

Introduzione alla Generazione di Immagini Fotorealistiche

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

Dott. Ugo Erra

13° Lezione – Area Light e Mirror Reflection

Sommario

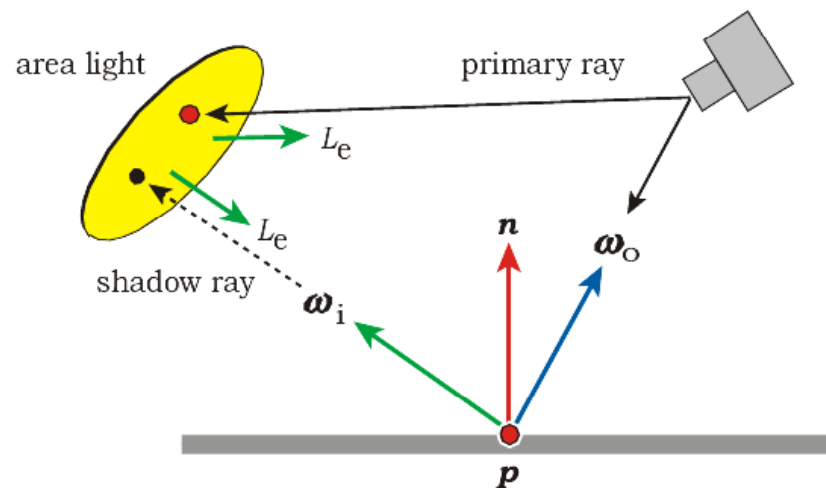
- Definizione di Area Light
 - Architettura
 - Direct Rendering
 - Stima dell'illuminazione diretta
 - Mirror reflection
 - Il modello di illuminazione
 - Ray Tracing
-

Area Lights

- Modellare sorgenti di luce puntiformi è semplice comporta la generazione di ombre con contorni troppo netti
 - In natura le sorgenti di luci hanno un'area finita e permettono la generazione di ombre di tipo soft shadow
 - Utilizzare sorgenti di luce con estensione finita è computazionalmente più costoso e comporta l'utilizzo del metodo Monte Carlo
-

Illuminazione diretta

- Il rendering di scena con sorgenti di luce di tipo *area light* deve prendere in considerazione
 - La sorgente di luce potrebbe essere visibile
 - La sorgente di luce illumina gli oggetti presenti nella scena
- Ogni raggio che colpisce la superficie luminosa restituisce una radianza emessa L_e



Materiale emissivo

- L'implementazione di una area light considera un oggetto geometrico in modo tale da essere gestito e renderizzato come qualunque altro oggetto
- All'oggetto sarà applicato un materiale di tipo emissivo in modo da fornire la radianza emessa $L_e(\mathbf{p}, \omega_o)$ nell'equazione di rendering

$$L_o(\mathbf{p}, \omega_o) = L_e(\mathbf{p}, \omega_o) + L_r(\mathbf{p}, \omega_o)$$

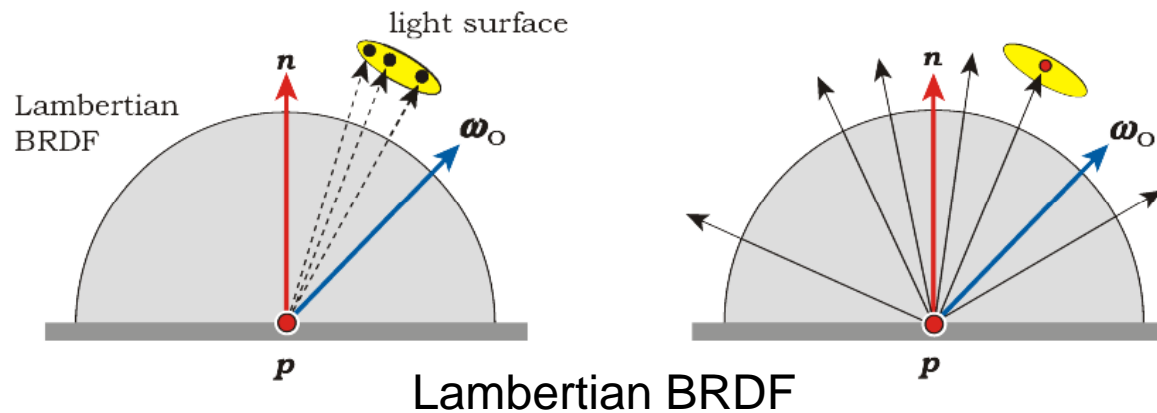
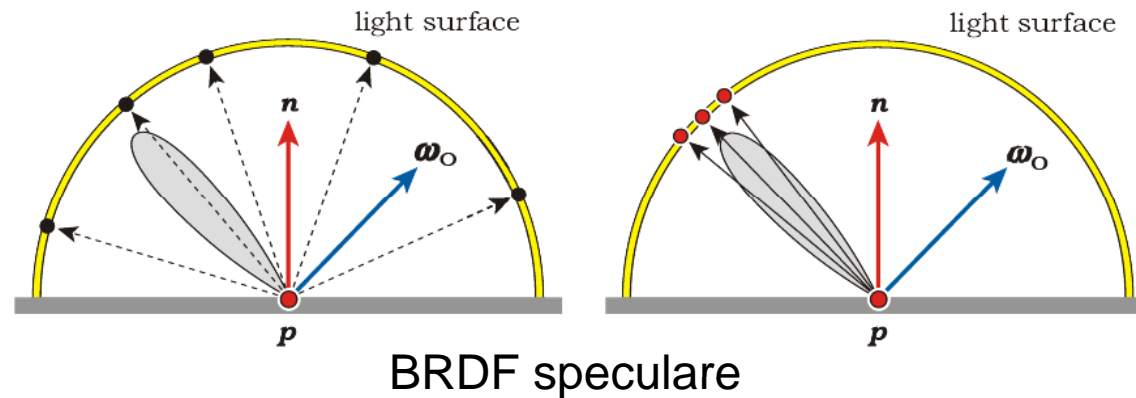
ovvero

