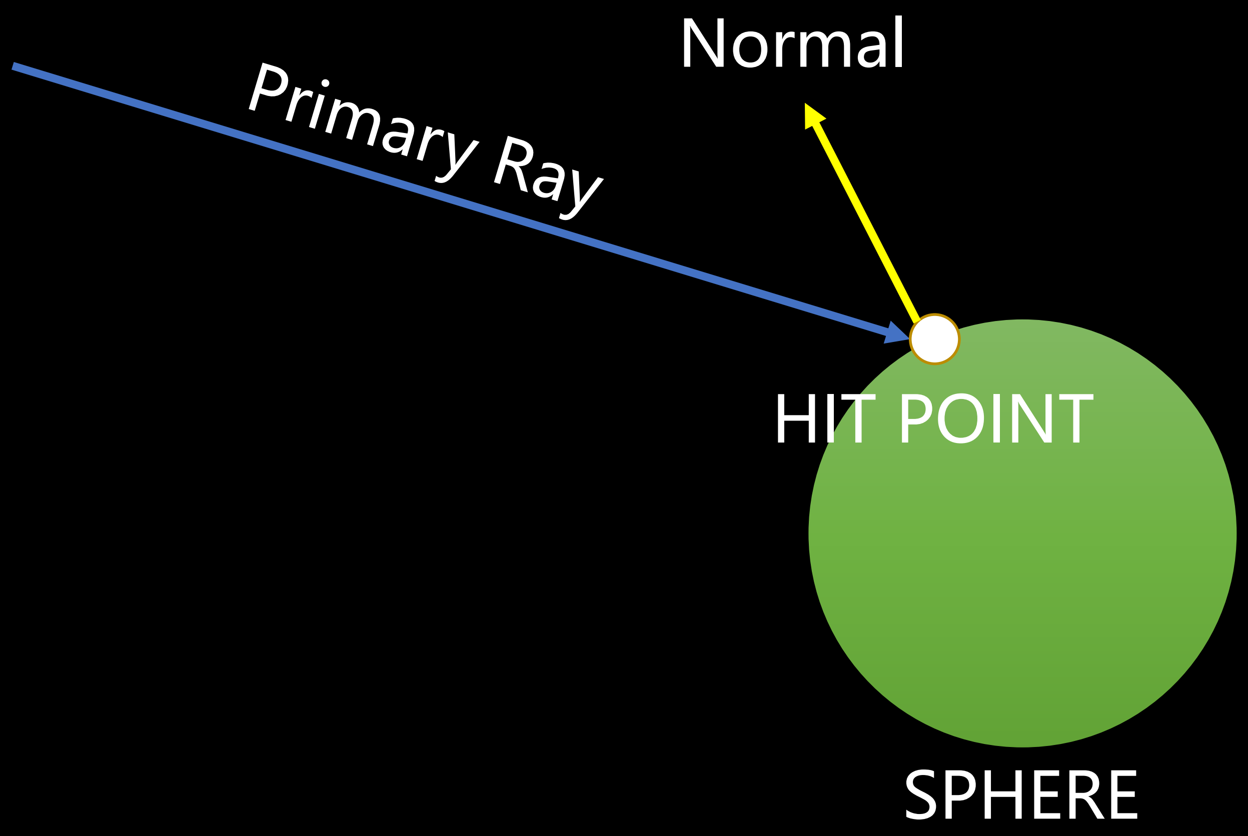
Chapter_6
交点 法线

主讲人：王世元

西南石油大学 2020



Normal
Primary Ray
HIT POINT
SPHERE

光 线 球 体 求 交 点

$$\begin{cases} a: & d * d \\ b: & 2\left(d * (O - C)\right) \\ c: & (O - C) * (O - C) - r^2 \end{cases}$$

$$\begin{cases} \Delta = b^2 - 4\,ac \\ t = \dfrac{-b \pm \sqrt{b^2 - 4\,ac}}{2a} \end{cases}$$

