

# Introduzione alla Generazione di Immagini Fotorealistiche

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

Dott. Ugo Erra

*14° Lezione – Global Illumination*

# Sommario

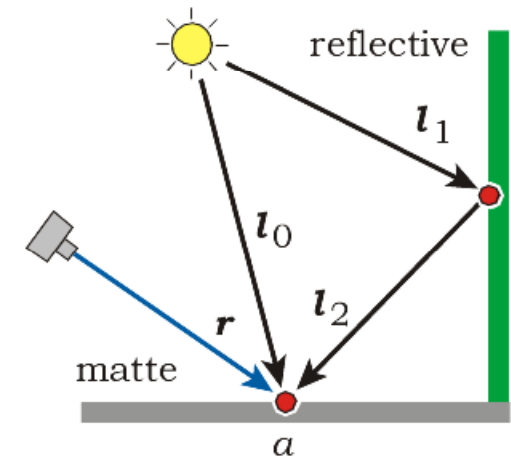
- Path tracing
  - Combinare illuminazione diretta e indiretta
  - Limiti del path tracing
-

# Light Transport

- L'illuminazione può essere diretta oppure indiretta
    - Nel caso diretto abbiamo considerato diversi tipi di luci
    - Nel caso indiretto abbiamo considerato superfici perfettamente speculari oppure glossy
  - Il caso generale è rappresentato da BRDF completamente arbitrarie
  - Ci poniamo il problema di calcolare la distribuzione della luce all'interno di un ambiente composto da materiali di qualunque tipo
-

# I percorsi della luce

- Il punto  $a$  riceve radianza incidente in due modi:
  - Direttamente dalla sorgente di luce attraverso il raggio  $l_0$
  - Indirettamente attraverso i raggi  $l_1$  ed  $l_2$
- Supponendo che il raggio primario  $r$  colpisca la superficie nel punto  $a$ . Come facciamo a capire da quale direzione proviene la luce?
- Gli algoritmi di illuminazione globale cercano di simulare tutti i percorsi che luce compie lungo un percorso



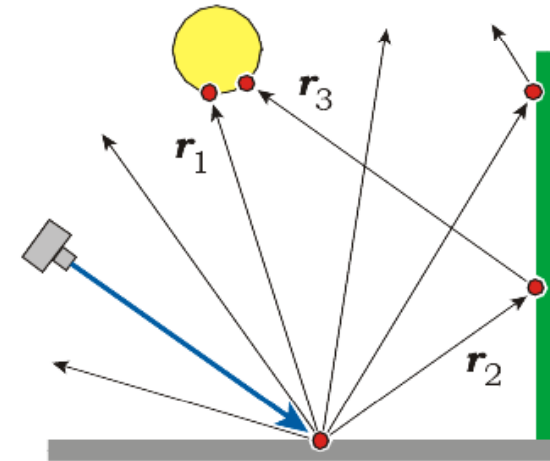


# Osservazioni sul Path Tracing

- Per ogni raggio che colpisce una superficie il metodo `sample_f` genera un nuovo raggio in maniera casuale
    - Non ci sono restrizioni sui tipi di BRDF
  - Il path tracing permette di simulare ogni tipo di superficie assumendo che tutti i materiali riflettono i raggi di luce quando sono colpiti
  - Poiché un raggio termina il suo percorso solo se eventualmente colpisce una sorgente di luce non è possibile utilizzare luci puntiformi
    - Inoltre qualunque oggetto può emettere luce non è necessario generare dei punti sull'oggetto
-