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

Come stimiamo l'illuminazione diretta?

- Sono possibili tre soluzioni:
 1. Considerare dei punti campioni generati sulla area light che rappresentino degli shadow ray
 - Per ogni oggetto dobbiamo essere in grado di campionare dei punti sulla sua superficie indipendentemente dalla forma
 2. Considerare gli shadow ray generati da un angolo solido sotteso da p e verificare se intersecano l'area light
 3. Campionare dei raggi dalla BRDF a partire da p
 - I raggi non sono shadow ray ed inoltre potrebbero anche non colpire la sorgente di luce
-

Considerazioni sulle tre tecniche...



Stima dell'illuminazione diretta - 1

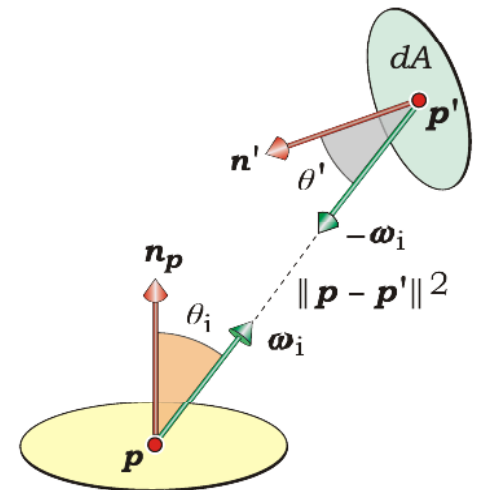
- Per la stima dell'illuminazione diretta da parte di una sorgente di luce finita consideriamo l'equazione di rendering per superfici

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_A f_r(p, \omega_i, \omega_o) L_o(p', -\omega_i) V(p, p') G(p, p') dA$$

dove

$$G(p, p') = \cos\theta_i \cos\theta' / \|\mathbf{p}' - \mathbf{p}\|^2$$

è il geometry term



Stima dell'illuminazione diretta - 2

- La quantità $L_o(\mathbf{p}', -\omega_i)$ nell'equazione di rendering diventa $L_e(\mathbf{p}', -\omega_i)$ da cui otteniamo che la radianza riflessa su di punto p di una superficie è

$$L_r(p, \omega_o) = \int_{A_{lights}} f_r(p, \omega_i, \omega_o) L_e(p', -\omega_i) V(p, p') G(p, p') dA'$$

- Se consideriamo n_l sorgenti di luci presenti nella scena allora abbiamo

$$L_r(p, \omega_o) = \sum_{k=1}^{n_l} \int_{A_{l,k}} f_r(p, \omega_i, \omega_o) L_e(p', -\omega_i) V(p, p') G(p, p') dA'$$

Stima dell'illuminazione diretta - 3

- Per stimare l'integrale usiamo uno stimatore Monte Carlo

$$\langle L_r(p, \omega_o) \rangle = \frac{1}{n_s} \sum_{j=1}^{n_s} \frac{f_r(p, \omega_{i,j}, \omega_o) L_e(p', -\omega_{i,j}) V(p, p'_j) G(p, p'_j) dA'}{p(\mathbf{p}'_j)}$$

- dove n_s sono il numero di campioni \mathbf{p}'_j per $j=1, \dots, n_s$
- Il modo più semplice per stimare l'integrale è scegliere dei punti uniformemente distribuiti sulla superficie della luce
-

Il materiale emissivo

- Dobbiamo definire un materiale emissivo ovvero che genera radianza L_e nella direzione ω_o
 - In generale questo valore dipende da p e ω_o infatti $L_e(p, \omega_o)$ ma assumiamo che:
 - Il materiale è isotropico cioè L_e è indipendente da p e ω_o
 - Il materiale emissivo non riflette ulteriori raggi ad esempio raggi primari o secondari
 - Come per le luci il materiale utilizza un fattore di scala della radianza l_s ed il colore emesso c_e
-

Il metodo AreaLighting::trace_ray

```
RGBColor AreaLighting::trace_ray(const Ray ray, const int depth) const {
    if (depth > world_ptr->vp.max_depth)
        return (black);
    else {
        ShadeRec sr(world_ptr->hit_objects(ray));

        if (sr.hit_an_object) {
            sr.depth = depth;
            sr.ray = ray;

            return (sr.material_ptr->area_light_shade(sr));
        }
        else
            return (world_ptr->background_color);
    }
}
```

