğinde t=s+q ile hesaplanır.

```
class Sphere : public Shape {
    public:
        Vertex Center;
        float Radius;

    float Intersect(Vertex Ro, Vertex Rd) {
        Vertex l = Center - Ro;
        float s = l * Rd;
        float l2 = l * l;
        float r2 = Radius * Radius;
        if ( s<0 && l2 >r2) return 0; // Kesişim yok.
        float s2 = s*s;
        float m2 = l2-s2;
        if (m2 >r2) return 0;
        float q = (float) Math::Sqrt (r^2-m^2)
        if ( l2>r2) return s-q;
        else return s+q;
    }
```

örn1:



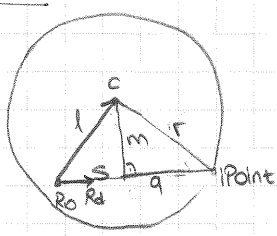
Ro(0,0,0)
c(0,0,100)
r= 75

l = c - Ro = (0,0,100)
s = l * Rd = (0,0,100) * (0,0.6,0.8)
s = 80

if(s<0 && l^2>r^2) → false
l2 = (0,0,100) * (0,0,100) = 10000
s2 = 6400
m2 = 10000 - 6400 = 3600
if(m2>r2) → false
q = sqrt(r^2-m^2) = 45
if(l2>r2) → true ⇒ t = s-q ⇒ t = 80-45 = 35

iPoint = Ro + t*Rd
(0,0,0) + 35 * (0,0.6,0.8)
iPoint = (0, 21, 28)

örn2:



Ro(0,-60,95)
Rd(0,0,1)
c(0,0,100)

l = c-Ro = (0,60,5)
s = l * Rd = (0,60,5) * (0,0,1) = 5
l2 = 3625
if (s<0 && l2>r2) false
m2 = l2 - s2 = 3600
if(m2>r2) false
q = sqrt(r^2-m^2) = 45
if (l2 > r2) false
else t = s+q → t = 5 + 45 = 50

iPoint = Ro + t*Rd = (0,-60,95) + 50*(0,0,1)
= (0,-60, 145)

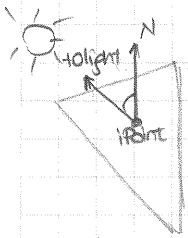
Phong Boyama Modeli

Işınlar ile üçgenler/küreler kesişim testi yapılıp görüntü düzlemine en yakın olan üçgen/küre belirlendiğinde bu u/k rengi doğrudan ilgili piksele setlenmez. Çünkü ışık kaynağının o noktayı ne oranda aydınlattığı dikkate alınmalıdır. Phong boyama modeline göre ışık kaynağına bağlı olarak ambient, diffuse ve specular olmak üzere üç renk bileşeni, toplamları 1 olan katsayılarla çarpılıp toplanarak pixelin son rengi elde edilir.

1-) Diffuse Renk Bileşeni

* Diffuse katsayı yüzey normali ile ışık kaynağına doğru olan vektör skaler çarpılarak belirlenir. Her ikisinin de birim vektör olduğu varsayıldığında skaler çarpımları aralarındaki açının kosinüsünü verir.
- Negatif değerler için yüzey ışık kaynağı tarafından aydınlatılmıyor demektir.
- Işık kaynağı yüzeye tam dik vuruyorsa, yani yüzey normali ile aralarındaki açı 0° ise $\cos 0 = 1$ olduğundan maximum aydınlatma
- Açı 90° ise de $\cos 90 = 0$ minimum (sıfır) aydınlatma söz konusudur.

Hesaplanan diffuse katsayı ile yüzeyin rengi çarpılarak Phong boyama modelinin diffuse renk bileşeni belirlenir.



açı > 90° ise ışık kaynağı noktayı aydınlatmaz.

diffuseKatsayı, ışık kaynağının yüzeyi aydınlatma oranı

* Bir daire için her iPoint noktasının normali farklıdır. Normal hesaplama:

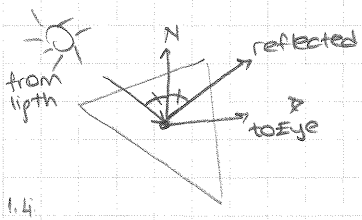
```
Vertex NormalAt(Vertex p) {
    return (p - center) / Radius;   => Radius'a bölerek normalize ediyoruz
}                  iPoint
*
Color ShadeDiffuse(Shape* S, Vertex iPoint){
    Vertex light = Vertex(0,30,60);
    Vertex toLigth = (light - iPoint).Normalize();
    Vertex normal = S->NormalAt(ipoint).Normalize();
    Color c = S->ShapeColor;   => cismin rengi
                                  => scaler çarpım.
    float diffuseKatsayı = normal * toLigth;
    if(diffuseKatsayı < 0.0F) return Color::Black;  => ışık kaynağı o yüzeyi
                                                       aydınlatmıyor demektir.
```

```
float r=0; g=0; b=0;
r += diffuseKatsayı * c.R;
g += diffuseKatsayı * c.G;
b += diffuseKatsayı * c.B;

r = r>255 ? 255 : r;    r = r<0 ? 0 : r;
g = g>255 ? 255 : g;    g = g<0 ? 0 : g;
b = b>255 ? 255 : b;    b = b<0 ? 0 : b;
sınır değerleri aşılırsa float değerler kaynaklı
return Color::FromArgb((int)r,(int)g,(int)b);
}
```

Işık kaynağının koordinatları (0,40,0) olduğu durumda yüzey normali (0,1,0) olan kırmızı renkli bir üçgen üzerindeki (0,0,30) noktasının diffuse bileşeni:
toLight = (0,40,0) - (0,0,30) = (0,40,30)
Normalize edersek (0, 0.8, -0.6)
diffuse katsayı: (0,0.8,-0.6) * (0,1,0)
= 0 + 0.8*1 - 0.6*0 = 0.8
kırmızı (255,0,0)
diffuse renk bileşeni (204,0,0)

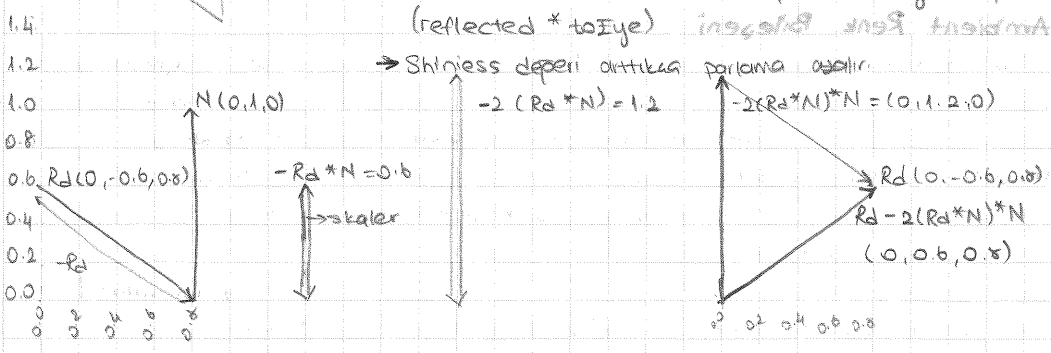
2-> Specular Renk Bileşeni

- Işığın yüzey üzerindeki parlamasını temsil eder.
- Specular katsayı ışık kaynağından yüzeye doğru olan vektörün yüzey normalinden yansıma vektörü ile bakış noktasına doğru olan vektör arasındaki açının belli bir skaler değer kadar kuvveti alınarak hesaplanır. Hesaplanan specular katsayı ile ışık kaynağının rengi çarpılarak Phong boyama modelinin specular renk bileşeni belirlenir.



Gelen ışının doğrultusu Rd, yüzey normali N olduğu durumda yansıma vektörü Rd - 2Rd*N*N ile hesaplanır.

Shiniess (parlamanın yarıçapı)
(reflected * toEye)
→ Shiniess değeri arttıkça parlama azalır.



```
Color ShadeSpecular (Shape* S, Vertex iPoint, Vertex camera) {
    Vertex light = Vertex(0,30,60);
    Vertex fromLight = (iPoint - light).Normalize();
    Vertex toCamera = (camera - iPoint).Normalize();  => toEye'a eşdeğer

    Color lamba = Color::White; →parlayacak ışığın rengi
    Vertex reflected = (fromLight - 2*(normal*fromLight)*normal).Normalize();
    float dotProduct = reflected * toCamera;
    if (dotProduct < 0.0f) return Color::Black;

    float specularKatsayı = (float) Math::Pow((double)dotProduct, (double) 8);

    float r=0, g=0, b=0;
    r += specularKatsayı * lamba.R;
    g +=        "        * lamba.G;
    b +=        "        * lamba.B;

    r = r > 255 ? 255 : r;  r = r < 0 ? 0 : r;
    g = g > 255 ? 255 : g;  g = g < 0 ? 0 : g;
    b = b > 255 ? 255 : b;  b = b < 0 ? 0 : b;

    return Color::FromArg((int)r, (int)g, (int)b);
}
```

Ambient Renk Bileşeni

Bir odada masa sehpa gibi eşyaların altları gibi ışık kaynağı tarafından doğrudan aydınlatılmayan yüzeyler vardır. Buralara duvarlardan veya diğer eşyalardan yansıyan ışıkların vurması nedeniyle az da olsa aydınlatılmaktadır. Phong modelinin ambient renk bileşeni bu dolaylı aydınlatmayı modeller ve cismin kendi rengi 0..1 arası bir katsayı ile çarpılarak ambient renk bileşeni hesaplanır.

ka + kd + ks = 1

Pixel Rengi = ka x AmbientColor + kd x Diffuse Color + ks x Specular Color (biz hesapladık.)

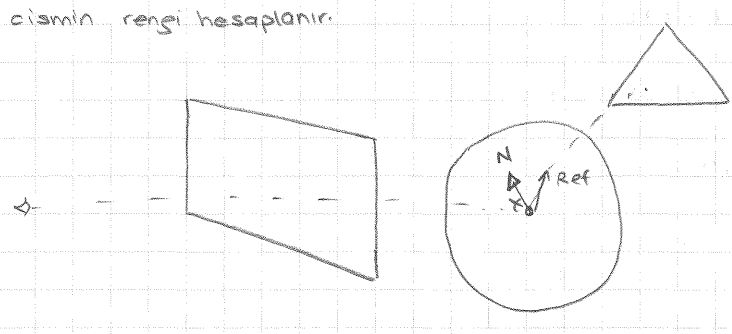
```
Color ShadingModel(Shape* S, Color diffuseColor, Color specularColor, float amb, float dif, float spec) {
Color ambientColor = S -> shapeColor;
int r = Math::Min(255, (int)(amb * ambientColor.R + dif * diffuseColor.R + spec * specularColor.R));
int g =                       .G                .G                .G));
int b =                       .B                .B                .B));
return Color::FromArgb(r,g,b);
```

S->Ambient, S->Dif, S->Spec değerleri ka, kd ve ks değerlerine denk gelir. Şekil duşturulurken setlenir.

Reflection (Yansima)

Yansıma doğrultusu $R_d - 2R_d * N * N$ ile hesaplanır.

Yansıma doğrultusu hesaplandıktan sonra yeni doğrultu boyunca yollanan ışın ile cisimler üzerinde kesişim testleri yapılarak en yakın cisim belirlenir. Onun rengi 0-1 arasında bir katsayı ile çarpılarak yansıma ile gelen cismin rengi hesaplanır.



TraceRay fonk.un içinde

if (S -> Refl != 0.0F) -> cisim aynasalsa

```
* Vertex CalculateReflection( Shape* S, Vertex iPoint, Vertex Rd)
{
   Vertex normal = S -> NormalAt(iPoint).Normalize();  -> iPoint noktasının normalini hesaplıyor
   return (Rd - 2 * (normal x Rd) * normal).Normalize();
}
```

-> calculateReflection fonk.u ile reflectedDirection hesaplanır.

```
reflectedColor = TraceRay(iPoint, reflectedDirection, Shapes, camera, depth + 1, S);
```

iPoint → Ro oldu; reflectedDirection → Rd oldu; S → yansıma özellikli olan nesneyi verdik, ilk çağırımda NULL idi.

```
Color TraceRay(Vertex Ro, Vertex Rd, Shape* Shapes[], Vertex camera, int depth, Shape* prevShape)
{
   if (depth > 4) {
      return prevShape -> ShapeColor;
   }

   vector<Intersection> Intersections;
```

/* depth değişkeni ile yolladığımız ışın yansıma sayısını sınırlıyor. İlk çağırımda 0 'dır. */
True olduğunda önceki kesişimdeki nesnenin rengini döndürür.

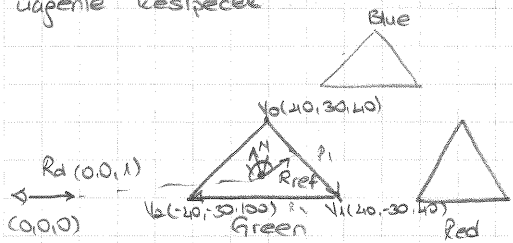
```
for( int i=0; i<sb; i++){
    float t = Shapes[i] -> Intersect(Ro,Rd)
    if( t > 0.1F)  ⇒ Float değerlerdeki hatalardan dolayı nokta zeminin altında-
    {  Intersection.distance = t;                  imiş gibi gözükebilir bunu
       Intersection.indice = i;                    önlemek için. Bu yüzden
       Intersections.push_back(Intersection);      yansımada tekrar nesne-
    }                                              nin kendisiyle kesişmesine
                                                   neden olabilir.

if(Intersections.size() >0){
        :
        :
  Vertex iPoint = Ro+ min_distance*Rd;
  Shape* S = Shapes[min_indis];
  Color reflectedColor;

  if( s -> Refl != 0.0F){
      Vertex reflectedDirection = CalculateReflection(S, iPoint, Rd);
      reflectedColor = TracePay(iPoint, reflectedDirection, shapes, camera, depth+1, S)
  }
  Color diffuseColor = ShadeDiffuse(S, iPoint);
  Color specularColor = ShadeSpecular(S, iPoint, camera);

  return ShadingModel(S, diffuseColor, specularColor, reflectedColor, S->Ambient,
                      S->Dif, S->Spec, S->Refl);
}
return Color::Black;
}
*
Color ShadingModel(Shape* S, Cola diffuseColor, Color specularColor, Color reflectedColor,
                   float amb, float dif, float spec, float refl)
{
  Color ambientColor = S -> ShapeColor;
  int r = Math::Min(255,(int)(amb* ambientColor.R + dif.diffuseColor.R + spec* specularColor.R
          + refl * reflectedColor.R));
  int g=
  int b=
  return Color::FromArgb(r,g,b);
}
```

if (t > 0.01F) yapmak yerine

iPoint = iPoint + 0.001F * reflectedDirection ile iPointi zemine ulaşabiliriz.

