◊ V.2

Intersecting a Ray with a Cylinder

Joseph M. Cychosz
Purdue University CADLAB
West Lafayette, IN 47907
3ksnn64@ecn.purdue.edu

Warren N. Waggenspack, Jr.
IMRLAB, Mechanical Engineering
Louisiana State University
Baton Rouge, LA 70803-6413
mewagg@mewnw.dnet.lsu.edu

This Gem presents a geometric algorithm for locating the points of intersection between a ray and a circular cylinder and the means for determining the normal to the surface at the point of intersection. It also presents an algorithm for intersecting a finite cylinder bounded by two planar end-caps of arbitrary orientation.

The cylinder is a common modeling primitive used in a variety of computer graphics and computer-aided design applications (Haines 1989), (Roth 1982). To render an image consisting of cylinders using ray tracing, the intersection points, the corresponding distances along the intersecting ray, and the surface normal required in lighting model computations all must be computed. The authors in a previous Gem (Cychosz and Waggenspack 1992) presented a general solution for intersecting a ray with a general quadric surface; however, much like the well-known solution for intersecting a ray with a sphere (Haines 1989), (Hultquist 1991), a simpler geometric solution exists. Summarized below is an algorithm for locating the circular cylinder intersection points, distances along the ray, and the means for determining the surface normal at the points of intersection.

◊ Intersection of a Ray with a Cylinder ◊

Let the semi-infinite ray be defined by a base point B_R and unit vector (direction cosines) $\mathbf{R}$. The associated infinite cylinder is described by its radius, r, a unit vector A defining its centerline or axis, and a base point B_C located on the cylinder axis. Analogous to the sphere intersection procedure, the first step in computing the intersection between a ray and cylinder is also a quick elimination test. It involves computing the shortest distance, d, between the ray and axis of the cylinder (see Figure 1a). This shortest path between the two lies along a direction, $\mathbf{D}$, perpendicular to both $\mathbf{A}$ and $\mathbf{R}$. Using

IBM ISBN 0-12-336155-9
Macintosh ISBN 0-12-336156-7

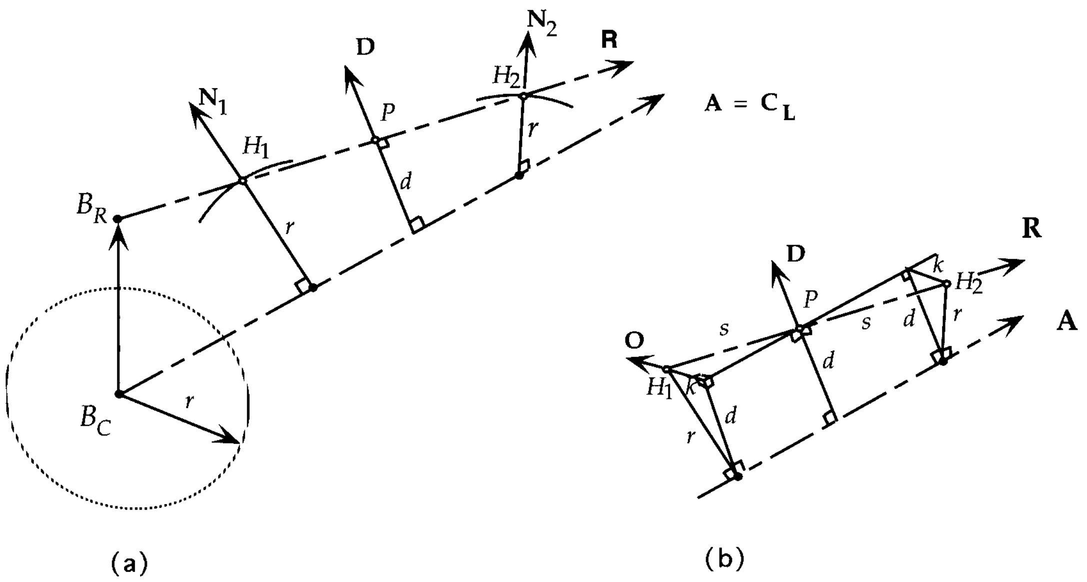

Figure 1. Geometric presentation of the ray-cylinder intersection problem.

