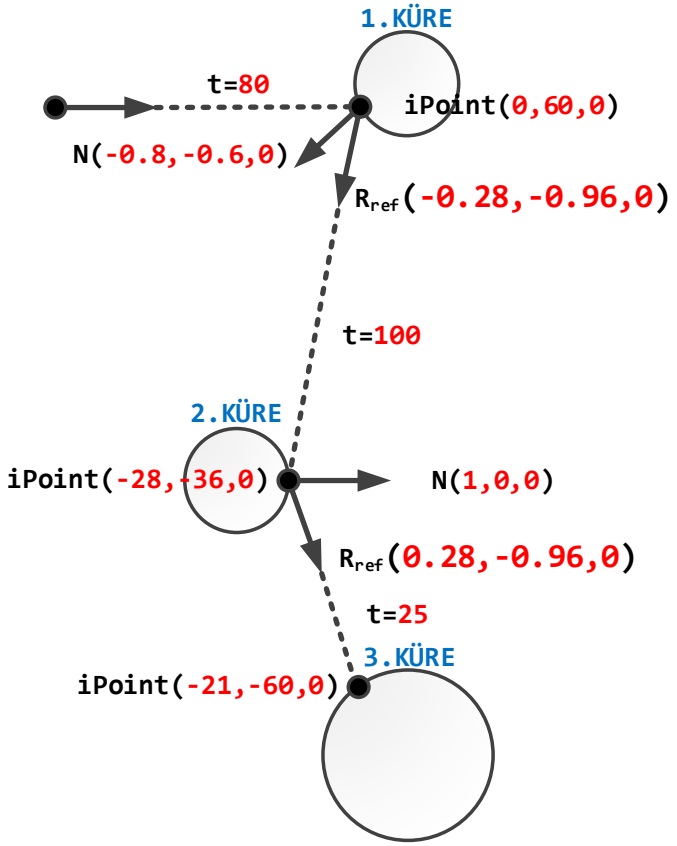
CEVAPLAR

```
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;
    float s2       = s * s;
    float m2       = l2 - s2;
    if (m2 > r2)   return 0;
    float  q       = (float)Math.Sqrt(r2 - m2);
    if (l2 > r2)   return s - q;
    else return s + q;
}
```

1) **R_o(-80,60,0)** noktasından **R_d(1,0,0)** doğrultusu boyunca giden bir ışın, merkezi **c(8,66,0)**, yarıçapı **r=10**br olan birinci küreden aynasal yansıyıp merkezi **c(-38,-36,0)**, yarıçapı **r=10**br olan ikinci küre ile kesişiyor. Sonra bu küreden de aynasal yansıyıp merkezi **c(-14,-84,0)**, yarıçapı **r=25**br olan üçüncü küre ile kesişiyor. Üçüncü küre üzerindeki kesişim noktasının koordinatlarını hesaplayınız. **(40P)**



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

  vector<intersections> Intersections;

  for (int i = 0; i < 26; i++)
  {
    float t = Shapes[i]->Intersect(Ro, Rd);

    if (t > 0.0) // önceden (t > 0.1) idi
    {
      intersection.distance = t;
      intersection.indice   = i;

      Intersections.push_back(intersection);
    }
  }

  if (Intersections.size() > 0)
  {
    float min_distance = FLT_MAX;
    int   min_indis = -1;

    for (int i = 0; i < Intersections.size(); i++)
    {
      if (Intersections[i].distance < min_distance)
      {
        min_indis    = Intersections[i].indice;
        min_distance = Intersections[i].distance;
      }
    }

    Vertex iPoint = Ro + min_distance * Rd;

    Shape* S      = Shapes[min_indis];

    Color reflectedColor;
    if (S->Refl != 0.0F)
    {
        Vertex reflectedDirection =
        CalculateReflection(S, iPoint, Rd);

        iPoint = iPoint +0.001 *reflectedDirection;

        reflectedColor = TraceRay(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;
}
```

2) Solda paylaşılan `TraceRay()`'in ilk `for()` döngüsünde 3D nesneler (`Shapes`) için kesişim testleri sonrası **`t`** uzaklıkları `0.0`'dan büyükse o kesişimlere dair **`t`** uzaklıkları ve nesne indisleri `Intersections` adlı vektöre eklenir. `if(t>0.0)` testindeki `0.0` kısmını derslerimizde `0.1` olarak anlatmıştık. Çünkü `0.0` olursa özellikle aynasal yansıma (reflection) ve gölge testinde gürültülü sonuçlarla karşılaşabiliriz. Sebebini kısaca açıklayacak olursak : Örneğin ışın, aynasal bir cisim ile kesiştikten sonra yansıma doğrultusu hesaplanıp o yeni doğrultu boyunca ışın yollarken yeni ışının başlangıç noktası aynasal cisim üzerindeki kesişim noktası olarak alınır. Bu noktanın aynasal yüzey üzerinde olduğu varsayılır. Floating point hatalarından veya virgülden sonraki kısmın yeterince hassas olmamasından bu nokta bazen gerçekte aynasal yüzeyin çok az da olsa altında kalabilir. Bu durumda yolladığımız yeni ışın yine aynı cisimle kesişebilir. Bu da hatalı sonuçlara yol açar. Işının cismin kendisi ile kesişmemesi için kesişim testinden dönen değerin minimum `0.1` olmasını isteriz. Böylece kendisi ile kesiştiğinde **`t`** uzaklığı `0.1` 'den küçük olacağından bu kesişim dikkate alınmaz ve hatalı sonuçlarla karşılaşılmaz.

Derste, yukarıdaki yönteme alternatif olarak Cem Yüksel'in **YouTube** kanalından alıntı yaparak bahsettiğimiz ve kodunu yazdığımız başka bir yöntem daha vardı. Bu yönteme göre, örneğin aynasal bir cisim üzerindeki kesişim noktası için yansıma doğrultusu hesaplandıktan sonra yeni ışın yollanırken ışının başlangıç noktası yansıma doğrultusu boyunca küçük bir değer (örneğin `0.001`) kadar uzaktan seçilir.

Soldaki kodda `iPoint = ......` olarak boş bırakılan kısmı Cem Yüksel'in bahsettiği yönteme göre güncelleyiniz. Yeni başlangıç noktası öncekinden `0.001` birim uzakta olsun. **(30P)**

Y0(28,35,120) Y1(60,35,96) Y2(28,-25,120)

3) Yukarıda köşe noktaları verilen **Y** üçgeni üzerindeki **iPoint(44,11,108)** noktasının **(u,v)** barisentrik koordinatlarını hesaplayınız. **(30P)**

```
L = iPoint - Y0 = (16,-24,-12)

Rdu = (Y1-Y0).Normalize() = (0.8,0,-0.6)
Su = L * Rdu = 20
u = Su / (Y1-Y0).Length() = 20 / 40 = 0.5


Rdv = (Y2-Y0).Normalize() = (0,-1,0)
Sv = L * Rdv = 24
v = Sv / (Y2-Y0).Length() = 24 / 60 = 0.4
```