

Introduzione alla Generazione di Immagini Fotorealistiche

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

Dott. Ugo Erra

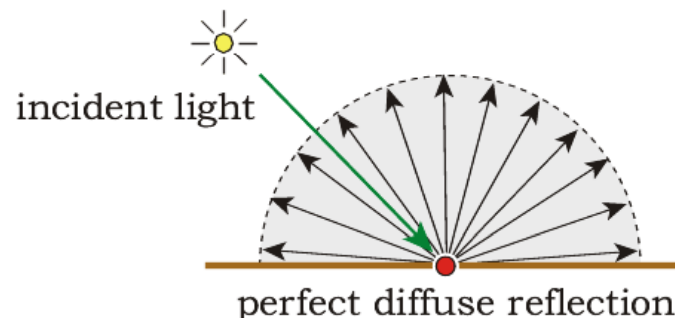
9° Lezione – Luci e Materiali

Sommario

- Perfect diffuse reflection
 - La classe BRDF
 - La sottoclasse Lambertian
 - Tipi di illuminazione
 - La classe Light
 - La classe ShadeRec
-

Perfect Diffuse BRDF - 1

- Una superficie ideale diffusa riflette la radianza incidente in maniera equa in tutte le direzioni
 - Definita anche *superficie di lambert*
- In natura non esiste una superficie del genere ma alcuni materiali possono essere approssimati in questi modo



Perfect Diffuse BRDF - 1

- Poiché non ha importanza la direzione della radianza riflessa abbiamo che è indipendente da ω_o

$$L_o(\mathbf{p}, \omega_o) = L_{r,d}(\mathbf{p})$$

- La BRDF per un materiale del genere sarà:

$$f_{r,d}(\mathbf{p}) = L_{r,d}(\mathbf{p}) / E(\mathbf{p})$$

- Siamo interessati a calcolare la riflettanza ovvero la proporzione di luce incidente che una data superficie è in grado di riflettere

$$\rho_d(\mathbf{p}) = d\Phi_o / d\Phi_i = f_{r,d}(\mathbf{p}) \pi$$

da cui

$$f_{r,d}(\mathbf{p}) = \rho_d(\mathbf{p}) / \pi$$

Bihemispherical reflectance

- La frazione di flusso incidente che viene riflessa è chiamata *bihemispherical reflectance* ρ_{hh}
- La riflettanza è definita come

$$\rho(p, \Omega_i, \Omega_o) = \frac{\int_{\Omega_o} \int_{\Omega_i} f_r(p, \omega_i, \omega_o) L_i(p, \omega_i) \cos \theta_i \cos \theta_o d\omega_i d\omega_o}{\int_{\Omega_i} L_i(p, \omega_i) \cos \theta_i d\omega_i}$$

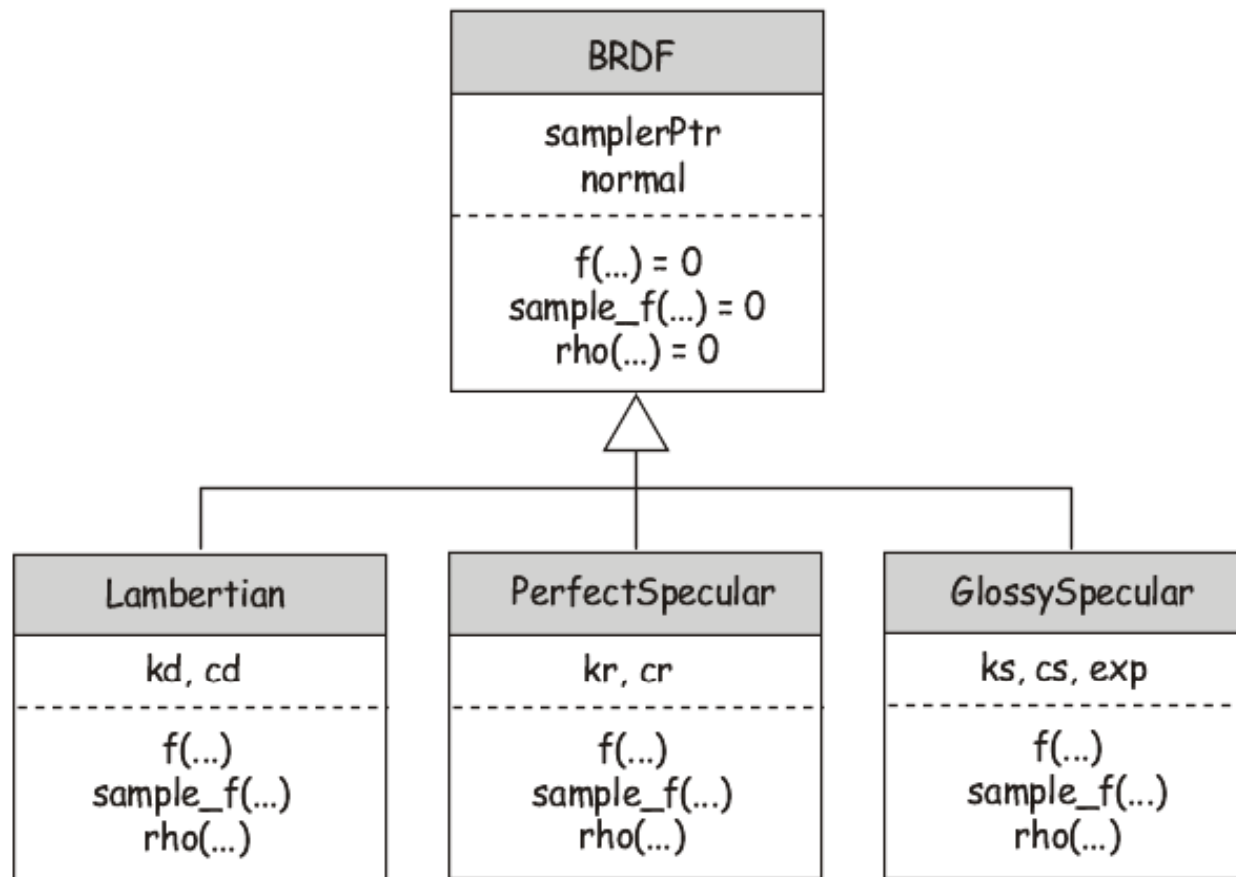
- Poiché $L_i(\mathbf{p}, \omega_i) = L_i$ e $f_r(\mathbf{p}, \omega_i, \omega_o) = f_{r,d}(\mathbf{p})$ abbiamo che

