

Introduzione alla Generazione di Immagini Fotorealistiche

*Corso di Dottorato in Matematica e Informatica
Università degli Studi della Basilicata*

Dott. Ugo Erra

2° Lezione – Ray Casting - I parte

Sommario

- Ray Casting
 - Raggi
 - Intersezioni
 - Un Ray Tracing semplice per viste ortografiche
-

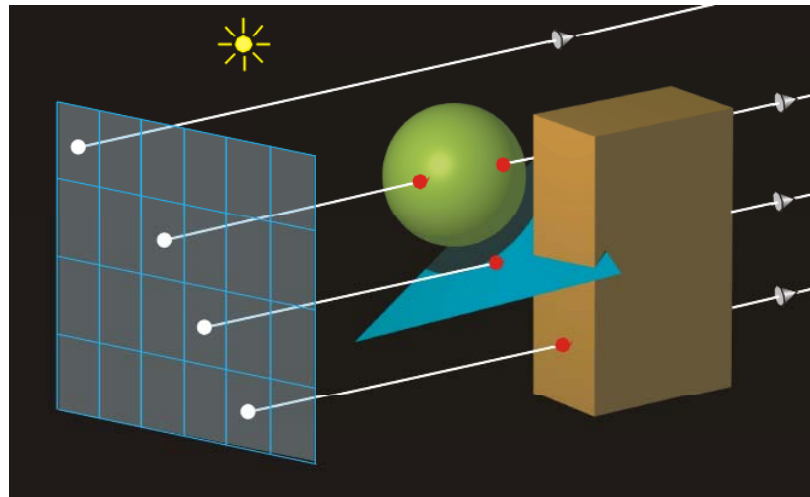
Cosa accade in natura

- In natura, una sorgente di luce emette un raggio che viaggia fino a raggiungere una superficie che ne interrompe il tragitto
 - Si può pensare al raggio come ad un fascio di fotoni che viaggia sulla stessa linea
- Lungo il suo tragitto può accadere che un raggio subisca uno o più di questi fenomeni
 - Assorbimento
 - Riflessione
 - Rifrazione
- Una superficie può riflettere tutto o parte del raggio di luce, in una o più direzioni
 - Sommando i valori di assorbimento, riflessione e rifrazione, si ottiene esattamente la potenza del raggio entrante
 - I raggi riflessi e/o rifratti possono colpire altre superfici, dove verranno assorbiti, riflessi e rifratti (di nuovo)
- Alcuni di questi raggi, alla fine del viaggio, colpiscono il nostro occhio, permettendoci di vedere la scena e contribuendo al disegno dell'immagine finale

(wikipedia)

Ray Casting

- Nel *ray casting* generiamo dei raggi per ogni pixel in direzione della camera
- Lo scopo è determinare quali oggetti l'occhio vede attraverso ogni pixel
- Per ogni raggio troviamo l'oggetto che ne blocca il percorso
- Se più di un oggetto si trova lungo il percorso del raggio consideriamo solo il più vicino
- Le proprietà del materiale e le luci presenti nella scena ci permetteranno di determinare il colore dell'oggetto



Ray Casting

- Define some objects
- Specify a material for each object
- Define some light sources
- Define a window whose surface is covered with pixel

- For each pixel
 - Shoot a ray towards the objects from the center of the pixel
 - Compute the nearest hit point of the ray with the objects (if any)