Reflection Triangle Sample

➡ İşin Red, Green üçgenlerden kameraya yakın olandan yansıyıp Blue renkli üçgenle kesişecek



Yansıma doğrultusu : $Rd - 2*(Rd*N)*N$

$N_{Green} = R_1 * R_2$

$R_1 = (0,-60,0)$

$R_2 = (-80,0,60)$

$N_G = (-60*60, 0, +60.(-80))$. Normalize()

$(-3600, 0, -4800)$. Normalize()

$N_G = (-0.6, 0, -0.8)$

$Rref \Rightarrow Rd * N = (0,0,1) * (-0.6,0,-0.8) = 0+0-0.8 = -0.8$

$\Rightarrow -2*(Rd*N) = 1.6$

$\Rightarrow -2*(Rd*N)*N = 1.6*(-0.6,0,-0.8) = (-0.96,0,-1.28)$

$\Rightarrow Rd - 2*(Rd*N)*N = (0,0,1) + (-0.96,0,-1.28) = (0.96,0,-0.28)$

iPointA = Ro + t * Rd = (0,0,0) + 70 * (0,0,1) = (0,0,70)

Yeni ışın için Ro = iPointA , Rd = Rref

mavi renkli üçgene t uzaklık tblue = 25

iPointB = (0,0,70) + 25 * (-0.96,0,-0.28) = (-24,0,63)

Reflection Sphere Sample

Ro = (0,0,0) başlangıç noktasından Rd = (0.96, 0.28, 0) doğrultusu boyunca giden bir ışın Cx (80, 15, 0) merkez noktasına ve rk = 10 birim yarı çapa sahip kırmızı renkli küreden yansıyıp Cm = (72, 121, 0) merkez koordinatlarına ve rm = 35 birim yarıçapına sahip mavi renkli küre ile kesişiyor.

a.) Mavi küre üzerindeki kesişim noktasını hesaplayınız.

Vertex l = (80,15,0) - (0,0,0) = (80,15,0)

float s = (80,15,0) * (0.96,0.28,0) = 80*(0.96) + 15*(0.28) = 81

float l2 = (80,15,0) * (80,15,0) = 6625

float r2 = 10 * 10 = 100

if(s<0 && l2 > r2) return 0 ⇒ false

float s2 = 81 * 81 = 6561

float m2 = 6625 - 6561 = 64

if(m2 > r2) return 0 ⇒ false

float q = $\sqrt{100-64}$ = 6

if (l2 > r2) return 81 - 6 ⇒ true return 75

iPointKirmizi = (0,0,0) + 75 * (0.96, 0.28, 0) = (72, 21, 0)

Rref = Rd - 2 * (Rd * N) * N

NKirmizi = (iPoint - center) / radius = (72, 21, 0) - (80, 15, 0) / 10 = (-0.8, 0.6, 0)

Rref ⇒ (0.96, 0.28, 0) * (-0.8, 0.6, 0) = 0.96 * (-0.8) + 0.28 * 0.6 = -0.6

⇒ 1.2

⇒ 1.2 (-0.8, 0.6, 0) = (-0.96, 0.72, 0)

⇒ (0.96, 0.28, 0) + (-0.96, 0.72, 0) = (0, 1, 0)

Yeni Ro = (72, 21, 0) , Rd = (0, 1, 0)

Vertex l = (72, 121, 0) - (72, 21, 0) = (0, 100, 0)

float s = (0, 100, 0) * (0, 1, 0) = 100

float l2 = (0, 100, 0) * (0, 100, 0) = 10000

float r2 = 35 * 35 = 1225

if(s<0 ...) → return 0 ⇒ false

float s2 = 100 * 100 = 10000

float m2 = 0

if (m2 > r2) return 0 ⇒ false

float q = $\sqrt{1225 - 0}$ = 35

if (l2 > r2) return 100 - 35 = 65 ⇒ true,

iPointMavi = (72, 21, 0) + 65 * (0, 1, 0) = (72, 86, 0)

2012 Vize 2. Soru

Ro = (0, -80, -25) başlangıç noktasından Rd = (0, 0.8, 0.6) doğrusu boyunca giden bir ışın C = (0,0,0) merkez koordinatlarına ve r = 100 birim yarıçapa sahip kürenin içinden yansıyıp aşağıda köşe noktalarının koordinatları V0, V1, V2 olarak verilmiş, N = (0, -0.6, 0.8) normaline sahip düzlem ile kesişiyor.

V0 = (0, 20, -34) V1 = (-10, -4, -52) V2 = (10, -4, -52)

a.) Küre üzerindeki kesişim noktasını hesaplayınız.

l = C - Ro = (0,0,0) - (0, -80, -25) = (0, 80, 25)

s = l * Rd = (0, 80, 25) * (0, 0.8, 0.6) = 79

l2 = (0, 80, 25) * (0, 80, 25) = 7025

r2 = 100 * 100 = 10.000

if (s<0 && l2 > r2) → false

s2 = 79 * 79 = 6241

m2 = 7025 - 6241 = 784

if(m2 > r2) → false

q = $\sqrt{10.000 - 784}$ = 96

if (l2 > r2) return s-q → false

else return s+q = 79+96 = 175

iPointküre = (0,-80,-25) + 175 * (0, 0.8, 0.6) = (0, 60, 80)

b.) Yansıyan ışının doğrultusunu hesaplayınız.

Nküre = (Cküre - iPoint) / Radius

= (0,0,0) - (0,60,80) / 100 = (0, -0.6, -0.8)

Rref = Rd - 2*(Rd*N)*N

=> (0, 0.8, 0.6) * (0, -0.6, -0.8) = -0.48 - 0.48 = -0.96

=> -2 * -0.96 = 1.92

=> 1.92 * (0, -0.6, -0.8) = (0, -1.152, -1.536)

=> (0, 0.8, 0.6) + (0, -1.152, -1.536) = (0, -0.352, -0.936)

Kullanılan ışın küremin içinden yansıyorsa N = c - iPoint'ten/Radius bulunur.

Yeni Ro = (0, 60, 80), Rd = (0, -0.352, -0.936)

Ax + By + Cz + D = 0 → (0) kullanalım

0 * 0 + (-0.6) * 20 + 0.8 * (-34) + D = 0 => D = 39.2

$t = -\frac{N*Ro+D}{N*Rd} \Rightarrow -\frac{(0,-0.6,0.8)*(0,60,80)+D}{(0,-0.6,0.8)*(0,-0.352,-0.936)} = -\frac{28+39.2}{-0.5376} = 125$

iPointüçgen = (0, 60, 80) + 125 * (0, -0.352, -0.936) = (0, 16, -37)

2019 Vize 1. Soru

Ro(24, -12, 0) noktasından Rd(0.8, 0.6, 0) doğrultusu boyunca giden bir ışın, merkezi c(0,0,0), yarıçapı r = 40 br olan kürenin içinden 2 kere yansıyıp köşe noktaları verilen N(0,1,0) normaline sahip üçgenle kesişiyor. Kesişim noktasının koordinatlarını hesaplayınız.

Vertex l = (0,0,0) - (24, -12, 0) = (-24, 12, 0)

float s = (-24, 12, 0) * (0.8, 0.6, 0) = -12

float l2 = (-24, 12, 0) * (-24, 12, 0) = 720

float r2 = 40 * 40 = 1600

if (s < 0 && l2 > r2) → false

float s2 = 144

float m2 = l2 - s2 = 576

if (m2 > r2) → false

float q = $\sqrt{1600 - 576}$ = 32

if (l2 > r2) → false

else return -12 + 32 = 20

iPoint1 = (24, -12, 0) + 20 * (0.8, 0.6, 0)

= (40, 0, 0)

N1 = (0,0,0) - (40,0,0) / 40 = (-1, 0, 0)

Rref1 => Rd - 2 * (Rd * N) * N

(0.8, 0.6, 0) * (-1, 0, 0) = -0.8

-2 * -0.8 = 1.6

(-1, 0, 0) * 1.6 = (-1.6, 0, 0)

(0.8, 0.6, 0) + (-1.6, 0, 0) = (-0.8, 0.6, 0)

Yeni $R_0 = (-11.2, 38.4, 0)$, $R_d = (-0.8, 0.6, 0)$

Vertex $l = (0,0,0) - (-40,0,0) = (-40,0,0)$

float $s = (-0.8, 0.6, 0) * (-40,0,0) = 32$

float $l2 = (-40,0,0) * (-40,0,0) = 1600$

float $r2 = 1600$

if ( ) → false

float $s2 = 1024$

float $m2 = 1600 - 1024 = 576$

if ( ) → false

float $q = \sqrt{1600 - 576} = 32$

if ( ) → false

else return $32 + 32 = 64$

$iPoint2 = (-40,0,0) + 64 * (-0.8, 0.6, 0)$

$= (-11.2, 38.4, 0)$

$N2 = (0,0,0) - (-11.2, 38.4, 0) / 40$

$= (0.28, -0.96, 0)$

$R_{ref2} = R_d - 2 * (R_d * N) * N$

$\Rightarrow (-0.8, 0.6, 0) * (0.28, -0.96, 0)$

$= -0.8$

$\Rightarrow -2 * -0.8 = 1.6$

$\Rightarrow 1.6 * (0.28, -0.96, 0) =$

$= (0.448, -1.536, 0)$

$\Rightarrow (-0.8, 0.6, 0) + (0.448, -1.536, 0)$

$= (-0.352, -0.936, 0)$

Üçgen için $R_0 = (-11.2, 38.4, 0)$, $R_d = (-0.352, -0.936, 0)$

D uzaklığı $\rightarrow (0,1,0) * (-25, 15, 0) = D \Rightarrow -15$

$$t = - \frac{(0,1,0) * (-11.2, 38.4, 0) - 15}{(0,1,0) * (-0.352, -0.936, 0)} = - \frac{23.4}{-0.936} = 25$$

$iPoint_{üçgen} = (-11.2, 38.4, 0) + 25 * (-0.352, -0.936, 0) = (-20, 15, 0)$

Transparency (Saydamlık)

→ Işın transparan küre ile kesiştiğinde yoluna devam eder.

Saydam cisimlerde ışın cismin içinden geçerken doğrultusu değişmez.

Transmission direction gelen ışın doğrultusu R_d ile aynı alınır

```
Vertex CalculateTransmission(Shape* S, Vertex iPoint, Vertex Rd){
    return Rd;
}
```

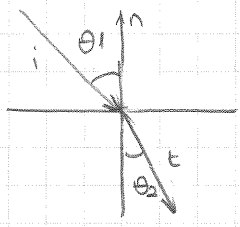
→ Işının küremin içinden geçebilmesi için depth değerinin min 2 olması gerekir.

→ Transmitted color arkadaki cismin rengi

```
transmittedColor = TraceRay(iPoint, transmittedDirection, shapes, camera, depth+1, S);
                   (iPoint: transparan resme üzerindeki kesişmenin olduğu nokta; transmittedDirection: Rd)
```

```
Color ShadingModel(shape* S, Color diffuseColor, Color specularColor, Color reflectedColor,
      Color transmittedColor, float amb, float dif, float spec, float refl, float trans)
{ Color ambientColor = S→ ShapeColor;
int r = Min(255, (int)(amb*ambientColor.R + dif*diffuseColor.R + spec*specularColor.R +
    refl*reflectedColor*R + trans.transmittedColor.R))
```

Refraction (Kırılma)



i: gelen ışının doğrultusu yani Rd
t: kırıldıktan sonraki doğrultu.
n: yüzey normali
$n_1 \cdot \sin(\theta_1) = n_2 \cdot \sin(\theta_2)$

$t = r * i + (w - k) * n$

$r = n1 / n2$

$w = -(i * n)\, r$

$k = \sqrt{1 + (w - r)(w + r)}$

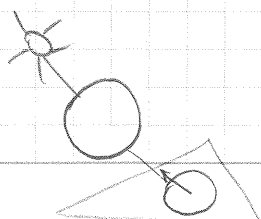
```
if(S->Refrac != 0.0f){
  Vertex refractedDirection = CalculateRefraction(S, iPoint, Rd, 1.0f, 1.33f);
  refractedColor = TraceRay(iPoint, refractedDirection, shapes, camera, depth+1, S);
}
Vertex CalculateRefraction(Shape* S, Vertex iPoint, Vertex Rd, float n1, float n2)
{
  Vertex normal = S->NormalAt(iPoint).Normalize();
  float  r = n1/n2;
  float  w = -(Rd * normal) * r;
  float  k = (float)Math::Sqrt((double)(1+(w-r)*(w+r)));
  return (r * Rd + (w - k) * normal).Normalize();   // t: ışının yeni doğrultusu
}
```

Rd: gelen ışının doğrultusu
n1: 1. ortamın kırılma indisi
n2: 2. ortamın kırılma indisi

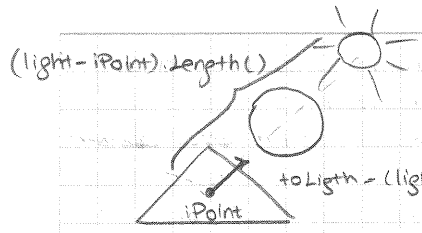
→ Shading Modele bir parametre daha eklenir.

Shadows (Gölgeleme)

Herhangi bir yüzeyin başka bir yüzeyin gölgesinde kalıp kalmadığını belirlemek için yüzey üzerindeki kesişim noktasından ışık kaynağına doğru gölge test etme ışını yollanır. Bu ışın ile üçgenler/küreler arasında kesişim testleri yapılarak en yakın üçgen/küre belirlenir. Hesaplanan uzaklık değeri ışık kaynağına olan uzaklıktan küçük ise yüzey bu üçgenin/ küremin gölgesinde kalıyor demektir.



→ Işık kaynağı tarafından doğrudan aydınlatılmasada ambient color sebebiyle tamamen siyah çıkmaz.




```
bool TestShadow(Shape* shapes[], Vertex iPoint)
{
  Vertex light = Vertex(0, 30, 40);
  Vertex toLight = (light - iPoint).Normalize()
  vector<intersections> Intersections;

   for(int i=0; i< 25; i++){
       float t= shapes[i]->Intersect(iPoint, toLight);
       ;
   }

  if( min_distance <(light - iPoint).Length())
     return true;
  else
     return false
}
```

⇒ Tüm şekillerde iPointten toLighta doğru gönderilen ışınla kesişim var mı diye test yapılır.

iPoint ile ışık kaynağı arasındaki uzaklık

```
if(TestShadow(shapes, iPoint){
  return ShadingModel(5, Color::Black, C::B, C::B, C::B, C::B, S->Ambient, S->Dif, S->Spec,
                      S->Refl, S->Trans, S->Refrac);
```

Düzlemsel Yüzey üzerine Doku Kaplama

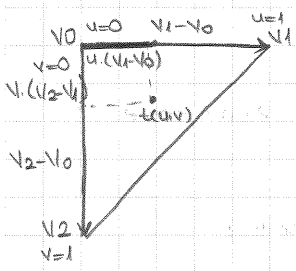
Doku bir image, dOku kapladığımız yüzey düzlemseldir ve yüzeyi 2 dik üçgenle temsil ederiz.

→ Möller'in yöntemine göre kesişim testi yapıldığında yalnızca kesişim noktasına uzaklık olan t değeri değil aynı zamanda kesişim noktasının barisentrik koordinatlarında hesaplanabilir. Barisentrik koordinatlar ve üçgen köşe koordinatları kullanılarak üçgen üzerinde herhangi bir noktaya gidebilmek için aşağıdaki ifade kullanılır:

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

→ Üçgenin köşe noktaları uygun sırada tutulursa barisentrik koordinatlarla doku kaplama için kullanılan resim dosyasındaki doğru piksel koordinatları rahatlıkla bulunabilir. Bunun için bazı varsayımlar yapılmıştır. Birincisi üzerine doku kaplaması yapılacak yüzey düzlemseldir. İkincisi yüzey iki dik üçgenden oluşmaktadır. Dik üçgenlerin V_0, V_1 ve V_2 köşe koordinatlarından dik kenarları birleştiren V_0 olmalıdır.

