Raytracing shapes

Chris Dragan

This is a short article about hitting miscellaneous shapes with a ray. The topic of raytracing is quite broad and so there are many aspects beyond this, nevertheless it is probably a good start.

I am not explaining the basics here, i.e. how to construct a primary (eye) ray, what is a ray, the difference between intersection and shadow rays, how to calculate a surface color, how to succeed with texture mapping, etc., etc. I only show how to hit more or less simple shapes. I am also not explaining how to hit Constructive Solid Geometry (CSG, the composition of shapes with union, difference and intersection operators). Therefore you may want to refer to other raytracing articles in previous and possibly future issues of Hugi.

It is assumed that you have basic knowledge about vector mathematics. I use here a few simplifications:

- Vector dot product is denoted with "|".
- len(V) is the length of vector V.
- dot(V) is the square length of vector V (dot product with itself).
- nrm(V) is normalized V (V/sqrt(V|V)).

I present here the following shapes:

- Plane
- Sphere
- Cylinder
- Cone
- Ellipsoid
- Paraboloid
- Triangle
- Surface of Revolution
- Torus
- Blob

A ray is defined in the following way:

```
P = O + D*t
```

where O is the ray origin and D is the ray direction. From this we create a shape-specific equation that we solve for scalar t. Obtained t is actually the distance of the hit point from the ray origin. If t is negative, the hit point lies behind the origin, and is beyond our interest (except the CSG case, in which the point may interest us unless all hit points of the shape are behind the origin).

Throughout this article I will use a substitution for the P equation:

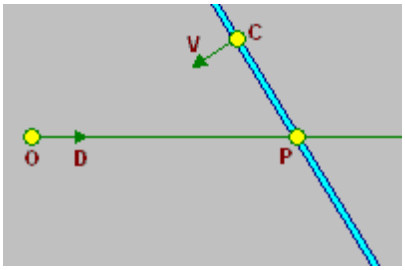
```
P - C = D*t + X
```

where C is a center point of a shape that we hit and X equals O-C.

Plane

Definition:

- C is a point that lies on the plane
- V is the plane normal (unit length)



To hit a plane we notice that:

```
(P-C)|V = 0
```

This means that the P-C vector is perpendicular to the normal, which is true when the point P lies on the plane.

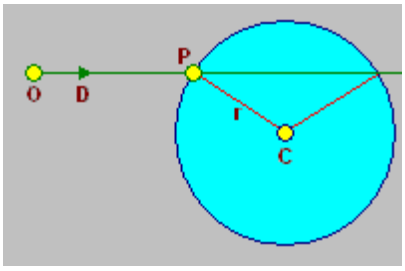
Solution:

```
(D*t+X)|V = 0
D|V*t = -X|V
t = -X|V / D|V
```

Before the division we should check whether D|V is nonzero. We can also check if the signs of D|V and X|V are different (if not, resulting t will be negative).

Surface normal vector at point P equals to the plane normal, unless D|V is negative, in which case N=-V.

Sphere



Definition:

- C is the center of the sphere
- r is the radius of the sphere

To hit a sphere we notice that:

```
len(P-C) = r
```

This means that the distance between the sphere center and the hit point equals r, which is true when P lies on the surface of the sphere.

Solution:

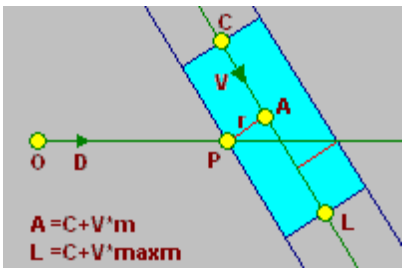
```
len(D*t+X) = r
dot(D*t+X) = r^2
D|D*(t^2) + 2*D|X*t + X|X - r^2 = 0
```

Hence we have a quadratic equation that we have to solve. To simplify, we have the following trinomial coefficients:

```
a   = D|D
b/2 = D|X
c   = X|X - r*r
```

Surface normal is N=nrm(P-C).