La sottoclasse Emissive

```
class Emissive: public Material {
private:
    float    ls;           // radiance scaling factor
    RGBColor ce;          // color

public:
    // constructors, set functions, etc ...

    void scale_radiance(const float _ls);

    void set_ce(const float r, const float g, const float b);

    virtual RGBColor get_Le(ShadeRec& sr) const;

    virtual RGBColor shade(ShadeRec& sr);

    virtual RGBColor area_light_shade(ShadeRec& sr);
};
```

Il metodo Emissive::area_light_shade

- Il metodo AreaLighting::trace_ray deve essere invocato quando il punto più vicino di un raggio primario colpisce una superficie definita come area light

```
RGBColor Emissive::area_light_shade(ShadeRec& sr) {  
    if (-sr.normal * sr.ray.d > 0.0)  
        return (ls * ce);  
    else  
        return (black);  
}
```

Il metodo Matte::area_light_shade

```
RGBColor Matte::area_light_shade(ShadeRec& sr) {
    Vector3D wo = -sr.ray.d;
    RGBColor L = ambient_brdf->rho(sr, wo) * sr.w.ambient_ptr->L(sr);
    int num_lights = sr.w.lights.size();

    for (int j = 0; j < num_lights; j++) {
        Vector3D wi = sr.w.lights[j]->get_direction(sr);
        float ndotwi = sr.normal * wi;

        if (ndotwi > 0.0) {
            bool in_shadow = false;

            if (sr.w.lights[j]->casts_shadows()) {
                Ray shadow_ray(sr.hit_point, wi);
                in_shadow = sr.w.lights[j]->in_shadow(shadow_ray, sr);
            }

            if (!in_shadow)
                L += diffuse_brdf->f(sr, wo, wi) * sr.w.lights[j]->L(sr) *
                    sr.w.lights[j]->G(sr) * ndotwi / sr.w.lights[j]->pdf(sr);
        }
    }
    return (L);
}
```

La classe geometric object

- La classe geometric object deve fornire i seguenti metodi alla classe AreaLight
 - Campionare un punto sulla sue superficie
 - Una densità di probabilità per ogni punto campionato
 - La normale per ogni punto campionato
-

La sottoclasse Rectangle::GeometricObject

```
class Rectangle: public GeometricObject {
public:

    virtual void set_sampler(Sampler* sampler);

    virtual Point3D sample(void);

    virtual Normal get_normal(const Point3D& p);

    virtual float pdf(ShadeRec& sr);

private:

    Point3D p0;           // corner vertex
    Vector3D a;          // side
    Vector3D b;          // side
    double a_len_squared; // square of the length of side a
    double b_len_squared; // square of the length of side b
    Normal normal;
    float area;          // for rectangular lights
    float inv_area;      // for rectangular lights
    Sampler* sampler_ptr; // for rectangular lights
    static const double kEpsilon;
};
```

Campionare un punto

- Per generare un punto sul rettangolo utilizziamo la posizione p_0 e la dimensione dei lati a e b del rettangolo

```
Point3D Rectangle::sample(void) {  
    Point2D sample_point = sampler_ptr->sample_unit_square();  
  
    return (p0 + sample_point.x * a + sample_point.y * b);  
}
```

La sottoclasse AreaLight::Light

```
Vector3D AreaLight::get_direction(ShadeRec& sr) {  
    sample_point = object_ptr->sample();           // used in the G function  
    light_normal = object_ptr->get_normal(sample_point);  
    wi = sample_point - sr.hit_point;             // used in the G function  
    wi.normalize();  
    return (wi);  
}
```

```
RGBColor AreaLight::L(ShadeRec& sr) {  
    float ndotd = -light_normal * wi;  
    if (ndotd > 0.0) return (material_ptr->get_Le(sr));  
    else return (black);  
}
```

```
bool AreaLight::in_shadow(const Ray& ray, const ShadeRec& sr) const {  
    float t;  
    int num_objects = sr.w.objects.size();  
    float ts = (sample_point - ray.o) * ray.d;  
  
    for (int j = 0; j < num_objects; j++)  
        if (sr.w.objects[j]->shadow_hit(ray, t) && t < ts)  
            return (true);  
  
    return (false);  
}
```