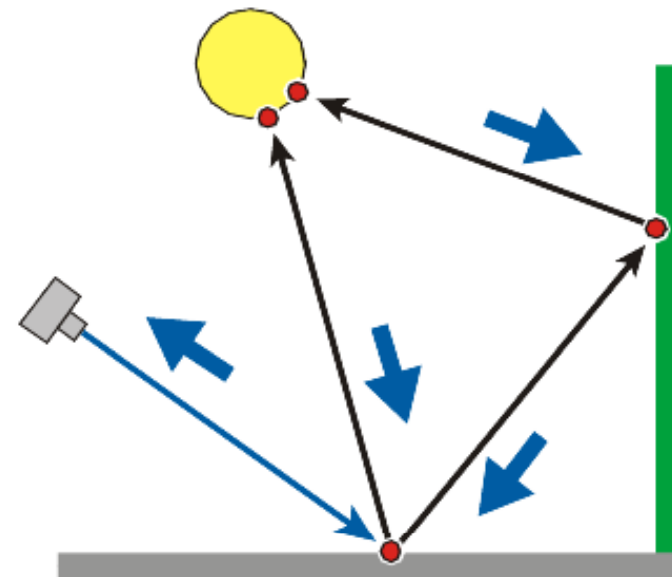
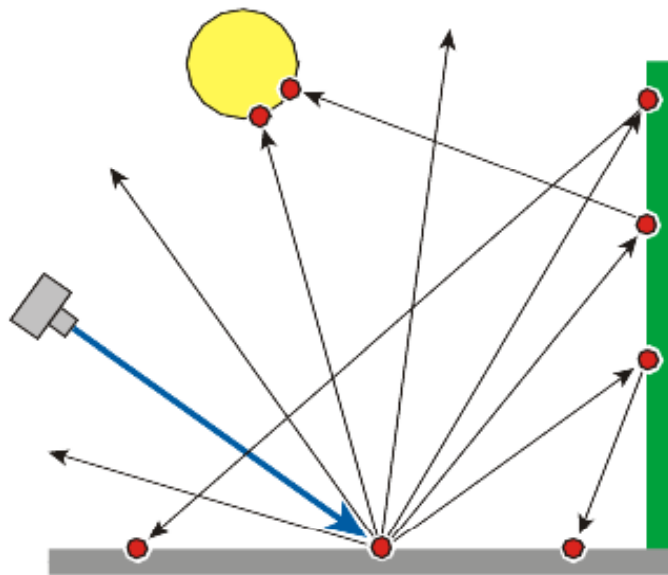
# Esempio

- Consideriamo una scena in cui
  - La superficie grigia è di tipo Lambert
  - La superficie verde è un specchio
- Per ogni raggio che colpisce una superficie abbiamo che:
  - Nel caso della superficie di Lambert la direzione del raggio riflesso è scelta attorno alla normale in maniera casuale
  - Nel caso dello specchio il raggio riflesso è scelto il solo raggio riflesso possibile
- La possibilità che alcuni raggi colpiscano una sorgente di luce è proporzionale al numero di campioni che generiamo
  - I raggi  $r_1$ ,  $r_2$  ed  $r_3$  sono “utili”



# Esempio

- Tutte le superfici sono di tipo Lambert
  - Il trasferimento di radianza tra superfici di tipo lambert produce un effetto noto come *color blending*
- Tutti i raggi secondari sono scelti in maniera random
- Solo due percorsi o path colpiscono la sorgente di luce e quindi restituiranno radianza



# Calcolo dell'equazione di rendering

- Per ogni pixel stimiamo la quantità di radianza ricevuta attraverso il metodo Monte Carlo

$$\langle L_r(p, \omega_o) \rangle = \frac{1}{n_s} \sum_{j=1}^{n_s} \frac{f_r(p, \omega_{i,j}, \omega_o) L_i(p', \omega_{i,j}) \cos \theta_{i,j}}{p(\omega_{i,j})}$$

- L'espressione è valutata in maniera ricorsiva considerando l'operatore di casting
  - Il path tracing fornisce una soluzione esatta dell'equazione di rendering se il numero di raggi e la `max_depth` sono scelti opportunamente
  - Poiché la dimensione della sorgente di luce è piccola rispetto alla scena è probabile la presenza di noise nell'immagine finale
-

