

# Introduzione alla Generazione di Immagini Fotorealistiche

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

Dott. Ugo Erra

*11° Lezione – Ombre e Ambient Occlusion*

# Sommario

- Definizione delle ombre nel ray tracing
  - Generazione delle ombre
  - Implementazione delle ombre
  - Definizione dell'ambient occlusion
-

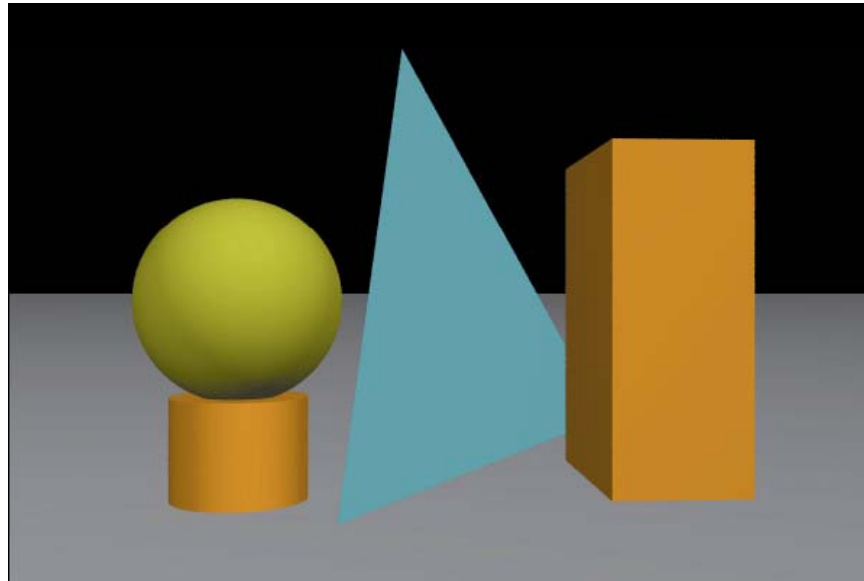
# Introduzione

- L'ombra è un'area contenuta da un perimetro che si genera a causa della presenza della luce
- Le ombre sono un fenomeno diffuso in natura sia in ambienti interni che esterni
- La percezione delle ombre è un fenomeno fondamentale per la comprensione della realtà
- Inoltre possono caratterizzare emotivamente una scena



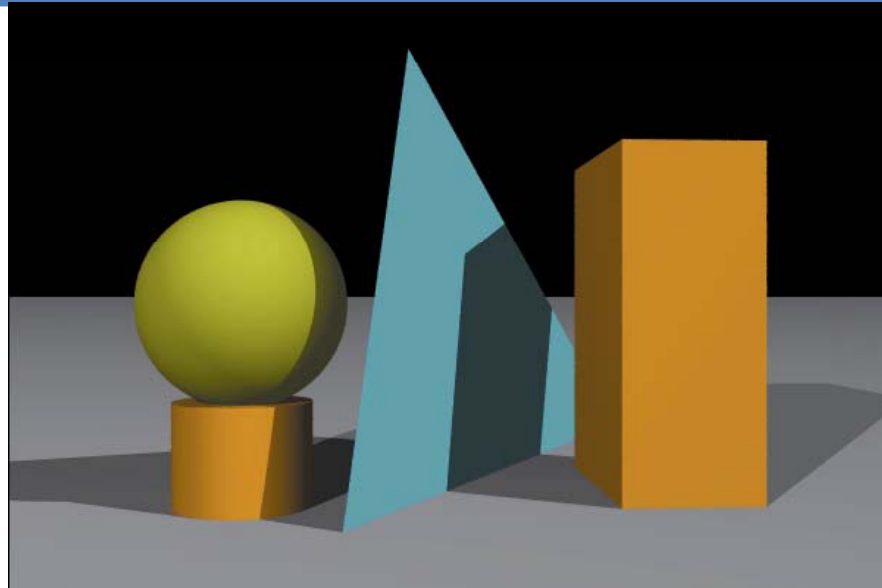
Mélancolie et mystère d'une rue  
*Giorgio De Chirico*

# L'importanza delle ombre - 1



- Consideriamo una scena senza ombre e proviamo a rispondere ad alcune semplici domande:
    - Quanto distano gli oggetti dal piano?
    - Quanto distano gli oggetti dalla camera e quanto sono grandi?
    - Quante sorgenti di luce ci sono nella scena e da che direzione proviene la luce?
    - Che tipi di luce sono presenti nella scena?
-