Cylinder



Definition:

- C is the start cap point of the cylinder
- V is a unit length vector that determines cylinder's axis
- r is the cylinder's radius
- maxm determines cylinder's end cap point

To hit a cylinder we notice that:

```
A = C + V*m
( P-A )|V = 0
```

```
len( P-A ) = r
```

where m is a scalar that determines the closest point on the axis to the hit point. The P-A vector is perpendicular to V, what guarantees the closest distance to the axis. P-A is the cylinder's radius.

Solution:

```
(P-C-V*m)|V = 0
(P-C)|V = m*(V|V) = m    (len(V)=1)
m = (D*t+X)|V
m = D|V*t + X|V

len(P-C-V*m) = r
dot( D*t+X - V*(D|V*t + X|V) ) = r^2
dot( (D-V*(D|V))*t + (X-V*(X|V)) ) = r^2
   dot( A-V*(A|V) ) = A|A - 2*(A|V)^2 + V|V * (A|V)^2 =
    = A|A - (A|V)^2
a*t^2 + b*t + c = 0
a = D|D - (D|V)^2
c = X|X - (X|V)^2 - r^2
b = 2 * (D-V*(D|V))|(X-V*(X|V)) =
  = 2 * (D|X - D|V*(X|V) - X|V*(D|V) + (D|V)*(X|V)) =
  = 2 * (D|X - (D|V)*(X|V))
```

Finally:

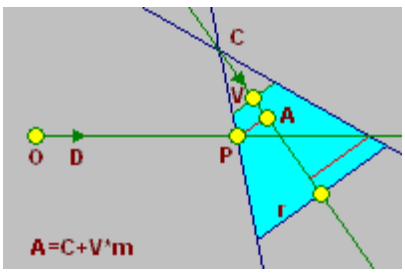
```
a   = D|D - (D|V)^2
b/2 = D|X - (D|V)*(X|V)
c   = X|X - (X|V)^2 - r*r
```

Surface normal is:

```
m = D|V*t + X|V
N = nrm( P-C-V*m )
```

There are two points on the cylinder that we hit (it can be the same point twice). We have to calculate two m values and test whether they fall in the range of [0,maxm]. If any falls out, we can either throw the point that corresponds to it away (the cylinder will have a hole) or we can cap the cylinder with planes. One of the planes is defined by a pair (C,-V) and the other by (C+V*maxm,V). We hit the planes like a typical plane; the dot products we have already calculated, we only need to do the division(s).

Cone



Definition:

- C is the vertex of the cone
- V is the axis vector
- k is the tangent of half angle of the cone
- minm, maxm define cap points

To hit a cone we notice that:

```
A = C + V*m
( P-A )|V = 0
len( P-A )/m = k
```

The (P-A)|V=0 equation is identical to the equation we know from cylinder, since cylinder is a special case of a cone.

Solution:

```
m = D|V*t + X|V     (like for cylinder)
len( P-C-V*m ) = m*k
dot( D*t+X - V*(D|V*t + X|V) ) = k^2 * ( D|V*t + X|V )^2
dot( (D-V*(D|V))*t + X-V*(X|V) ) = k^2 * ( D|V*t + X|V )^2
```

Now the coefficients of the left-side trinomial are similar like for cylinder:

```
a   = D|D - (D|V)^2
b/2 = D|X - (D|V)*(X|V)
c   = X|X - (X|V)^2
```

And the right-side coefficients:

```
k^2 * ( D|V*t + X|V )^2 =
 = k^2 * ( (D|V)^2 * t^2 + 2*(D|V)*(X|V)*t + (X|V)^2 )
a   = k*k*(D|V)^2
b/2 = k*k*(D|V)*(X|V)
c   = k*k*(X|V)^2
```

Finally:

```
a   = D|D - (1+k*k)*(D|V)^2
b/2 = D|X - (1+k*k)*(D|V)*(X|V)
c   = X|X - (1+k*k)*(X|V)^2
```

To calculate surface normal it is enough to notice that we have to normalize vector (P-C-V*m) - V*a. How to calculate a ? The angle between the normal and P-C-V*m must be the same as half angle of the cone. Hence:

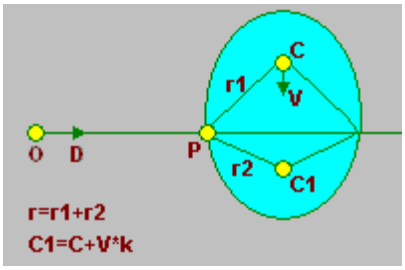
```
a/r = k
r/m = k
a = m*k*k
N = nrm( P-C-V*m - V*m*k*k )
N = nrm( P-C - (1+k*k)*V*m )
```

We cap the cone like a cylinder (except that the start cap point does not have to be at 0).