# Pseudocode Path Tracing

```
Color TracePath(Ray r,depth)
{
    if(depth==MaxDepth)
        return Black;

    if(r.hitSomething==false)
        return Black;

    emittance=m.emittance;

    newRay.origin=r.pointWhereObjWasHit;
    newRay.direction=RandomUnitVectorInHemisphereOf(r.normalWhereObjWasHit);
    float cost=DotProduct(newRay.direction,r.normalWhereObjWasHit); Color

    BRDF=m.reflectance/PI; float scale=1.0*PI; Color
    reflected=TracePath(newRay,depth+1);

    return emittance + ( BRDF * scale * cost * reflected );
}
```

# Implementazione

- Per implementare il path tracing abbiamo bisogno di:
    - Un nuovo metodo `PathTrace`
    - Il metodo `sample_f` deve restituire un raggio riflesso
    - Un metodo `path_shade` per tutti i materiali
-

# Il metodo PathTrace::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->path_shade(sr));
        }
        else
            return (world_ptr->background_color);
    }
}
```

---

# Il metodo Matte::path\_shade

```
RGBColor Matte::path_shade(ShadeRec& sr) {
    Vector3D      wo = -sr.ray.d;
    Vector3D      wi;
    float         pdf;
    RGBColor      f = diffuse_brdf->sample_f(sr, wo, wi, pdf);
    float         ndotwi = sr.normal * wi;
    Ray           reflected_ray(sr.hit_point, wi);

    return (f * sr.w.tracer_ptr->trace_ray(reflected_ray, sr.depth + 1) * ndotwi
/ pdf);
}
```

# Il metodo Lambertian::sample\_f

```
RGBColor Lambertian::sample_f(const ShadeRec& sr, const Vector3D& wo, Vector3D&
wi, float& pdf) const {
    Vector3D w = sr.normal;
    Vector3D v = Vector3D(0.0034, 1, 0.0071) ^ w;
    v.normalize();
    Vector3D u = v ^ w;

    Point3D sp = sampler_ptr->sample_hemisphere();
    wi = sp.x * u + sp.y * v + sp.z * w;
    wi.normalize();

    pdf = sr.normal * wi * invPI;

    return (kd * cd * invPI);
}
```

- La funzione genera un raggio campionando un punto sulla semisfera e ritorna la componente diffusa da utilizzare in path\_shade
-

# Il metodo Reflective::path\_shade

```
RGBColor Reflective::path_shade(ShadeRec& sr) {
    Vector3D      wo = -sr.ray.d;
    Vector3D      wi;
    float         pdf;
    RGBColor      fr = reflective_brdf->sample_f(sr, wo, wi, pdf);
    float         ndotwi = sr.normal * wi;
    Ray           reflected_ray(sr.hit_point, wi);

    return (fr * sr.w.tracer_ptr->trace_ray(reflected_ray, sr.depth + 1) * ndotwi
/ pdf);
}
```

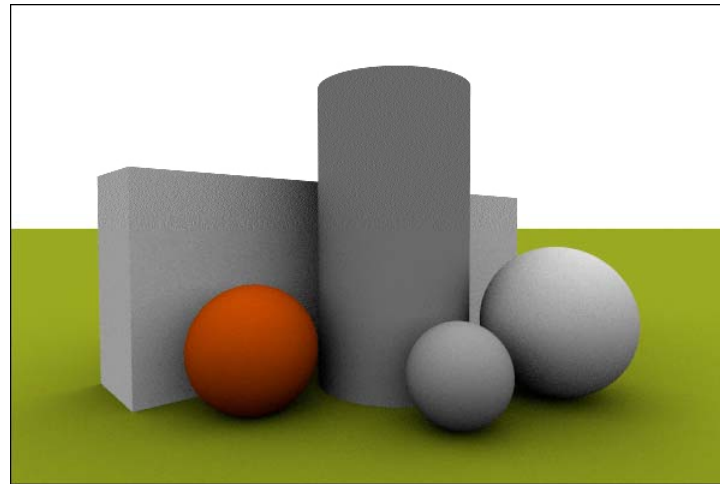
# Il metodo Lambertian::sample\_f

```
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;