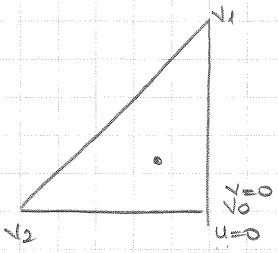
Doku kaplamada kullanılacak resmin genişliği w, yüksekliği h olduğunda, üzerine doku kaplama yapılacak olan yüzeye ait dik üçgenlerden birincisi için resim dosyasındaki koordinatları bulmak için $(u*w, v*h)$ formülü kullanılır. İkinci dik üçgen içinde $((1-u)*w, (1-v)*h)$ formülü kullanılır.



$t(u,v) = V_0 + u(V_1 - V_0) + v.(V_2 - V_0)$ $\quad u \geq 0, v \geq 0, u+v \leq 1$

İPoinden resmin pixeline gitme → $(u*w, v*h)$

bu pixeldeki rengi alıp ekrana basarız.



pixel koordinatları: $((1-u)*w, (1-v)*h)$

u ve v'nin hesaplanma mantığına bakmadık.

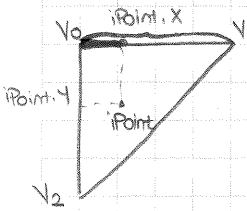
→ Bir nesnenin üzerine doku kaplayıp kaplamayacağımızı int tex parametresi ile belirtiriz. 0 ise yapılmayacak.

Her bir texture için farklı int tex kullanırız ki hangi kaplamayı yapacağımızı bilelim. Üst üçgen ve alt üçgen için farklı tex numaraları kullanırız.

```
void TextureMapping(Vertex R0, Vertex Rd, Shape* S){
    switch(S->Tex){
        float* uv;
        float u;
        float v;
        int x, y;
        case 1:{
         uv = IntersectUV(R0,Rd);
         u = uv[0]
         v = uv[1]
```

```
x=(int)(u * 800)      (w)
y=(int)(v * 800)      (h)
S->ShapeColor = myBitmap->GetPixel(x,y);
}
```

2-) u ve v koordinatlarını kendimiz iPoint'e bağlı olarak hesaplayabiliriz



$$u = \frac{iPoint.X - V0.X}{V1.X - V0.X}$$

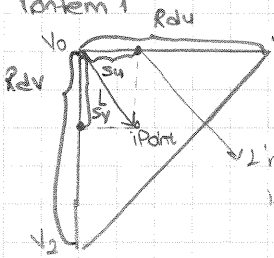
$$v = \frac{iPoint.Y - V0.Y}{V2.Y - V0.Y}$$

→ uzaklıkların oranı u ve v değerlerini verir.

→ soldan sağa doğru giderken sadece x ve aşağı doğru giderken sadece Y'nin değiştiğini varsayıyoruz.

3-) V0'dan V1'e giderken sadece x değil x,y,z'nin hepsi değişebilir. Vektörel bir oku kaplama işlemi gerçekleştirmeliyiz.

Yöntem 1



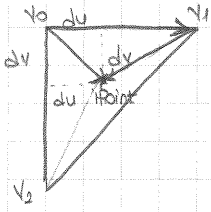

```
L = iPoint - V0          // Vertex
Rdu = (V1 - V0).Normalize // Vertex
Su = L * Rdu             // float  => |L| * |Rdu| * cosθ
```

L'nin V1-V0 üzerine izdüşümü

```
float u = Su / (S->V1 - S->V0).Length()
Vertex Rdv = (S->V2 - S->V0).Normalize()
float Sv = L * Rdv
float v = Sv / (S->V2 - S->V0).Length()
```

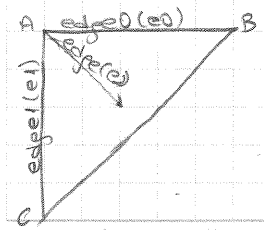
→ Yaptığımız işlemler vektörel olduğu için x,y,z koordinatlarının değişiminden etkilenmeyiz.

Yöntem 2




```
Vertex X0_X1 = iPoint - S->V0
Vertex X0_X2 = iPoint - S->V1
Vertex X2_X1 = S->V1 - S->V0
float dv = (X0_X1.CrossProduct(X0_X2)).Length() / X2_X1.Length()
float v = dv / (S->V2 - S->V0).Length()

X0_X2 = iPoint - S->V2
X2_X1 = S->V2 - S->V0
float du = (X0_X1.CrossProduct(X0_X2)).Length() / X2_X1.Length()
float u = du / (S->V1 - S->V0).Length()
```

Yöntem 3



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

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

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

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

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

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

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

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

$u * d10 * d10 + v * d10 * d11 = d10 * d21$

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

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

Vertex e0 = S->V1 - S->V0;

Vertex e1 = S->V2 - S->V0

Vertex e2 = iPoint - S->V0

float d00 = e0 * e0

float d10 = e1 * e0

float d11 = e1 * e1

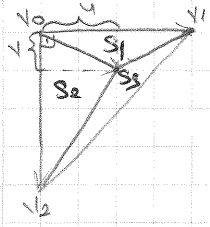
float d20 = e2 * e0

float d21 = e2 * e1

float u = (d20 * d11 - d21 * d10) / (d00 * d11 - d10 * d10)

float v = (d20 * d01 - d21 * d00) / (d01 * d01 - d11 * d00)

Yöntem 4



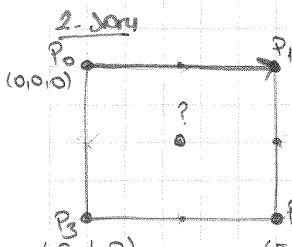
$$v = \frac{S_1}{S}, \quad u = \frac{S_2}{S}$$

float v = ((S->V1 - S->V0).CrossProduct(iPoint - S->V0)).Length() / (S->V1 - S->V0).CrossP(S->V2 - V0).Leg

float u = ((S->V2 - S->V0).CrossProduct(iPoint - S->V0)).Length()/((S->V1 - S->V0).CrossProduct()
(S->V2 - S->V0).Length()

2022

2. Soru



A) $P_0 + 0.5(P_1 - P_0) + 0.5(P_2 - P_1)$ ✓

B) $P_1 + 0.5(P_2 - P_1) + 0.5(P_3 - P_2)$ ✓

C) $P_2 + 0.5(P_3 - P_2) + 0.5(P_0 - P_3)$ ✓

D) $P_3 + 0.5(P_0 - P_3) + 0.5(P_0 - P_1)$ ✗

E) $0.5(P_0 + P_2)$ ✓

F) $0.5(0.5(P_0 + P_3) + 0.5(P_1 + P_2))$ ✓

$(P_1 - P_0)\ 0.5 = (2.5, 0, 0)$

$P_0 + 0.5(P_1 - P_0) = (2.5, 0, 0)$

$(P_2 - P_1) * 0.5 = 0.5\ (0, 4, 0) = (0, 2, 0)$

$\Rightarrow (2.5, 2, 0)$

$P_1 \qquad 0.5(P_2 - 1)$

$(5, 0, 0) + (0, 2, 0) = (5, 2, 0)$

$0.5(P_3 - P_2) = 0.5(-5, 0, 0) = (-2.5, 0, 0)$

$\Rightarrow (2.5, 2, 0)$

$(5, 4, 0) + (-2.5, 0, 0) = (2.5, 4, 0)$

$0.5(P_0 - P_3) = 0.5\ (0, -4, 0) = (0, -2, 0)$

$\Rightarrow (2.5, 2, 0)$

$(0, 4, 0) + (0, -2, 0) = (0, 2, 0)$

$0.5(P_0 - P_1) = 0.5\ (-5, 0, 0) = (-2.5, 0, 0)$

$\Rightarrow (-2.5, 0, 0)$ ✗

$0.5(P_0 + P_2) = 0.5(5, 4, 0) = (2.5, 2, 0)$

$0.5(P_0 + P_3) = 0.5(0, 4, 0) = (0, 2, 0)$

$0.5(P_1 + P_2) = 0.5(10, 4, 0) = (5, 2, 0)$

$\Rightarrow 0.5\ (5, 4, 0) = (2.5, 2, 0)$ ✓

3. Soru

$R_0(42, 56, 0)$, $R_d(0.96, -0.28, 0)$, $c(0, 0, 0)$, $r = 284$ küresinin içinden aynasal yansıyıp merkezi $(114, -35, 0)$, yarıçapı $r = 25$ olan başka bir küre ile kesişiyor. Kesişim noktasının koordinatları nedir?

$l = (-42, -56, 0)$

$s = (-42, -56, 0) * (0.96, -0.28, 0) = -24.64$

$l2 = (-42, -56, 0) * (-42, -56, 0) = 4.900$

$r2 = 54.956$

if ($s < 0$ && $l2 > r2$) → false

$s2 = 607.1296$

$m2 = 4.292.8704$

if ($m2 > r2$) → false

$q = \sqrt{54956 - 4292.8704} = 224.64$

if ($l2 > r2$) → false

else return $-24.64 + 224.64 = 200$

iPoint1 = (142, 56.0) + 200*(0.96, -0.28, 0) = (234, 0, 0)

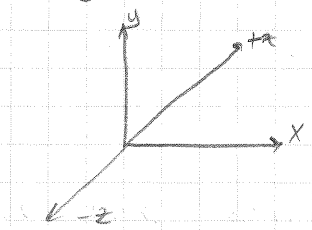
Nküre (0,0,0) - (234,0,0) / 234 = (-1,0,0)
Rref = Rd - 2*(Rd*N)*N
⇒ (0.96, -0.28, 0) * (-1,0,0) = -0.96
⇒ -2 * -0.96 = 1.92
⇒ (-1,0,0) * 1.92 = (-1.92, 0, 0)
⇒ (0.96, -0.28, 0) + (-1.92, 0.0) = (-0.96, -0.28, 0)

Yeni Ro(234,0,0), Rd(-0.96, -0.28, 0)
l = (114, -35, 0) - (234, 0, 0) = (-120, -35, 0)
s = (-120, -35, 0) * (-0.96, -0.25, 0) = 125
l2 = (-120, -35, 0) * (-120, -35, 0) = 15625
r2 = 625
if → false
s2 = 125 * 125 = 15625
m2 = 0
if → false
q = √625 - 0 = 25
if( ) return 125 - 25 = 100

iPoint2 = (234, 0, 0) + 100*(-0.96, -0.28, 0) = (138, -28, 0)

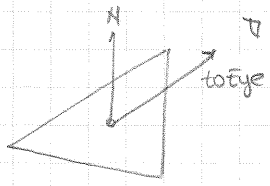
Backface Culling

→ Normalin dik olduğu yüz önyüz, dik olmadığı yüz arka yüz.
→ Üçgenin köşe noktaları sırası dik belirlenirse normal dik çıkar.
→ Kartezyen koordinat düzlemi



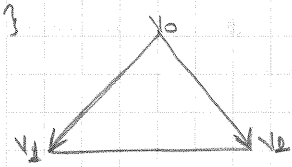
OpenGL'de +z ve -z yönleri terstir.
OpenGL'de noktalar saat yönünün tersinde tanımlanır.

1. Yöntem:

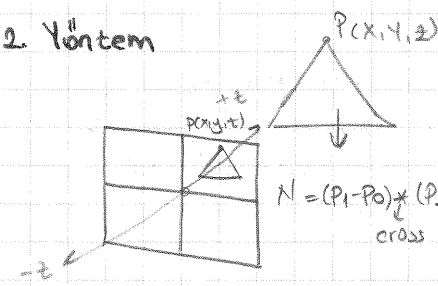


if(N* totye < 0) then backface
N ve totye arasındaki açı $\frac{\Pi}{2}$'den büyükse nesne göstergeye bakmıyordur
Başlangıçta Shape'in backface = false'tur.

```
void DotProductBackfaceTest(Shape* Shapes[7], Vertex camera) {
    for(int i=0; i<94; i++) {
  Vertex normal = (Shapes[i]->V1 - S[i]->V0).CrossProduct(S[i]->V2 - S[i]->V0);
  Vertex toEye = (camera - Shapes[i]->V0);

  if((normal * toEye) < 0.0F)
  {
    Shapes[i]->backface = true;
  }
  }
}
```



2. Yöntem



$N = (P_1 - P_0) * (P_2 - P_0)$ (cross)

Normalin z değeri negatif ise frontface

Normal.z > 0 ise backface

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

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

$p.z = d$

```
Vertex Projection(Vertex P;)
{
  Vertex p(0,0,0);
  p.x = 10* P.x / P.z
  p.y = 10* P.y / P.z
  p.z = 10
}
```

Bu fonk. üçgenin her köşe noktasının izdüşümünü hesaplar.

```
void CrossProductBackfaceTest(Shape* S[7]) {
  for(int i=0; i<94; i++) {
    Vertex A = Projection(S->V0);
    Vertex B =           (S->V1);
    Vertex C =           (S->V2);
    Vertex normal = (B-A).CrossProduct(C-A);
    if(normal.z > 0.0F) { S[i]->backface = true; }
  }
}
```

Axis Aligned Bounding Box

Nesneler bir yerde yoğunlaşmışsa bir dikdörtgen prizma tanımlanır. İlk olarak bu prizma ile kesişim testi yapılır. Kesişiyorsa nesnelerle kesişim testi yapılır.

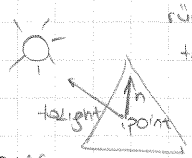
float AABB(Vertex Ro, Vertex Rd, Vertex B, float U, float V, float W)
B: prizmanın merkezi

2019 Vize

1. a)

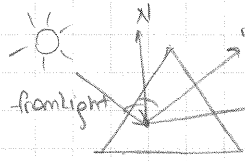
Diffuse renk bileşeni bir kaynağının yüzeyi aydınlatma oranını verir. toLight vektörünün yüzey normali ile skaler çarpımından elde edilir.

toLight = light - iPoint



Diffuse

Specular renk bileşeni ışığın yüzey üzerinde parlamasını temsil eder. reflected vektörü ve toEye'ın skaler çarpımının shinness kadar üssü alınarak hesaplanır.



reflected = fromLight - 2 * (fromLight * N) * N

(reflected * toEye)^shinness

Ambient renk bileşeni bir kaynağı tarafından doğrudan aydınlatılmayan yüzeylerin dolaylı aydınlatılmasını modeller. Cismin kendi renginin 0..1 arasında bir katsayı ile çarpılmasıyla elde edilir.

1. b) light(0,40,0) ve camera(0,14,78), N(0,1,0) olan kırmızı renkli bir üçgen üzerindeki (0,0,30) noktasının specular renk bileşeni ne olur.

fromLight = (0,0,30) - (0,40,0) = (0,-40,30). Normalize() ⇒ (0,-0.8,0.6)

Rref ⇒ (0,-0.8,0.6) * (0,1,0) = -0.8

⇒ -2 * -0.8 = 1.6

⇒ 1.6 * (0,1,0) = (0,1.6,0)

⇒ (0,-0.8,0.6) + (0,1.6,0) = (0,0.8,0.6)

toEye = (0,14,78) - (0,0,30) = (0,14,48). Normalize() = (0,0.28,0.96)

(0,0.28,0.96) * (0,0.8,0.6) = 0.8

(255,0,0) * 0.8 = (204,0,0)

4-) Ro(-48, -60, 36) noktasından Rd(0.48, 0.6, 0.64) doğrultusu boyunca giden bir ışın, merkezi c(0,0,0) yarıçapı r = 100 br olan kürenin içinden yansıyıp yine aynı küre ile kesişiyor. Kesişim noktasının koordinatları?