a combination of vector and scalar products it can be shown that the direction $\mathbf{D}$ and the distance d are computed as follows.

$$\mathbf{D} = \frac{\mathbf{R} \times \mathbf{A}}{|\ \mathbf{R} \times \mathbf{A}\ |} \qquad d = |(B_R - B_C) \cdot \mathbf{D}|$$

If d is greater than r, then the ray misses the cylinder and no further processing is necessary. Otherwise the ray intersects the cylinder in two points, H_1 and H_2, located symmetrically about the point P where the ray passes closest to the axis of the cylinder. The distance t along the ray from the base point to the point P is determined by:

$$t = \frac{\{(B_R - B_C) \times \mathbf{A}\} \cdot \mathbf{D}}{|\ \mathbf{R} \times \mathbf{A}\ |} = \frac{\{(B_R - B_C) \times \mathbf{A}\} \cdot \mathbf{R} \times \mathbf{A}}{|\ \mathbf{R} \times \mathbf{A}\ |^2}$$

From the symmetry formed about the midpoint P, a set of right triangles can be assembled where $k^2 + d^2 = r^2$ (see Figure 1b). The distance s, along the ray between the intersections and the midpoint P, can now be computed by defining the unit vector orthogonal to both $\mathbf{A}$ and $\mathbf{D}$

$$\mathbf{O} = \frac{\mathbf{D} \times \mathbf{A}}{|\ \mathbf{D} \times \mathbf{A}\ |}$$

and noting that

$$s\mathbf{R} \cdot \mathbf{O} = \pm k$$

thus

$$s = \left| \frac{k}{\mathbf{R} \cdot \mathbf{O}} \right| = \left| \frac{\sqrt{r^2 - d^2}}{\mathbf{R} \cdot \mathbf{O}} \right|$$

The intersection distances, t_{in} and t_{out}, and the corresponding points of intersection, H_{in} and H_{out} are then simply:

$$t_{in} = t - s$$
$$t_{out} = t + s$$

with

$$H_{in} = B_R + t_{in}\mathbf{R}$$
$$H_{out} = B_R + t_{out}\mathbf{R}$$

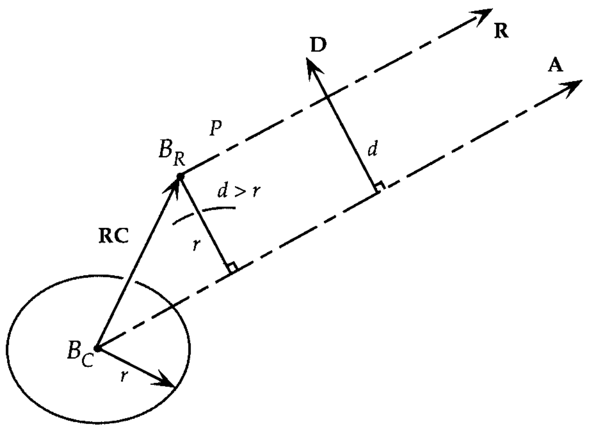

Figure 2. Geometric description for parallel ray and cylinder.