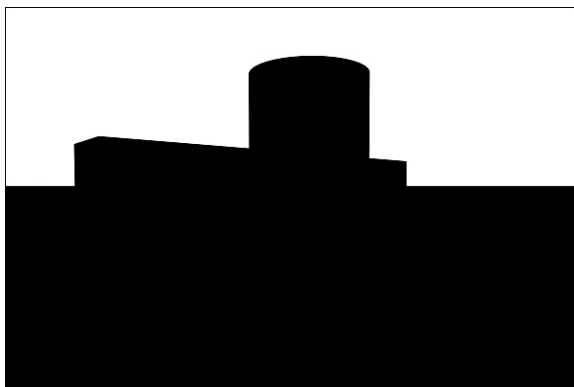
    return (kr * cr / fabs(sr.normal * wi));
}
```

- La funzione genera un raggio riflesso e ritorna la componente riflessa da utilizzare in path\_shade
-

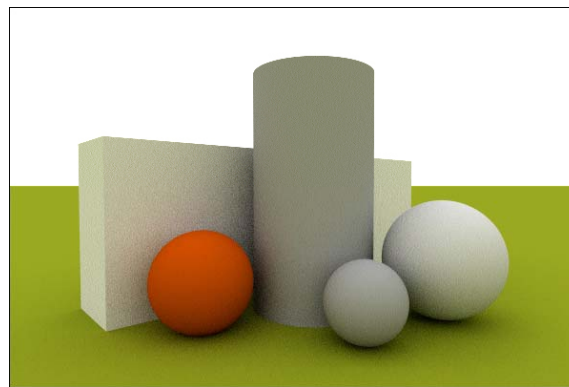
# Esempio con luce ambientale



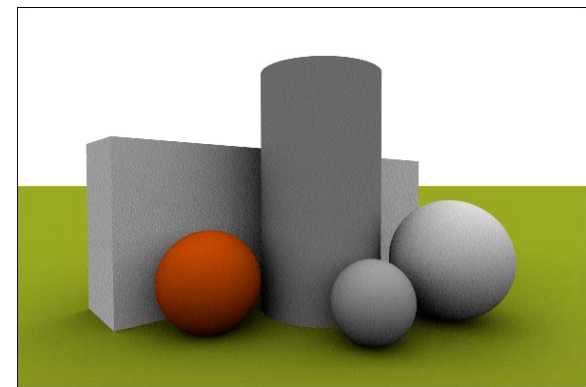
Solo illuminazione diretta



`max_depth = 0`



`max_depth = 1`



`max_depth = 5`

# Cornell Box

- La cornell box è una scena di test diventata standard ed utilizzata per la prima volta Cornell University
- La scena è composta da materiali di tipo Lambert e da una sorgente di luce al centro del soffitto
- Successivamente furono inseriti dei parallelepipedi



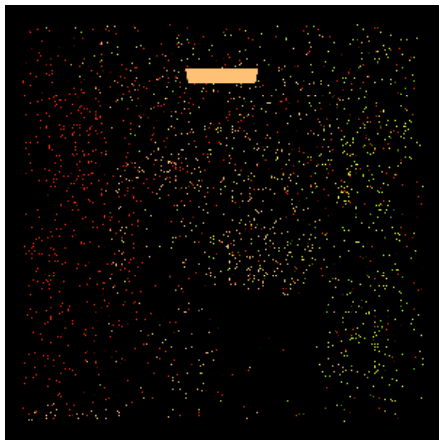
foto



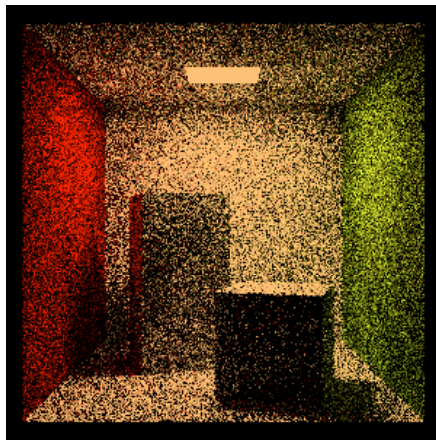
rendering



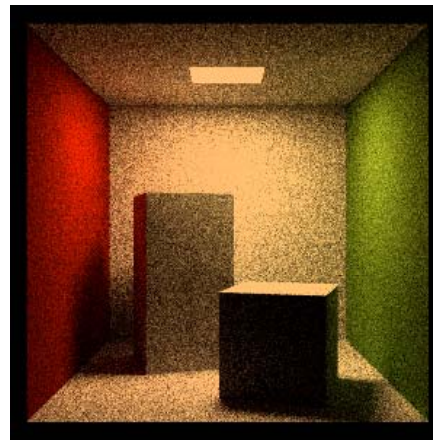
# Path Tracing con la Cornell Box



samples = 1



samples = 100



samples = 1024



samples = 10000

max\_depth = 10

# Altri algoritmi di global illumination

- Radiosity
  - Beam tracing
  - Metropolis light transport
  - Photon Mapping
-

# Radiosity

- Il radiosity è un algoritmo per descrivere accuratamente la riflessione diffusa di una superficie