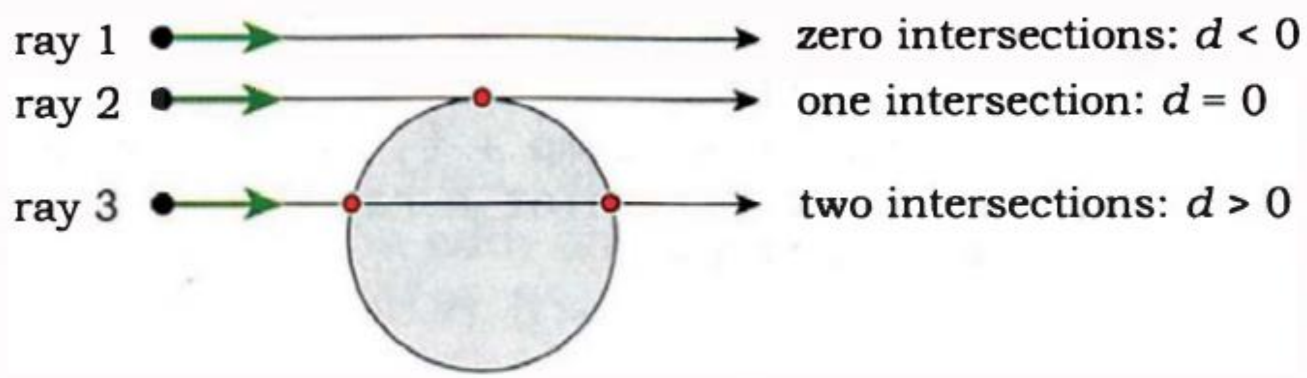


Figure 3.7. Ray-sphere intersections.

我们的第一个版本

Ray.cs
(后续扩展)

```
//射线是否与球体相交
0 个引用
public bool isHit(Sphere sphere)
{
    Vector3D oc = Origin - sphere.Center;

    double a = Direction * Direction;
    double b = 2.0 * (Direction * oc);
    double c = oc * oc - sphere.Radius * sphere.Radius;

    double delta = b * b - 4.0 * a * c;

    return delta > 0;
}
```

光线参数方程

Ray

t_0

$$P(t) = o + t_0 * d$$

线上某点

射线起点

点乘

射线方向

求交点

$$P(t) = o + t_0 * d$$

Point3D.cs

```
//点 + 向量
1 个引用
public static Point3D operator +(Point3D p, Vector3D v)
{
    return new Point3D(p.X + v.X, p.Y + v.Y, p.Z + v.Z);
}
```

Ray.cs

```
//依据参数t的值，得到射线上某点
0 个引用
public Point3D GetPoint(double t)
{
    return Origin + t * Direction;
}
```

求交点

Ray.cs

```
//射线是否与球体相交
1 个引用
public bool isHit(Sphere sphere)
{
    //计算delta
    Vector3D oc = Origin - sphere.Center;
    double a = Direction * Direction;
    double b = 2.0 * (Direction * oc);
    double c = oc * oc - sphere.Radius * sphere.Radius;
    double delta = b * b - 4.0 * a * c;

    //有交点
    if (delta > 0)
    {
        //求t
        double t = (-b - Math.Sqrt(delta)) / (2.0 * a);

        if (t < 0)
        {
            t = (-b + Math.Sqrt(delta)) / (2.0 * a);
        }

        //求交点
        Point3D hitPoint = GetPoint(t);

        return true;
    }
    else
    {
        return false;
    }
}
```

验证你的交点是否计算正确

Example

Given a ray with an origin at [1 −2 −1] and a direction vector of [1 2 4], find the nearest intersection point with a sphere of radius $S_r = 3$ centered at [3 0 5].

This means the ray intersects the sphere. From this we can calculate t_0 from (A6):

