from Equation (2.2), square both sides, and add them to get

$$x^2 + z^2 - r^2 = 0. \tag{19.17}$$

Note that this doesn't involve y. This is the equation for the infinite cylinder with $y \in (-\infty, \infty)$. The finite cylinder in Figure 19.11 is for $y \in [y_0, y_1]$. Substituting the ray equation into Equation (19.17) gives the quadratic

$$a\,t^2 + b\,t + c = 0,$$

where

$$\begin{aligned} a &= d_x^2 + d_z^2, \\ b &= 2(o_x d_x + o_z d_z), \\ c &= o_x o_x + o_z o_z - r^2. \end{aligned}$$

The outward-facing unit normal at a hit point p is

$$n = (p_x/r, 0, p_z/r).$$

Because the code for the cylinder hit function is similar to that of the sphere hit function in Listing 3.6, I'll leave its implementation as an exercise. However, see Listing 19.9.

19.5.4 Generic Torus

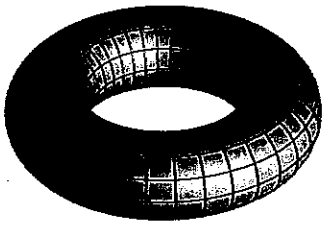

Figure 19.12. A torus.

A torus is a doughnut-shaped object, as illustrated in Figure 19.12.

We can construct a torus as follows. Consider a circle of radius b in the (y, z) plane with center at $z = a$, as shown in Figure 19.13(a). This figure also shows an arbitrary point on the circle at distance z from the y-axis. The equation of the circle is

$$(z - a)^2 + y^2 - b^2 = 0. \tag{19.18}$$

We generate the torus by rotating this circle through 360° around the y-axis. Figure 19.13(b) is a top-down view of the circle at some stage during the rotation. If we can write down the equation for the rotated circle in terms of x, y, and z, we have the implicit equation of the torus. The key observation is as follows. The red point on the circle in Figure 19.13(a) at distance z along the z-axis maintains the same distance from the y axis as the circle rotates, where it becomes $(x^2 + z^2)^{1/2}$. The equation of the torus is therefore obtained by replacing z in Equation (19.18) with $(x^2 + z^2)^{1/2}$:

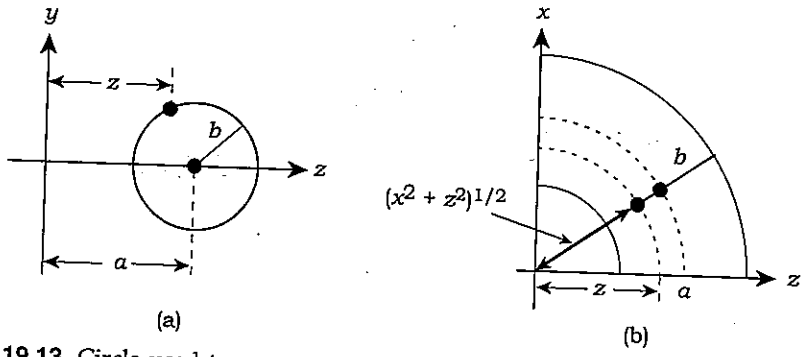

Figure 19.13. Circle used to construct a torus: (a) initial location in the (y, z) plane; (b) rotated location for generating the torus.

$$[(x^2 + z^2)^{1/2} - a]^2 + y^2 - b^2 = 0. \tag{19.19}$$

Squaring Equation (19.19) to get rid of the square root gives

$$f(x, y, z) = (x^2 + y^2 + z^2)^2 - 2(a^2 + b^2)(x^2 + y^2 + z^2) + 4a^2y^2 + (a^2 - b^2)^2 = 0. \tag{19.20}$$

This is the implicit equation of a generic torus whose central axis is the y-axis and that is bisected by the (x, z) plane. Note that this is a *fourth-degree* polynomial in x, y, and z, which is also known as a quartic equation. The two parameters a and b define the shape of the torus; a is called the *swept radius*, and b is called the *tube radius*.[4] These control the size and shape of the torus. Figure 19.14 shows cross sections of tori with different relative values of a and b.

By substituting the ray equation into Equation (19.20), we can derive the equation to solve for the ray parameter t. Unfortunately, there's a fair bit

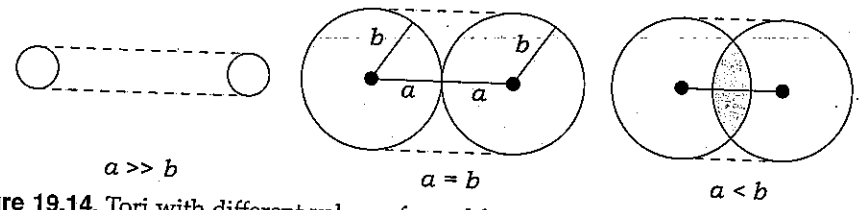

Figure 19.14. Tori with different values of a and b. Note that there is no hole in the torus when $a \leq b$.