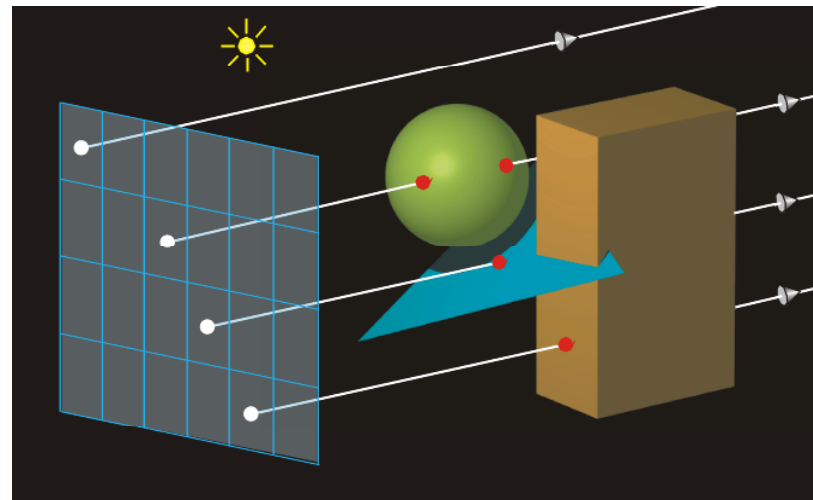
 - If the ray hits an object
 - Use the object's material and the light to compute the pixel color
 - else
 - Set the pixel color to black

Simuliamo

- I raggi del ray casting sono differenti rispetto ai raggi di luce (o fotoni):
 - Nel ray casting i raggi viaggiano dall'osservatore verso gli oggetti
 - Un raggio di luce viaggia dalla sorgente di luce all'osservatore
 - Assumiamo che i raggi attraversano gli oggetti
 - Per poter determinare l'oggetto colpito più vicino all'osservatore
-

Alcune definizioni sul Ray Casting

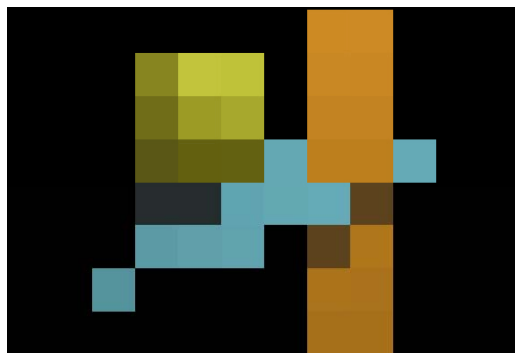
- Il *view plane* è il piano sul quale proiettiamo i pixel
- Se tutti i raggi sono perpendicolare tra di loro e producono una *proiezione ortografica*
- Lo *shading* è il processo con il quale determiniamo il colore del pixel utilizzando le proprietà del materiale
- La fase in cui determiniamo gli oggetti colpiti è chiamata *ray object intersection*