## L'importanza delle ombre - 2



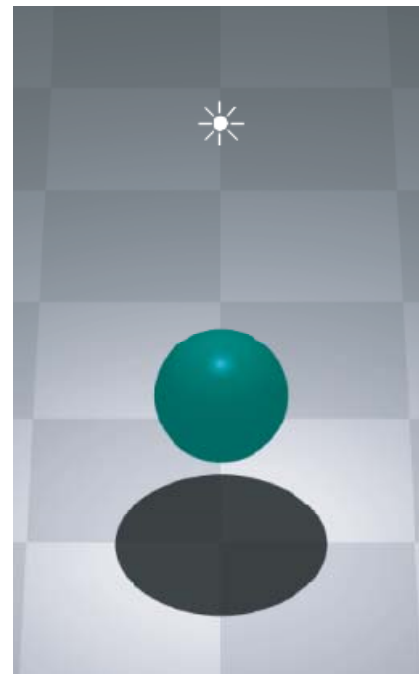
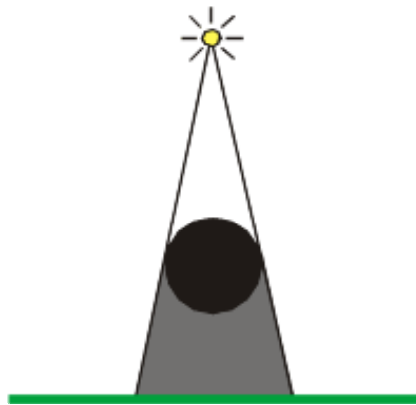
- Con le ombre possiamo facilmente rispondere alle domande
    - Gli oggetti hanno una distanza identica dalla camera
    - Gli oggetti non penetrano all'interno del piano
    - Ci sono due luci direzionali ed una puntiforme
-

# Definizioni

- In una scena con una luce puntiforme o direzionale l'ombra è quella parte della scena che non riceve luce poiché un oggetto ostruisce il suo passaggio
  - Una definizione alternativa è che l'ombra è quella parte di scena non visibile dalla sorgente di luce
-

# Ombre per luci puntiformi

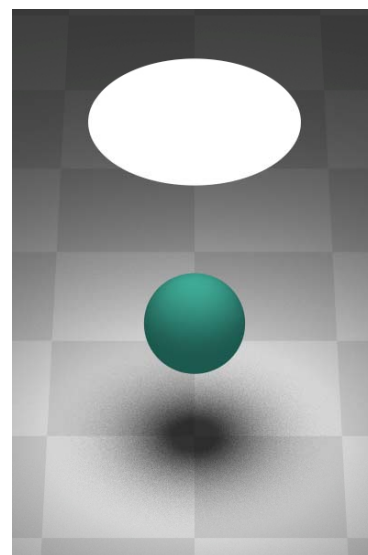
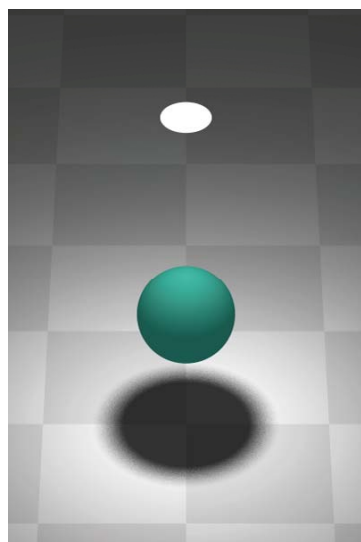
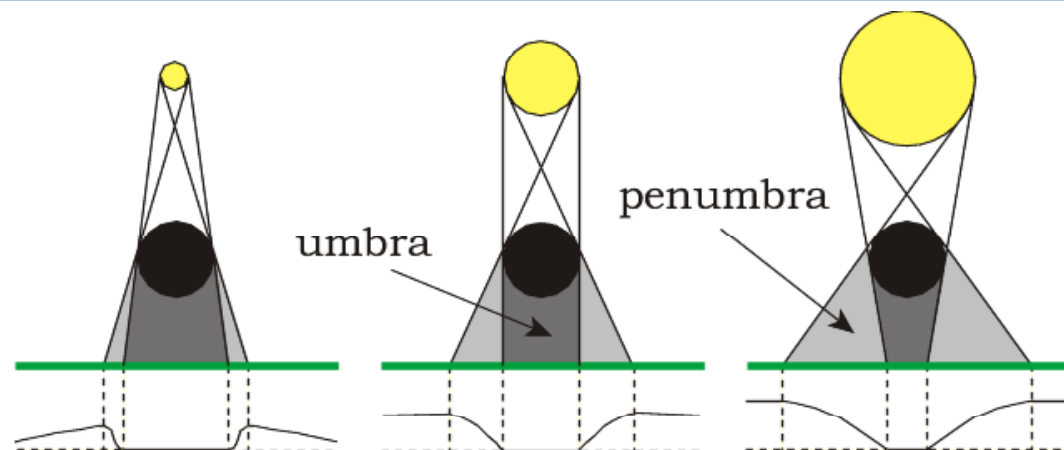
- Nel caso di una luce puntiforme l'ombra generata avrà contorni netti