For the special case where the ray is parallel to the axis of the cylinder (i.e., $\mathbf{R} \times \mathbf{A} = \mathbf{0}$), the distance, d, between the ray base point and the axis of the cylinder dictates if the ray travels inside or outside of the cylinder. The alternate formulation for d is shown below and, again, if d is greater than r, then the ray falls outside the cylinder and no further computation is required. (See Figure 2.)

$$\begin{aligned} \mathbf{RC} &= B_R - B_C = \mathbf{RC}_{\perp\mathbf{A}} + \mathbf{RC}_{\parallel\mathbf{A}} = d\mathbf{N} + (\mathbf{RC} \cdot \mathbf{A})\mathbf{A} \\ d\mathbf{N} &= \mathbf{RC} - (\mathbf{RC} \cdot \mathbf{A})\mathbf{A} \\ d &= |\mathbf{RC}_{\perp\mathbf{A}}| = |\ \mathbf{RC} - (\mathbf{RC} \cdot \mathbf{A})\mathbf{A}\ | \end{aligned}$$

Should the ray travel inside the cylinder ($d \leq r$), the entering and exiting distances, t_{in} and t_{out} are set to negative and positive infinity, respectively. As detailed in a section to follow, these results are used in determining the appropriate bounded cylinder intersections.

A different derivation of the intersection of a ray with an infinite cylinder is contained in another article in this volume (Shene 1994).

◇ Determining the Surface Normal ◇

Much like determining the normal on the surface of a sphere, where the normal is defined as the vector originating from the center of the sphere and passing through the point on the surface, a simple geometric solution exists for the cylinder also. The normal, $\mathbf{N}$, at a given location H on the surface of the cylinder is simply the vector perpendicular to the cylinder's axis that passes through H (see Figure 1). It can be formulated in terms of the unit vector parallel to the component of $\mathbf{HB} = \mathbf{H} - \mathbf{B_C}$ that is perpendicular to $\mathbf{A}$.

$$\begin{aligned}
\mathbf{HB} &= H - B_C = \mathbf{HB}_{\perp\mathbf{A}} + \mathbf{HB}_{\parallel\mathbf{A}} = r\mathbf{N} + (\mathbf{HB}\cdot\mathbf{A})\mathbf{A} \\
r\mathbf{N} &= \mathbf{HB} - (\mathbf{HB}\cdot\mathbf{A})\mathbf{A} \\
\mathbf{N} &= \frac{\mathbf{HB} - (\mathbf{HB}\cdot\mathbf{A})\mathbf{A}}{r}
\end{aligned}$$

◇ Intersecting a Ray with a Finite Cylinder ◇

Rarely are infinite cylinders used as modeling primitives; instead the cylinder is often bounded by two planar end-caps. In addition to testing the infinite cylinder, a comparison must be made with the planar end-caps to determine if the ray does indeed intersect the bounded portion of the cylinder, one of the planar end-caps, or misses the finite surfaces altogether.

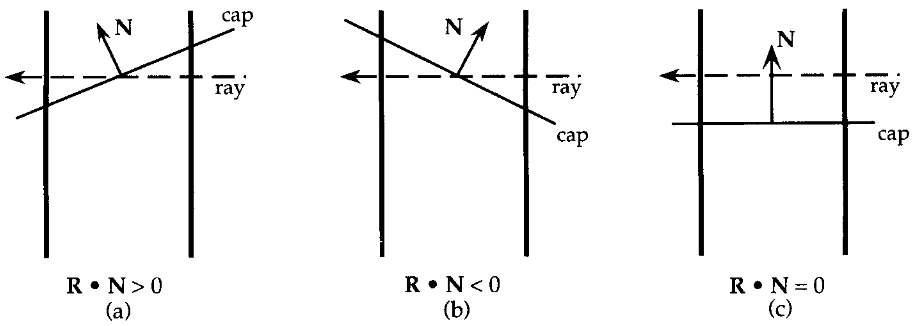

Figure 3. Determination of intersecting surfaces for a cylinder bounded by two end-cap planes.