$$\rho_{hh}(p) = \frac{f_{r,d}(p)}{\pi} \int_{2\pi+} \int_{2\pi+} \cos \theta_i \cos \theta_o d\omega_i d\omega_o = \pi f_{r,d}(p) = \rho_d(p)$$

Implementazione della BRDF

- Un oggetto BRDF implementa i processi di riflessione necessari a fornire ai materiali il proprio aspetto
 - Ogni materiale dovrà avere almeno una BRDF
 - Ed ogni poligono dovrà avere almeno un materiale
 - La classe BRDF implementa i meccanismi di base
-

La gerarchia delle BRDF



La classe BRDF

```
class BRDF{
public:
    ...
    virtual RGBColor f(const ShadeRec& sr, const Vector3D& wi, const Vector3D& wo)
        const = 0;

    virtual RGBColor sample_f(const ShadeRec& sr, Vector3D& wi,
        const Vector3D& wo) const = 0;

    virtual RGBColor rho(const ShadeRec& sr, const Vector3D& wi)
        const = 0;

protected:
    Sampler* sampler_ptr;
};
```

La sottoclasse Lambertian

```
class Lambertian: public BRDF{
public:
    // constructors, etc.
    // access functions for kd and cd
    ...
    virtual RGBColor f(const ShadeRec& sr, const Vector3D& wi, const Vector3D& wo)
        const = 0;

    virtual RGBColor sample_f(const ShadeRec& sr, Vector3D& wi,
        const Vector3D& wo) const = 0;

    virtual RGBColor rho(const ShadeRec& sr, const Vector3D& wi)
        const = 0;

private:

    float kd;
    RGBColor cd;
};
```

Implementazione della sottoclasse Lambertian

- La riflettanza ρ_d è un colore RGB espresso come

$$\rho_d = k_d \mathbf{c}_d$$

dove

$k_d \in [0,1]$ è un coefficiente di riflessione diffusa

\mathbf{c}_d è il colore diffuso

- In questo modo possiamo modificare la frazione di luce riflessa variando solo k_d
-

La sottoclasse Lambertian

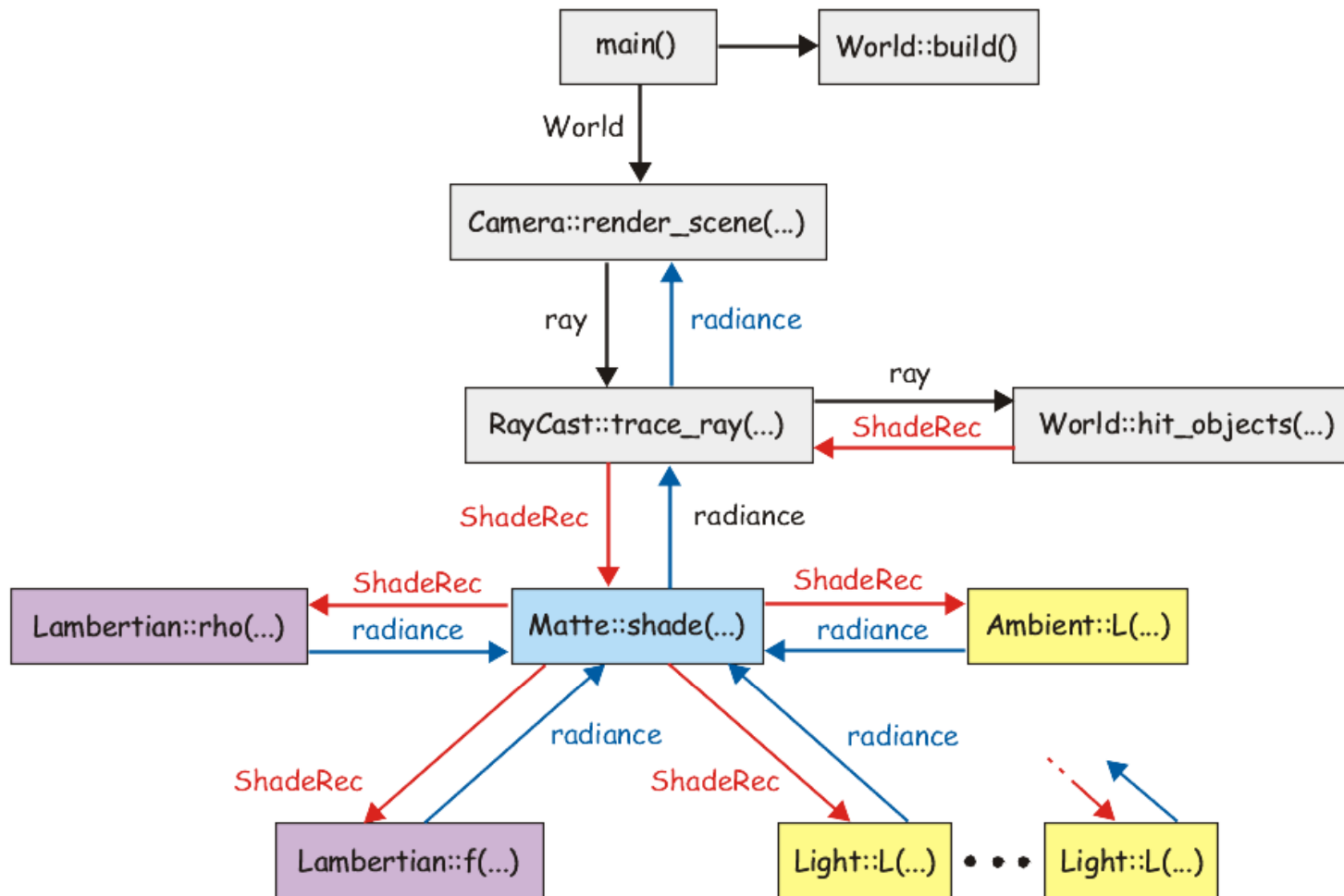
```
RGBColor Lambertian::f(const ShadeRec& sr, const Vector3D& wi, const Vector3D& wo)
const
{
    return (kd * cd * invPI);
}

RGBColor Lambertian::rho(const ShadeRec& sr, const Vector3D& wi, const Vector3D& wo)
const
{
    return (kd * cd);
}
```