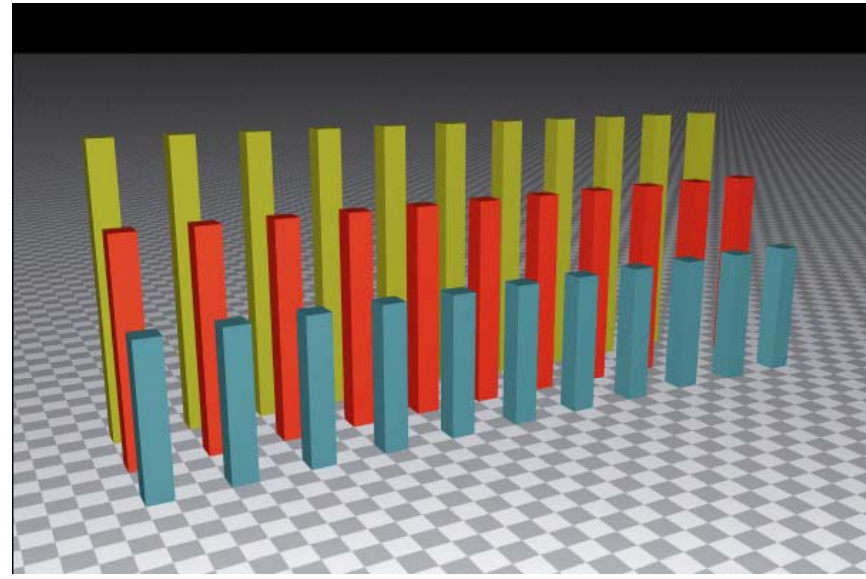
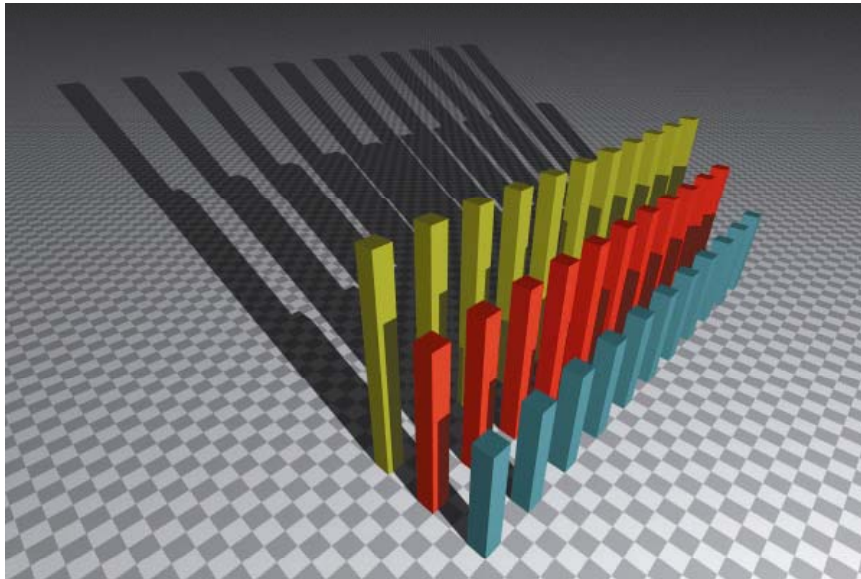
# Ombre nel caso reale

- Nel caso in cui consideriamo una sorgente di luce di superficie finita queste possono anche essere parzialmente oscurate dagli oggetti
  - Una sorgente di luce di superficie finita genera una cosiddetta soft shadow ed composta da due parti l'*ombra* (umbra) e la *penombra* (penumbra)
    - L'ombra è quella parte della soft shadow da cui non è visibile la luce
    - La penombra è quella parte della soft shadow da cui è parzialmente visibile la luce
-

# Esempi



# Esempi



# Implementazione

- Consideriamo il caso di una luce puntiforme e consideriamo la funzione di visibilità

$$V(p, p') = \begin{cases} 1 & \text{se } p \text{ e } p' \text{ sono visibili} \\ 0 & \text{se } p \text{ e } p' \text{ non sono visibili} \end{cases}$$

tra la sorgente di luce ed un punto della scena

- La radianza riflessa in quel punto sarà

$$L_o(\mathbf{p}, \omega_o) = f_r(\mathbf{p}, \mathbf{l}(\mathbf{p}), \omega_o) l_s c_l V(\mathbf{p}, \mathbf{l}_p) \cos\theta_l$$

dove  $\mathbf{l}_p$  è la posizione della luce

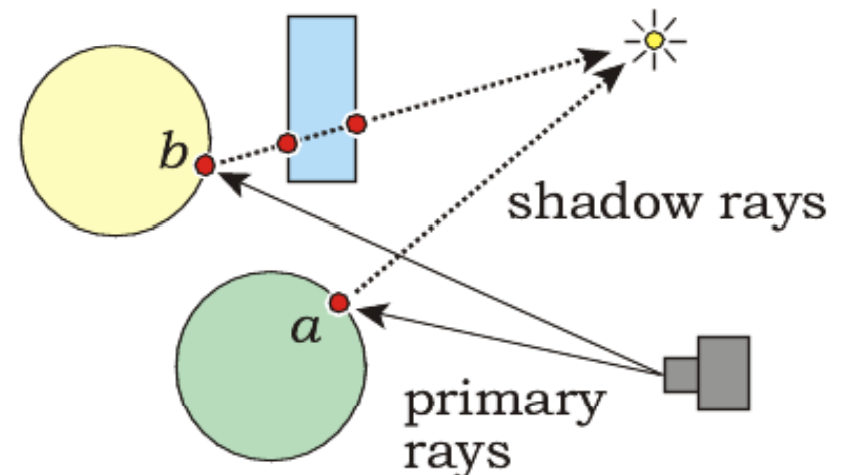
---

# Shadow ray

- L'unico modo per valutare la funzione  $V(p, l_p)$  è “sparare” un raggio definito *shadow ray* dal punto  $p$  verso la locazione della luce  $l_p$
- La direzione dello shadow ray è definita come