$$t_0 = \frac{-B - \sqrt{(B^2 - 4 * C)}}{2}$$

$$= \frac{13.092 - \sqrt{(31.400)}}{2}$$

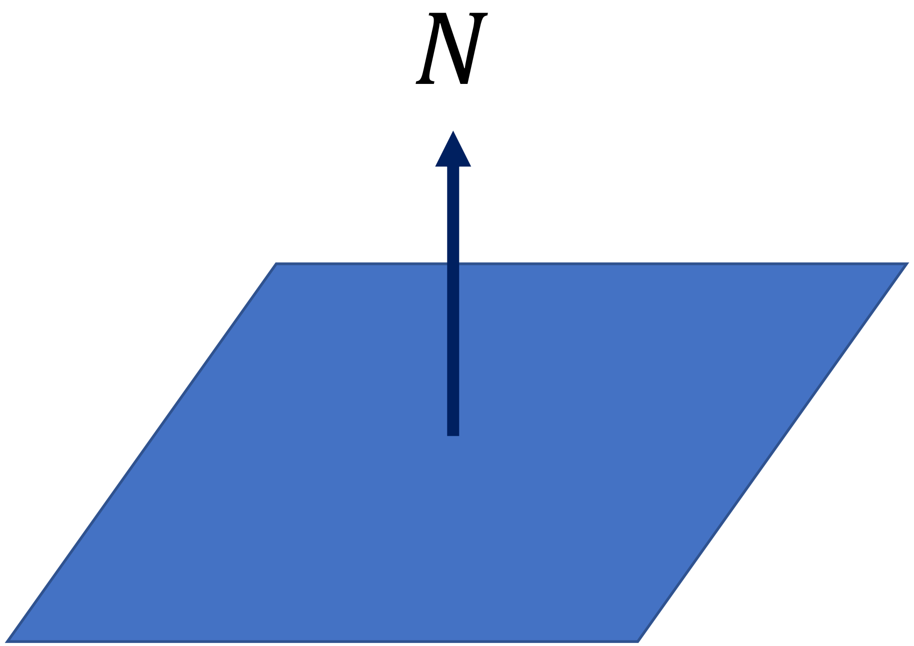
$$= 3.744$$

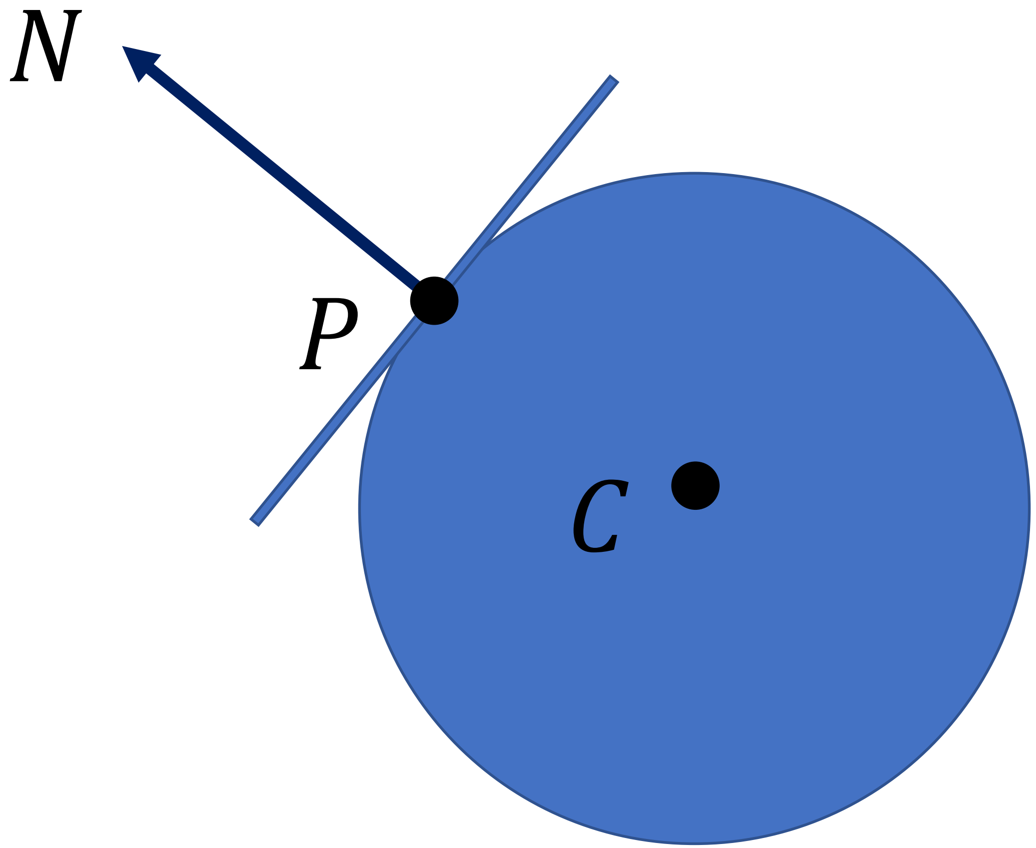
$$\mathbf{r}_i = [X_0 + X_d * t \quad Y_0 + Y_d * t \quad Z_0 + Z_d * t]$$
$$= [1 + 0.218 * 3.744 \quad -2 + 0.436 * 3.744 \quad -1 + 0.873 * 3.744]$$
$$= [1.816 \quad -0.368 \quad 2.269].$$

交点处的法线向量

球面某点的法线：若过该点的直线与过该点的切线垂直,则该直线为球面某点的法线.