Shading

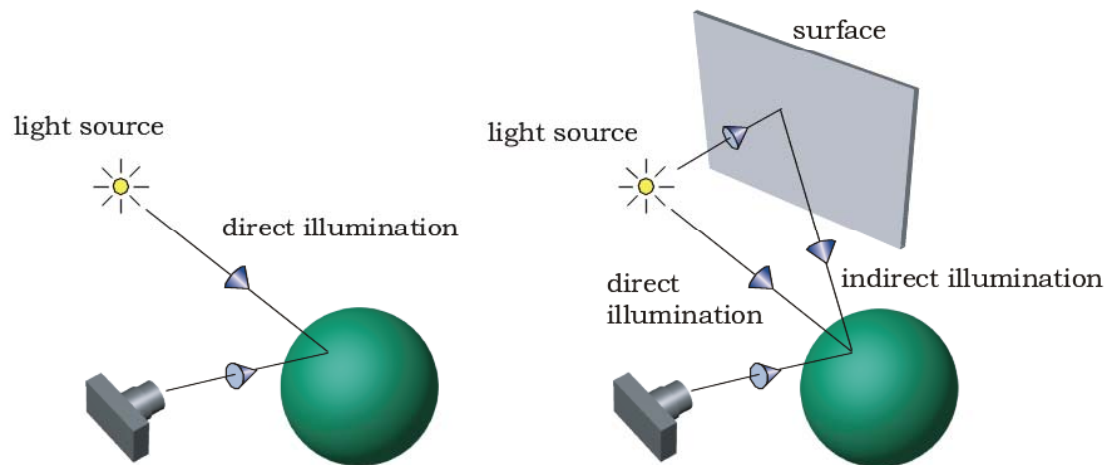
- Lo shading è il processo con il quale calcoliamo l'apparenza degli oggetti all'interno della scena
 - L'interazione della luce con il materiale avviene attraverso la BRDF
-

La gestione dello shading



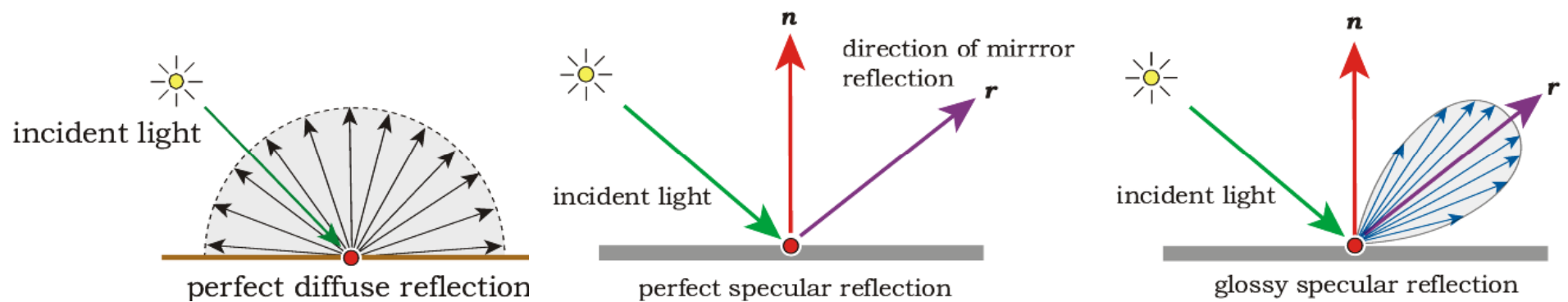
Illuminazione e riflessione

- Il processo di *illuminazione* di una scena descrive in che modo la luce dalle sorgenti interagisce con la superfici degli oggetti
- Distinguiamo due tipi di illuminazione:
 - L'*illuminazione diretta* descrive l'interazione della luce con una superficie che proviene direttamente da una sorgente (*local illumination*)
 - L'*illuminazione indiretta* descrive l'interazione della luce con una superficie dopo che è stata riflessa da uno o più oggetti (*global illumination*)

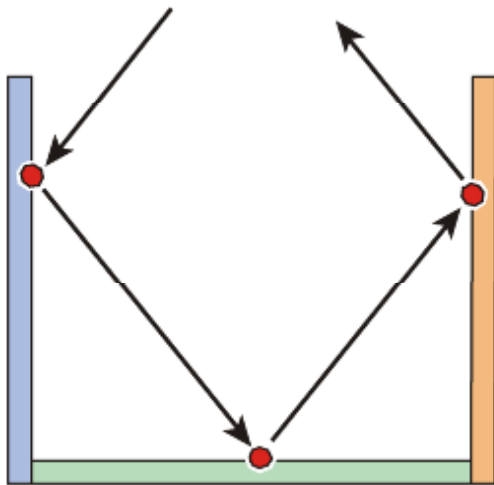


Come modellare la riflessione?