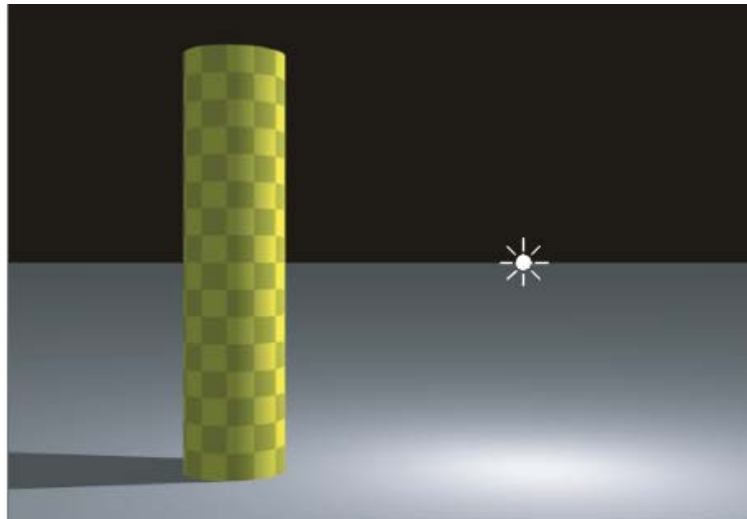
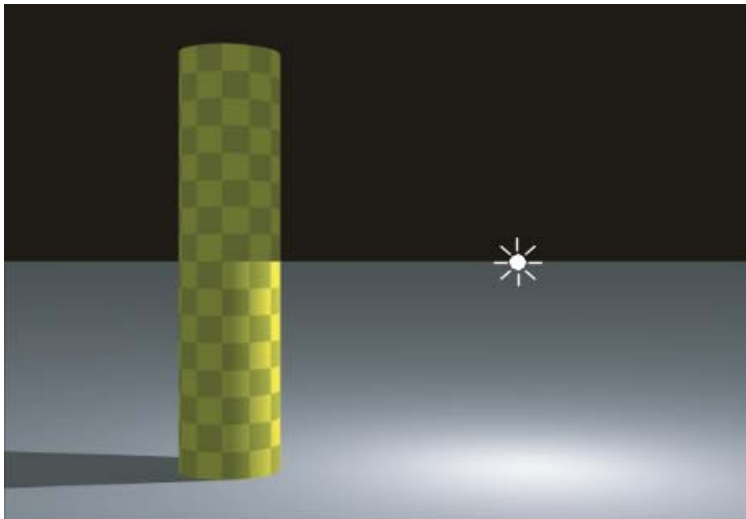
$$d = (l_p - p) / \|l_p - p\|$$

- Nell'esempio abbiamo:
  - Il punto  $a$  è illuminato dalla luce ambientale più da una sorgente di illuminazione diretta
  - Il punto  $b$  è illuminato solo da luce ambientale

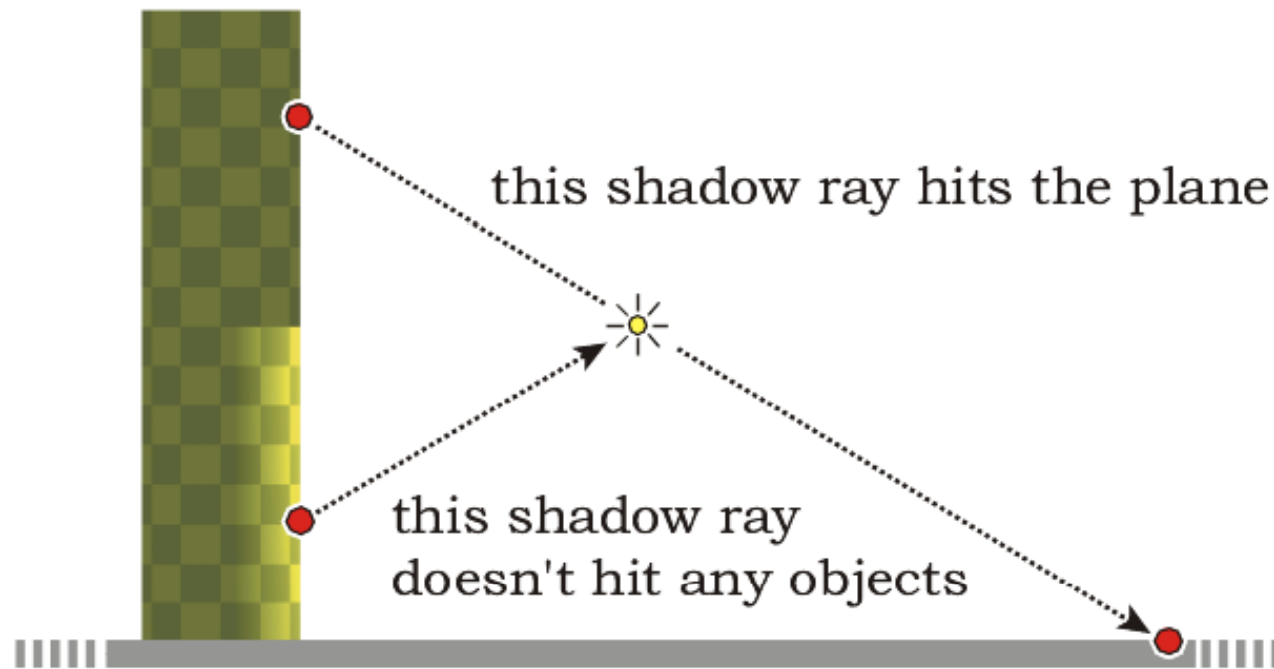


# Valutazione dello shadow ray

- Per valutare se un punto è in ombra bisogna determinare se ci sono oggetti tra il punto e la sorgente di luce
- L'ordine di eventuali oggetti colpiti rispetto alla luce è importante!