Esempio

```
Sampler* sampler_ptr = new MultiJittered(num_samples);
background_color = RGBColour(0.5);
tracer_ptr = new AreaLighting(this);

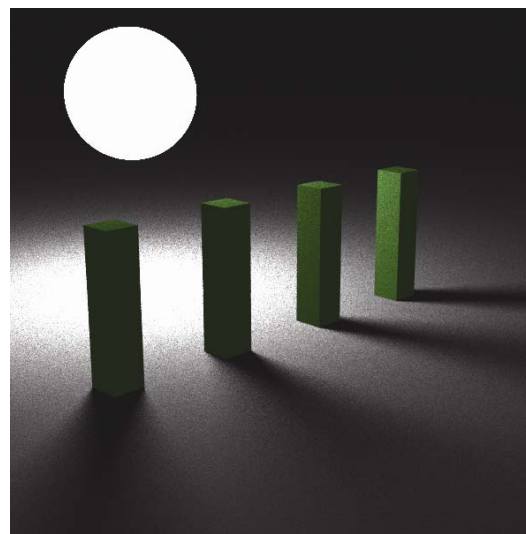
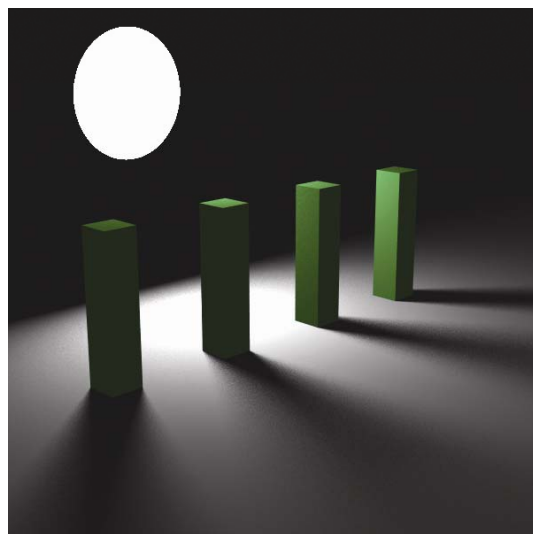
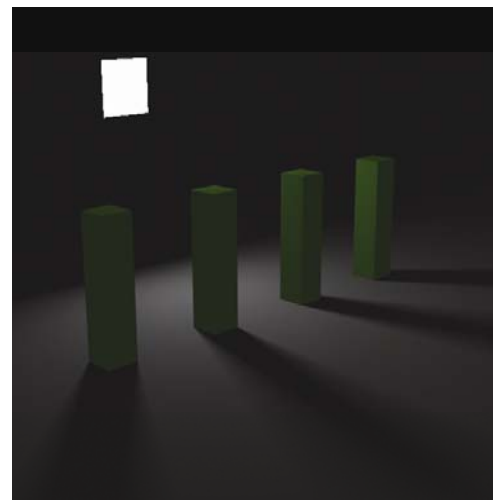
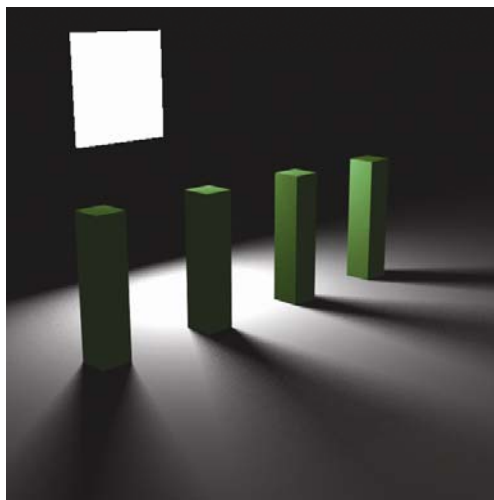
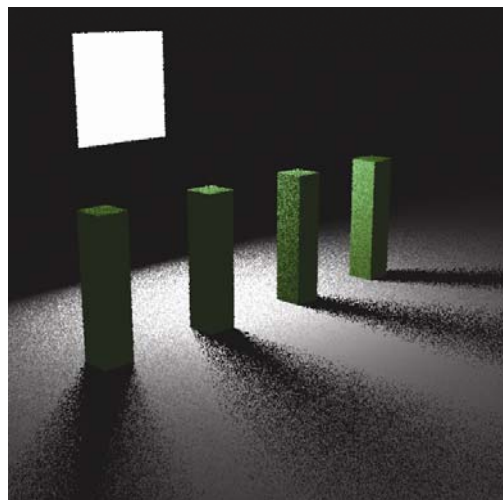
Pinhole* camera = new Pinhole;
camera->set_eye(-20, 10, 20);
camera->set_lookat(0, 2, 0);
camera->set_view_distance(1080);
camera->compute_uvw();
set_camera(camera);

Emissive* emissive_ptr = new Emissive;
emissive_ptr->scale_radiance(40.0);
emissive_ptr->set_ce(white);

Rectangle* rectangle_ptr = new Rectangle(p0, a, b, normal);
rectangle_ptr->set_material(emissive_ptr);
rectangle_ptr->set_sampler(sampler_ptr);
rectangle_ptr->set_shadows(false);
add_object(rectangle_ptr);

AreaLight* area_light_ptr = new AreaLight;
area_light_ptr->set_object(rectangle_ptr);
area_light_ptr->set_shadows(true);
add_light(area_light_ptr);
```

Esempio



Environment Lights

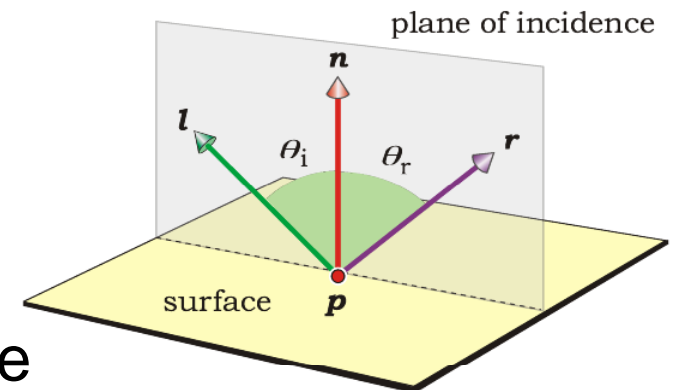
- L'environment lights permette di aggiungere una sfera che conterrà l'intera scena e che fornisce illuminazione in tutte le direzioni
 - La sfera è composta da materiale emissivo che può essere costante oppure variabile
 - Il colore può essere basato ad esempio su un tramonto
-