Ellipsoid

Definition:

- C is one of the two centers of the ellipsoid
- k is the distance between the two centers
- V is a unit length vector from center C to the other center
- r is the radius sum



To hit an ellipsoid we notice that:

```
len( P-C ) + len( P-(C+V*k) ) = r
```

This means that the sum of distances from P to the two centers is equal r.

Solution:

```
len( P-C-V*k) ) = r - len( P-C )
dot( D*t + X-V*k ) = r*r - 2*r*len( D*t+X )
   dot( D*t + X-V*k ) =
      = D|D*t^2 + 2*D|(X-V*k)*t + dot(X-V*k) =
      = D|D*t^2 + 2*(D|X-D|V*k)*t + X|X-2*X|V*k+k
   dot( D*t + X ) =
      = D|D*t^2 + 2*(D|X)*t + X|X
2*D|V*k*t - 2*X|V*k + k - r*r = -2*r*len( D*t + X )
(2*D|V*k*t - 2*X|V*k + k-r*r)^2 =
  = 4*r*r*( D*D*t^2 + 2*(D|X)*t + X|X )
```

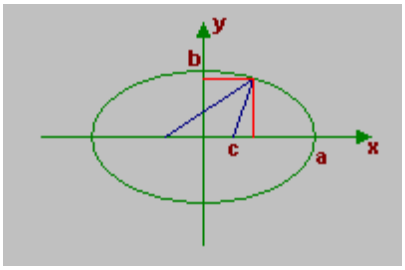
And the final coefficients:

```
a   = 4*r*r*D|D - 4*k*k*(D|V)^2
b/2 = 4*r*r*D|X - 2*(D|V)*k * (r*r+2*X|V*k-k)
c   = 4*r*r*X|X - (r*r+2*X|V*k-k)^2
```

To make it simple:

```
A1  = 2*k*(D|V)
A2  = r^2 + 2*k*(X|V) - k
a   = 4*r^2*D|D - A1^2
b/2 = 4*r^2*D|X - A1*A2
c   = 4*r^2*X|X - A2^2
```

Ellipsoid's surface normal is more complex to calculate than the ones before. We could calculate it like for a sphere, but then it would be inaccurate. If the ellipse that makes our ellipsoid was like on the image, it would have equation:




```
x^2/a^2 + y^2/a^2 = 1
```

For the upper part of the chart, the equation can be rewritten as:

```
y = b * sqrt( 1 - x^2/a^2 )
```

The derivative of this function is:

```
f'(x) = -b/a^2 * x / sqrt( 1 - x^2/a^2 )
```

But what really iterests us, is the reciprocal of the derivative (`1/f'(x)`), which is the ratio of the surface normal's `y/x`:

```
1/f'(x) = a^2/b * sqrt( 1/x^2 - 1/a^2 )
```

However if we calculated the normal like for a sphere, the ratio would be simply `y/x` (values of the function itself):

```
y/x = b * sqrt( 1/x^2 - 1/a^2 )
```

Now we see that this differs from what we need by a factor of `a^2/b^2`. Note that this factor is always greater than 1 in our case, since the ellipsoid's centers lie on the X axis.

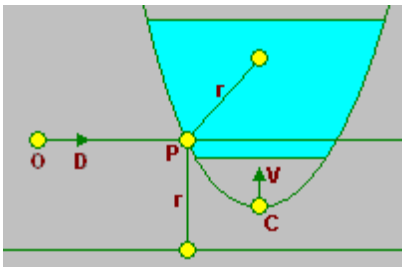
The original function is related to our ellipsoid definition in that the vector `V` specifies the X axis. Now we want to know how much of `V` we have to add to vector `P-C-V*k/2` (`C+V*k/2` is the point between the two ellipsoid's centers) to obtain a correct surface normal (after normalizing the resulting vector, of course).