Each end-cap plane is described with an outward pointing unit normal $\mathbf{N}$, and an offset d defining an oriented, shortest distance from the plane to the origin. A point lies on the plane iff $(\mathbf{P}\cdot\mathbf{N} + d = 0)$. A positive d indicates the origin lies above the plane or on the outside in terms of bounding the cylinder. The distance along the ray to the intersection point on a planar end-cap can be computed as:

$$t = -\frac{B_R \cdot \mathbf{N} + d}{\mathbf{R} \cdot \mathbf{N}}$$

Once computed, the intersection distances to the end-caps are compared with those from the infinite cylinder (t_{in} and t_{out}). If the ray points in the same direction as the corresponding end-plane normal ($\mathbf{R} \cdot \mathbf{N} > 0$), then it potentially exits the finite cylinder volume there. This intersection is thus compared with the exiting distance of the cylinder, t_{out}. The minimum of these two distances dictates which of the two surfaces defines the true intersection (see Figure 3a). Where the ray direction opposes that of the corresponding end-plane normal ($\mathbf{R} \cdot \mathbf{N} > 0$), it may be that it enters the finite cylinder volume there. The resulting intersection distance is therefore compared to t_{in} and the maximum of the two indicates which of the two surface intersections is actually where the ray enters the bounded volume (see Figure 3b). Upon integrating the results from the two end-caps, a quick comparison of the intersection distances will indicate if the ray completely misses the bounded cylinder (i.e., $t_{out} < t_{in}$).

Special attention must be given where the ray is determined to be parallel to an end-cap plane ($\mathbf{R} \cdot \mathbf{N} = 0$). In this instance, the location of the ray base point relative to the cap plane is enough to establish where the ray passes with respect to the bounded volume. If the numerator in the ray-plane intersection computation is positive, ($B_R{\cdot}\mathbf{N}{+}d > 0$), then the ray misses the cylinder (see Figure 3c). Otherwise, the current t_{in} and t_{out} remain valid intersection distances.

The object clipping presented here can be used for any object with one pair of intersection points such as the family of quadric surfaces presented in (Cychosz and Waggenspack 1992). An analogous intersection procedure can also be derived for other quadrics such as the cone from the plane-quadric intersection procedures described in (Johnston and Shene 1992). For the simplified case of a right circular cylinder where the end-cap planes are parallel and the normals are aligned with the axis of the cylinder, the computation of the intersection distances to the end-caps is greatly simplified and all that must be determined when the ray is parallel to the end-cap planes is whether the ray lies between the two planes.

◇ Summary ◇

A benefit of this geometric approach to the ray-cylinder intersection is that it deals directly in object space coordinates. This avoids the overhead of transforming either the rays and/or the bounded cylinder into a canonical form for testing. This is especially true for instances when the cylinder is in motion (i.e., rendering a cylinder that is changing either size or position with motion blur), which would require re-computation of the transfrom at each instance. For stationary cylinders, the approach presented here basically requires the computation of 3 cross products (each cross product requires 6 multiplies and 3 additions), whereas the approaches that perform a transformation require the computation of 1 cross product and 1 vector-matrix multiplication (a vector-matrix multiplication that ignores perspective requires 12 multiplies and 9 additions). In the context of this problem, there are going to be times where transforming an object in a canonical space may be faster, but one still can't handle the arbitrary end-caps in a "canonical" fashion. It would require non-linear transformations to normalize the end-caps while not distorting the cylinder!