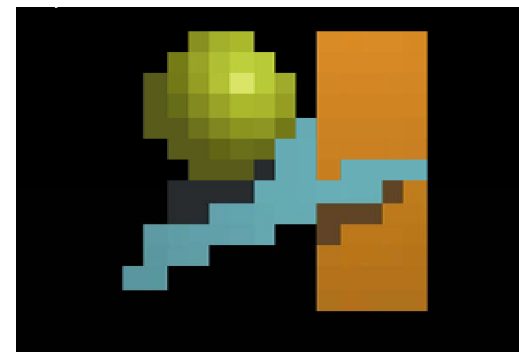
Risoluzione



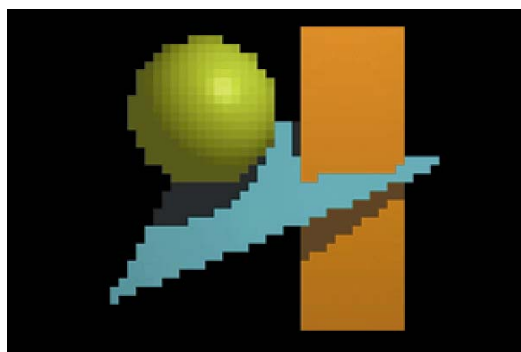
6 x 4



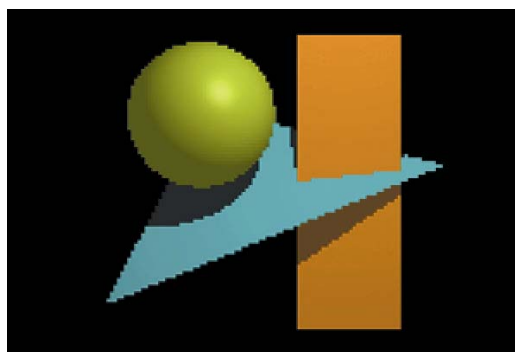
12 x 8



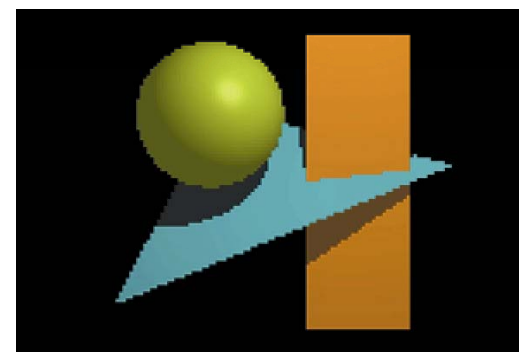
24 x 16



60 x 40



120 x 80



150 x 100

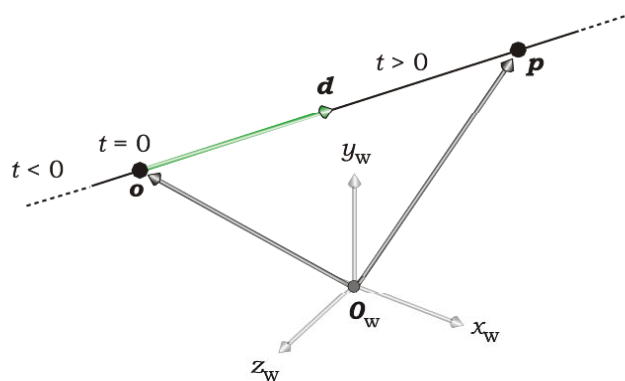
World coordinate

- Nel ray tracing la geometria degli oggetti, le luci, la camera, il view plane ed il colore di fondo rappresenta la *scena*
 - Tutti gli elementi della scena si trovano nel mondo (*world space*) e la loro posizione ed orientamento è definita attraverso le *world coordinate*
 - Chiamate anche *coordinate assolute* nel mondo
 - Attualmente abbiamo solo il view plane e degli oggetti
 - Le coordinate di tipo world saranno indicate come (x_w, y_w, z_w)
-

In che modo definiamo un raggio?

- Un raggio è una linea retta infinita definita attraverso
 - Un punto o chiamato origine
 - Un vettore d chiamato direzione
- Ogni raggio è definito parametricamente attraverso un parametro t nel seguente modo

$$p = o + t d$$



Tipi di raggi

- Raggi primari (primary rays)
 - Sono “sparati” dal centro del pixel a partire dalla posizione della camera
 - Raggi secondari (secondary rays)
 - Sono raggi riflessi e rifratti a partire dai punti della superficie colpiti dai raggi primari
 - Ombra
 - Sono raggi che generati dalle superfici in direzione delle sorgenti di luce
 - Raggi di luce
 - Sono raggi generati dalle sorgenti di luce ed utilizzati per simulare alcuni aspetti della simulazione della luce
-