Because we have to multiply the y/x ratio by a^2/b^2, it's enough if we divide x by that factor. b^2/a^2<1, so we subtract exactly (1-b^2/a^2)*x from x to obtain a correct value.

Because V is directly related to x, we have to multiply it by that factor and then subtract it from P-C-V*k/2. The value of x is in our case m=(P-C-V*k/2)|V, which is similar to the cylinder's case (we cast the vector on the straight line determined by V).

```
Cmid = C + V*k/2
R = P - Cmid
N = nrm( R - V*(1-b^2/a^2)*(R|V) )
```

Paraboloid



Definition:

- C is the extremum point of the paraboloid
- V is a unit length vector that specifies the direction of the paraboloid
- k is a scalar that specifies the distance of the paraboloid's kernel and the plane from C

To hit a paraboloid we notice that every point on the paraboloid's surface has the same distance to the kernel C+V*k as to the plane (C-V*k, V).

How to calculate the distance to the plane ? Let A be a point on the plane, made by casting P on that plane. And let r be our distance that we are looking for.

```
P = A + V*r
A = P - V*r
(A-(C+V*k))|V = 0
(P-C+V*k-V*r)|V = 0
(P-C+V*k)|V - V|V*r = 0
r = (P-C+V*k)|V
```

Now we go for the equation:

```
(P-C+V*k)|V + k = r
len( P-(C+V*k) ) = r
```

Solution:

```
len( P-C-V*k ) = (P-C+V*k)|V
dot( D*t + X-V*k ) = (D|V*t + X|V+k)^2
D|D*(t^2) + 2*(D|(X-V*k))*t + dot(X-V*k) =
  = (D|V)^2*(t^2) + 2*D|V*(X|V+k)*t + (X|V+k)^2

a   = D|D - (D|V)^2
b/2 = D|X - D|V*k - D|V*(X|V+k) =
    = D|X - D|V*( X|V + 2*k )
c   = X|X - 2*X|V*k + k^2 - (X|V)^2 - 2*(X|V)*k - k^2 =
    = X|X - (X|V)^2 - 4*(X|V)*k =
    = X|X - X|V*( X|V + 4*k )
```

Finally:

```
a   = D|D - (D|V)^2
b/2 = D|X - D|V*( X|V + 2*k )
c   = X|X - X|V*( X|V + 4*k )
```

We can cap the paraboloid, limiting m value calculated as m=(P-C)|V (like for cylinder).

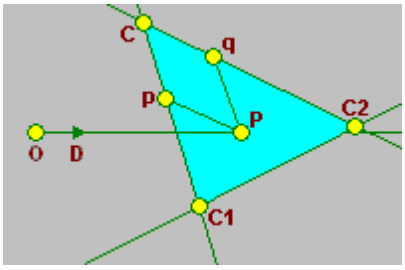
To calculate the normal, we notice that we can normalize the vector P-(C+V*m) + V*a where a is a secret scalar that we want to find. In fact, the ratio len(P-(C+V*m))/a is the same as ratio Ny/Nx, where Nx and Ny are values of 2D normal vector, if we treat our paraboloid as a typical 2D function y^2=2*k*x. Let's proceed:

```
f(x) = sqrt( 2*k*x )          our function
f'(x) = k / sqrt( 2*k*x )     derivative
Ny/Nx = 1/f'(x)               the ratio we need
Ny/Nx = sqrt( 2*x/k )
Nx = Ny * sqrt( k/(2*x) )     we are looking for a...
Ny = sqrt( 2*k*x )            ...while we know f(x)
Nx = sqrt( 2*x*k*k/(2*x) )
Nx = k                        and there we are!
```

Interestingly, the value of a does not depend on the length of vector P-(C+V*m), and equals the distance from the extremum to the kernel point.

```
N = nrm( P-C-V*(m+k) )
```

Triangle



Definition:

- C is a vertex of our choice
- V is a normal of the plane on which the triangle lies
- V1 is a vector from vertex C to vertex C1 (V1=C1-C)
- V2 is a vector from vertex C to vertex C2 (V2=C2-C)