    - Calcola l'energia radiante (*radiosity*) tra superfici all'interno di una scena
  - L'algoritmo è noto per i risultati fotorealistici ma anche l'enorme costo computazionale
  - Assunzioni
    - Solo superfici di tipo diffuso
    - Suddivide le mesh della scena in patch
    - Per ogni patch determina un fattore di visibilità rispetto alle altre patch della scena (*form factor*)
-

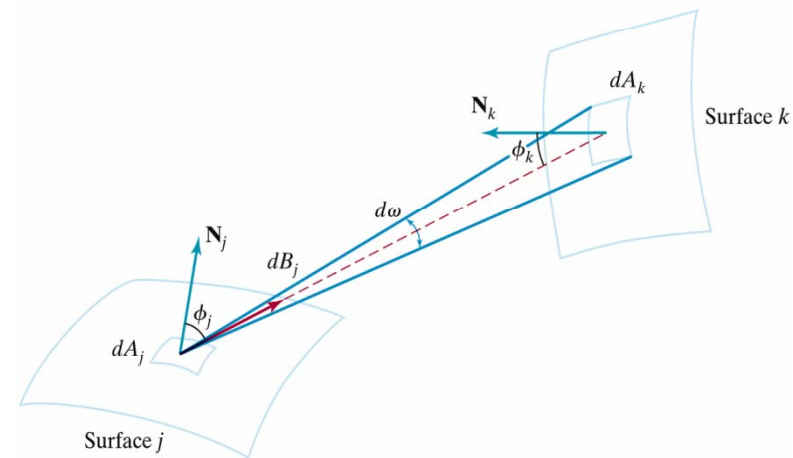
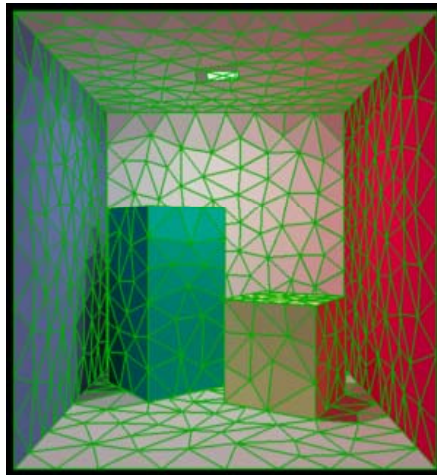
# Equazione del radiosity

- L'equazione:

$$B_k = E_k + \rho_k \sum_{j=1}^n B_j F_{jk}$$

- Dove

- $B_k$  è la radiosity della patch  $k$
- $E_k$  è la luce emessa dalla patch  $k$
- $F_{jk}$  è la frazione di luce emessa dalla patch  $j$  in direzione della patch  $k$
- $\rho_k$  è il coefficiente di riflessione della patch  $k$



# Metodi di risoluzione

- L'equazione del radiosity genera un sistema di equazioni

$$\begin{bmatrix} 1 - \rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & -\rho_2 F_{22} & \cdots & -\rho_2 F_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1 - \rho_n F_{nn} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix}$$

- Il “sistemone” si risolve attraverso tecniche numeriche standard
    - Gauss-Seidel
    - Decomposizione LU
-

# Radiosity