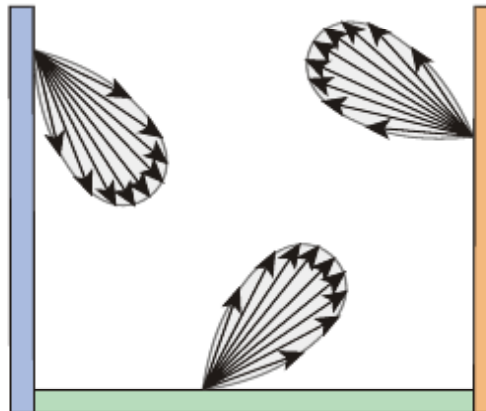
- Poiché modellare la riflessione da una superficie è un processo complesso si preferisce utilizzare una combinazione di diversi modelli di riflessione
- I modelli di riflessione utilizzati sono:



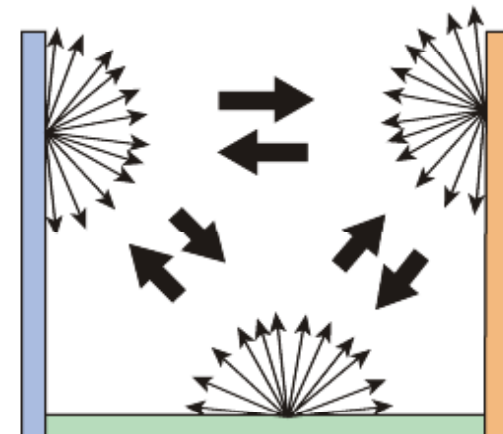
Illuminazione della scena



Le superfici perfettamente speculari possono essere utilizzate per gli specchi



Le superfici glossy sono adoperate per l'illuminazione diretta e indiretta e richiedono l'uso di molti raggi



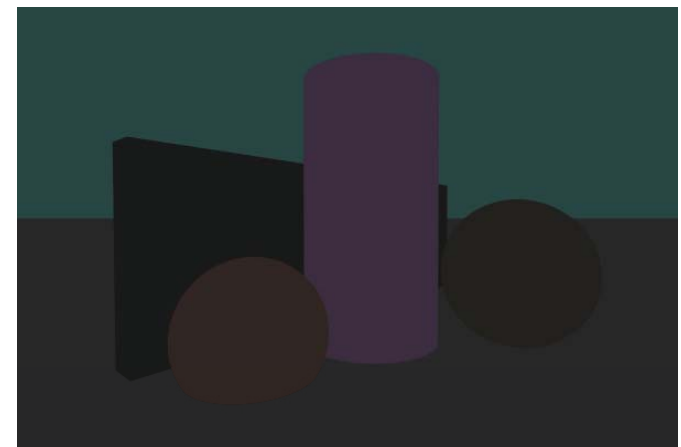
Anche le superfici diffuse sono adoperate per l'illuminazione diretta e indiretta e richiedono l'uso di molti raggi

Tipi di luce

- I tipi di luce che avremo all'interno della scena sono:
 - Ambientale (ambient light)
 - Direzionale (directional light)
 - Puntiforme (point light)
 - Per ogni luce piuttosto che utilizzare il flusso radiante utilizzeremo un colore c_l ed un fattore di scala radiante $l_s \in [0, \infty)$
 - Un raggio di luce trasferisce una quantità di radianza pari a $l_s c_l$
-

Ambient light

- Simulare l'illuminazione indiretta è un processo complesso
- Una possibile soluzione è assumere che l'illuminazione indiretta sia costante all'interno della scena e definirla come *ambient light*
 - Sebbene non sia corretto è sempre meglio questo fattore costante e non nullo
- Lo scopo è di illuminare anche quella parti della scena che non ricevono illuminazione diretta



Ambient light

- La radianza incidente $L_i(\mathbf{p}, \omega_i)$ dalla luce ambientale è

$$L_i = l_s \mathbf{c}_l$$

in quanto è indipendente da \mathbf{p} e ω_i

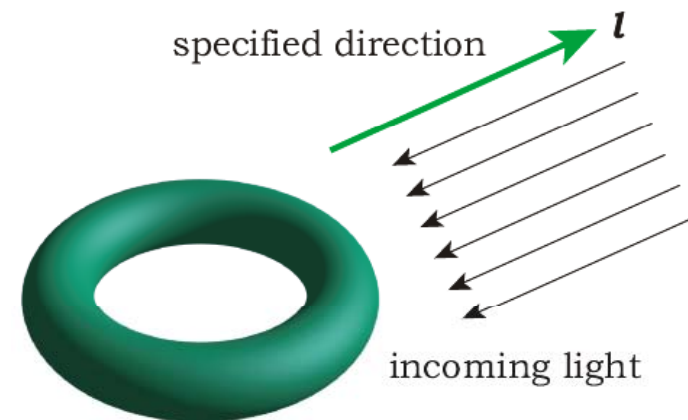
- La radianza riflessa ambientale è data da

$$L_o(\mathbf{p}, \omega_i) = \rho_{hh}(\mathbf{p}) * l_s \mathbf{c}_l$$

- L'operatore $*$ moltiplica componente per componente due colori RGB
-

Directional light

- Una luce direzionale ha la caratteristica che i raggi sono tutti paralleli tra di loro
- Il sole può essere considerato una luce direzionale data la distanza dalla terra ma nella realtà non esistono luci direzionali
- Indichiamo con l il vettore direzione della luce incidente opposto alla direzione della luce



Directional light

- La luce direzionale incide su di un punto p di una quantità pari a

$$L_o(p, \omega_o) = \int_{2\pi^+} f_r(p, \omega_i, \omega_o) L_i(p, \omega_i) \cos \theta_i d\omega_i$$