4. The parameter a is also called the *outer radius*, and b is called the *inner radius*.

of algebra involved, but the result is the following fourth-degree polynomial in t:

$$c_4 t^4 + c_3 t^3 + c_2 t^2 + c_1 t + c_0 = 0, \tag{19.21}$$

where the coefficients are

$$\begin{aligned}
c_4 &= \left(d_x^2 + d_y^2 + d_z^2\right)^2, \\
c_3 &= 4\left(d_x^2 + d_y^2 + d_z^2\right)\left(o_x d_x + o_y d_y + o_z d_z\right), \\
c_2 &= 2\left(d_x^2 + d_y^2 + d_z^2\right)\left[o_x^2 + o_y^2 + o_z^2 - \left(a^2 + b^2\right)\right] + 4\left(o_x d_x + o_y d_y + o_z d_z\right)^2 + 4a^2 d_y^2, \\
c_1 &= 4\left[o_x^2 + o_y^2 + o_z^2 - \left(a^2 + b^2\right)\right]\left(o_x d_x + o_y d_y + o_z d_z\right) + 8a^2 o_y d_y, \\
c_0 &= 4\left[o_x^2 + o_y^2 + o_z^2 - \left(a^2 + b^2\right)\right]^2 - 4a^2\left(b^2 - o_y^2\right).
\end{aligned} \tag{19.22}$$

Since Equation (19.21) is a quartic in t, it can be written in the form

$$(t - t_1)(t - t_2)(t - t_3)(t - t_4) = 0,$$

where t_1–t_4 are the four roots. The fact that there can be from one to four real roots reflects the fact that a ray can hit a torus up to four times. Figure 19.15 illustrates this for four rays in the (x, z) plane. Rays with one or three hits are analogous to rays hitting a sphere once—they involve a tangential intersection with the torus to machine precision. This will rarely happen in practice.

Because we have to solve a quartic to ray trace a torus, these are more difficult to intersect than spheres and planes, but fortunately, help is at hand. Not only can quartics be solved in closed form, but there's public-domain

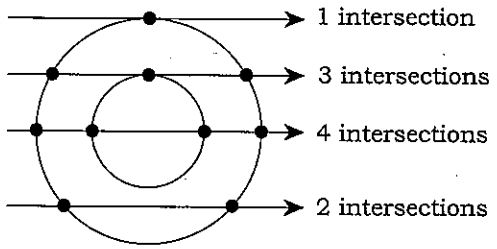

Figure 19.15. Some of the ways that a ray can hit a torus.

```
bool
Torus::hit(const Ray& ray, double& tmin, ShadeRec& sr) const {
        if (!bbox.hit(ray))
                return (false);

        double x1 = ray.o.x; double y1 = ray.o.y; double z1 = ray.o.z;
        double d1 = ray.d.x; double d2 = ray.d.y; double d3 = ray.d.z;

        double coeffs[5];               // coefficient array
        double roots[4];                // solution array

        // define the coefficients

        double sum_d_sqrd = d1 * d1 + d2 * d2 + d3 * d3;
        double e = x1 * x1 + y1 * y1 + z1 * z1 - a * a - b * b;
        double f = x1 * d1 + y1 * d2 + z1 * d3;
        double four_a_sqrd = 4.0 * a * a;

        coeffs[0] = e * e - four_a_sqrd * (b * b - y1 * y1);   // constant term
        coeffs[1] = 4.0 * f * e + 2.0 * four_a_sqrd * y1 * d2;
        coeffs[2] = 2.0 * sum_d_sqrd * e + 4.0 * f * f + four_a_sqrd * d2 * d2;
        coeffs[3] = 4.0 * sum_d_sqrd * f;
        coeffs[4] = sum_d_sqrd * sum_d_sqrd;                   // coefficient of t^4

        // find the roots

        int num_real_roots = SolveQuartic(coeffs, roots);

        bool intersected = false;
        double t = kHugeValue;

        if (num_real_roots == 0)  // ray misses the torus
                return (false);

        // find the smallest root greater than kEpsilon, if any

        for (int j = 0; j < num_real_roots; j++)
                if (roots[j] > kEpsilon) {
                        intersected = true;
                        if (roots[j] < t)
                                t = roots[j];
                }
```

Listing 19.9. The function `Torus::hit`.

```
        if (!intersected)
            return (false);

        tmin = t;
        sr.local_hit_point = ray.o + t * ray.d;
        sr.normal = computeNormal(sr.local_hit_point);

        return (true);
}
```

Listing 19.9 (continued). The function `Torus::hit`.

C code available for solving them. Schwarze (1990) discusses the solutions of cubic and quartic equations. His code is on the book's website. Herbison-Evans (1995) also discusses the solution of quartic equations.

Listing 19.9 shows the function `Torus::hit`, the first half of which computes the coefficients (19.22) and stores them in a C array. Note that the first coefficient in the array is the constant term c_0, and the last is the coefficient c_4 of t^4. All of the calculations are performed in Schwarze's function `SolveQuartic`, which returns the number of real roots and the roots themselves in the `roots` array. Because the roots in this array are not ordered by increasing t, the following code has to search the array to find the smallest t value.

The normal to the torus is given by the gradient of the function $f(x, y, z)$ in Equation (19.20), evaluated at the hit point:

$$n = \nabla f(x, y, z) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right).$$

See, for example, Thomas and Finney (1996). Performing the partial differentiations gives

$$\begin{aligned} n_x &= 4x\,[x^2 + y^2 + z^2 - (a^2 + b^2)], \\ n_y &= 4y\,[x^2 + y^2 + z^2 - (a^2 + b^2) + 2a^2], \\ n_z &= 4z\,[x^2 + y^2 + z^2 - (a^2 + b^2)]. \end{aligned}$$

As n is not a unit normal, it must be normalized. Figure 19.16 shows three ray-traced tori with the same viewing parameters and $a = 2.0$ but with different values of b.