如果球心坐标知道,只要将球心坐标与该点坐标相减即为球面某点的法向量





Sphere.cs

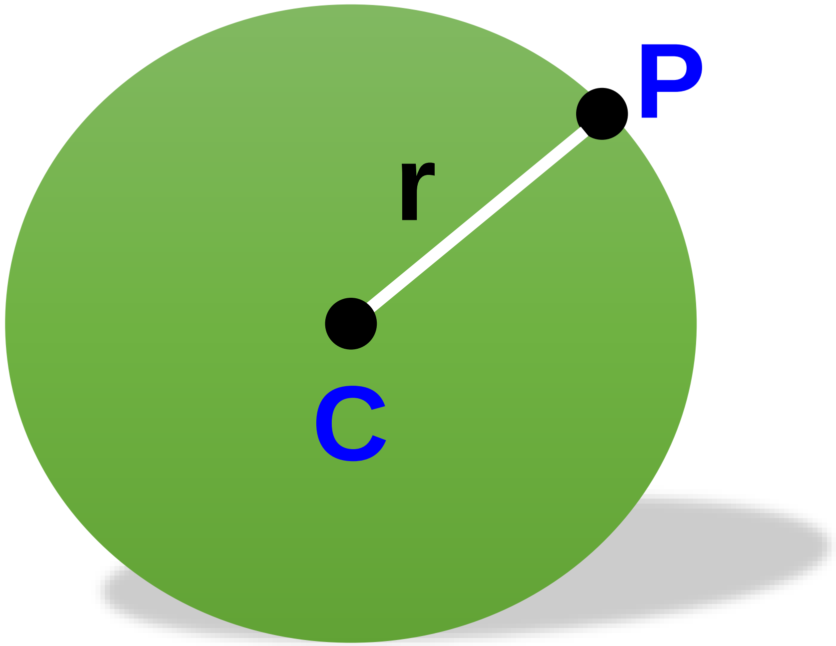
```
//获得球面某点的法线向量
0 个引用
public Vector3D GetNormalVector(Point3D p)
{
    Vector3D v = p - Center;

    v.Normalize();       //归一化，可简化后续计算

    return v;
}
```

Ray.cs

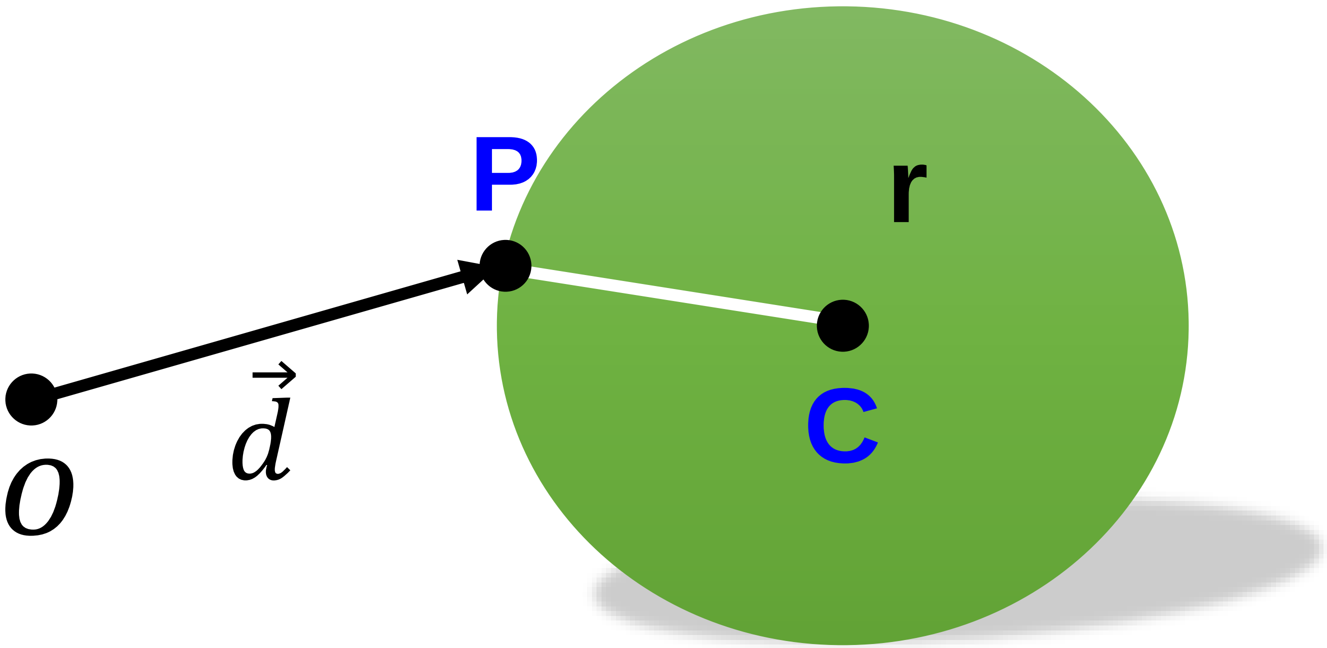
```
//求交点
Point3D hitPoint = GetPoint(t);

//求交点处的法线
Vector3D normalVector = sphere.GetNormalVector(hitPoint);
```