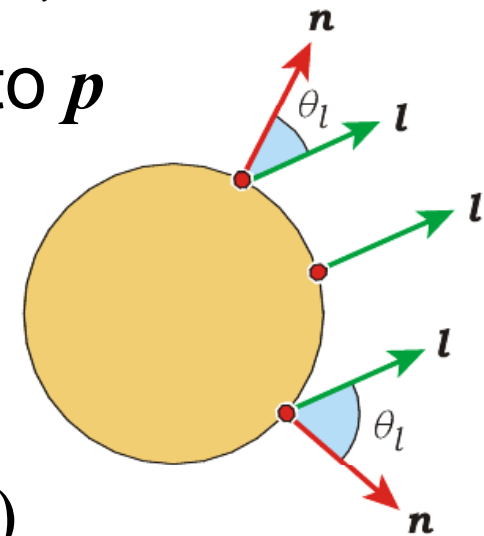
dove $\omega_i = l$ ed è indipendente dal punto p

- La radianza riflessa è

$$L_o(p, \omega_o) = f_r(p, l, \omega_o) * l_s c_l \cos \theta_l$$

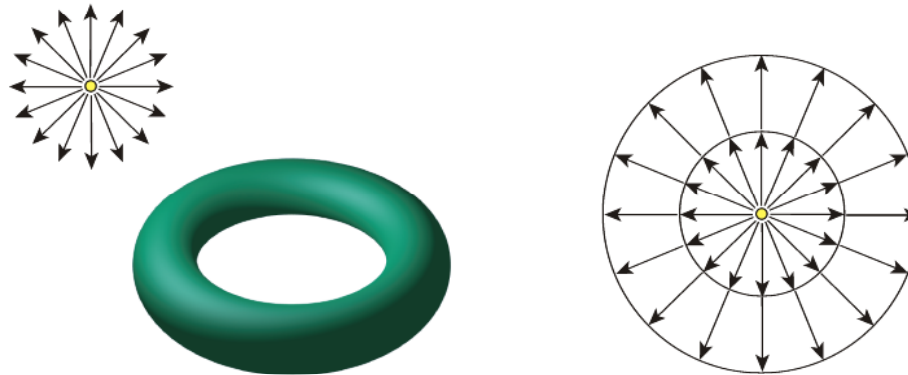
- La radianza incidente è

$$L_i(p, \omega_i) = l_s c_l \delta(\cos \theta_i - \cos \theta_l) \delta(\phi_i - \phi_l)$$



Point light

- Una luce puntiforme genera luce in tutte le direzioni a partire da un punto nello spazio

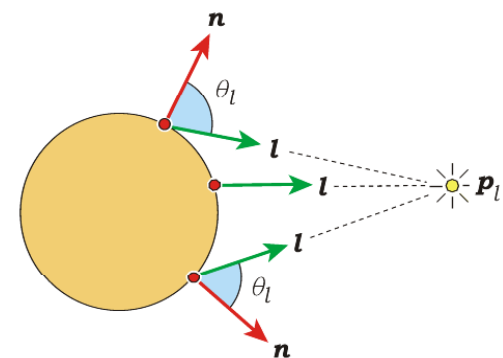


- La radianza riflessa è

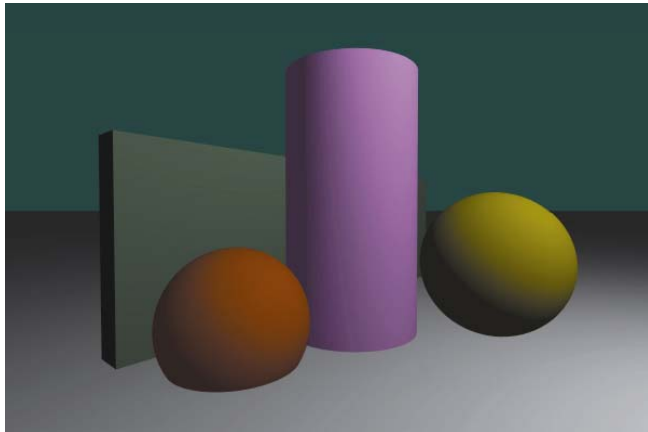
$$L_o(\mathbf{p}, \omega_o) = f_r(\mathbf{p}, \mathbf{l}(\mathbf{p}), \omega_o) * l_s c_l / (r^2) \cos \theta_l$$

- La radianza incidente è

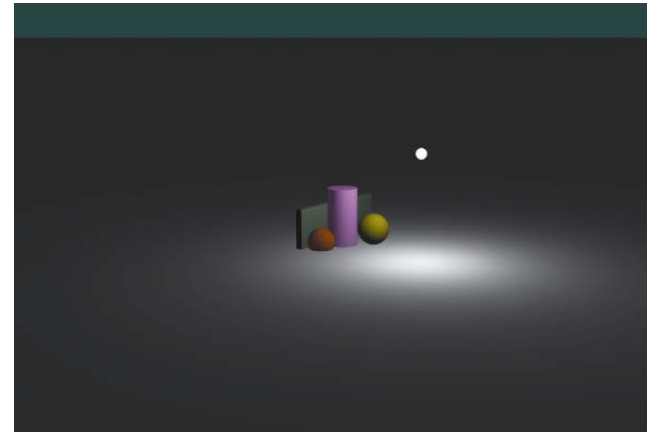
$$L_i(\mathbf{p}, \omega_i) = l_s c_l / (r^2) \delta(\cos \theta_i - \cos \theta_l) \delta(\phi_i - \phi_l)$$