Classe Ray

```
class Ray {
    public:

        Point3D    o;    // origin
        Vector3D   d;    // direction

        Ray(void);

        Ray(const Point3D& origin, const Vector3D& dir);

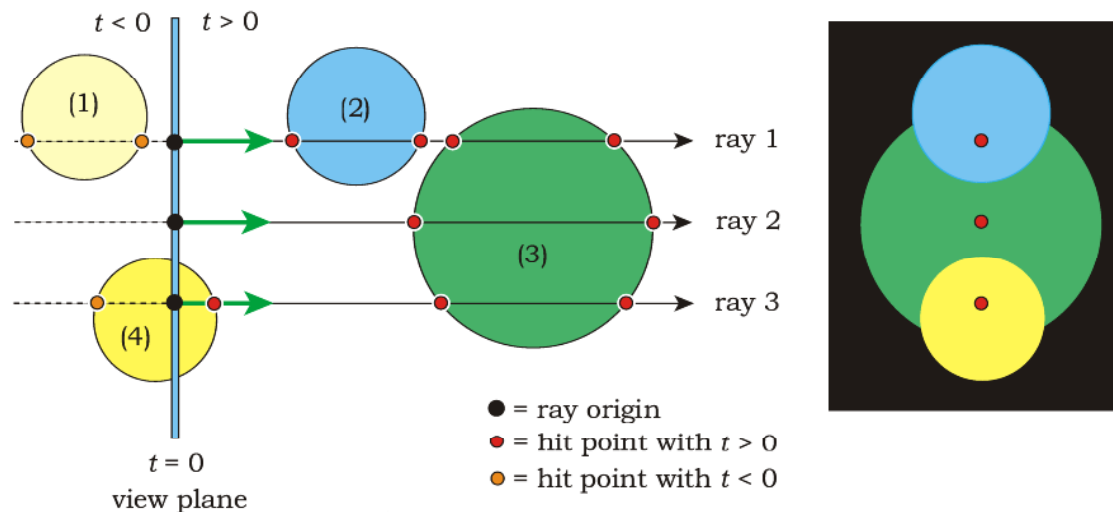
        Ray(const Ray& ray);

        Ray&
        operator= (const Ray& rhs);

        ~Ray(void);
};
```

Intersezioni

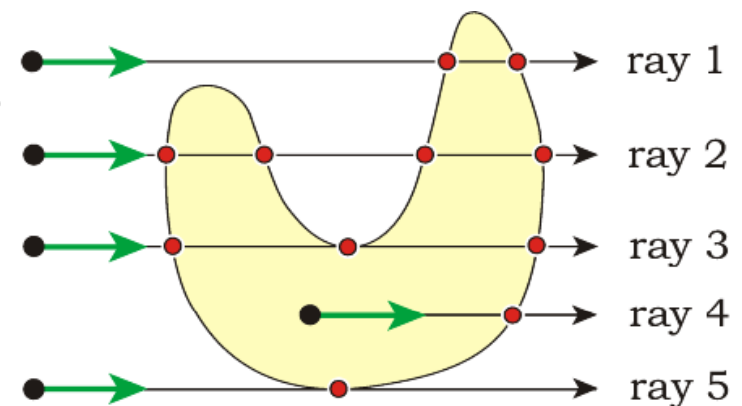
- L'operazione di base nel ray tracing è determinare il punto di intersezione t dell'oggetto colpito più vicino dal raggio di origine o lungo la direzione d
 - Per comodità il valore di $t \in [\varepsilon, +\infty)$ dove $\varepsilon = 10^{-6}$
- Poiché l'origine del raggio può trovarsi ovunque nel world space il valore di t può essere $<, =, >$ di zero



Intersezioni con superfici implicite

- Consideriamo l'intersezione raggio-superficie implicita
- Un raggio può colpire una superficie un certo numero di volte in base alla complessità della superficie

- Se la superficie è chiusa il raggio dovrà uscire una volta entrato
- Il raggio potrà essere generato anche internamente



Intersezioni con superfici implicite

- Dalla definizione di superficie implicita

$$f(x, y, z) = 0$$

consideriamo

$$f(\mathbf{p}) = 0$$

dove $\mathbf{p}=(x, y, z)$

- *Il punto di intersezione deve essere tale da soddisfare sia l'equazione del raggio che l'equazione della superficie implicita, ovvero*

$$f(\mathbf{o} + t\mathbf{d}) = 0$$

- Il valore più piccolo di t ci darà il punto di intersezione cercato sostituendolo in $\mathbf{p}=\mathbf{o} + t\mathbf{d}$
-