Mirror reflection

- Finora abbiamo considerato solo i raggi primari ed in che modo interagiscono con diversi tipi di materiali
 - Le riflessioni rappresentano un primo approccio alla simulazione dell'illuminazione indiretta
 - La tecnica è stata sviluppata da Whitted nel 1980 nell'algoritmo del ray tracing
 - In questo contesto simuliamo l'illuminazione indiretta attraverso superfici perfettamente speculari
-

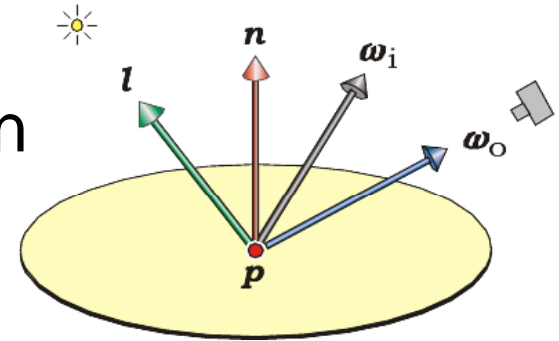
Raggi riflessi

- Se un raggio colpisce un oggetto costituito da materiale riflessivo determiniamo un **raggio riflesso** (*reflective ray*)
- Il raggio riflesso rappresenta un raggio secondario che può generare ulteriori altri raggi riflessi
- “Inseguire” tutti i raggi generati permette di modellare le riflessioni dovuti a materiali riflettenti presenti nella scena
 - Un oggetto non visibile alla camera potrebbe essere visibili indirettamente per riflessione



Il modello di illuminazione - 1

- L'illuminazione indiretta considera la radianza incidente nella direzione ω_i non solo proveniente dalle sorgenti di luci ma anche da eventuali altri sorgenti
- La radianza uscente nella direzione ω_o sarà quindi la somma di una quantità diretta e indiretta



$$L_r(\mathbf{p}, \omega_o) = L_{\text{direct}}(\mathbf{p}, \omega_o) + L_{\text{indirect}}(\mathbf{p}, \omega_o)$$

dove $L_{\text{direct}}(\mathbf{p}, \omega_o)$ è data da (area light)

$$L_{\text{direct}}(\mathbf{p}, \omega_o) = \int_{A_{\text{lights}}} f_r(\mathbf{p}, \omega_i, \omega_o) L_e(\mathbf{p}', -\omega_i) V(\mathbf{p}, \mathbf{p}') G(\mathbf{p}, \mathbf{p}') dA'$$

Il modello di illuminazione - 2

- La radianza indiretta invece è data da

$$L_{\text{indirect}}(p, \omega_o) = \int_{2\pi^+} f_r(p, \omega_i, \omega_o) L_i(r_c(p, \omega_i), -\omega_i) \cos \theta_i d\omega_i$$

dove $L_i(r_c(p, \omega_i), -\omega_i)$ è la radianza incidente ottenuta sparando un raggio riflesso sulla semisfera centrata su p

- L'equazione L_{indirect} non è semplice da risolvere ma in questo contesto ci limitiamo solo a considerare superfici perfettamente speculari (specchi)
-

Il modello di illuminazione - 3

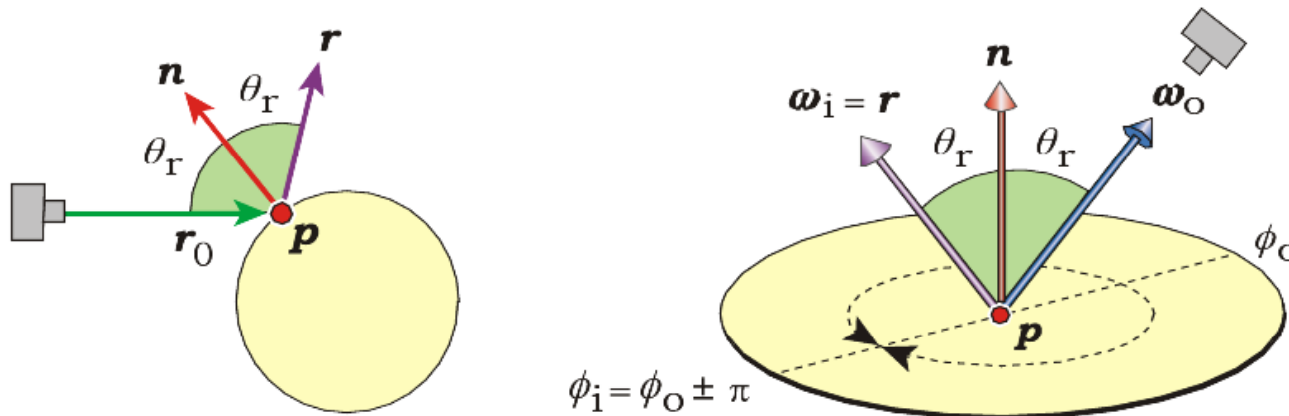
- Se r_0 è il raggio primario e p è il punto di intersezione possiamo ottenere il raggio riflesso r a partire dalla normale n nel seguente modo