球体参数方程

Sphere $$(P-C)*(P-C)-r^2=0$$



光线 球体求交点



将射线表示 带入球体方程

$$(P(t) - C) * (P(t) - C) - r^2 = 0$$

光线 球体求交点

$$(P(t)-C)*(P(t)-C)-r^2=0$$

$$(O+{\color{red}t}*\vec{d}-C)*(O+{\color{red}t}*\vec{d}-C)-r^2=0$$

射线起点

射线方向

向量点乘

未知参数

球心

半径

光 线 球 体 求 交 点

$$(O + t * \vec{d} - C) * (O + t * \vec{d} - C) - r^2 = 0$$

$$\Downarrow$$

$$t^2 \vec{d} * \vec{d} + t2\left(\vec{d} * (O - C)\right) + (O - C) * (O - C) - r^2 = 0$$

一元二次方程

$$\begin{cases} a: \ \vec{d} * \vec{d} \\ b: \ 2\left(\vec{d} * (O - C)\right) \\ c: \ (O - C) * (O - C) - r^2 \end{cases}$$

光线 球体求交点

$$\begin{cases} a: & d * d \\ b: & 2\left(d * (O - C)\right) \\ c: & (O - C) * (O - C) - r^2 \end{cases}$$

$$\begin{cases} \Delta = b^2 - 4\,ac \\ t = \dfrac{-b \pm \sqrt{b^2 - 4\,ac}}{2a} \end{cases}$$

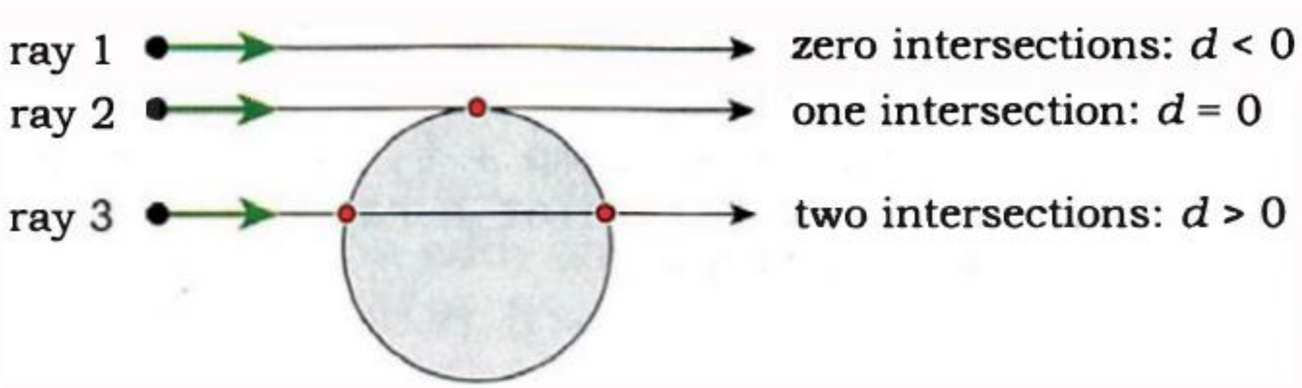


Figure 3.7. Ray-sphere intersections.