Vertex l = (0,0,0) - (-48, -60, 36) = (48, 60, -36)
float s = (48, 60, -36) * (0.48, 0.6, 0.64) = 36
float r2 = 100.00
float s2 = 1296
float l2 = (48, 60, -36) * (48, 60, -36) = 7200
float m2 = 7200 - 1296 = 5904
float q = $\sqrt{10000 - 5904}$ = 64
return 36 + 64 = 100

iPoint = (-48, -60, 36) + 100 * (0.48, 0.6, 0.64) = (0, 0, 100)

N1 = (0,0,0) - (0,0,100) = (0,0,-100) → Normalize() → (0,0,-1)

Rref → Rd - 2 * (Rd * N) * N1
→ (0.48, 0.6, 0.64) * (0,0,-1) = -0.64
→ -2 * -0.64 = 1.28
→ 1.28 (0,0,-1) = (0,0,-1.28)
→ (0.48, 0.6, 0.64) + (0,0,-1.28) = (0.48, 0.6, -0.64)

Yeni Ro(0,0,100) , Rd(0.48, 0.6, -0.64)
tz için
Vertex l = (0,0,0) - (0,0,100) = (0,0,-100)
s = (0,0,-100) * (0.48, 0.6, -0.64) = 64
l2 = 10000
r2 = 10000
s2 = 4096
m2 = 10000 - 4096 = 5904
q = $\sqrt{10000 - 5904}$ = 64
return 64 + 64 = 128

iPoint2 = (0,0,100) + 128 * (0.48, 0.6, -0.64) = (61.44, 76.8, 18.08)

2.a) Backface culling nasıl yapılır? (Perspektif dönüşüm ile)

3D uzaydaki nesnenin görüntü düzlemine izdüşümü alınır ve normali hesaplanır. Normalin z değeri negatif ise frontfacetir.

görüntü düzlemine izdüşüm ama

$p.x = \frac{d * P.X}{P.z}$, $p.y = \frac{d * P.Y}{P.z}$, $p.z = d$

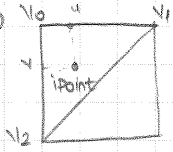
2.b) (60,80,100) noktasının görüntü düzlemi üzerine perspektif izdüşümünü hesaplayınız.

Bakış noktası (0,0,0) ve $d = 10$

$p.x = \frac{10 * 60}{100} = 6$, $p.y = \frac{10 * 80}{100} = 8$, $p.z = 10$ $\Rightarrow (6,8,10)$

2018 Arasınav

1.)



$u = (iPoint.X - V0.X) / (V_{1}.X - V0.X)$

$v = (iPoint.Y - V0.Y) / (V_{2}.Y - V0.Y)$

2017 1. Soru

$R_0(24,-12,0)$ $R_d(0.8,0.6,0)$ $c(0,0,0)$, $r = 40$ küreden 2 kere yansıyıp $N(0,1,0)$ üçgene kesişiyor. Kesişim noktasının koordinatları?

1. Uzaklık

$l = (0,0,0) - (24,-12,0) = (-24,12,0)$

$s = (-24,12,0) * (0.8,0.6,0) = -12$

$l2 = (-24,12,0) * (-24,12,0) = 720$

$r2 = 1600$

$s2 = 144$

$m2 = 720 - 144 = 576$

$q = \sqrt{1600 - 576} = 32$

return $-12 + 32 = 20$

$iPoint = (24,-12,0) + 20 * (0.8,0.6,0) = (40,0,0)$

$N = (0,0,0) - (40,0,0) / 40 = (-1,0,0)$

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

$\Rightarrow (0.8,0.6,0) * (-1,0,0) = -0.8$

$\Rightarrow -2 * -0.8 = 1.6$

$\Rightarrow 1.6 * (-1,0,0) = (-1.6,0,0)$

$\Rightarrow (0.8,0.6,0) + (-1.6,0,0) = (-0.8,0.6,0)$

Yeni Ro (20,0,0) Rd (-0.8, 0.6, 0)

2. üçgenlik

l = (0,0,0) - (20,0,0) = (-40,0,0)

s = (-20,0,0) * (-0.8, 0.6, 0) = 32

l2 = (-20,0,0) * (-20,0,0) = 1600

r2 = 1600

s2 = 1024

m2 = 1600 - 1024 = 576

q = √(1600 - 576) = 32

return 32 + 32 = 64

Rref = Rd - 2*(Rd*N)*N

= (-0.8, 0.6, 0) * (0.28, -0.96, 0) = -0.8

= -2 * -0.8 = 1.6

= 1.6 * (0.28, -0.96, 0) = (0.448, -1.536, 0)

+(-0.8, 0.6, 0) + (0.448, -1.536, 0)

= (-0.352, -0.936, 0)

iPoint = (20,0,0) + 64*(-0.8, 0.6, 0)

iPoint = (-11.2, 38.4, 0)

Normal = (0,0,0) - (-11.2, 38.4, 0)

= (11.2/20, -38.4/20, 0) . Normalize() = (0.28, -0.96, 0)

Yeni Ro (-11.2, 38.4, 0) Rd (-0.352, -0.936, 0)

A*x + B*y + C*z + D = 0 → Yo'i kullanalım

0 * -25 + 1 * 15 + 0 * 0 + D = 0 → D = -15

t = - (N * Ro + D) / (N * Rd) = (0,1,0) * (-11.2, 38.4, 0) + 15 / (0,1,0) * (-0.352, -0.936, 0) = 23.4 / -0.936 = 25

iPointÜçgen = (-11.2, 38.4, 0) + 25 * (-0.352, -0.936, 0) = (-20, 15, 0)

2018)

1) U0(-60, 35, 96) U1(-28, 35, 128) U2(-60, -25, 96) üçgeninin üzerindeki iP(-44, 17, 108) noktasının (u,v) barisentrik koordinatlarını hesaplayınız

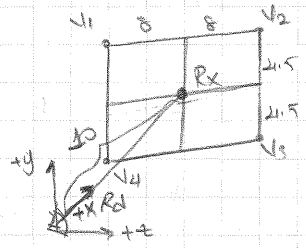
u = (iPoint.x - Vo.x) / (V1.x - Vo.x)

v = (iPoint.y - Vo.y) / (V2.y - Vo.y)

u = (-44 - (-60)) / (-28 - (-60)) = 16/32 = 0.5

v = (17 - 35) / (-25 - 35) = -18/-60 = 0.3

3-) Camera (24, -12, 0) bakış noktasından Rd (0.8, 0.6, 0) doğrultusu boyunca ilerlendiğinde bakış noktasına 10 br uzaklıktaki 16×9 birimlik görüntü düzleminin köşe koordinatlarının başlangıç konumları ne olur?



2 (24, -12, 0)

Rx = camera + t * Rd

(24, -12, 0) + 10 * (0.8, 0.6, 0) = (32, -6, 0)

(0,0,1) * Rd = (+0.6, 0.8, 0) = u

u * Rd = (0,0,-1) = v

V1 = (32, -6, 0) - 8 * (0,0,-1) + 4.5 (-0.6, 0.8, 0) = (29.3, -2.4, 8)

V2 = (32, -6, 0) + 8 (0,0,-1) + 4.5 (-0.6, 0.8, 0) = (29.3, -2.4, -8)

V3 = (32, -6, 0) + 8 (0,0,-1) - 4.5 (-0.6, 0.8, 0)

V4 = (32, -6, 0) - 8 (0,0,-1) - 4.5 (-0.6, 0.8, 0)

2017

2-) R0 (0,0,0) noktasından Rd (0,0,1) doğrultusu boyunca ilerlendiği varsayıldığında görüntü düzlemine ait P noktasının N (0, 0.6, -0.8) iPoint (0, 0, 100) noktasından yansıdıktan sonraki konumlarını hesaplayınız.

P0 (-8, 4.5, 10)

P1 (8, 4.5, 10)

P2 (8, -4.5, 10)

P3 (-8, -4.5, 10)

Rref = Rd - 2 * (2 * N) * N →

3-) Camera (0, -26, 59) ve (0, -34, 63) olduğunda N (0, 0.8, -0.6) üçgenin arkayüz olup olmadığını belirleyiniz. Uo (0, 60, 180)

N * toEye < 0 ise backface

toEye = (0, -26, 59) - (0, 60, 180) = (0, -86, -123)

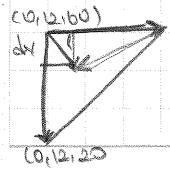
(0, 0.8, -0.6) * (0, -86, -123) = 5 > 0 önyüz.

toEye = (0, -34, 63) - (0, 60, 180) = (0, -94, -117)

(0, 0.8, -0.6) * (0, -94, -117) = -5 < 0 arkayüz.

2016

1-) V0 (0, 12, 60) V1 (18, -12, 60) V2 (0, 12, 20) V dik üçgeni üzerindeki iPoint (9, 0, 48) noktasından E1 = V1 - V0 ve E2 = V2 - V0 kenarlarına uzaklıkları ile ilgili bir yöntem öneriniz. Uzaklıkları bulunuz.



V0X1 = (9, 0, 48) - (0, 12, 60) = (9, -12, -12)

V0X2 = (9, 0, 48) - (18, -12, 60) = (-9, 12, -12)

X2X1 = (18, -12, 60) - (0, 12, 60) = (18, -24, 0)

dV = (9, -12, -12) * (-9, 12, -12) length / (18, -24, 0) length

-12 * -12 - (-12) * 12, -12 * -9 - 9 * -12, 9 * 12 - (-12) * (-9)

X0X2 = (9,0,28) - (0,12,20) = (9,-12,28)
X2X1 = (0,12,20) - (0,12,60) = (0,0,-40)
X0X1 = (9,-12,-12)
du = (9,-12,-12)*(9,-12,28) . 1 / (0,0,-40) . 1 ⇒ 600/40 = 15
-12*28 - (-12)*(-12), (-12)*9 - 9*28, 9*-12 - (-12)*9 ⇒ (-480, -360, 0)

2-) V üçgeninin V0 köşe noktasının aynı zamanda yarıçapı r=30 birim, merkezi C(0,-12,78) olan küre üzerinde olduğu varsayılsın. Üçgenin V2 noktasından yollanan bir ışın, V0 noktasından o noktanın küre normaline göre aynasal yansıyıp köşe noktaları verilen Nu(0,-1,0) normaline sahip U üçgeni ile kesişiyor. Kesişim noktasının koordinatları?
U0(0,36,40), U1(30,36,60), U2(-30,36,60)

Rd = (V0 - V2).Normalize()
= (0,12,60) - (0,12,20) = (0,0,40).Normalize() = (0,0,1)

N_V0 = (0,12,60) - (0,-12,78) / 30 = (0,24,-18) / 30 = (0,0.8,-0.6)
Rref = Rd - 2*(Rd*N)*N
= (0,0,1) * (0,0.8,-0.6) = -0.6
= -2 * -0.6 = 1.2
= 1.2 * (0,0.8,-0.6) = (0,0.96,-0.72)
= (0,0,1) + (0,0.96,-0.72) = (0,0.96,0.28) → Rd

0*0 + 36*-1 + 40*0 + D = 0 ⇒ D = 36 — N * R
t = - ((0,-1,0)*(0,12,60) + 36) / ((0,-1,0)*(0,0.96,0.28)) = -(-12 + 36)/-0.96 = 24/0.96 = 25 ✓

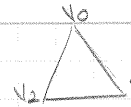
IPamüçgen = (0,12,60) + 25*(0,0.96,0.28) = (0,36,67)

2015
1-) IPam2 (5.6, 0, 58.8) = (7,0,54) + t*Rd
Rd2 = -(5.6,0,58.8) + (7,0,54) = (1.4/5, 0/5, -2.8/5).Normalize() = (0.28, 0, -0.96)

Rref = (0.28,0,-0.96)*(-1,0,0) = -0.28
= -2 * -0.28 = 0.56
= (0.56,0,0) + (0.28,0,-0.96) = (-0.28, 0, -0.96)

Ro = (7,0,54) + 25*(-0.28,0,-0.96) = (0,0,30)

2-) toEye * N < 0 (backface)



N = (V1 - V0) * V2 - V0

N_U = (30, -60, -80) * (-30, -60, -80)

= -60 * -80 - (-80) * (-60), (-80) * (-30) - 30 * (-80), 30 * (-60) - (-60) * (-30)

= (0, 4800, -3600). Normalize() = (0, 0.8, -0.6)

toEye = (0,0,0) - (0, 30, 160) = (0, -30, -160)

(0, 0.8, -0.6) * (0, -30, -160) = 72 > 0 önyüz.

N_V = (30, 60, -80) * (-30, 60, -80)

2014

float t = S.Intersect(Ro, Rd)

Vertex iPoint = Ro + t * Rd

∠(100, 117, 0)

(100, 117, 0) - (72, 21, 0) = (28, 96, 0) Normalize() → Rd

(72, 121, 0) - (72, 21, 0) = (0, 100, 0)

22.04.24

Interactive Ray Tracing

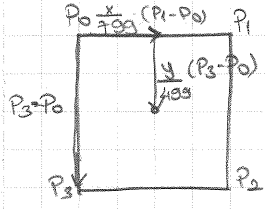
3 boyutlu ortamla etkileşim halindeyiz.

W yakınlaş, A sola, S uzaklaş, D sağa, Z yukarıda bakış, X aşağıdan bakış

*Dönme işlemi yaparken kameranın konumu değişmez. Görüntü düzleminin köşe koordinatları değişir. Bu köşelerle ara koordinatlar hesaplanır.

Bu köşelerle ara koordinatlar hesaplanır.