Intersezioni raggi-oggetti

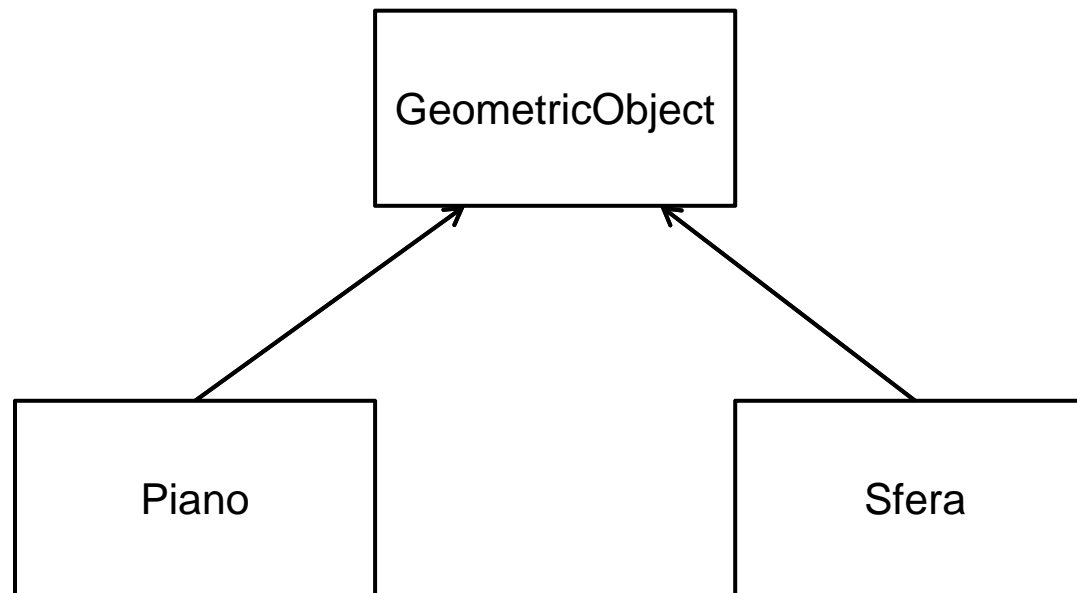
- Scene con superfici implicite differenti prevedono procedure di intersezioni specifiche per ogni tipo di oggetto presente
 - Alcune semplici casi:
 - Piano
 - Sfera
-

Alcune considerazioni sulle intersezioni

- Poiché all'interno della scena possono esserci diversi oggetti l'intersezione deve essere verificata per tutti gli oggetti
 - Tra tutti gli oggetti “colpiti” dal raggio sceglieremo il valore di t più piccolo
 - L'esatto valore del punto di intersezione sarà calcolato in fase di shading
 - Non è necessario calcolare il punto di intersezione fino a quando non abbiamo determinato il valore di t minimo
-

Gestione degli oggetti

- La classe degli oggetti che utilizzeremo avrà la classe `GeometricObject` come classe base



Classe GeometricObject

```
class GeometricObject {  
  
    public:  
  
        virtual bool  
        hit(const Ray& ray, double& t, ShadeRec& s) const = 0;  
  
    protected:  
  
        RGBColor color;  
  
};
```

Intersezione raggio-piano

- Per ottenere il punto di intersezione raggio-piano sostituiamo l'equazione del raggio