◇ C Implementation ◇

The following code calls vector arithmetic routines defined in Chapter X.4.

```
#include        "GraphicsGems.h"
#include        <math.h>

/* ---- intcyl - Intersect a ray with a cylinder. --------------------- */
/*                                                                      */
/*                                                                      */
/*      Description:                                                    */
/*          Intcyl determines the intersection of a ray with a          */
/*          cylinder.                                                   */
/*                                                                      */
/*      On entry:                                                       */
/*          raybase = The base point of the intersecting ray.           */
/*          raycos  = The direction cosines of the above ray. (unit)    */
/*          base    = The base location of the cylinder.                */
/*          axis    = The axis of symmetry for the cylinder.  (unit)    */
/*          radius  = The radius of the cylinder.                       */
/*                                                                      */
/*      On return:                                                      */
/*          in      = The entering distance of the intersection.        */
/*          out     = The leaving  distance of the intersection.        */
/*                                                                      */
/*      Returns:  True if the ray intersects the cylinder.              */
/*                                                                      */
/*      Note:     In and/or out may be negative indicating the          */
/*                cylinder is located behind the origin of the ray.     */
/*                                                                      */
/* -------------------------------------------------------------------- */

#define HUGE            1.0e21          /* Huge value                   */
```

```
boolean intcyl  (raybase,raycos,base,axis,radius,in,out)

        Point3          *raybase;       /* Base of the intersection ray */
        Vector3         *raycos;        /* Direction cosines of the ray */
        Point3          *base;          /* Base of the cylinder         */
        Vector3         *axis;          /* Axis of the cylinder         */
        double          radius;         /* Radius of the cylinder       */
        double          *in;            /* Entering distance            */
        double          *out;           /* Leaving distance             */

{
        boolean         hit;            /* True if ray intersects cyl   */
        Vector3         RC;             /* Ray base to cylinder base    */
        double          d;              /* Shortest distance between    */
                                        /*   the ray and the cylinder   */
        double          t, s;           /* Distances along the ray      */
        Vector3         n, D, O;
        double          ln;
const   double          pinf = HUGE;    /* Positive infinity            */


        RC.x = raybase->x - base->x;
        RC.y = raybase->y - base->y;
        RC.z = raybase->z - base->z;
        V3Cross (raycos,axis,&n);

        if  ( (ln = V3Length (&n)) == 0. ) {    /* ray parallel to cyl  */
            d    = V3Dot (&RC,axis);
            D.x  = RC.x - d*axis->x;
            D.y  = RC.y - d*axis->y;
            D.z  = RC.z - d*axis->z;
            d    = V3Length (&D);
            *in  = -pinf;
            *out =  pinf;
            return (d <= radius);               /* true if ray is in cyl*/
        }

        V3Normalize (&n);
        d    = fabs (V3Dot (&RC,&n));           /* shortest distance    */
        hit  = (d <= radius);

        if  (hit) {                             /* if ray hits cylinder */
            V3Cross (&RC,axis,&O);
            t = - V3Dot (&O,&n) / ln;
            V3Cross (&n,axis,&O);
            V3Normalize (&O);
            s = fabs (sqrt(radius*radius - d*d) / V3Dot (raycos,&O));
            *in  = t - s;                       /* entering distance    */
            *out = t + s;                       /* exiting  distance    */
        }

        return (hit);
}
```

```
/* ---- clipobj - Clip object with plane pair. ----------------------- */
/*                                                                     */
/*                                                                     */
/*      Description:                                                   */
/*          Clipobj clips the supplied infinite object with two        */
/*          (a top and a bottom) bounding planes.                      */
/*                                                                     */
/*      On entry:                                                      */
/*          raybase = The base point of the intersecting ray.          */
/*          raycos  = The direction cosines of the above ray. (unit)   */
/*          bot     = The normal and perpendicular distance of the     */