# Errore nello shadow ray



# Come determinare l'ombra

- I passi necessari per determinare se un punto è in ombra sono:
    - Inglobare il test di visibilità all'interno della funzione `shade`
    - Implementare una funzione `in_shadow` per ogni luce che determina se un punto può vedere la luce. Quindi valutare la funzione  $V(\mathbf{p}, \mathbf{l}_p)$
    - Implementare una funzione `shadow_hit` per ogni tipo di oggetto che interseca uno shadow ray con un oggetto
-

# Il metodo Phong::shade

```
RGBColor Phong::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 shadowRay(sr.hitPoint, wi);
                in_shadow = sr.w.lights[j]->in_shadow(shadowRay, sr);
            }

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

    return (L);
}
```

# Il metodo PointLight::in\_shadow

```
bool PointLight::in_shadow(const Ray& ray, const ShadeRec& sr) const {
    float t;
    int numObjects = sr.w.objects.size();
    float d = location.distance(ray.o);

    for (int j = 0; j < num_objects; j++)
        if (sr.w.objects[j]->shadow_hit(ray, t) && t < d)
            return (true);

    return (false);
}
```

---

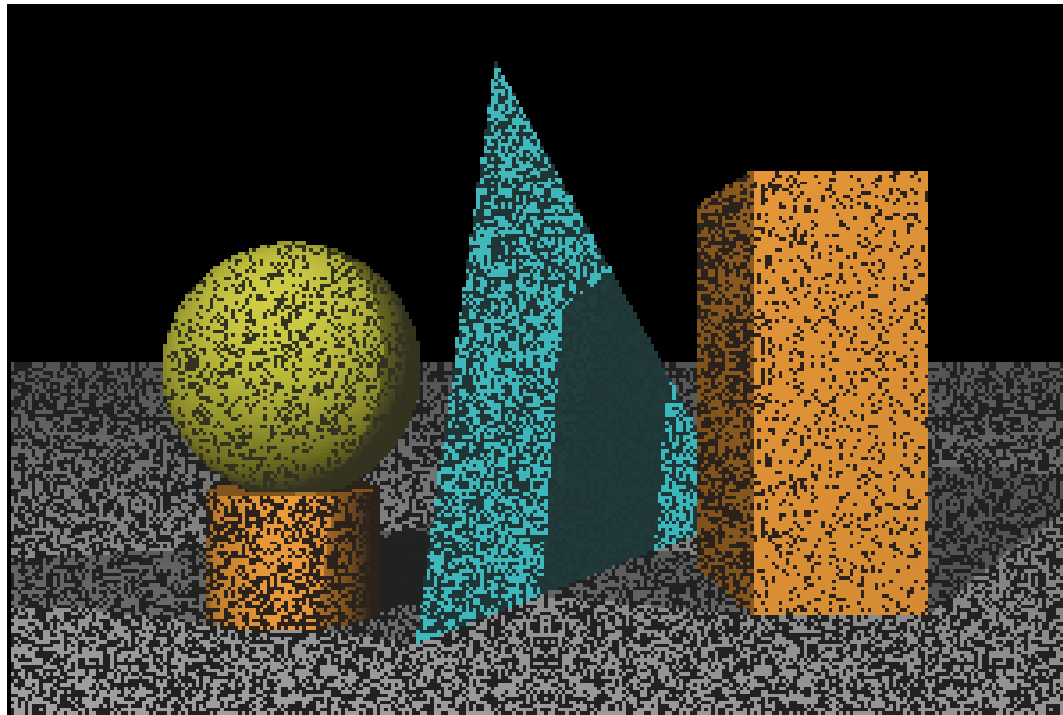
# Il metodo Plane::shadow\_hit

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