$$r = \omega_0 + 2(n \cdot \omega_0) n$$

- Poiché la radianza incidente da ω_i proviene da una sola direzione possiamo semplificare l'espressione della radianza indiretta come

$$L_{\text{indirect}}(p, \omega_o) = f_{r,s}(p, \omega_o, \omega_i) L_i(p, \omega_i)$$

dove $f_{r,s}$ è la BRDF di un materiale perfettamente speculare



BRDF perfettamente speculare

- La BRDF $f_{r,s}$ può essere modellata in diversi modi
 - Ad esempio utilizzando le equazioni di Fresnel
- In questo contesto supponiamo che $f_{r,s}$ sia costante ovvero

$$f_{r,s} = \rho_s = k_r \mathbf{c}_r$$

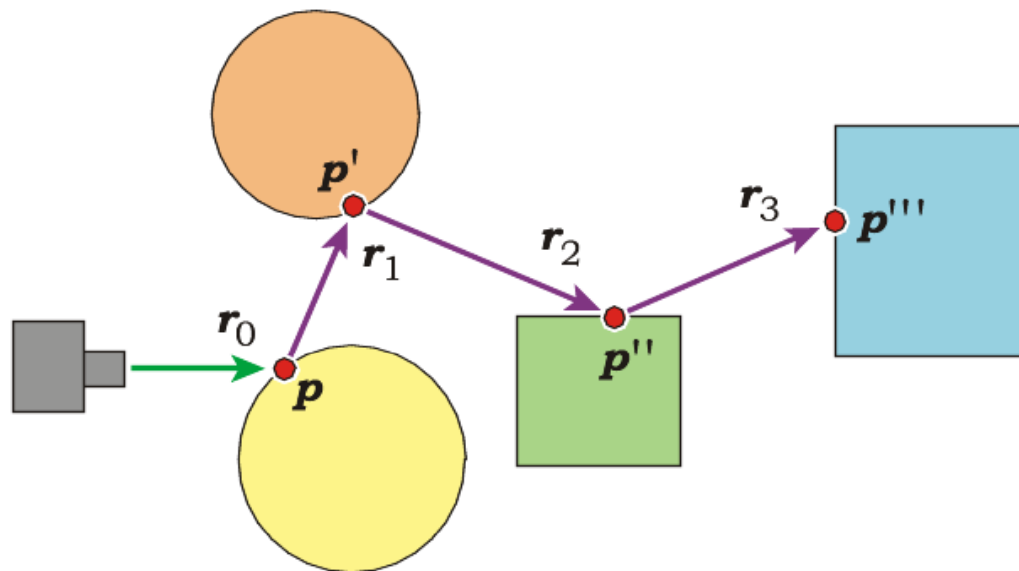
- dove $k_r \in [0,1]$ è un *coefficiente di riflessione* e \mathbf{c}_r è il *colore riflesso*
- Quindi l'equazione per la radianza indiretta diventa

$$L_{\text{indirect}}(\mathbf{p}, \omega_o) = k_r \mathbf{c}_r L_i(\mathbf{p}, \omega_i)$$

- In altre parole la BRDF deve essere zero per tutti valori $\theta_i \neq \theta_r$ e $\phi_i \neq \phi_o \pm \pi$
-

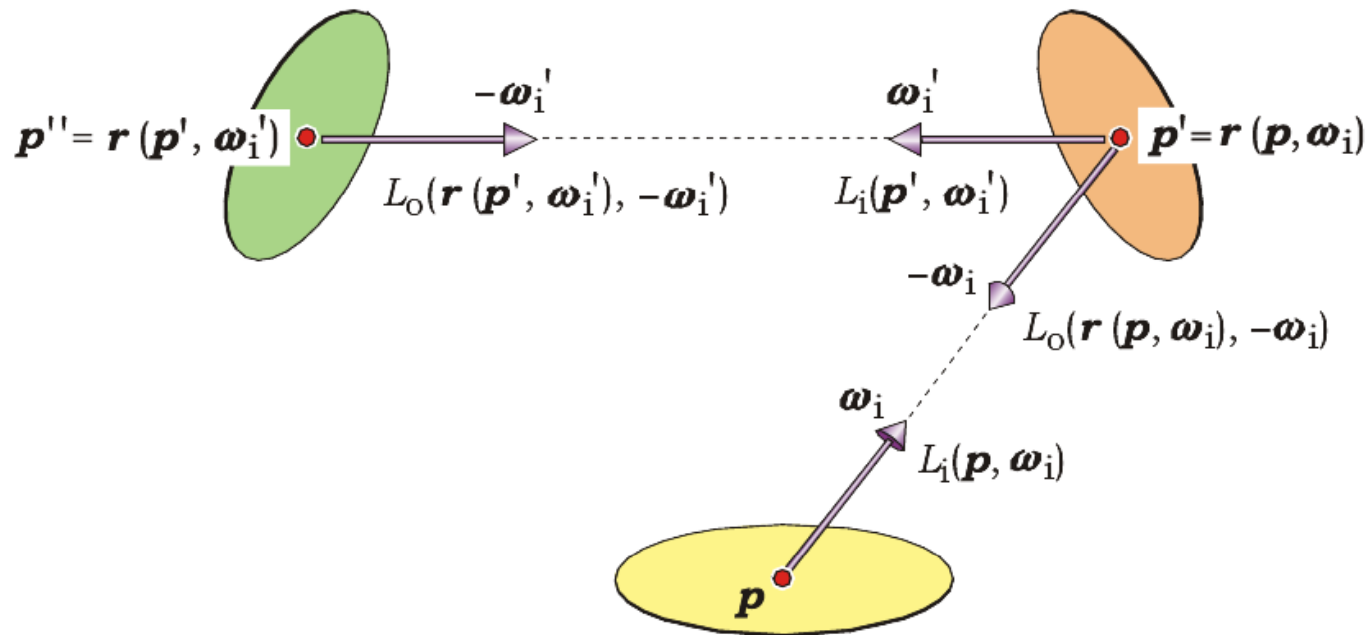
Riflessioni continue...