光 线 球 体 求 交 点

$a:\quad d * d$

射线方向

点乘

射线方向

$b:\quad 2\left(d * (O - C)\right)$

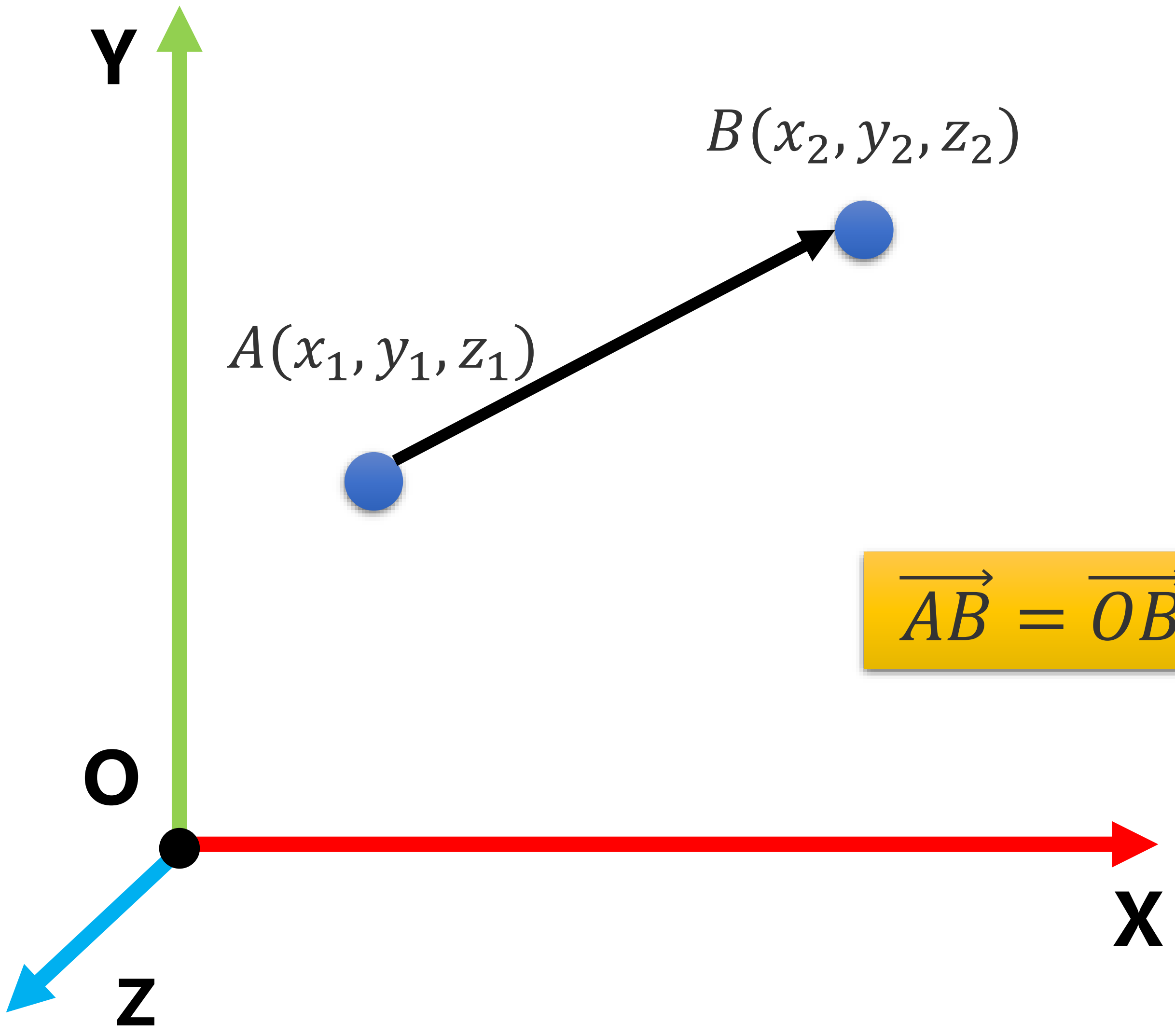
射线方向

点乘

射线起点

球心位置

两 个 点 减 法------得到从A指向B的向量



$$\overrightarrow{AB} = \overrightarrow{OB} - \overrightarrow{OA} = (x_2 - x_1, y_2 - y_1, z_2 - z_1)$$