    if (t > kEpsilon) {
        tmin = t;
        return (true);
    }
    else
        return (false);
}
```

---

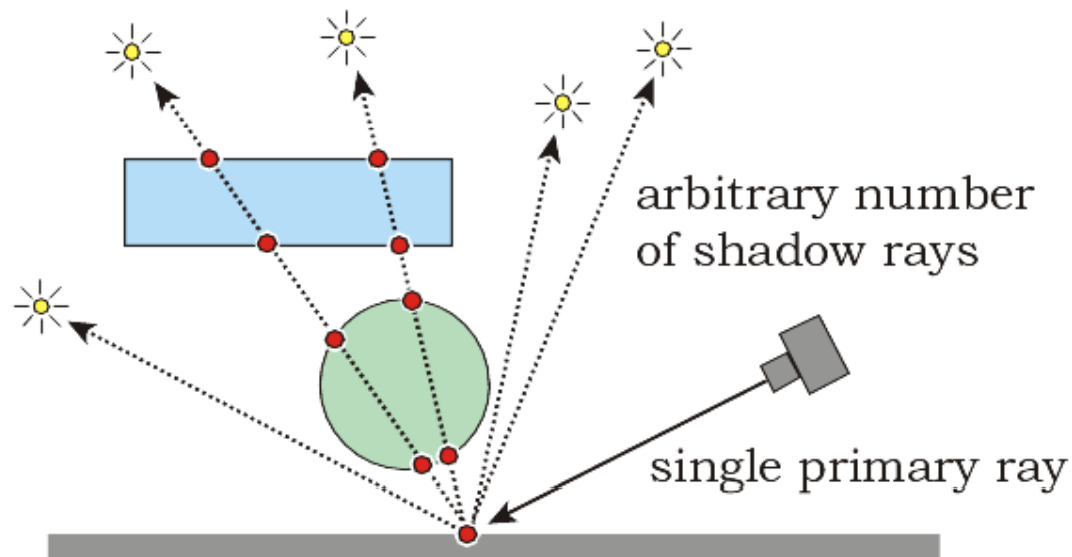
# Salt-and-pepper noise



- La scelta del punto di origine dello shadow ray è cruciale per evitare artefatti di tipo self-shadow
-

# Costi

- Il costo per generare le ombre all'interno della scena dipende dal numero di luci presenti nella scena
  - Può essere anche superiore al costo per i raggi primari

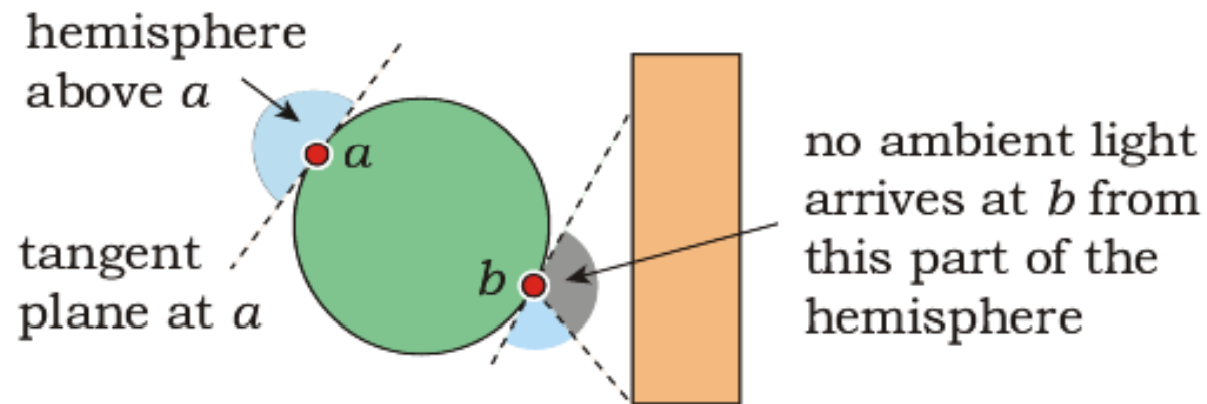


# Ambient Occlusion

- Considerare l'illuminazione ambientale costante è utile per ridurre la quantità di calcolo ma non fornisce immagini corrette
  - L'*ambient occlusion* viene adoperato per determinare la quantità di luce ambientale che ogni superficie riceve
  - L'idea di base è considerare la visibilità di un punto rispetto a tutti gli oggetti presenti nella scena
-

# Esempio

- Il punto  $a$  riceve dalla semisfera la massima luce ambientale non essendo occluso da nessun oggetto
- Il punto  $b$  riceve dalla semisfera parte della luce ambientale essendo occluso da un oggetto