Pixel koordinatları hesaplama



Vertex pixel = P0 + (x/799.0) * (P1 - P0) + (y/449.0) * (P3 - P0)

→ Bu kode ile tüm pixelleri dolaşabiliriz

→ (P1 - P0) / 799 pixelin eni

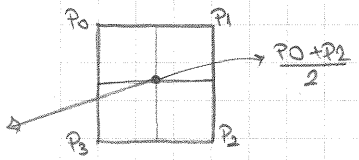
(P3 - P0) / 449 pixelin boyu

```
for(int y=0;  y< 450; y++){
    for(int x=0;  x<800;  x++) {
        Vertex pixel =P0+ (x/799.0)*(P1-P0) + (y/449.0) *(P3-P0);
        Vertex Rd = (pixel - camera).Normalize()
        Color  c  = TraceRay(camera, Rd, Shapes, camera, 0, NULL),
         surface -> SetPixel(x,y,c);
      }
    if ((y%20) == 0)   surfacePictureBox -> Refresh();
}
```

→ Dönme işleminde bakış nok. sabit, köşe noktaları değişir
İlerlerken hem bakış nok. hem köşe noktaları değişir.

```
switch(e->KeyChar)
{
  case 'w': {
    Vertex norm = (P1-P0).CrossProduct(P2-P1);
    norm.Normalite()

    P0 = P0 - 10* norm;   // -normal doğrultusunda noktaları 10 br ileri taşıdık.
    P1 = P1 - 10* norm;
    P2 = P2 - 10* norm;
    P3 = P3 - 10* norm;

    camera = (P0+P2)/2 + 10* norm;
  }
  break;
```

* Normal bize bakar ama w ile ileri gittiğimiz için -Normal eklenir P değerlerine.



$$camera = \underbrace{(P0+P2)/2}_{R_0} + \underbrace{10}_{t} * \underbrace{norm}_{R_d}$$

Kamerayı görüntü düzleminin 10 br gerisine taşıdık.

```
  case 's': {
    Vertex norm = (P1-P0).CrossProduct(P0-P1);
    norm.Normalite();
    P0 = P0 + 10* norm;
    P1 = P1 + 10* norm;
    P2 = P2 + 10* norm;
    P3 = P3 + 10* norm;
    camera = (P0+P2)/2 + 10* norm;
```

```
case 'a':{
   P0 = rotateLeft(P0,camera)
   P1 =  //    (P1, //  )
   P2 =  //    (P2, //  )
   P3 =  //    (P3, //  )
}
break;
case 'd':{
  P0 = rotateRight(P0,camera)
  P1 =  //    (P1, //  )
  P2 =  //    (P2, //  )
  P3 =  //    (P3, //  )
}
break;
case '2':{
   P0= P0 + Vertex(0,10,0)
   P1 = P1 + Vertex(0,10,0)
   P2 = P2 + Vertex(0,10,0)
   P3 = P3 + Vertex(0,10,0)

   camera = camera + Vertex(0,10,0);
}
break;
case 'x':{
   P0 = P0 - Vertex(0,10,0)
   P1 = P1 - Vertex(0,10,0)
   P2 = P2 - Vertex(0,10,0)
   P3 = P3 - Vertex(0,10,0)

   camera = camera - Vertex(0,10,0)
}
```

```
// ccw Rotation around Left
Vertex rotateLeft(Vertex P, Vertex camera){
   P = P - camera;
   float tmpX = P.X                 sin(15)
   P.X = P.X * 0.966F - P.Z * 0.259F;
   P.Z = tmpX * 0.259F + P.Z * 0.966F;
   P = P + camera                   cos(15)
   return P;
}
```

→ Y ekseninde rotasyon yapılırken y sabit kalır ve bu yüzden hesaplanmaz.
→ P = P - camera ⇒ Odanın içini bakış noktası etrafında döndürmek istiyoruz. Kamera etrafında dönebilmek için döndürülecek nokta kameradan çıkarılır.
tmpX değişkenini P.Z hesaplanırken güncellenmeyen P.X değerini kullanabilmek için oluşturduk.

```
// CW Rotation around Y-axis
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;
}
```

* ccw = saat yönünün tersine , cw = clock wise (saat yönünde)
* Sola dönerken saat yönünün tersine, sağa dönerken saat yönünde dönüyor.

* ccw Rotation around Y-axis

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

* cw Rotation around Y-axis

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

* X tuşu ile kameranın ve P değerlerinin konumu aşağı hareket ettirilir.
  2 tuşu ile kameranın ve P değerlerinin konumu yukarı hareket ettirilir. } y ekseninde rotasyon.

Rotation Object

```
switch (e->keyChar)
    case 'd': {
        rotateCubeY_CW(Vertex(0,0,50))
    }
    break;
    case 'd': {
        rotateY_CCW(Vertex(0,0,50))
    }
    break;
    case 'w': {
        rotateCubeX_CW(V(0,0,50))
    }
    b:
    :
void rotateCubeY_CCW(Vertex center){
  for(int i=0; i<12; i++){
    Vertex P = Shapes[i]->V0;
    P = P - center;
    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 + center;
    Shapes[i] = P;
}

* Vertex YCW(Vertex P, Vertex center){
    P = P - center;
    float tmpX = P.X;
    P.X = P.X * 0.966F + P.Z * 0.259F;
    P.Z = -tmp.X * 0.259F + P.Z * 0.966F;
    P = P + center;
    return P;
}
```

*Küpü kendi orjini etrafında döndürmek için rotate fonklarına küpün orjinin parametre olarak geçilir.

→ Vertex(0,0,50) ⇒ küpün merkez koordinatları

→ Küpün 6 yüzeyi olduğu için 12 üçgen temsilinde kullanılır.

→ Her üçgenin V0, V1, V2 koordinatları üzerinde rotasyon yapılır.

→ V0, V1 ve V2 için aynı işlem uygulanır.

```
* void rotateCubeY_CW(Vertex center){
    for(int i=0; i<12; i++) {
        Shapes[i]->V0 = YCW(S[i]->V0, center)
        S[i]->V1 = YCW(S[i]->V1, center)
        S[i]->V2 = YCW(S[i]->V2, center)
    }
}
```

```
* Vertex XCCW(Vertex P, Vertex center){
  P = P-center;
  float tmpY = P.Y;
  P.Y = P.Y * 0.966F + P.Z * 0.259F;
  P.Z = -tmpY * 0.259F + P.Z * 0.966F;
  P = P + center
  return P;
}
```

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

```
* Vertex XCW(Vertex P, Vertex center){
  P = P-center;
  float tmpY = P.Y
  P.Y = P.Y * 0.966F - P.Z * 0.259F;
  P.Z = tmpY * 0.259F + P.Z * 0.966F;
  P = P + center;
  return P;
}
```

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

```
* Vertex ZCCW(Vertex P, Vertex center){
  P = P - center;
  float tmpX = P.X;
  P.X = P.X * 0.966F - P.Y * 0.259F;
  P.Y = tmpX * 0.259F + P.Y * 0.966F;
  P = P+center;
  return P;
}
```

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

```
* Vertex ZCW(Vertex P, Vertex center){
  P = P-center;
  float tmpX = P.X;
  P.X = P.X * 0.966F + P.Y * 0.259F;
  P.Y = -tmpX * 0.259F + P.Y * 0.966F;
  P = P+center;
  return P;
}
```

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

DIRECTX 12

hlsl ⇒ high level shading language ⇒ GPU'da koşar.
•cpp ⇒ CPU'da koşar bir kısmı

Ekran kartında buffer'lar var CPU buffer'lara veri yükler
mul ⇒ multiply

PSMain koordinatların boyanması
Önce bir pencere düzenlenir üzerine directx çizilir
Winmain ⇒ V c++ main fonksiyonu
InitWindow ⇒ pencereyi oluşturur bas

DirectX
- OnInit() ⇒ Üçgenin köşe noktalarının, renginin setlendiği, grafikle ilgili matrisler setlenir
- UpUpdate() ⇒ frameden frame değişiklik varsa hesaplar.
- OnRender ⇒ çizim yapılır

OnInit: programın başında bir kez koşar
msp: kullanıcıdan giriş, mesaj alır

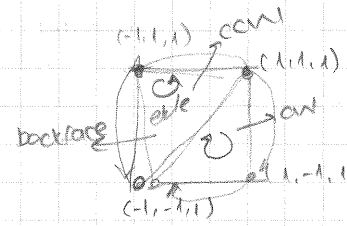
Hangi buffer'lar setlenir
XMFLOAT3 ⇒ (x,y,z) position
XMFLOAT4 ⇒ (R,G,B,A) color ⇒ (0-1) arası değişen floating point değerler

m_VertexBuffer } Ekran kartında buffer tanımlamada kullanılan pointer değişkenler.
m_VertexBufferView }
↓
ekran kartında oluşturduğumuz m_VertexBuffera işaret eder.

IASetVertexBuffer ile hangi vertex buffer'ı kullanacağımızı setleriz.

DrawInstanced(Hangi vertexler kullanılır, 1, StartVertexLocation

Kare yapalım
Vertex triangleVertex[7] =
{
V0 { XMFLOAT3(1.0, 1.0, 1.0), _rnk
V1 {
V2 {



DrawInstanced(6,1,0,0)
6 yerine

Özellikle söylenmediği müddet backface culling yapılır
psoDesc.RasterizerState.CullMode = none ⇒ backface culling yapma
FRONT ⇒ frontfaceleri çizmez.
BACK ⇒ backfaceleri çizmez

Visual C++
OnInit() ⇒ Bufferların içeriğinin doldurulması
OnUpdate() ⇒ OnInitteki settlemelerdeki değişiklik yapılır

```
struct Vertex
{
    XMFLOAT3 position; (x,y,z)
    XMFLOAT4 color;  (R,G,B,A)  örn kırmızı (1,0,0,1)
}
```

* Üçgen oluşturabilmemiz için ekran kartında buffer oluşturmalıyız.
ComPtr<ID3D12Resource> m_vertexBuffer; ⇒ GPU tarafında vertex buffera işaret eden değ
D3D12_VERTEX_BUFFER_VIEW m_vertexBufferView; ⇒ CPU " " " " " "

* // Create the vertex buffer
Vertex triangleVertices[] =
{ { XMFLOAT3(0.0f, 1.0f, 1.0f), XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f) },
{ XMFLOAT3(1.0f, -1.0f, 1.0f), XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f) },
{ XMFLOAT3(-1.0f, -1.0f, 1.0f), XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f) }
};
const UINT vertexBufferSize = sizeof(triangleVertices);
ThrowIfFailed(m_device->CreateCommitedResource(
&CD3DX12_HEAP_PROPERTIES(D3D12_HEAP_TYPE_UPLOAD),
D3D12_HEAP_FLAG_NONE,
&CD3DX12_RESOURCE_DESC::Buffer(vertexBufferSize),
D3D12_RESOURCE_STATE_GENERIC_READ,
nullptr,
IID_PPV_ARGS(&m_vertexBuffer)));

* Buffer settlemesi yapıldı.

* m_vertexBufferView.BufferLocation = m_vertexBuffer->GetGPUVirtualAddress();
m_vertexBufferView.StrideInBytes = sizeof(Vertex)
m_vertexBufferView.SizeInBytes = vertexBufferSize

m_vertexBufferView'in başlangıç değerleri settlendi, birden fazla buffer varsa =ettlemesi burada yapılır.

onRender() { → her bir framede koşar

m_commandList → SetGraphicsRootConstantBufferView(0, m_constantBuffer → GetGPUVirtualAdress())

m_commandList → IASetVertexBuffers(0,1,&m_VertexBufferView);
m_commandList → DrawInstanced(3,1,0,0);
Vertex bufferdaki kaç vertex çizilecek → StartVertexLocation: Bufferdaki hangi vertex indisinden başlanacağı

}

Struct SceneConstantBuffer { 256 baytlık aralıklarla içeriğine veri yazılabilir. Vertex bufferdan küçük

XMMATRIX mWorld;
XMMATRIX mView;
XMMATRIX mProjection;
}

OnInit() içerisinde g_World = XMMatrixIdentity(); birim matrise setlenir.
OnUpdate() içinde g_World = XMMatrixRotationY(rotation) olarak setlenir

→ g_World çizim için hangi transformları uygulayacağımızı belirtir. (Döndürme, öteleme, büyütme)

* View matrisin setlenmesi

XMVECTOR Eye = XMVectorSet(0.0f, 0.0f, -0.5f, 0.0f); ⇒ bakış noktasının konumu
XMVECTOR At = XMVectorSet(0.0f, 0.0f, 1.0f, 0.0f); ⇒ hangi doğrultu boyunca bakıyoruz
XMVECTOR Up = XMVectorSet(0.0f, 1.0f, 0.0f, 0.0f); ⇒ yukarı doğru dan vektör
g_View = XMMatrixLookAtLH(Eye, At, Up); ⇒ hangi noktadan baktığımızın bilgisi

→ View matrisi üç boyutlu düzlemi hangi açıdan gözlemlediğimizi temsil eder.

* g_Projection = XMMatrixPerspectiveFovLH(XM_PIDIV4 (fovAngleY), 1280/(FLOAT)720 (16/9 oranı), 0.01f, 100.0f);
3 boyutlu düzlemdeki nesnenin görüntü düzlemine izdüşümü alınır.
-ZBuffer ile öndeliğin rengi basılır.
z ekseninde iki düzlem tanımlıyoruz bu iki düzlem arasındaki noktaları render edeceğiz.

* ComPtr<ID3D12Resource> m_constantBuffer; * constantbuffer oluşturma
SceneConstantBuffer m_constantBufferData * bufferdaki veriyi temsil eder *
→ Constant buffer OnInit içerisinde setlenir

ThrowIfFailed(m_device → CreateCommitedResource(
&CD3DX12_HEAP_PROPERTIES(D3D12_HEAP_TYPE_UPLOAD),
D3D12_HEAP_FLAG_NONE,
&CD3DX12_RESOURCE_DESC::Buffer(1024*64) (max boyut),
D3D12_RESOURCE_STATE_GENERIC_READ,
nullptr,
IID_PPV_ARGS(&m_constantBuffer)));

m_constantBufferData.mWorld = XNMatrixTranspose(g_World); *constant buffer
m_constantBufferData.mView = XNMatrixTranspose(g_View); değerlerini
m_constantBufferData.mProjection = XNMatrixTranspose(g_Projection); setleme

OnUpdate() içinde
memcpy(m_pCbvDataBegin, &m_constantBufferData, sizeof(m_constantBufferData));
ile ekran kartındaki constantbuffera yazılır

memcpy(pVertexDataBegin, triangleVertices, sizeof(triangleVertices)); *triangle için memcpy
memcpy triangleVertices bellek bölgesindeki veriyi pVertexDataBegin bellek bölgesine kopyalar

→ onRender() içinde kosulan drawx komutlarında aslında hlsl kodları kosar. Önce VSMain sonra PsMain
→ shaders.hlsl

```
cbuffer SceneConstantBuffer : register(b0)      // cbuffer = constant buffer
{ matrix World;                                  ekran kartındaki b0. buffera yazdığımızı
  matrix View;                                   gösterir
  matrix Projection;
}
struct VSOutput
{ float4 position : SV_POSITION;
  float4 color    : COLOR;
}
VSOutput VSMain(float3 position:POSITION, float4 color: COLOR)
{ VSOutput result;
  result.position = mul(float4(position, 1), World);
  result.position = mul(result.position, View);
  result.position = mul(result.position, Projection);
  result.color = color;
  return result;
}
float4 PsMain(VSOutput input) : SV_TARGET
{ return input.color; }
```

⇒ memcpy ile ekran kartına kopyalanan vertexler bu fonka parametre olarak gelir. Bu işlem input layout setlemesi ile gerçekleşir

* constanbuffer aracılığıyla world, view ve projection matrisleri ekran kartına yollanır memcpy ile

```
// define the vertex input layout
D3D12_INPUT_ELEMENT_DESC inputElementDescs[] {
  {"POSITION", 0, DXGI_FORMAT ...
  {"COLOR", 0 ...
}
```

Alıştırma 1 - Üçgeni kare yapma

1-) Üst köşe noktasını sağa kaydırarak dik üçgen yapma ⇒ XMFLOAT3(1.0f, 1.0f, 1.0f)

2-) Yeni bir üçgen ekleme ⇒ XMFLOAT3(1.0f, 1.0f, 1.0f)
XMFLOAT3(-1.0f, 1.0f, 1.0f)
XMFLOAT3(-1.0f, -1.0f, 1.0f)
} saat yönünün tersinde

3-) DrawInstanced(6, ...) olarak güncelledik (6 vertex var artık)

* XMMatrixRotationY(rotation) ile üçgen Y ekseni etrafında belli bir yarıçapla döner kendi ekseni etrafında dönsün istesek vertexlerin z değerini 0 yapmalıyız.