两 个 点 减 法------得到从A指向B的向量

$$\overrightarrow{AB} = (x_2 - x_1, y_2 - y_1, z_2 - z_1)$$

Point3D.cs

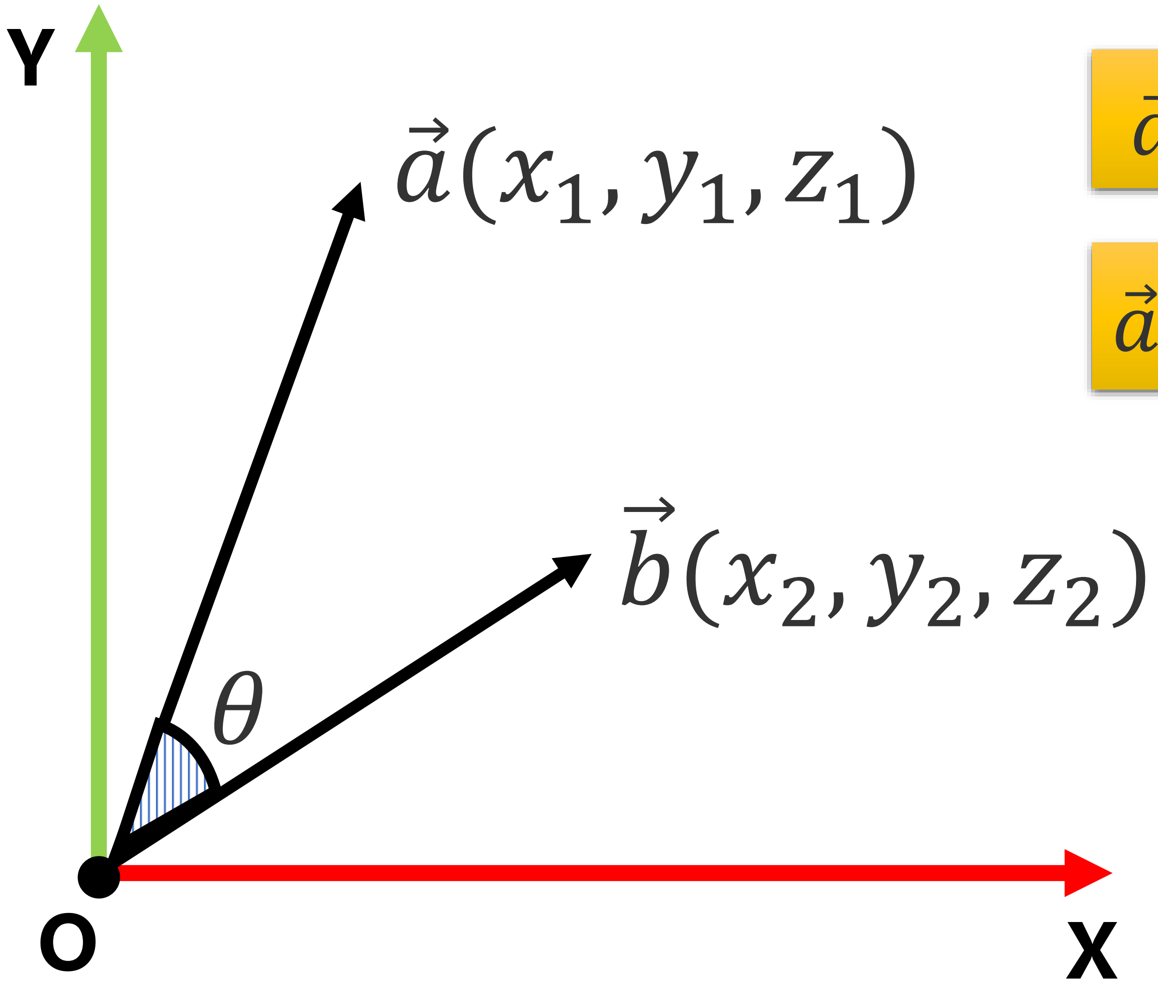
```
//两个点之间的减法，得到向量
0 个引用
public static Vector3D operator -(Point3D p1, Point3D p2)
{
    return new Vector3D(p1.X-p2.X, p1.Y - p2.Y, p1.Z - p2.Z);
}
```

Form1.cs（测试）

```
        //光线类测试
        1 个引用
        private void btnTest_Click(object sender, EventArgs e)
        {
            Point3D p1 = new Point3D(1, 2, 10);
            Point3D p2 = new Point3D(-10, 1, 3);

            Vector3D v = p1 - p2;
        }  已用时间 <= 2ms
    }
}
```

```
v {SyRayTracer.Vector3D}
X 11
Y 1
Z 7
_x 11
_y 1
_z 7
```

向量点乘



$$\vec{a} * \vec{b} = (x_1 * x_2 + y_1 * y_2 + z_1 * z_2)$$

$$\vec{a} * \vec{b} = |\vec{a}| * |\vec{b}| * cos\theta$$

向量点乘

Vector3D.cs

```
//向量点乘
0 个引用
public static double operator*(Vector3D v1, Vector3D v2)
{
    return v1.X * v2.X + v1.Y * v2.Y + v1.Z * v2.Z;
}
```

Form1.cs
(测试)

```
        //光线类测试
        1 个引用
        private void btnTest_Click(object sender, EventArgs e)
        {
            Vector3D v1 = new Vector3D(1, 2, 2);
            Vector3D v2 = new Vector3D(-1, 3, 0);

            double d = v1 * v2;
        }
    }
```

向量的其它运算

Vector3D.cs

```
//数 * 向量
0 个引用
public static Vector3D operator *(double d, Vector3D v)
{
    return new Vector3D(d * v.X, d * v.Y, d * v.Z);
}

//向量 * 数
0 个引用
public static Vector3D operator *( Vector3D v, double d)
{
    return new Vector3D(d * v.X, d * v.Y, d * v.Z);
}

//向量加法
0 个引用
public static Vector3D operator +(Vector3D v1, Vector3D v2)
{
    return new Vector3D(v1.X + v2.X, v1.Y * v2.Y, v1.Z * v2.Z);
}

//向量减法
0 个引用
public static Vector3D operator -(Vector3D v1, Vector3D v2)
{
    return new Vector3D(v1.X - v2.X, v1.Y - v2.Y, v1.Z - v2.Z);
}
```

向量的模，及归一化

Vector3D.cs

```
//向量的模，及向量的大小
2 个引用
public double Magnitude()
{
    return Math.Sqrt(X * X + Y * Y + Z * Z);
}

//对本向量进行归一化，本向量被改变
0 个引用
public void Normalize()
{
    double d = Magnitude();
    X = X / d;
    Y = Y / d;
    Z = Z / d;
}

//返回本向量的归一化向量，本向量不变
0 个引用
public Vector3D GetNormalizeVector()
{
    double d = Magnitude();
    return new Vector3D(X/d, Y/d, Z/d);
}
```