To hit a triangle we first hit a plane on which it lies, in the same manner as we hit a regular plane. Then the two other triangle vertices give us a point to calculate two coefficients that tell us where exactly we are on the triangle.

```
t = - X|V / D|V
P = C + V1*p + V2*q
```

Solution:

```
P - C = V1*p + V2*q
[ Px-Cx ] = [ V1.x V2.x ] * [ p ]
[ Py-Cy ]   [ V1.y V2.y ]   [ q ]
```

To calculate p and q we don't need the whole V1 and V2 vectors, we just need their x and y values (or z are good as well). We yet have to calculate an inverse of the matrix in the above equation, but this is a precalculation step.

To determine whether the point P we hit actually lies within the triangle, we just have to ensure that the scalar values p, q and their sum are within range [0,1].

If we hit the triangle, we can use the p and q values to calculate (smooth) normals and to calculate texture coordinates. Usually triangles build up bigger meshes, so we don't have to worry about flipping the surface normal if we hit it from the back (like in case of a plane).

A triangle for raytracing actually takes a little bit more space than for a regular scanline rendering algorithm (e.g. OpenGL). That's because we have one more vector (V) and we have to keep original V1 and V2 for rotations and scaling from frame to frame. We also have to calculate the inverse matrix per frame, what takes additional time. What's more, we have to adapt a special hit test scheme for meshes, so that we don't waste cycles on testing all triangles in a mesh. Perhaps a bounding volume hierarchy is a solution. Nevertheless every additional thing takes up memory, and that's why triangles are not the most efficient objects for raytracing, especially in complex scenes.

Surface of Revolution

A surface of revolution is defined as a spline rotated about an axis (a line). Typically this is a one to three degree spline (linear, quadratic or cubic). Every separate segment of the SOR is defined as follows:

- C is the starting point of the segment
- V is the unit length vector that specifies the axis
- maxm is the end cap point of the segment
- a, b, c, d are the coefficients of the segment (two to four coefficients depending on the degree of the segment).

We calculate m like for a cylinder (cylinder, cone and paraboloid are just special cases of SOR).

```
m = D|V*t + D|X
```

Now we calculate P according to the function defined by a,b,c,d coefficients (the distance of P from the axis is actually equal to value of the function in point m - f(m)).

Solution:

```
len( P-(C+V*m) ) = f(m)
(P-(C+V*m))|V = 0

dot( P-C-V*((P-C)|V) ) = f(m)^2
dot( D*t+X-V*(D|V*t+X|V) ) = f(m)^2
dot( (D-V*(D|V))*t + X-V*(X|V) ) = f(m)^2
```

Left side coefficients:

```
a = D|D - (D|V)^2
b = 2*( D|X - (D|V)*(X|V) )
c = X|X - (X|V)^2
```

Right side coefficients:

```
f(m)^2 = (am^3 + bm^2 + cm + d)^2
f(m)^2 = m^6 * a^2 +
       + m^5 * 2*a*b +
       + m^4 * (b^2 + 2*a*c) +
       + m^3 * 2*(a*d + b*c) +
       + m^2 * (c^2 + 2*b*d) +
       + m^1 * 2*c*d +
       + m^0 * d^2
m = D|V*t + X|V
```

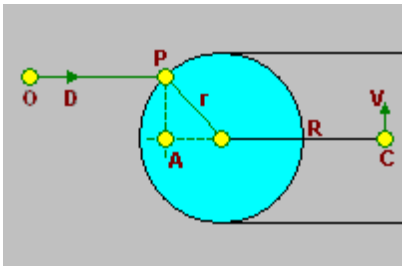
Hence we have a 2nd to 6th degree equation for t (for 1st to 3rd degree spline). Solving equations of degree higher than 2 is a non-trivial topic that requires a lot of processing power. However it's still better to model SORs itselves instead of converting them into a bunch of triangles.

Calculating normal is a brute force method, as follows:

```
m = D|V*t + X|V
dval = f'(m) = 3*a*m^2 + 2*b*m + c
R = P - C - V*m
N = nrm( R - V*len(R)/dval )
```

We find how much of V to add through the derivative of our spline segment function.