- L'ambient occlusion resta un'approssimazione della diffusione della luce ambientale poiché nel caso del punto  $b$  potrebbe essere riflessa da altri oggetti



# Il modello

- Sappiamo che la radianza incidente  $L_i$  dalla luce ambientale è uguale a

$$L_i = l_s c_1$$

ed è indipendente da  $p$  e da  $\omega_i$

- Introducendo la funzione di visibilità otteniamo una nuova espressione per la radianza di luce ambientale incidente

$$L_i(p, \omega_i) = V(p, \omega_i) l_s c_1$$

dove ora  $L_i$  dipende da  $p$  e da  $\omega_i$

- La funzione di visibilità è come sempre

$$V(p, p') = \begin{cases} 1 & \text{se } p \text{ e } p' \text{ sono visibili} \\ 0 & \text{se } p \text{ e } p' \text{ non sono visibili} \end{cases}$$

---

# Stimare l'illuminazione ambientale - 1

- Per stimare l'illuminazione ambientale calcoliamo il numero di oggetti colpiti sparando degli shadow ray nella semisfera centrata sul punto  $p$
  - L'illuminazione ambientale può essere stimata considerando il numero di raggi che non colpiscono gli oggetti
-

## Stimare l'illuminazione ambientale - 2

- La stima deve essere calcolata considerando l'equazione della riflessione

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

- Sostituendo la BRDF

$$f_r = f_{r,d} = k_a \mathbf{c}_d / \pi$$