* D3D12_CULL_MODE_NONE ⇒ BACK FACE yapma
_BACK ⇒ BACK FACE yap
_FRONT ⇒ FRONT FACE'leri alamaz.

Alıştırma 2 - Vertex buffer sayısını 2 yapma

1-) Yeni vertex buffer ve vertexbufferView değişkenleri oluşturduk
2-) OnInit içerisinde setleme yaptık.
3-) Vertex değerlerini üzerine yazmasın diye ötelediik ✓
4-) OnRender içerisinde draw komutu ekleme

Alıştırma 3 - Constant buffer sayısını 2 yapma

Üçgen dönerken sadece world matrisinin değeri değişir ama biz onUpdate içerisinde memcpy ile her bir frame için constantbuffer içindeki world, view ve projection matrisini tekrar tekrar ekran kartına kopyalıyoruz. View ve projection matrisini 1 kez kopyalasak ve sadece değişen world matrisini birden çok kez kopyalasak daha iyi bir yaklaşım olur.

1-) Yeni struct constant buffer oluşturma ⇒ changesEveryFrame, NeverChanges
m_world mView, projection

2-) constant buffer değişkeni oluşturma
3-) OnInit içinde setleme yapma
4-) mWorld, mView, Projection değerlerini uygun struct için setleme
5-) onUpdate içindeki memcpyi sadece changesEveryFrame için yapma
6-) hlsl tarafındaki constant buffer sayısını 2 yapma, register değerleri 2 farklı register olarak setleme
7-) Root signature nesnesi içinde rootparameter[2] yapılır. Artık 2 constant bufferimiz olduğu için. Bunun üyeleri içinde shaderRegister değerleri hlsl tarafındaki register değerleriyle uyumlu yapılır. Constant buffer'ların cpp tarafından hangi registera yazılacağı belirtilmiş oldu.
8-) RootConstantBufferView'in indisleri setlenir.
9-) NeverChanges için memcpy yazma

Alıştırma 4 - Birden çok kare çizme tek vertex buffer ile

Constant buffera veriyi 256 baytlık paketler şeklinde peş peşe yazabilirsiniz.

1. memcpy (...)

2. memcpy (... +256 , ...)

OnRender içinde de +256 yaparak GetVirtualAdress +256

Sadece world matrisini güncelleyerek çizdik.

Karelerin renklerini değiştirmek istersek color değişkenini constant buffera eklemeliyiz 256 baytlık aralıklarla renkleri değiştiriyoruz.

2-Rendering Cube

* Küpü vertex buffer ile render etmek istesek vertex buffer'da 12x3 = 36 vertex tanımlamak gerekir.

Köşe noktaları vertex buffer'da kendini tekrar ederdi. Bu yüzden index buffer kullanıyoruz.

* Index buffer tanımlama

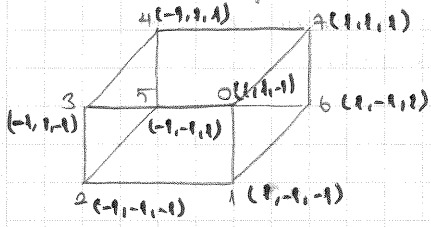
ComPtr<ID3D12Resource> m_indexBuffer;

D3D12_INDEX_BUFFER_VIEW m_indexBufferView;

Index buffer'ı vertex buffer ile birlikte kullanırız.

Vertex buffera her bir köşe noktası bir kez yazılır.

Index buffer'da ise köşe noktaları indexlenerek üçgenler tanımlanır. Her bir üçlü bir üçgeni temsil eder.



DWORD indices[] = {

//FRONT 0,1,2 — 0,2,3

//BACK 4,5,6 — 4,6,7

//RIGHT 7,6,1 — 7,1,0

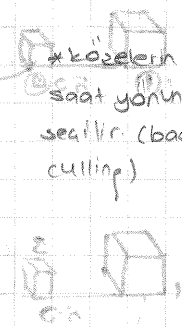
//LEFT 3,2,5 — 3,5,4

//TOP 7,0,3 — 7,3,4

//BOTTOM 5,2,1 — 5,1,6

};

* köşelerin sırası saat yönünde seçilir (backface culling)



→ View değişkenleri birden fazla buffer kullanılacaksa 0 an hangisinin kullanıldığını belirtir.

* m_commandList → IASetIndexBuffer(&m_indexBufferView)

m_commandList → DrawIndexedInstanced (36, 1, 0, 0, 0)

↓ Toplam indis sayısı

*DirectX'te köşe noktaları için renk veriyoruz. Birden çok renk verirsek geçişler kendi kendine yapılır.

3-) Transformations

Rotate - Translate - Scale (ölçekleme)

→ XNMatrixScaling(0.3f, 0.3f, 0.3f) ⇒ her eksende 0,3 küçülttük, bir eksende küçültme yapmak istemiyorsak değerini 1 yaparız.

→ g-World'de çarpma soldan sağa doğru yapılır.

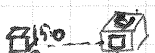
rotation += 0.01;

g-World = XNMatrixRotationY(rotation) ⇒ Y ekseninde kendi etrafında dönen küp

→ XNMatrix mRotate = XNMatrixRotationY(-rotation)

// mTranslate = XNMatrixTranslation(-5.0f, 0.0f, 0.0f);

// mScale = XNMatrixScaling(0.3f, 0.3f, 0.3f);

* g-World = mScale * mTranslate * mRotate ⇒  3 Büyük küpün etrafında döner.

→ orijinden uzaklaştırdığımız için bir çember oluşturacak şekilde döner.

* g-World = mScale * mRotate * mTranslate

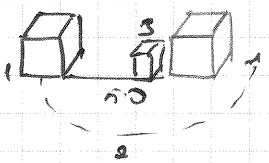
*Büyük küpten 5.0f uzakta kendi etrafında döner.

* g-World = mTranslate * mScale * mRotate

*Büyük küpe yakın bir şekilde dönmeye başlar.

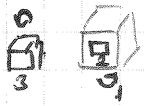
! Cisim orijinden uzakta iken scale yapılırsa orijine olan uzaklıkta scale edilmiş olur.

* g-World = mTranslate * mRotate * mScale;



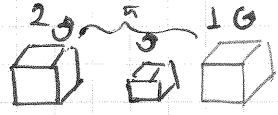
*üstteki ile aynı

* g-World = mRotate * mScale * mTranslate



kendi ekseninde dönen küçük bir küp 3 ile aynı

* gWorld = mRotate * mTranslate * mScale



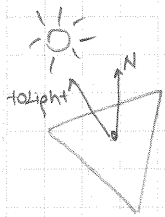
Lighting

VSMain → Vertex shader, vertexler üzerinde koşar.
PSMain → Pixel shader, VSMain ile ekrana izdüşürülen pixelleri boyar.

→ float4 PS_Flat(VSOutput input) : SV_TARGET
{ return MeshColor; } ⇒ Tüm pixelleri aynı renge boyar.

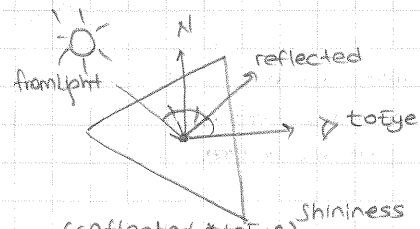
→ Artık VSMain'e parametre olarak üçgenin position ve normal bilgisi gelir.

→ **Diffuse**



toLight * N

→ **Specular**



(reflected * toEye)^Shininess

reflected = Rd - 2Rd * N * N

→ struct VSOutput
{ float4 position : SV_POSITION; ⇒ 2 boyutlu ekran koordinatlarına izdüşürülmüş position
float4 positionW : POSITION; ⇒ 3 boyutlu position değeri, işlemlerde kullanılır.
float4 normal : NORMAL;
}

VSOutput VSMain(float3 position : POSITION, float3 normal : NORMAL) * VSMain'in outputu PSMain'e parametre geçilir.
{ VSOutput result;
result.position = mul(float4(position, 1), World);
result.position = mul(result.position, View);
result. " = " (" " , Projection);
result.positionW = mul(position, World)
result.normal = mul(normal, World)
return result;
}

float4 PS_Phong(VSOutput input) : SV_TARGET {
float3 toLight = normalize(LightPos - input.positionW);
float dotLightNorm = max(dot(toLight, input.normal), 0) ⇒ dot skaler çarpım fonksiyonu max kullanıldı çünkü dot negatif değer alabilir.
float4 diffuseColor = dotLightNorm * MeshColor

diffuse color

// specular color

```
float3 fromLight = normalize(input.positionW - lightPos);
float3 toEye = normalize(EyePos - input.positionW);
float3 reflected = fromLight - 2*dot(fromLight, input.normal) * input.normal;
float dotEyeReflected = max(dot(toEye, reflected), 0);
float4 specularColor = pow(dotEyeReflected, 32.0f) * lightColor;
                                                  ↓ shininess
ambient color.
return 0.2 * MeshColor + 0.5 * diffuseColor + 0.3 * specularColor; ⇒ ShadingModel()
          cismin kendi
          rengi
}
```

PS_FLAT ⇒ cismin için kullanılmış.

* Pipeline state nesnesini kullanarak cisimleri render ederken hangi pixel shader'ı hangi vertex shader'ı kullanacağımızı setleriz.

```
ComPtr<ID3DBlob> vertexShader;
//               pixelShaderPhong;
//               pixelShaderFlat;
```
} pointer değişkenler.

2 tane pipeline state nesnesi oluşturduk.

1. içinde

```
psoDesc.PhongVS = CD3DX12_SHADER_BYTECODE(vertexShader.Get());
//      .PS =                             (pixelShaderPhong.Get());
```

2. içinde 1.'nin kopyası alınıp PS değiştirilir.

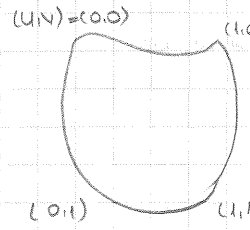
```
      .PS =                             (pixelShaderFlat.Get());
```

WIREFRAME ile cismin sadece kenarlarını çizdik.

onRender() içinde hangi pipeline state'i kullandığımızı setleriz.

Texture Mapping

Doku için default u,v değerleri vardır



küpün köşe noktaları için u,v değerlerinin setlenmesi yeterlidir, küpün her yüzeyi için hesaplama gerekli değildir

```
struct Vertex {                    ⇒ texture parametresi (u,v) koordinatlarını tutar
    XMFLOAT3 position;
    XMFLOAT3 normal;
    XMFLOAT2 texture;
};
ThrowIfFailed(DirectX::CreateDDSTextureFromFile(   ⇒ texture'ı ekran kartının ilgili
    m_device.Get(),                                  bufferına yüklememizi sağlar.
    m_commandList.Get(),
    L"...", "dds formatında olmalı
    textureBuffer,                                 ⇒ textureBuffer pointer değişkeni
    textureBufferUploadHeap));                       aracılığıyla ekran kartına yükleyiz.
```

Shaders.hlsl

```
Texture2D textureMap : register(t0);   ⇒ texture 0 nolu registera yazılacak
                                         RootSignature'da setlemesi yapılır.
⇒CD3DX12_DESCRIPTOR_RANGE1 range(... , 1, 0, 0 ... )
rootParameters[1].InitAsDescriptorTable(1, &range, ...)
OnRender içinde:
m_commandList->SetGraphicsRootDescriptorTable(1, m_descriptorHeap->GetGPU...());

SamplerState sampleLinear : register(s0);  ⇒texture için bir filtre uygulanacaksa o
                                             filtreyi tutar
VSMain(float3 position : POSITION, float3 normal : NORMAL, float2 tex : TEXCOORD)
{
    ...
    result.tex = tex;
}
float4 PSMain(VSOutput input) : SV_TARGET {                    (u,v)
    float4 ShapeColor = textureMap.Sample(sampleLinear, input.tex);
                        Registerdaki textureden Sample fonk. ile örnekler alır.
```

Stenciling

* Back buffer - front buffer:
Çizim komutlarımızla back buffera çizim yaparız. İşlemler bitene kadar içeriği ekrana yollamayız o sırada front bufferı görürüz. Present fonku koştuktan sonra backbuffer front buffer olur, front bufferda back buffer olur.
Böylece çizimler arası kapukluk olmaz.
Swapchain nesnesi bufferları kontrol eder.
Backbuffer her bir pixel için RGBA değerleri tutar.

* Stencil buffer:
Ayrısallıştırılmış modeller, görünürlüğü back buffer ile aynıdır. Her pixel için 8 bitlik bir değer tutar (0-255 arası).
Stencil bufferdaki değerlere bağlı olarak bazı bileşenleri ayrı olarak işaretleriz.

Ortamdaki cisimler back buffera çizilirken ayna çizimi sırasında back buffer daki ayraya karşılık gelen pixellerin stencil bufferdaki karşılıklarına 0-255 arası int sayı yazılır.
Yansımalar çizilirken stencil bufferda int değer içeren pixellerin back bufferdaki karşılıklarına yansımalar çizilir.

m_commandList->OMSetStencilRef(61) ⇒ stencil bufferda aynanın olduğu pixellere 61 sayısı yazılır

m_commandList->SetPipelineState(m_pipelineState_markMirrors.Get());

m_commandList->ClearDepthStencilView(...,0,...) ile önce stencil bufferdaki değerler 0 ile clearlanır

Yansımaların çizilmesi:

m_commandList->OMSetStencilRef(61)

" ->SetPipelineState(m_pipelineState_drawReflections.Get())

⇒ Sadece 61 yazılan pixellere çizim yapılmasını sağlar.

→ Pipeline state'in içi (markMirror)

* mirrorDSS.FrontFace.StencilFunc = D3D12_COMPARISON_FUNC_ALWAYS (hep true döner.)

⇒ Bu fonksiyon OMSetStencilRef(61) deki değer ile stencil bufferdaki o anki değeri karşılaştırır. Bool değer döndürür.
StencilFunc true dönmesi durumunda koşar:

* mirrorDSS.FrontFace.StencilPassOp = D3D12_STENCIL_OP_REPLACE;

⇒ Replace ile aynayı temsil eden pixellere 61 yazar.

* drawReflection

→ reflectionDSS.FrontFace.StencilFunc = D3D12_COMPARISON_FUNC_EQUAL;

⇒ stencilRef ve stencil buffer'daki değerleri karşılaştırır eşit ise true döndürür.
Aynaya sahip pixeller için true döner.
Stencil func true ise kalsın.

→ // // StencilPassOp = D3D12_STENCIL_OP_KEEP; ⇒ aynayı temsil eden pixellerin değerini sabit tutar, replace olsaydı olurdu çünkü stencil Ref'in değeri 01 ise yine aynı şeyi yazacak.

R transformasyon matrisi ile yansımayı mirror planın arkasına taşırız

A B C D Ax+By+Cz+D=0 ⇒ N(A,B,C)

XMVECTOR mirrorPlane = XMVectorSet(0.0f, 0.0f, -1.0f, 0.0f) ⇒ 0x+0y-z+0=0 z=0 düzlemi

XMMATRIX R = XMMatrixReflect(mirrorPlane) ⇒ bir transformasyon matrisi üretir.