/*                    bottom plane.                                    */
/*          top     = The normal and perpendicular distance of the     */
/*                    top plane.                                       */
/*          objin   = The entering distance of the intersection with   */
/*                    the object.                                      */
/*          objout  = The exiting  distance of the intersection with   */
/*                    the object.                                      */
/*                                                                     */
/*      On return:                                                     */
/*          objin   = The entering distance of the intersection.       */
/*          objout  = The exiting  distance of the intersection.       */
/*          surfin  = The identifier for the entering surface.         */
/*          surfout = The identifier for the leaving surface.          */
/*                                                                     */
/*      Returns:  True if the ray intersects the bounded object.       */
/*                                                                     */
/* ------------------------------------------------------------------- */

#define         SIDE    0               /* Object surface              */
#define         BOT     1               /* Bottom end-cap surface      */
#define         TOP     2               /* Top end-cap surface      */

typedef struct  {                       /* Plane: ax + by + cz + d = 0  */
        double  a ,b ,c, d;
}       Plane;


boolean clipobj         (raybase,raycos,bot,top,objin,objout,surfin,surfout)

        Point3          *raybase;       /* Base of the intersection ray */
        Vector3         *raycos;        /* Direction cosines of the ray */
        Plane           *bot;           /* Bottom end-cap plane         */
        Plane           *top;           /* Top end-cap plane            */
        double          *objin;         /* Entering distance            */
        double          *objout;        /* Exiting  distance            */
        int             *surfin;        /* Entering surface identifier  */
        int             *surfout;       /* Exiting  surface identifier  */

{
        boolean hit;
        double  dc, dw, t;
        double  in, out;                /* Object  intersection dists.  */
        *surfin = *surfout = SIDE;
        in  = *objin;
        out = *objout;

/*      Intersect the ray with the bottom end-cap plane.                */

        dc = bot->a*raycos->x  + bot->b*raycos->y  + bot->c*raycos->z;
        dw = bot->a*raybase->x + bot->b*raybase->y + bot->c*raybase->z + bot->d;

        if  ( dc == 0.0 ) {             /* If parallel to bottom plane  */
            if  ( dw >= 0. ) return (FALSE);
        } else {
            t  = - dw / dc;
            if  ( dc >= 0.0 ) {                     /* If far plane     */
                if  ( t > in && t < out ) { out = t; *surfout = BOT; }
                if  ( t < in  ) return (FALSE);
             } else {                               /* If near plane    */
                if  ( t > in && t < out ) { in  = t; *surfin  = BOT; }
                if  ( t > out ) return (FALSE);
            }
        }

/*      Intersect the ray with the top end-cap plane.                   */

        dc = top->a*raycos->x  + top->b*raycos->y  + top->c*raycos->z;
        dw = top->a*raybase->x + top->b*raybase->y + top->c*raybase->z + top->d;

        if  ( dc == 0.0 ) {             /* If parallel to top plane     */
            if  ( dw >= 0. ) return (FALSE);
        } else {
            t  = - dw / dc;
            if  ( dc >= 0.0 ) {                     /* If far plane     */
                if  ( t > in && t < out ) { out = t; *surfout = TOP; }
                if  ( t < in  ) return (FALSE);
             } else {                               /* If near plane    */
                if  ( t > in && t < out ) { in  = t; *surfin  = TOP; }
                if  ( t > out ) return (FALSE);
            }
        }

        *objin  = in;
        *objout = out;
        return (in < out);
}
```

◇ Bibliography ◇

(Cychosz and Waggenspack 1992) J. M. Cychosz and W. N. Waggenspack, Jr. Intersecting a ray with a quadric surface. In David Kirk, editor, *Graphics Gems III*, pages 275–283. Academic Press, Boston, 1992.

(Haines 1989) Eric Haines. Essential ray tracing algorithms. In A. Glassner, editor, *An Introduction to Ray Tracing*, pages 33–77. Academic Press, London, 1989.

(Hultquist 1991) Jeff Hultquist. Intersection of a ray with a sphere. In A. Glassner, editor, *Graphics Gems*, pages 275–283. Academic Press, Boston, 1991.

(Johnston and Shene 1992) John K. Johnston and Ching-Kuang Shene. Computing the intersection of a plane and a natural quadric. *Computers and Graphics*, 16(2):179–186, 1992.

(Roth 1982) S. Roth. Ray casting for modeling solids. *Computer Graphics and Image Processing*, 18:109–144, 1982.

(Shene 1994) Ching-Kuang Shene. Computing the intersection of a line and a cylinder. In Paul Heckbert, editor, *Graphics Gems IV*, 353–355. Academic Press, Boston, 1994.