Torus



Definition:

- C is the center of the torus
- V is the vector that shows the direction of the torus' inside
- R is the big radius of the torus
- r is the small radius of the torus

Torus is actually easier to hit than a 3rd degree SOR, because it produces only 4th degree equation. We notice that if we cast P on the torus' main plane (C,V), we have a simple equation made by torus' radii.

```
P = A + V*k
(len( A-C ) - R)^2 + k^2 = r^2
```

Solution:

```
A = P - V*k
(A-C)|V = 0
(P-C-V*k)|V = 0
k = (P-C)|V

r^2 - k^2 = ( len(P-C-V*k) - R )^2
Z = P-C = D*t+X          temporary substitution
r^2 - (Z|V)^2 = ( len( Z-V*(Z|V) ) - R )^2
dot(Z-V*(Z|V)) + R^2 - r^2 + (Z|V)^2 = -2*R*len(Z-V*(Z|V))
Z|Z + R^2 - r^2 = -2*R*len(Z-V*(Z|V))
4*R^2*dot(Z-V*(Z|V)) = (Z|Z + R^2 - r^2)^2
4*R^2*(Z|Z-(Z|V)^2) = (Z|Z)^2 + 2*(R^2-r^2)*(Z|Z) + (R^2-r^2)^2
(Z|Z)^2 - 2*(R^2+r^2)*(Z|Z) + 4*R^2*(Z|V)^2 + (R^2-r^2)^2 = 0
```

Let's make a small substitution:

```
m = D|D, n = D|X, o = X|X, p = D|V, q = X|V

Z|Z = dot(D*t+X) = m*t^2 + n*t + o
Z|V = (D*t+X)|V = p*t + q

(m*t^2 + n*t + o)^2 - 2*(R^2+r^2)*(m*t^2 + n*t + o) +
 + 4*R^2*(p*t + q)^2 + (R^2-r^2)^2 = 0
m^2*t^4 + 4*n^2*t^2 + o^2 + 4*n*o*t + 4*m*n*t^3 +
 + 2*m*o*t^2 + 4*R^2*(p^2*t^2 + 2*p*q*t + p^2) +
 + (R^2-r^2)^2 = 0
```

And final coefficients are:

```
a = m^2
b = 4*m*n
c = 4*m^2 + 2*m*o - 2*(R^2+r^2)*m + 4*R^2*p^2
d = 4*n*o - 4*(R^2+r^2)*n + 8*R^2*p*q
e = o^2 - 2*(R^2+r^2)*o + 4*R^2*q^2 + (R^2-r^2)^2
```

And that's the fourth degree equation to solve.

To calculate normal, we have to use k (the distance of the hit point to the torus' main plane).

```
k = (P-C)|V
A = P - V*k
m = sqrt( r^2 - k^2 )
N = nrm( P - A - (C-A)*m/(R+m) )
```

Blob

Blob is an interesting type of object to raytrace. There is no straightforward way to raytrace blobs, though. 

A blob is defined as a list of components. Inhere, I will cover only a sphere component, but you can use virtually any type of object. An influence coming from an object you can perceive as a kind of force field (e.g. magnetic field):

```
d = s * ( 1 - (len(P-C)/r)^2 )^2
```

where d is a calculated density, s is strength of field coming from the blob component, C is component's center and r is component's radius. There are also other density functions for other types of objects. The density from one component varies from s in the center to 0 at the radius. If a blob is constructed from multiple components, you simply add the density factors coming from the components that affect the ray.

Because you immediately know whether a component affects the ray or not (for a sphere component it simply has to enter the sphere to be affected), you can throw away components that don't affect it while you hit-test. You can assume a certain density threshold that will be the surface of the blob when you sum up densities of your components. Now if you sort your components in an ascending order (from the closest to the farthest) you can test them by adding and subtracting their density functions as you proceed through the groups of them covering one another. Unfortunately you have to solve a fourth degree equation each test.

After all, knowing all components that finally make up the hit point, you can use their strengths to calculate weighted sum of normals to produce the final surface normal. You can also proceed like this with other kinds of object parameters (texture, refraction, etc.).