光 线 球 体 求 交 点

$$\begin{cases} a: & d * d \\ b: & 2\left(d * (O - C)\right) \\ c: & (O - C) * (O - C) - r^2 \end{cases}$$

$$\begin{cases} \Delta = b^2 - 4ac \\ t = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a} \end{cases}$$

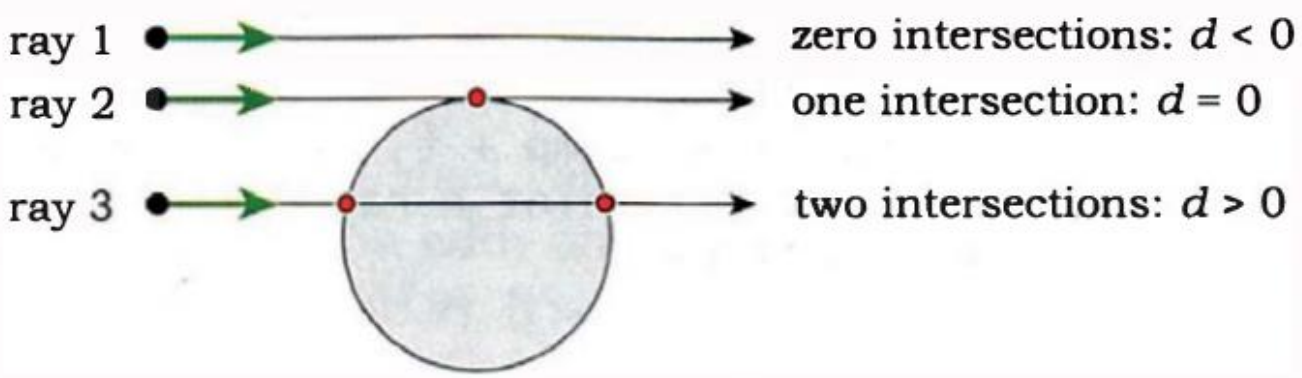


Figure 3.7. Ray-sphere intersections.

参考

```
bool hit_sphere(const vec3& center, float radius, const ray& r) {
    vec3 oc = r.origin() - center;
    float a = dot(r.direction(), r.direction());
    float b = 2.0 * dot(oc, r.direction());
    float c = dot(oc, oc) - radius*radius;
    float discriminant = b*b - 4*a*c;
    return (discriminant > 0);
}
```

```
bool
Sphere::hit(const Ray& ray, double& tmin, ShadeRec& sr) const {
     double     t;
     Vector3D temp    = ray.o - center;
     double     a       = ray.d * ray.d;
     double     b       = 2.0 * temp * ray.d;
     double     c       = temp * temp - radius * radius;
     double     disc    = b * b - 4.0 * a * c;

     if (disc < 0.0)
          return(false);
     else {
```

我们的第一个版本

Ray.cs
(后续扩展)

```
//射线是否与球体相交
0 个引用
public bool isHit(Sphere sphere)
{
    Vector3D oc = Origin - sphere.Center;

    double a = Direction * Direction;
    double b = 2.0 * (Direction * oc);
    double c = oc * oc - sphere.Radius * sphere.Radius;

    double delta = b * b - 4.0 * a * c;

    return delta > 0;
}
```

测试它是否工作正常

Example

Given a ray with an origin at $[1\ \ -2\ \ -1]$ and a direction vector of $[1\ 2\ 4]$, find the nearest intersection point with a sphere of radius $S_r = 3$ centered at $[3\ 0\ 5]$.

加入断点，逐句调试，是否正确?

First normalize the direction vector, which yields:

$$
\begin{aligned}
&\text{direction vector magnitude} = \sqrt{(1 * 1 + 2 * 2 + 4 * 4)} = \sqrt{21} \\
&\mathbf{R_d} = [1/\sqrt{21} \quad 2/\sqrt{21} \quad 4/\sqrt{21}] \\
&\quad\;\; = [0.218\ 0.436\ 0.873].
\end{aligned}
$$

Now find A, B, and C, using equation (A5):

$$
\begin{aligned}
&A = 1 \text{ (because the ray direction is normalized)} \\
&B = 2 * (0.218 * (1 - 3) + 0.436 * (-2 - 0) + 0.873 * (-1 - 5)) \\
&\quad = -13.092 \\
&C = (1 - 3)^2 + (-2 - 0)^2 + (-1 - 5)^2 - 3^2 \\
&\quad = 35.
\end{aligned}
$$