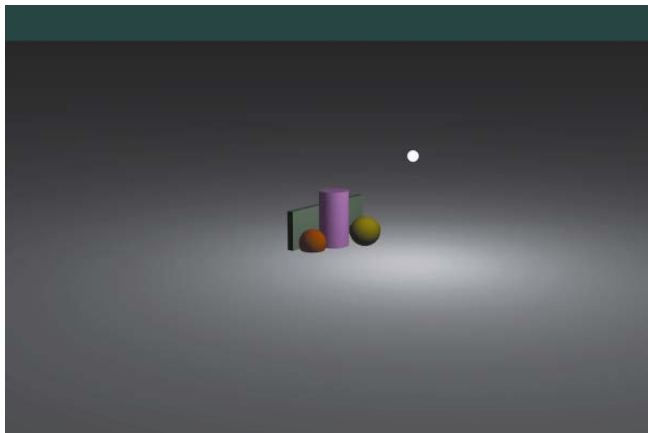
Alcuni esempi



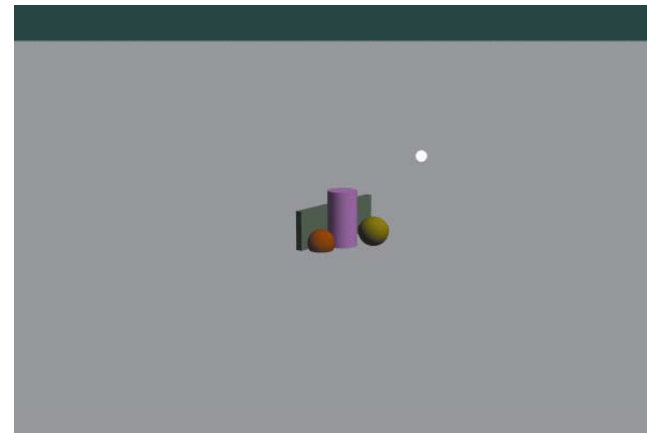
Scena con una sorgente di luce puntiforme



Scena con una sorgente di luce puntiforme ed attenuazione

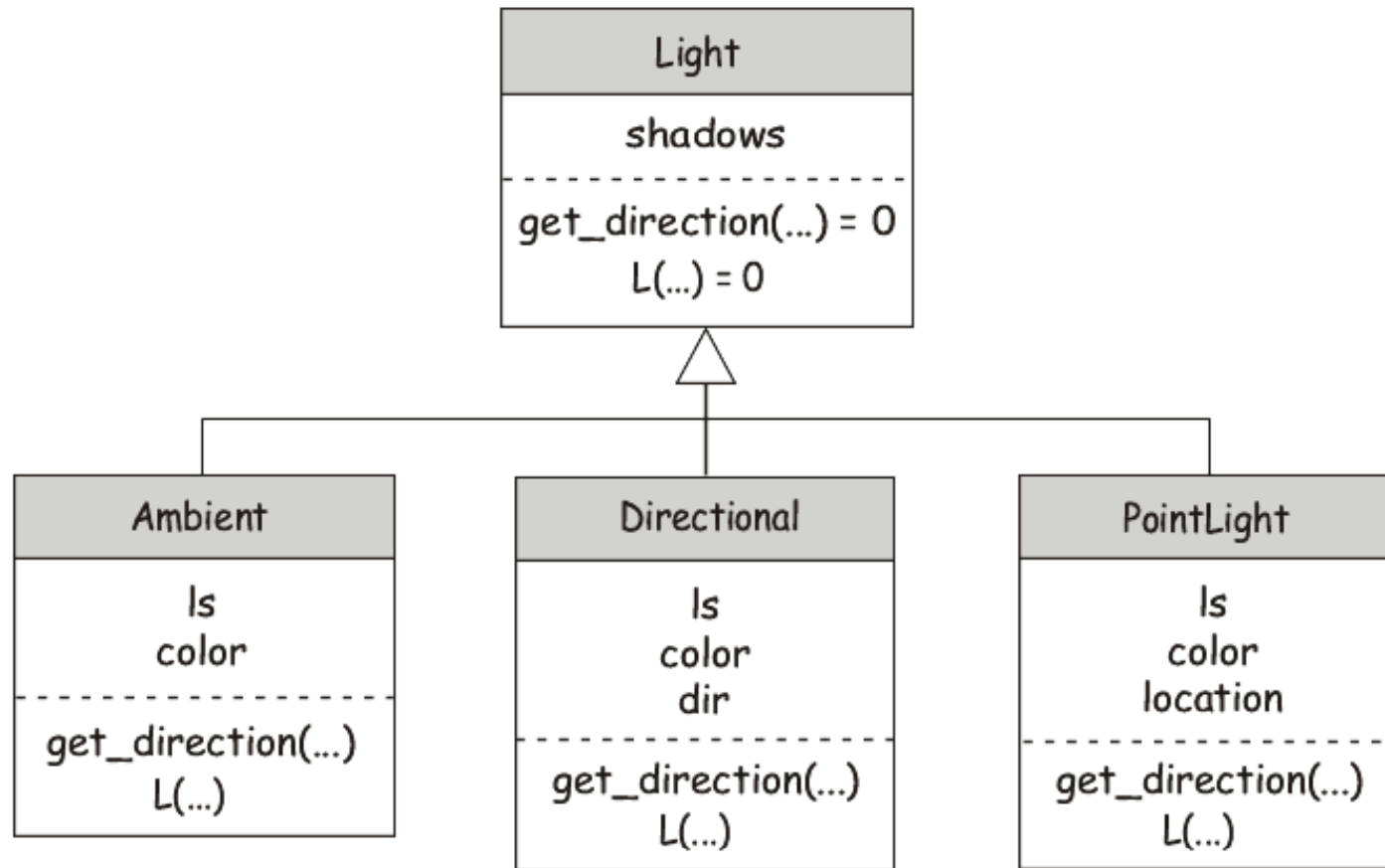


Scena con una sorgente di luce puntiforme senza attenuazione



Scena con una sorgente di luce direzionale

Implementazione della classe Light



La classe Light

```
class Light {
    public:
        ...
        virtual Vector3D get_direction(ShadeRec& sr);

        virtual RGBColor L(ShadeRec& sr);

    protected:

        bool shadows;
};
```