→ Aynanın z değerleri 0 old. için

m_constantBufferData.mWorld = XMMatrixTranspose(mTranslation * R)

W-A-S-D tuşları ile hareketi verir.

memcpy(- - -)

* küpün gölgesi

XMVECTOR shadowPlane = XMVectorSet(0.0f, 1.0f, 0.0f, 2.0f) y=-2 düzlemi

ground'un y değerleri -2 küpün konumu

XMVECTOR toLight = XMVectorSet(m_constantBufferData.mLightPos.x - Translation.x,
// // .y - // .y,
// // .z - // .z, 0);

XMMATRIX S = XMMatrixShadow(shadowPlane, toLight); ⇒ Transformasyon matrisi döndürür.

XMMATRIX shadowOffsetY = XMMatrixTranslation(0.0f, 0.01f, 0.0f)

gölge tam olarak zemin ile çakışmasın diye y ekseninde 0.01 yukarı kaydırıyoruz.

m_constantBufferData.mWorld = XMMatrixTranspose(mTranslation * S * shadowOffsetY);

* küpün gölgesinin yansıması

m_constantBufferData.mWorld = XMMatrixTranspose(mTranslation * S * shadowOffsetY * R);

RENDERING MAYA MODELS

alt + mouse sol button ile döndürerek değişik açılardan bakılabilir.
space tuşu + pers = diğer görüntüler
alt + tekerlek = kaydırma
move tool / scale tool / rotate tool
extrude tool

→ Modeli kare içine alıp move tooluna basınca karşımıza 3 eksen çıkar. Bu eksenlerde move yapabiliriz.
→ Rotate tooluna basınca eksenler çemberler olarak karşımıza çıkar. Döndürme işlemi yapabiliriz.
→ scale tooluyla verilen 3 eksende scale yapabiliriz.
Cisim üzerinde sağa tıklayınca farklı mod seçenekleri çıkar karşımıza cismi yönlendirebileceğimiz. Default object mode
face ile yüzeyleri seçebiliriz.
edge → kenar
vertex → köşe (shift tuşuyla birden çok seçim yapılabilir)

extrude o anda seçilenin kopyasını çıkarır.

ctrl+d → seçilen modelin kopyasını oluşturur
mesh → triangulate → üçgenlere çevirme
mb → maya binary
→ Mayada z ekseni directX'e ters.

OBJ dosya formatı:

→ Dosya içinde 8 tane vertex satırı var (v). Bunlar küpün köşe noktaları (x,y,z) v.
vt: vertex texture, doku koordinatları → 14 tane
vn: vertex normal → 24 tane
f: face yani üçgenler. F kısımları index buffer
s: surface (yüzey)

f 1/1/1 2/2/2 3/3/3 *indisler 1'den başlar
V0 V1 V2

vertex indisi / texture indis / normal indisi
f 1,8 7 f 1,14 7 f 1,24 7

→ Objeden load ederken z değerlerinin negatifi alınır.
→ Köşe sırası ccw → cw dönüşüm gerekli. Köşe noktalarının sırası değiştirilmeli.
→ While döngüsü ile dosyanın tamamı gezilir.
'v' → vertexCount artır
'vt' → textureCount++
'vn' → normalCount++
'f' → faceCount++
} Bu boyutlarda dizi oluşturulur.

→ indisler 0'dan başlanacak şekilde setleme yapılır.

Vertices-Modelde tüm bilgiler tutuluyor

*DirectX'te koordinatlar sol el kuralına göre belirlenir

DirectX → LH

Maya, OpenGL → RH

V0-V1-V2 için Vertices Modele kopyalama yaptık.
→ face index 6*2 'den 12'ye kadar döner
→ OBJ_Loader bir kez koşsun diye onun içerisinde yapıyoruz

STL Dosya Formatı

Olmayan dosya formatını etkinleştirme

Windows
↳ settings / Preferences
↳ Plugin Manager

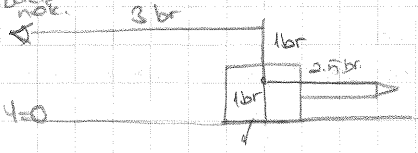
→ maya export STL scripti ile .stl (ASCII) formatına export etme
→ Scriptler ile mayaya yeni menüler eklenebilir. Mel (maya element language), python save script to shell ile script mayaya menü olarak eklenir.
→ stl dosya formatında texture yok.
face'in normalinin x,y,z koord. ve V0,V1,V2'nin x,y,z koord. tutulur → Bir üçgen

STL_Loader
→ STL'i yüklerken texture yükleyemediğimiz için PsMalnde değişiklik yapılır

Tank Oyunu

space tuşu ⇒ mermiyi ateşleme
R tuşu ⇒ Reload

→ First person shooter: Oyuncunun gözünden sahne render edilir. Arabamın içi gibi
→ Third person shooter: Kamera oyuncunun arkasında
→ Tank Position



Tank_Position = Eye + 3* (At-Eye) + XMVector3Set(0,2,0,0)
2 birim yaşağı
nereye bakıyorum
nereden bakıyorum

* At-Eye ⇒ Namlunun işaret ettiği doğruttu.
* Bakış noktasıyla tank aynı doğrultuda olsun diye At-Eye kullanılır.

* XMVECTOR ~ XMFLOAT4
x,y,z erişimi mümkün değil
x,y,z erişim mümkün

⇒ XMStoreFloat4() XMVECTOR'den XMFLOAT4'e dönüşüm yapar.
⇒ XMLoadFloat4() XMFLOAT4'ten XMVECTOR'e dönüşüm yapar.

* Tankın pozisyonu XNVector old. için parametrelerine erişmek için dönüşüm yaparız
XNFLOAT4 Tank_Position_Float4;
XNStoreFloat4(&Tank_Position_Float4, Tank_Position);

* Böylece tankı nereye çizeceğimizi bulduk
XNMATRIX mTranslate_Tank = XNMatrixTranslation(Tank_Position_Float4.x, T.P.Fay, T.P.z)

g_World_Tank = camRotationMatrix * mTranslate_Tank;
mouse'un sol tuşu ile yaptığımız rotasyon, tankı döndürmek için kullanırız. Tank sadece X ve Y eksenlerinde döner. (z=0)

⇒ camRotationMatrix = XNMatrixRotationRollPitchYaw(camPitch, camYaw, 0)
onMouseMove'da güncelliyoruz

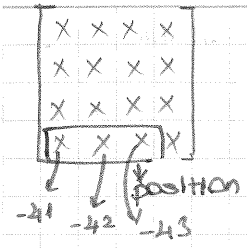
onMouseMove:
ML_LBUTTON ⇒ sol butona tıklanmışsa
O anki ve bir önceki tıklanmanın farkını alır.
Koordinatların farkının pozitif veya negatif olmasına göre sağa veya sola hareket belirlenir.
Eklenenler dönme açısı

* Mermiyi konumlandırılması
Ro_Tank_Missile = Eye + XMVectorSet(0,-1,0,0); ⇒ bakış noktasının 1 birim altında
Rd_Tank_Missile = XMVector3Normalize(At - Eye);
⇒ Mermiyi ateşlenmediği durum.
If(FireTankMissile) {
XMVECTOR initialPosition = Ro_Tank_Missile + (3+2.5) * Rd_TankMissile;
⇒ Mermiyi namlunun ucuna taşıdık
⇒ g_World_Missile ⇒ mermi için yeni world matrisi

g_World_Missile = g_World_Tank; ⇒ Tank ile mermi aynı rotasyona sahip olsun, tank nereye işaret ediyorsa mermide oraya işaret etsin

XNMATRIX ~ XMFLOAT4X4 ⇒ Dönüşüm yaparak erişiriz.
erişim yok erişim var

4x4 world matris



g-World_Missile matrisi XMMATRIX türünde olduğu için tür dönüşümleri yaparak parametrelerine erişebilelim diye

```
XMStoreFloat4x4(&g-World-Missile-4x4, g-World-Missile);
g-World-Missile-4x4._41 = initialPosition.Fx.x;   } Nesnenin
//                 ._42 =  //          .y;        } pozisyon
//                 ._43 =  //          .z;        } bilgilerini
                                                   aktardık
g-World-Missile = XMLoadFloat4x4(&g-World-Missile-4x4);
```

* XMStoreFloat4x4() => XMMatrix'ten XMFloat4x4'e dönüşüm yapar.
  XMLoadFloat4x4() => XMFloat4x4'ten XMMatrix'e " "

* FireTankMissile başlangıçta true, TraceTankMissile başlangıçta false birbirlerine eşittirler.

Space tuşuna basana kadar FireTankMissile bloğu koşar
" " basınca FireTankMissile false, TraceTankMissile true yapılır.

TraceTankMissile kod bloğu mermiyi tankın doğrultusu boyunca ilerletir.

```
XMFLOAT4 Rd_Tank_Missile_Float4;
XMStoreFloat4(&Rd_Tank_Missile_Float4, Rd_Tank_Missile);

if(TraceTankMissile){                          → hızı belirler
    XMStoreFloat4x4(&g-World-Missile-4x4, g-World-Missile);
    g-World-Missile-4x4._41 += 0.1 * Rd_Tank_Missile_Float4.x;  } pozisyonunu
//                  ._42 +=                      .y;            } güncelliyoruz. Yani
//                  ._43 +=                      .z;            } mermiyi ilerletiyoruz.
    g-World-Missile = XMLoadFloat4x4(&g_World_Missile_4x4);
}
```

* Yolladığımız merminin düşman tankına vurup vurmadığını anlama

```
Intersections = testIntersections(Ro_Tank_Missile, Rd_Tank_Missile, g-World-Enemy);
                                                                    düşman tankı hareketsiz ise önemli değil
```

```
vector<Intersect> testIntersections(XNVECTOR Ro, XNVECTOR Rd, XNMATRIX g_World_Enemy){

float t_Ground = IntersectTriangle(Ro, Rd, vertices_Ground, vertexCount_Ground, XNMIdentity());
float t_Walls = IntersectTriangle(Ro, Rd, vertices_Walls, vertexCount_Walls, XNMIdentity());
float t_Enemy = I.T.(Ro, Rd, vertices_Tank, vertexCount_Tank, g_World_Enemy);
```

// Karşımıza çıkacak cisimleri 3 farklı sınıfa ayırırız, sebebi mermi bu sınıflardan hangisi ile kesişecek
3 sınıf ile de kesişim testi yapılarak t değerleri bulunur

```
struct Intersect{
    float t = 0.0F
    bool isWall = false;     => yolladığımız ışın duvarla mı kesişiyor.
    bool isEnemy = false;    //  //  düşmanla mı  //
} Intersected;

if(t_Ground > 1.0){          => kesişiyorsa değeri 1'den büyük.
    Intersected.t = t_Ground;
    Intersections.pushback(Intersected);
}
if(t_Walls > 1.0){
    Intersected.t = t_Walls;
    Intersected.isWall = true;
    Intersections.push_back(Intersected);
}
if(t_Enemy > 1.0){
    Intersected.t = t_Enemy;
    Intersected.isEnemy = true;
    Intersections.push_back(Intersected);
}
```

* Birden çok kesişim olabilecepi için en yakın olanı bulmak istiyoruz:
nearestObject(Intersections) => en küçük t uzaklığını bulur.

* Nişan aldığımızda hem tankla hem duvarla veya yerle kesişebilir. En yakına sarı imleci alamamız gerekir t uzaklığı bunu tutar.

```
float RedDot_Distance = nearest.t
```

XNVECTOR RedDot_Position = Ro_Tank_Missile + RedDot_Distance * Rd_Tank_Missile; R = Rott*Rd

(Ro_Tank_Missile: mermin başlangıç konumu; Rd_Tank_Missile: mermin doğrultusu; RedDot_Position: Sarı imlecin konumu)

* RedDot_Position eğer tankın üzerine konumlanmışsa ve ateş edilmişse Missile_Position ve RedDot_Position aynı ise memi tanka çarpar.

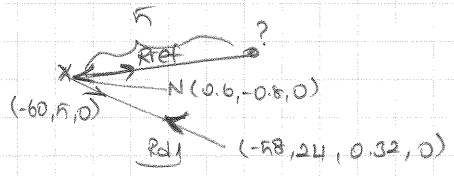
Missile_RedDot_Distance Missile_Position ve RedDot_Position arasındaki farkı tutar.
XMVector3Length fonku uzaklığı hesaplar vektör döndürür X,Y,t değerlerinde her birinde uzaklık tutulur. XMVectorGetX ile bir değeri alırız.

nearest.IsEnemy -> Düşmana mı dokunuyor
Çarpma sonucu OnRender() içinde
renderEnemy = false
renderTankMissile = false } yani artık çizilmezler.

if (FireEnemyMissile) renderEnemyMissile = false ⇒düşman bize ateş etmemişse onun mermisinide yok ederiz.

2015 Final A

1-) Topun direpe çarptığı nokta (-60,5,0), direkten dönüp yerden sektiği nokta (-58.24, 0.32, 0) direğin normali N(0.6,-0.8,0), şutun çekildiği noktanın direpe uzaklığı 5 m. Şut hangi noktadan çekilmiş.

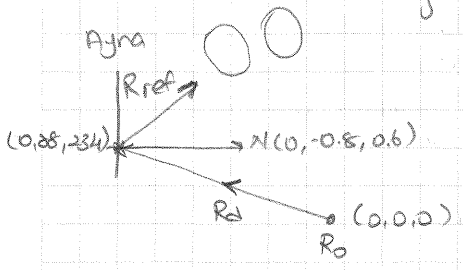


Rd1 = ((-60,5,0) - (-58,24, 0.32, 0)) Normalize()
= (-1.76, 4.68, 0) .Normalize()
= (-0.352, 0.936, 0)

Rref = Rd - 2(Rd * N) * N
= (-0.352, 0.936, 0) * (0.6, -0.8, 0) = -0.96
= 2 * -0.96 = -1.92
= -1.92 * (0.6, -0.8, 0) = (-1.152, 1.536, 0)
= (-0.352, 0.936, 0) - (-1.152, 1.536, 0) = (0.8, -0.6, 0)

R = Ro + t * Rd = (-60,5,0) + 5 * (0.8, -0.6, 0) = (-56, 2, 0)

2-) $P_0(0,0,0)$ noktasından çıkan bir ışın $N(0,-0.8,0.6)$ normaline sahip bir ayna üzerindeki $(0,88,234)$ noktasından yansıyıp kırmızı ve mavi renkli 2 küre ile kesişiyor. Kırmızı kürenin merkezi $C_k(0,148.8,319.6)$ yarıçapı $r_k=20$ mavi kürenin $C_M(0,189.6,265.2)$ $r_M=45$ dir. Ayna üzerindeki noktada bu kürelerin hangisinin yansıması pozitiktir.



$R_d = ((0,88,234) - (0,0,0)).N() \Rightarrow$ boyu 250

$= (0, 0.352, 0.936)$

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

$= (0,0.352,0.936) * (0,-0.8,0.6)$

$= -2 * 0.28 \Rightarrow -0.56$

$= -0.56 * (0,-0.8,0.6)$

$= (0,0.352,0.936) + (0,0.448,-0.336)$

$= (0,0.8,0.6)$

Intersect fonku için $R_o=(0,88,234)$ ve $R_d=(0,0.8,0.6)$

Kırmızı küre için

Vertex $l = (0,148.8,319.6) - (0,88,234) = (0,60.8,85.6)$

$s = (0,60.8,85.6) * (0,0.8,0.6) = 100$

$l2 = 11024$

$r2 = 1600$

$s2 = 10000$

$m2 = 1024$

$q = 24$

return $100 - 24 = 76$

mavi küre için

$l = (0,189.6,265.2) - (0,88,234) = (0,101.6,31.2)$

$s = 100$

$l2 = 11296$

$r2 = 2025$

$s2 = 10000$

$m2 = 1296$

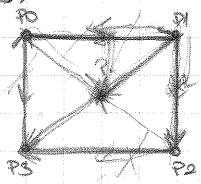
$q = 29$

return $100 - 29 = 93$

Mavi kürenin t değeri daha küçük old. için yani aynaya daha yakın old. için mavi küreyi görürüz.