- Un raggio riflesso potrebbe essere ulteriormente riflesso se lungo il suo cammino incontra un ulteriore oggetto speculare ovvero da r_0 a r_1 , r_2 , r_3 , etc...



...e ricorsive

- Per ogni raggio riflesso applichiamo ricorsivamente l'operatore di ray casting $r_c(\mathbf{p}, \boldsymbol{\omega}_i) = r(\mathbf{p}, \boldsymbol{\omega}_i)$



Strategia generale

- A partire da raggio riflesso r_1 generato da r_0 nel punto p si possono verificare le seguenti condizioni:
 - Se non colpisce nessun oggetto allora la radianza ottenuta è uguale colore di background ed restituita al punto p
 - Se colpisce una sorgente di luce allora otteniamo la radianza L_e
 - Se colpisce un oggetto non riflessivo nel punto p' allora determiniamo l'illuminazione diretta nel punto p' ed è restituita al punto p
 - Se colpisce un oggetto riflessivo nel punto p' allora determiniamo l'illuminazione diretta in p' e spariamo un raggi riflesso p'' . Tutte le radianze calcolate sono restituite al punto di origine p
-

Implementazione

- Per implementare il ray tracing abbiamo bisogno di
 - Una variabile `max_depth` che definisce il numero massimo di riflessioni consentite
 - Aggiungere una variabile `depth` nella classe `ShadeRec` per tenere traccia del numero di “rimbalzi”
 - Una nuova classe tracer chiamata `Whitted` che implementi il metodo `trace_ray`
 - Una BRDF perfettamente speculare
 - Un materiale riflessivo
-

Il metodo Whitted::trace_ray

```
RGBColor Whitted::trace_ray(const Ray ray, const int depth) const {
    if (depth > world_ptr->vp.max_depth)
        return(black);
    else {
        ShadeRec sr(world_ptr->hit_objects(ray));

        if (sr.hit_an_object) {
            sr.depth = depth;
            sr.ray = ray;
            return (sr.material_ptr->shade(sr));
        }
        else
            return (world_ptr->background_color);
    }
}
```

- All'interno del metodo `shade` se il materiale è riflessivo ricorsivamente sarà invocata nuovamente il metodo `trace_ray`
-

La classe PerfectSpecular

```
class PerfectSpecular: public BRDF
{
public:
    void set_kr(const float k);

    void set_cr(const RGBColor& c);

    void set_cr(const float r, const float g, const float b);

    void set_cr(const float c);

    virtual RGBColor f(const ShadeRec& sr, const Vector3D& wo, const Vector3D&
wi) const;

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

    virtual RGBColor rho(const ShadeRec& sr, const Vector3D& wo) const;

private:
    float kr;           // reflection coefficient
    RGBColor cr;       // the reflection colour
};
```

Il metodo PerfectSpecular::sample_f

```
RGBColor PerfectSpecular::sample_f(const ShadeRec& sr, const Vector3D& wo,
Vector3D& wi) const {
    float ndotwo = sr.normal * wo;
    wi = -wo + 2.0 * sr.normal * ndotwo;

    return (kr * cr / (sr.normal * wi));
}
```

Un materiale riflessivo

- Il materiale riflessivo può derivato dal classe Phong in modo tale da utilizzare il metodo shade per l'illuminazione diretta
- Inoltre il materiale phong include l'illuminazione ambientale, diffusa e speculare

```
class Reflective: public Phong {
public:
    void set_ks(const float k);

    void set_cr(const RGBColor& c);

    void set_cr(const float r, const float g, const float b);

    virtual RGBColor shade(ShadeRec& s);

private:
    PerfectSpecular* reflective_brdf;
};
```

Un materiale riflessivo

- Il materiale riflessivo può derivato dal classe Phong in modo tale da utilizzare il metodo shade per l'illuminazione diretta
- Inoltre il materiale phong include l'illuminazione ambientale, diffusa e speculare

```
class Reflective: public Phong {
public:
    void set_kr(const float k);

    void set_cr(const RGBColor& c);

    void set_cr(const float r, const float g, const float b);

    virtual RGBColor shade(ShadeRec& s);

private:
    PerfectSpecular* reflective_brdf;
};
```