- La funzione `get_direction` restituisce la direzione di un raggio incidente su di un punto
 - La funzione `L` restituisce la radianza incidente su di un punto
-

La sottoclasse Ambient

```
class Ambient: public Light {
    public:
        ...
        virtual Vector3D get_direction(ShadeRec& sr);
        virtual RGBColor L(ShadeRec& sr);

    private:
        float    ls;
        RGBColor color;
};

Ambient::Ambient(void) // default constructor
    : Light(),
      ls(1.0),
      color(1.0)      // white
{}

Vector3D Ambient::get_direction(ShadeRec& sr) {
    return (Vector3D(0.0));
}

RGBColor Ambient::L(ShadeRec& sr) {
    return (ls * color);
}
```

La sottoclasse PointLight

```
class PointLight: public Light {
    public:
        ...
        virtual Vector3D get_direction(ShadeRec& sr);
        virtual RGBColor L(ShadeRec& sr);

    private:
        float    ls;
        RGBColor color;
        Vector3D location;
};

Vector3D PointLight::get_direction(ShadeRec& sr) {
    return ((location - sr.hitPoint).hat());
}

RGBColor PointLight::L(ShadeRec& sr) const{
    return (ls * color);
}
```

La classe World

```
class World {
    public:

        ViewPlane      vp;
        RGBColor       background_color;
        Tracer*        tracer_ptr;
        RGBColor       background_color;
        Light*         ambient_ptr;
        vector<GeometricObject*> objects;
        vector<Light*> lights;

    public:
        // constructors, etc.

        void add_object(Object* object_ptr);

        void add_light(Light* light_ptr);

        void build(void);

        void display_pixel(const int row, const int column, const RGBColor& pixel_color)
const;};
```

La classe ShadeRec

```
class ShadeRec {
    public:

        bool        hit_an_object;        // did the ray hit an object?
        Material*   material_ptr;         // pointer to the nearest object's material
        Point3D     hit_point;           // world coordinates of intersection
        Point3D     local_hit_point;     // world coordinates of hit point on Normal
        Normal      normal;              // normal at hit point
        Ray         ray;                  // for specular highlights and area lights
        int         depth;                // recursion depth
        RGBColor    color;                // used in the Chapter 3 only
        double      t;                    // ray parameter
        float       u;                    // texture coordinate
        float       v;                    // texture coordinate
        World&      w;                    // world reference

        ShadeRec(World& wr);              // constructor
        ShadeRec(const ShadeRec& sr);     // copy constructor
        ~ShadeRec(void);                  // destructor
};
```

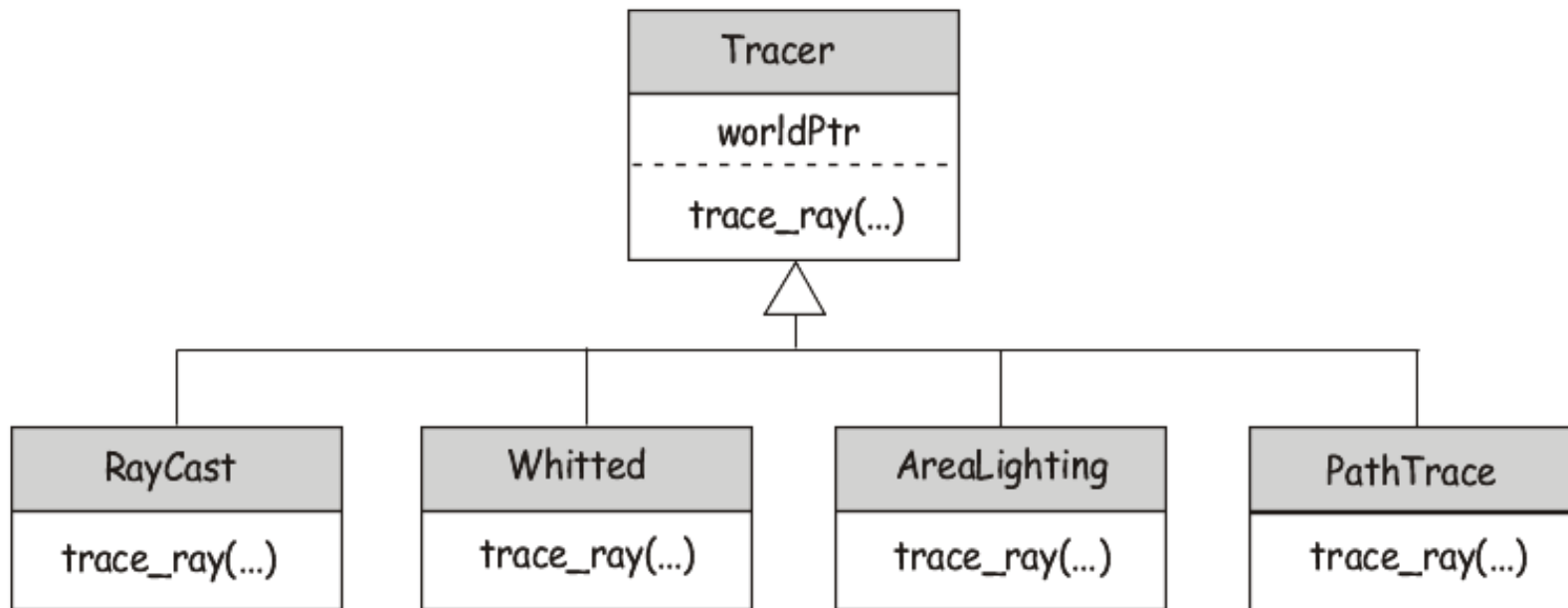
La classe ShadeRec

```
ShadeRec World::hit_objects(const Ray& ray) {
    ShadeRec    sr(*this);
    double      t;
    Normal      normal;
    Point3D     local_hit_point;
    double      tmin = kHugeValue;
    int         num_objects = objects.size();

    for (int j = 0; j < num_objects; j++)
        if (objects[j]->hit(ray, t, sr) && (t < tmin)) {
            sr.hit_an_object = true;
            tmin              = t;
            sr.material_ptr   = objects[j]->get_material();
            sr.hit_point      = ray.o + t * ray.d;
            normal            = sr.normal;
            local_hit_point   = sr.local_hit_point;
        }

    if(sr.hit_an_object) {
        sr.t                = tmin;
        sr.normal           = normal;
        sr.local_hit_point  = local_hit_point;
    }
    return(sr);
}
```