$$\mathbf{p} = \mathbf{o} + t\mathbf{d}$$

all'interno dell'equazione del piano

$$(\mathbf{p} - \mathbf{a}) \cdot \mathbf{n} = 0$$

ottenendo un'equazione lineare in t

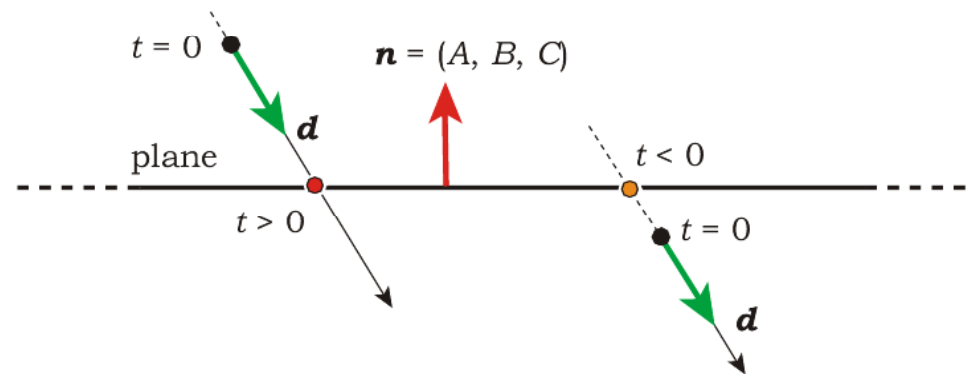
$$(\mathbf{o} + t\mathbf{d} - \mathbf{a}) \cdot \mathbf{n} = 0$$

- Il valore di t per il quale è soddisfatta l'equazione è

$$t = (\mathbf{a} - \mathbf{o}) \cdot \mathbf{n} / (\mathbf{d} \cdot \mathbf{n})$$

Intersezione raggio-piano

- Poiché un'equazione lineare può avere un sola soluzione avremo un solo valore per t
 - Il raggio interseca un piano in un solo punto



- Se il piano ed il raggio sono paralleli? In questo caso avremo che $d \cdot n = 0$

Sottoclasse Plane

```
class Plane: public GeometricObject {
public:
    Plane(void);

    Plane(const Point3D p, const Normal& n);
    ...
    virtual bool
    hit(const Ray& ray, double& t, ShadeRec& s) const;

private:

    Point3D          point;

    Normal           normal;
    static const double kEpsilon;    // for shadows and secondary rays
};
```

Metodo Plane::hit

```
bool
Plane::hit(const Ray& ray, double& tmin, ShadeRec& sr) const {
    float t = (a - ray.o) * n / (ray.d * n);

    if (t > kEpsilon) {
        tmin = t;
        sr.normal = n;
        sr.local_hit_point = ray.o + t * ray.d;

        return (true);
    }
    else
        return (false);
}
```

Intersezione raggio-sfera

- Per ottenere il punto di intersezione raggio-piano sostituiamo l'equazione del raggio

$$\mathbf{p} = \mathbf{o} + t\mathbf{d}$$

all'interno dell'equazione della sfera

$$(\mathbf{p} - \mathbf{c}) \cdot (\mathbf{p} - \mathbf{c}) - r^2 = 0$$

ottenendo

$$(\mathbf{o} + t\mathbf{d} - \mathbf{c}) \cdot (\mathbf{o} + t\mathbf{d} - \mathbf{c}) - r^2 = 0$$