Il metodo Reflective::shade

```
RGBColor Reflective::shade(ShadeRec& sr) {
    RGBColor L(Phong::shade(sr)); // direct illumination

    Vector3D wo = -sr.ray.d;
    Vector3D wi;
    RGBColor fr = reflective_brdf->sample_f(sr, wo, wi);
    Ray reflected_ray(sr.hit_point, wi);
    reflected_ray.depth = sr.depth + 1;

    L += fr * sr.w.tracer_ptr->trace_ray(reflected_ray, sr.depth + 1) *
(sr.normal * wi);

    return (L);
}
```

Esempio

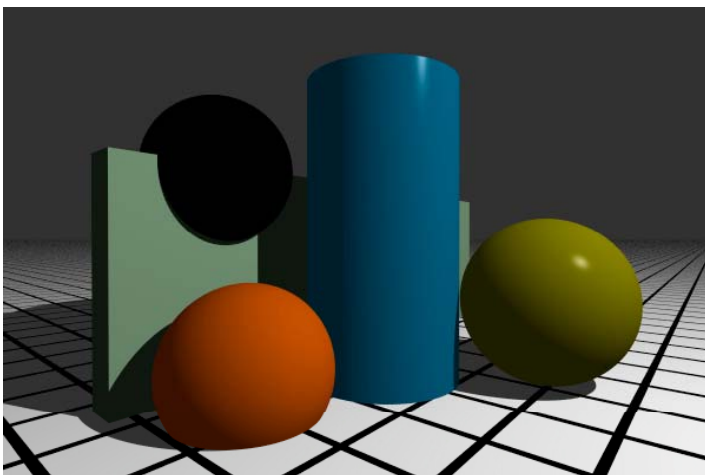
```
vp.set_hres(600);
vp.set_vres(400);
vp.set_samples(num_samples);
vp.set_max_depth(10);

tracer_ptr = new Whitted(this);

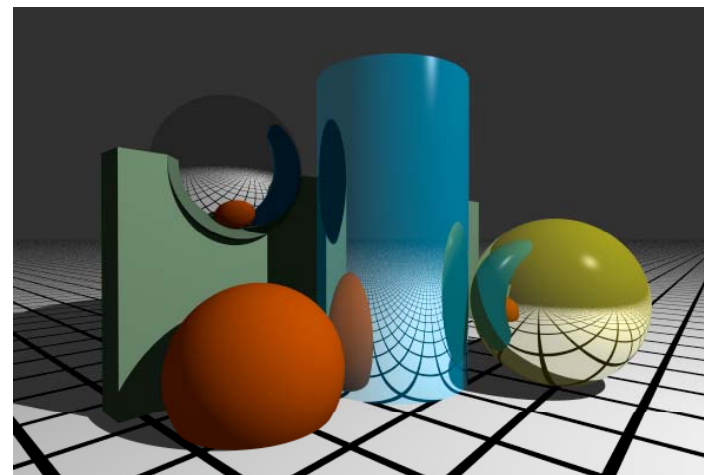
tracer_ptr = new Whitted(this);
background_color = RGBColor(0.15);

Reflective* reflective_ptr1 = new Reflective;
reflective_ptr1->set_ka(0.25);
reflective_ptr1->set_kd(0.5);
reflective_ptr1->set_cd(0.75, 0.75, 0);           // yellow
reflective_ptr1->set_ks(0.15);
reflective_ptr1->set_exp(100.0);
reflective_ptr1->set_kr(0.75);
reflective_ptr1->set_cr(white);
```

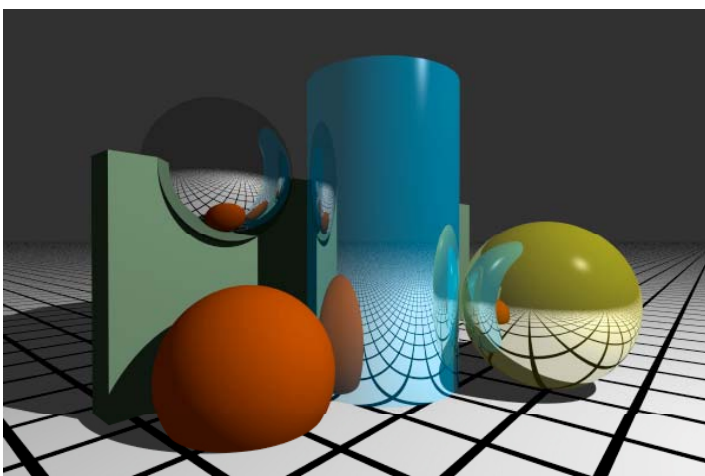
Esempi



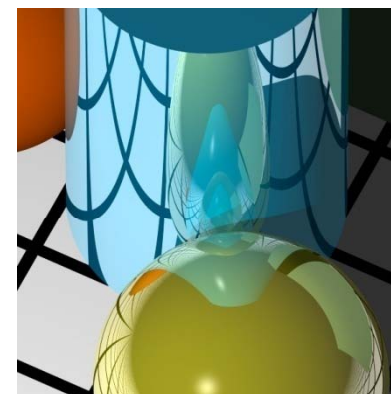
max_depth = 0



max_depth = 1



max_depth = 10



max_depth = 10