II Tracer



La classe Tracer

```
class Tracer {
public:

    Tracer(void);

    Tracer(World* _world_ptr);

    ~Tracer(void);

    virtual RGBColor trace_ray(const Ray& ray) const;

    virtual RGBColor trace_ray(const Ray ray, const int depth) const;

    virtual RGBColor trace_ray(const Ray ray, float& tmin, const int depth) const;

protected:

    World* world_ptr;
};
```

Il metodo RayCast::trace_ray

```
RGBColor RayCast::trace_ray(const Ray ray, const int depth) const {
    ShadeRec sr(world_ptr->hit_objects(ray));

    if (sr.hit_an_object) {
        sr.ray = ray; // used for specular shading
        return (sr.material_ptr->shade(sr));
    }
    else
        return (world_ptr->background_color);
}
```

Radianza totale riflessa

- La radianza riflessa $L_o(\mathbf{p}, \omega_o)$ di un materiale è dovuta alla componente ambientale e all'illuminazione diretta
 - La componente ambientale è pari a
$$L_o(\mathbf{p}, \omega_o) = \rho_{hh} * L_i(\mathbf{p}, \omega_i) = k_a \mathbf{c}_d * (l_s \mathbf{c}_l)$$
 - La componente ambientale è pari a dove k_a è il coefficiente di riflessione ambientale del materiale
 - Dovrebbe essere lo stesso di k_a poiché la riflessione diffusa è dovuta alla riflessione ambientale ma in questo modo si offre una maggiore flessibilità
-

Radianza totale riflessa

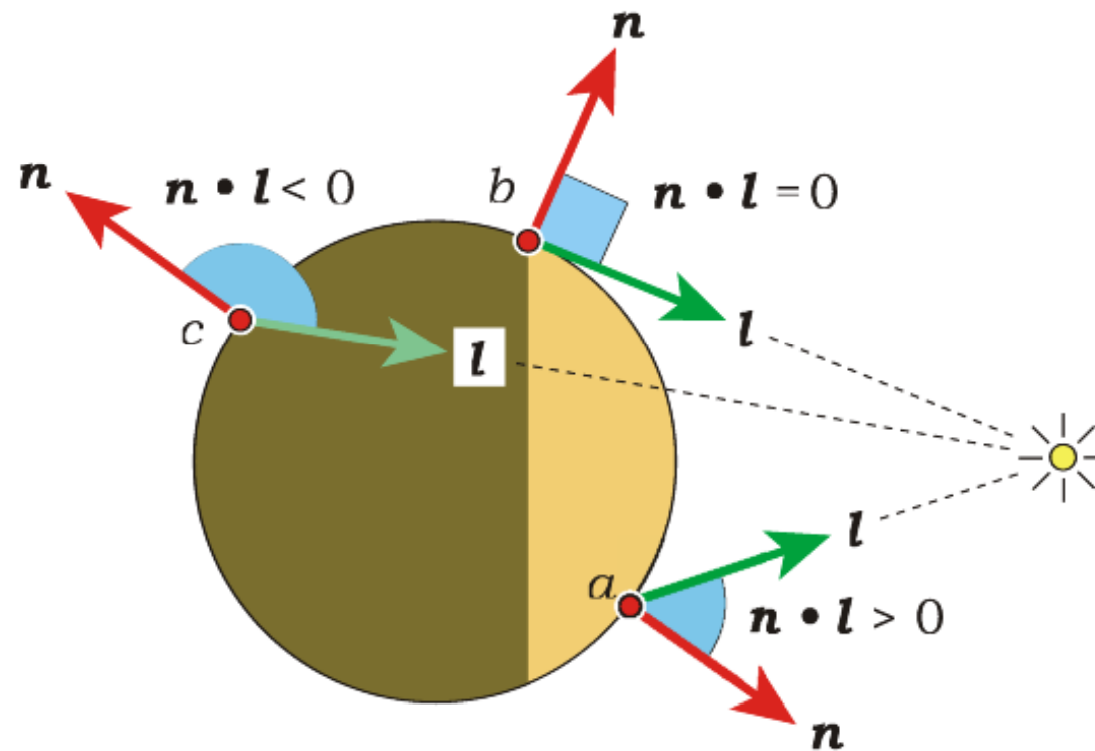
- In definitiva la radianza totale riflessa dal punto p è uguale a

$$L_o(p, \omega_o) = k_a \mathbf{c}_d * (l_s \mathbf{c}_l) + \sum_{j=1}^n (k_d \mathbf{c}_d / \pi) * (l_{s,j} \mathbf{c}_{l,j}) (n \cdot l_j)$$

- $k_d \in [0, 1]$ è un coefficiente di riflessione diffusa
 - \mathbf{c}_d è il colore diffuso
 - $k_a \in [0, 1]$ è il coefficiente di riflessione ambientale del materiale
 - \mathbf{c}_l e fattore di scala radiante $l_s \in [0, \infty)$ della luce
-

Controllo della visibilità

- Per verificare se un punto di un materiale riceve luce bisogna controllare il prodotto $n \cdot l$



Problemi sul controllo della visibilità