Intersezione raggio-sfera

- Sviluppando ulteriormente otteniamo una equazione di secondo grado

$$(d \cdot d)t^2 + [2(o - c) \cdot d]t + (o - c) \cdot (o - c) - r^2 = 0$$

- Impostando

$$a = (d \cdot d)t^2$$

$$b = 2(o - c) \cdot d$$

$$c = (o - c) \cdot (o - c) - r^2 = 0$$

otteniamo

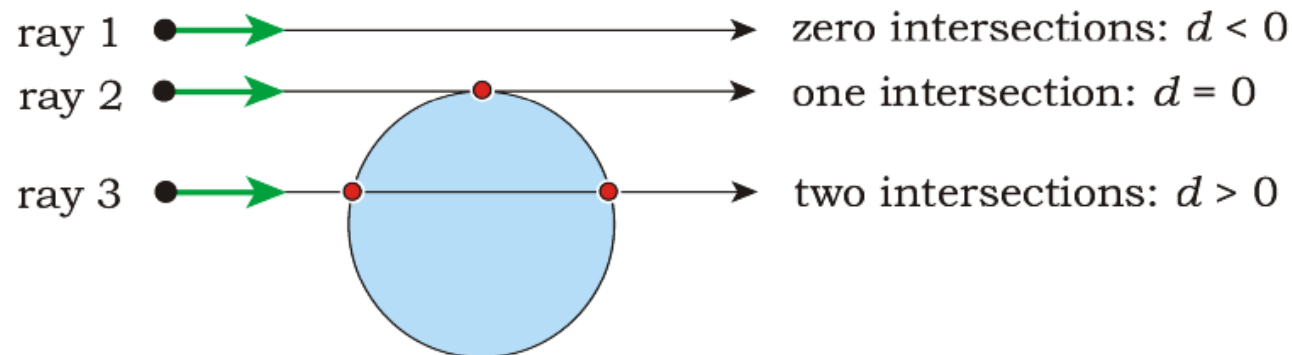
$$t = [-b \pm (b^2 - 4ac)^{1/2}] / 2a$$

Intersezione raggio-sfera

- Una equazione di secondo grado può avere zero, uno o due soluzioni nel campo reale in base al valore del discriminante

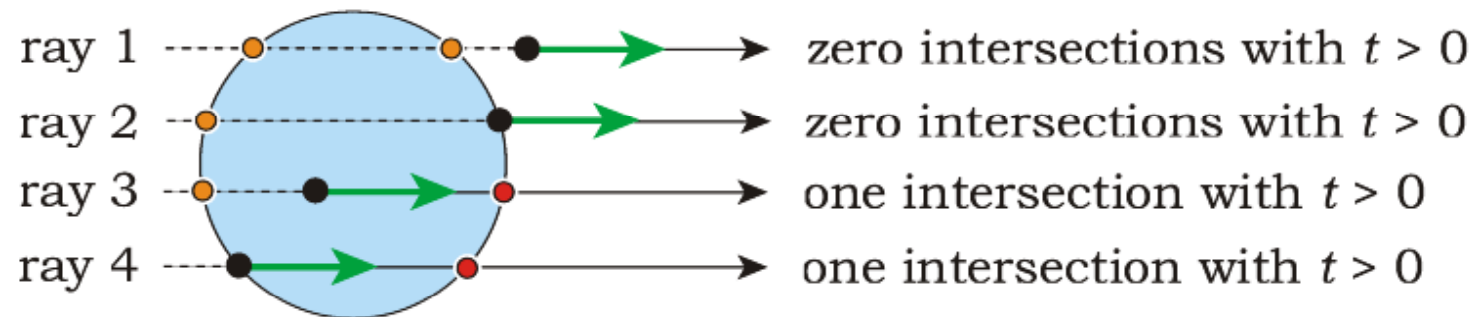
$$d = b^2 - 4ac$$

- Le soluzioni corrispondono ai modi con i quali un raggio può intersecare una sfera



Intersezione raggio-sfera

- La funzione che determina il punto di intersezione raggio-sfera dovrà gestire i seguenti casi:



- In definitiva durante il ray casting non ci interessano i punti di intersezioni di eventuali raggi generati sulla superficie della sfera
 - Corrispondono a tutti quelli per $t=0$
-

Sottoclasse Sphere

```
class Sphere: public GeometricObject {
public:
    Sphere(void);

    Sphere(const Point3D center, double r);
    ...
    virtual bool
    hit(const Ray& ray, double& t, ShadeRec& s) const;

private:
    Point3D          point;

    double           radius;
    static const double kEpsilon; // for shadows and secondary rays
};
```

Metodo Sphere::hit

```
Sphere::hit(const Ray& ray, double& tmin, ShadeRec& sr) const {
    double t;
    Vector3D temp      = ray.o - center;
    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 {
        double e = sqrt(disc);
        double denom = 2.0 * a;
        t = (-b - e) / denom;           // smaller root

        if (t > kEpsilon) {
            tmin = t;
            sr.normal      = (temp + t * ray.d) / radius;
            sr.local_hit_point = ray.o + t * ray.d;
            return (true);
        }

        t = (-b + e) / denom;           // larger root

        if (t > kEpsilon) {
            tmin = t;
            sr.normal      = (temp + t * ray.d) / radius;
            sr.local_hit_point = ray.o + t * ray.d;
            return (true);
        }
    }
    return (false);
}
```