2022 Final

1-)



Şekildeki P0, P1, P2, P3 köşe noktalarına bağlı olarak görüntü düzleminin merkezini hesaplayan aşağıdaki ifadelerden yanlış olanı işaretleyiniz.

A) $P_0 + (P_1 - P_0) + 0.5(P_3 - P_1)$ ✓

B) $P_1 + (P_2 - P_1) + 0.5(P_0 - P_2)$ ✓

(c) $P_2 + (P_3 - P_2) + 0.5(P_3 - P_1)$ → yönleri aynı

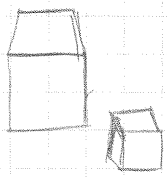
D) $P_3 + (P_0 - P_3) + 0.5(P_2 - P_0)$ ✓

2-) Rotlin = XNMatrixRotationY(XN.PI/2) CW 45

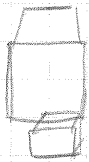
Trans = XNMatrix Translation(5.0f, 0.0f, 20f);

Scale = XNMatrix Scaling(0.5f, 0.5f, 0.5f);

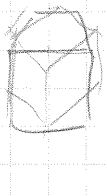
g-World_1 = Rotlin * Trans * Rotlin * Scale



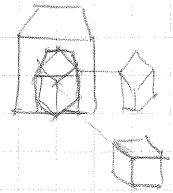
g-World_2 = Trans * Rotlin * Scale * Rotlin; ✓



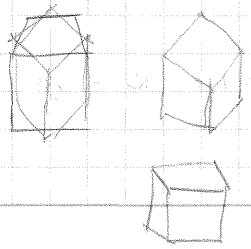
g-World_3 = Rotlin * Scale * Trans * Rotlin



g-World_4 = Scale * Rotlin * Trans * Rotlin



g-World_4 = Rotlin * Trans * Scale * Rotlin



2019 Final

3) X için 12 üçgen ⇒ 36 vertex

1 vertex → 7 float → 7*4 byte ⇒ 28

36 * 28 ⇒ X = 1008

Y için

8 vertex → 8*28 = 224

her yüzey için (DWORD)

6 yüzey → 6 index 6*6*4 = 144

Y = 224 + 144 = 368

X - Y = 368

4) OBJ Dosyasının STL hali nedir?

```
solid
  facet normal  0.0  1.0  0.0
    outer loop
      vertex    -0.5  0.0  0.5
      vertex     0.5  0.0  0.5
      vertex    -0.5  0.0 -0.5
    endloop
  endfacet
  facet normal  0.0  1.0  0.0
    outer loop
      vertex    -0.5  0.0 -0.5
      vertex     0.5  0.0  0.5
      vertex     0.5  0.0 -0.5
    endloop
  endfacet
endsolid
```

5)

P0(-10, 10, 20)
P1(10, 10, 20)
P2(10, -10, 20)
P3(-10, -10, 20)
P4(10, 10, 60)
P5(-10, 10, 60)
P6(-10, -10, 60)
P7(10, -10, 60)

CW Rotation around Y-axis

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

Köşe noktalarının koordinatları yukarıda verilen bir küp (0,0,1) doğrultusu boyunca 20 birim ilerledikten sonra y ekseninde saat yönünde 90 derece dönüyor ve bu sefer (1,0,0) doğrultusu boyunca 60 birim ilerliyor. Yeni köşe noktalarının yeni koordinatları

+z → 20 birim

+z → 20 birim		merkeze taşı (P-center)	döndür y
(-10, 10, 80)		(-10, 10, -10)	(-10, 10, 10)
(10, 10, 80)		(10, 10, -10)	(-10, 10, -10)
(10, -10, 80)		(10, -10, -10)	(-10, -10, -10)
(-10, -10, 80)		(-10, -10, -10)	(-10, -10, 10)
(10, 10, 100)	Center = (0, 0, 90)	(10, 10, 10)	(10, 10, -10)
(-10, 10, 100)		(-10, 10, 10)	(10, 10, 10)
(-10, -10, 100)		(-10, -10, 10)	(10, -10, 10)
(10, -10, 100)		(10, -10, 10)	(10, -10, -10)

$$\begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Eski konuma taşı (P+center)	+x → 60 br
(-10, 10, 100)	(50, 10, 100)
(-10, 10, 80)	(50, 10, 80)
(-10, -10, 80)	(50, -10, 80)
(-10, -10, 100)	(50, -10, 100)
(10, 10, 80)	(70, 10, 80)
(10, 10, 100)	(70, 10, 100)
(10, -10, 100)	(70, -10, 100)
(10, -10, 80)	(70, -10, 80)

2019 Bütünleme

1-) DirectX onRender()'da back buffer içeriğini ekranda görüntülemek için Swap Chain nesnesinin hangi fonk.u koşar. → **Present**

2-) HLSL programı ile tüm pixel shader fonku içerdiğinde onRender()'da o anda render edilecek cismin bu pixel shaderlardan hangisi ile render edileceği hangi nesnede setlenir? → **Pipeline State Object**

3-) HLSL kodunda register(b0) ile World, View, Projection matrislerinin sıfırıncı constant bufferdan okunacağı CPP programında hangi nesnede setlenir?
→ **Root Signature**

4-) STL Vertex index / texture index / normal index

```
facet normal  0.0   1.0  0.0
  vertex     -0.5   0.0  0.0
  vertex      0.5   0.0  0.5
  vertex      0.5   0.0  0.0
facet normal  0.0   1.0  0.0
  vertex     -0.5   0.0  -0.5
  vertex      0.5   0.0  0.0
  vertex      0.5   0.0  -0.5
```

+z 30 br	Merkeze taşı (P-Center)	90° döndür (Y)
5-) (-10,10,90)	(-10,10,-10)	(-10,10,10)
(10,10,90)	(10,10,-10)	(-10,10,-10)
(10,-10,90)	(10,-10,-10)	(-10,-10,-10)
(-10,-10,90)	(-10,-10,-10)	(-10,-10,10)
(10,10,110)	(10,10,10)	(10,10,-10)
(-10,10,110)	(-10,10,10)	(10,10,10)
(-10,-10,110)	(-10,-10,10)	(10,-10,10)
(10,-10,110)	(10,-10,10)	(10,-10,-10)

→ Center (0,0,100)

eski konuma taşı (P+center)	+x 100 br	Merkeze taşı (P-center)
(-10,10,110)	(90,10,110)	(-10,10,10)
(-10,10,90)	(90,10,90)	(-10,10,-10)
(-10,-10,90)	(90,-10,90)	(-10,-10,-10)
(-10,-10,110)	(90,-10,110)	(-10,-10,10)
(10,10,90)	(110,10,90)	(10,10,-10)
(10,10,110)	(110,10,110)	(10,10,10)
(10,-10,110)	(110,-10,110)	(10,-10,10)
(10,-10,90)	(110,-10,90)	(10,-10,-10)

center (100,0,100)

90° döndür (Y)	eski konuma taşı (P+center)	-x 100 br
(10,10,10)	(110,10,110)	(110,10,10)
(-10,10,10)	(90,10,110)	(90,10,10)
(-10,-10,10)	(90,-10,110)	(90,-10,10)
(10,-10,10)	(110,-10,110)	(110,-10,10)
(-10,10,-10)	(90,10,90)	(90,10,-10)
(10,10,-10)	(110,10,90)	(110,10,-10)
(10,-10,-10)	(110,-10,90)	(110,-10,-10)
(-10,-10,-10)	(90,-10,90)	(90,-10,-10)

2018 Final

1-) DirectX uygulamalarında kullanılan buffer'lardan bazıları .cpp programı ile RAM'de oluşturulup içeriği ekran kartına kopyalanır. Hangisi bunlara bir örnek değildir?

→ **Back buffer**: Render hedefi olan ve ekrana çıkartılacak görüntüyü tutan buffer'dır. Bu buffer doğrudan ekran kartında (GPU) oluşturulur ve işlenir.

2-) DirectX uygulamalarında kullanılan buffer'lardan bazıları sadece int/float sayı tutar. Bazıları sadece renk bilgisi tutar. Bazılarında hem int/float sayı hem de renk bilgisi tutulabilir. Hangisi her ikisinide tutulabilen buffer'a bir örnektir?

→ **Vertex buffer**

Index buffer ⇒ sadece integer

Texture buffer ⇒ genellikle renk veya doku verileri

Depth buffer ⇒ Derinlik bilgisi (float) ⇒ her pixel için z değeri (derinlik) saklar.

Stencil buffer ⇒ integer.

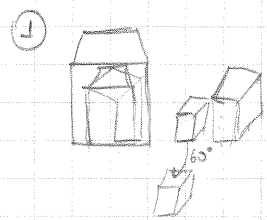
3-) Rot30 = XNMatrixRotationY(XN_PI/6)

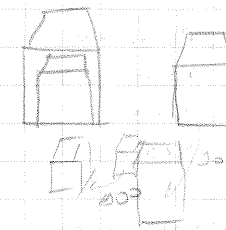
Rot60 = XNMatrixRotationY(XN_PI/3)

Trans = XNMatrixTranslation(5.0f, 0.0f, 0.0f);

Scale = XNMatrixScaling(0.5f, 0.5f, 0.5f),

g-World-1 = Scale * Rot30 * Trans * Scale * Rot60





g-World-2 = Rot30 * scale * Trans * Rot60 * Scale → Aynı

g-World-3 = Scale * Trans * Rot30 * Scale * Rot60

g-World-4 = Rot30 * Scale * Trans * Scale * Rot60 → Aynı

g-World-5 = Scale * Rot30 * Trans * Rot60 * Scale → Aynı

4-) merkete doğru scale: küçültme

merkezden dışarı scale: büyütme

D C B D A D E D A C B

5-) W → 50 birimle

Camera (0,0,50)	kamera etrafında çevirmek için taşı (P-camera)	A → CCW 90° → $\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$
P0(-8, 4.5, 60)	P0(-8, 4.5, 10)	P0(-10, 4.5, -8)
P1(8, 4.5, 60)	P1(8, 4.5, 10)	P1(-10, 4.5, 8)
P2(8, -4.5, 60)	P2(8, -4.5, 10)	P0(-10, -4.5, 8)
P3(-8, -4.5, 60)	P3(-8, -4.5, 10)	P3(-10, -4.5, -8)

Eski konuma taşı (P+camera)

P0(-10, 4.5, 42)

P1(-10, 4.5, 58)

P2(-10, -4.5, 58)

P3(-10, -4.5, 42)

* Kamera +t yönüne bakıyorken CCW 90° dönerse -X yönüne döner. (-1,0,0) → 95 birimle

P0(-85, 4.5, 42)

P1(-85, 4.5, 58)

P2(-85, -4.5, 58)

P3(-85, -4.5, 42)

2018 Bütünleme

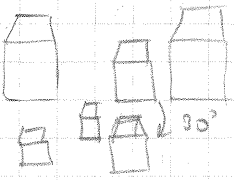
1-) DirectX bufferlarından hangisi üretilecek görüntü ile aynı çözünürlükte olmayabilir? → **Texture buffer** : çözünürlüğü doku boyutlarına bağlıdır ve ekran çözünürlüğünden farklı olabilir.

2-) DirectX buffer çiftlerinden hangisi yalnızca integer değer tutar?
→ **Index , Stencil buf.**

3-) Swap chain nesnesi aşağıdaki buffer çiftlerinden hangisini kullanır?
→ **Back , Front buf.**

4-) `Rot30 = XMMatrixRotation(XM_PI / 6);`
`Rot60 = XMMatrixRotation(XM_PI / 3);`
`Trans = XMMatrixTranslation(5.0f, 0.0f, 0.0f);`
`Scale = XMMatrixScaling(0.5f, 0.5f, 0.5f);`

aynı
- g-World-1 = Rot30 * Scale * Rot60 * Scale * Trans ⇒ 
- g-World-2 = Trans * Scale * Rot30 * Scale * Rot60 ⇒

aynı
- g-World-3 = Scale * Rot30 * Scale * Rot60 * Trans 

- g-World-4 = Trans * Rot30 * Scale * Rot60 * Scale

g-World-5 = Scale * Trans * Rot30 * Scale * Rot60 → farklı

5-)

2017 Final

1-) Bakış noktası (0,-4,91) ve (0,-12,99) old. N(0,-0.8,0.6) normaline sahip üçgenin backface olup olmadığını belirleyiniz. $U_0(0,60,180)$ $U_1(-60,0,100)$ $U_2(60,0,100)$

toEye1 = (0,-4,91) - (0,60,180) = (0,-64,-89)

N * toEye < 0 ⇒ backface

(0,-0.8,0.6) * (0,-64,-89) = -2.2 < 0 ⇒ backface

toEye2 = (0,-12,99) - (0,60,180) = (0,-72,-83)

(0,-0.8,0.6) * (0,-72,-83) = 9.8 ⪈ 0 → frontface

3-) Phong, Textured, Solid gibi 1'den fazla pixel shader olduğunda PSO'nun PS değişkeni ilgili pixel shadera settenerek farklı PSO'lar tanımlanır. Render() fonksiyonunda o anda çizilecek cisim Phong, Textured, Solid modlarından hangisinde çizilecekse o mode ait PSO SetPipelineState() ile settenir.

4-)

2024 ~~Final~~ Arasınav

1-) 1. kesişim

l = (88, 6, 0)

s = (88, 6, 0) * (1, 0, 0) = 88

l2 = (88, 6, 0) * (88, 6, 0) = 7780

r = 100

s2 = 7744

m2 = 36

q = 8

return s-q = 80

iPoint = (-80, 60, 0) + 80* (1, 0, 0) = (0, 60, 0)

Normal = (iPoint-center)/Radius = (-8, -6, 0) / 10

= (-0.8, -0.6, 0)

Rref = Rd - 2* (Rd * N) * N

= -0.8 * -2 = 1.6

⇒ (-0.8, -0.6, 0) * 1.6 = (-1.28, -0.96, 0)

⇒ (1, 0, 0) + (-1.28, -0.96, 0) = (-0.28, -0.96, 0)

Yeni Ro (0, 60, 0) Rd (-0.28, -0.96, 0)

2. kesişim

l = (-38, -96, 0)

s = (-38, -96, 0) * (-0.28, -0.96, 0) = 102.8

l2 = 10660

r2 = 100

s2 = 10767.84

m2 = 92.16

q = $\sqrt{12.84}$ = 2.8

t = 100

iPoint = (0, 60, 0) + 100 * (-0.28, -0.96, 0)

= (-28, -36, 0)

N = (1, 0, 0)

Rref = (0.28, -0.96, 0)

Yeni Ro (-28, -36, 0) Rd (0.28, -0.96, 0)

l = (14, -48, 0)

s = 50

l2 = 2500

r2 = 625

s2 = 2500

m2 = 0

q = 25

s-q = 25

iPoint = (-28, -36, 0) + 25* (0.28, -0.96, 0) = (-21, -60, 0)