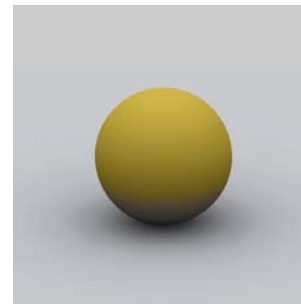
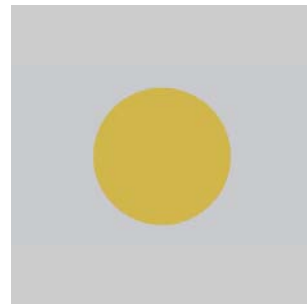
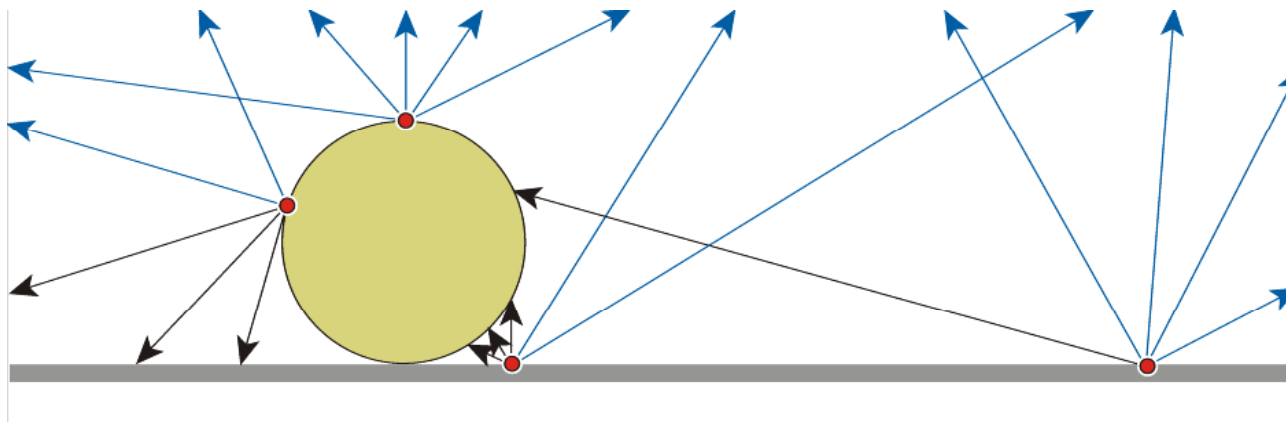
ed includendo la funzione di visibilità otteniamo

$$L_o(p, \omega_o) = (k_a \mathbf{c}_d / \pi) * (l_s \mathbf{c}_l) \int_{2\pi^+} V(p, \omega_i) \cos \theta_i d\omega_i$$

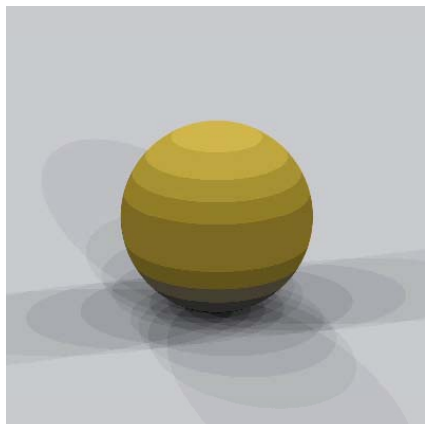
- La valutazione di questa espressione richiede di utilizzare il metodo Montecarlo campionando i raggi sulla semisfera centrata su  $p$
-

# Esempio

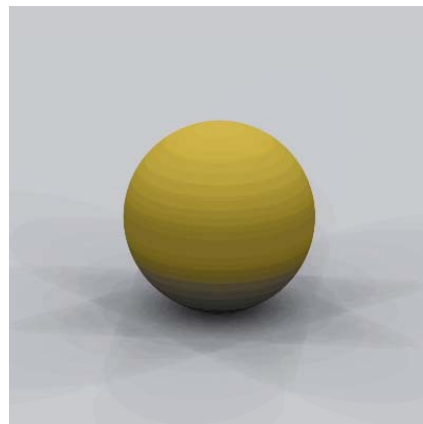
- Consideriamo una scena composta da una sfera poggiata su di un piano e consideriamo gli shadow ray generati



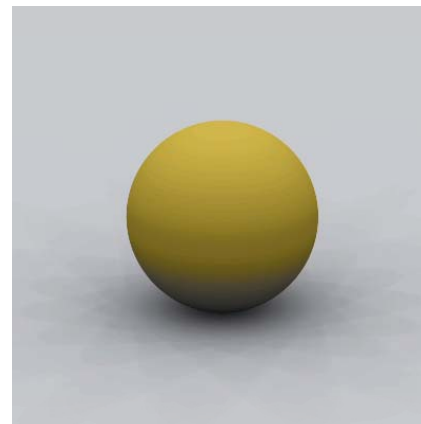
# Campionamento uniforme



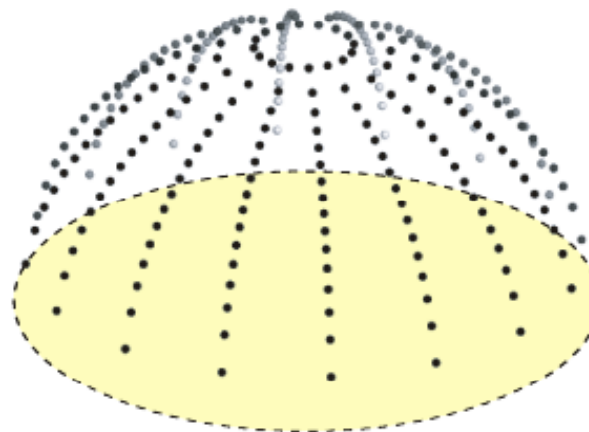
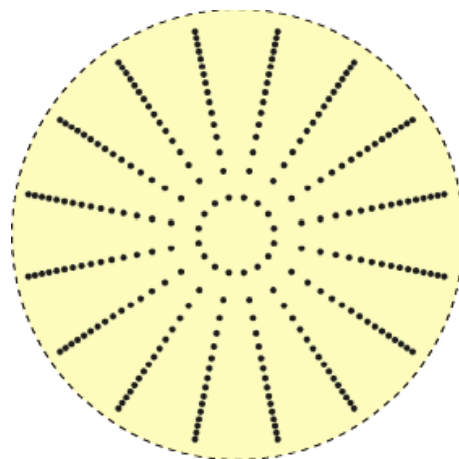
16 campioni



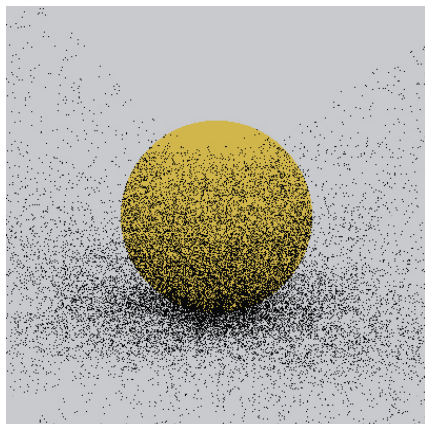
64 campioni



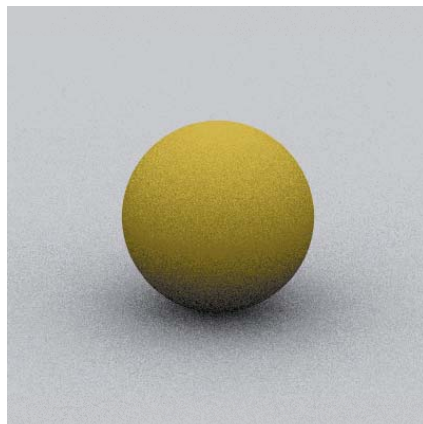
256 campioni



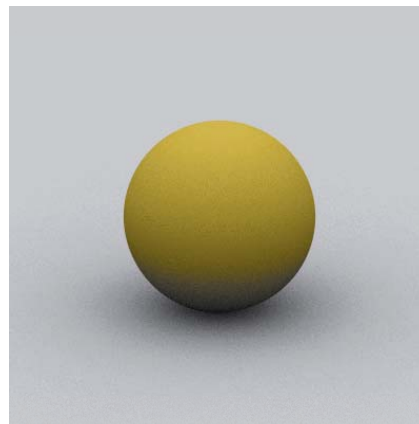
# Campionamento random/multi-jittere



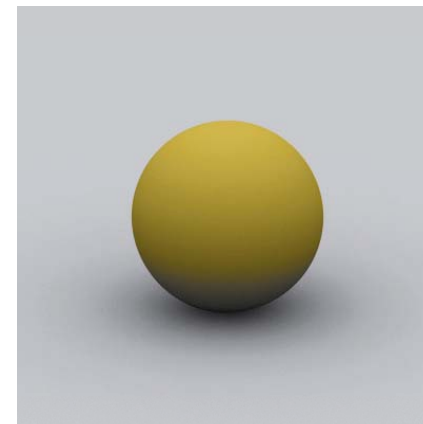
1 campione



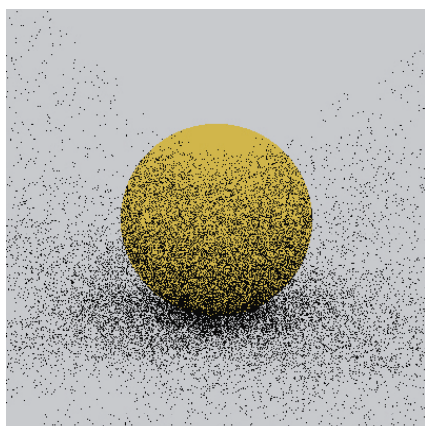
16 campioni



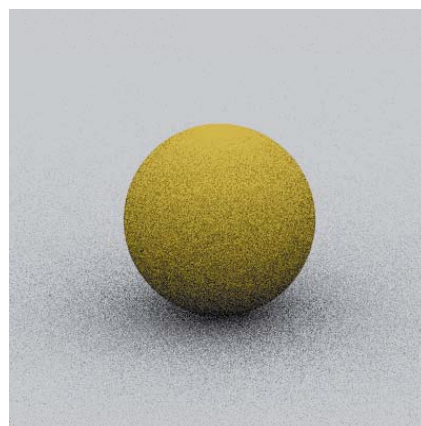
64 campioni



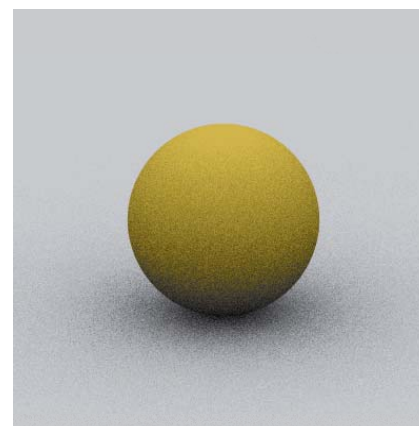
256 campioni



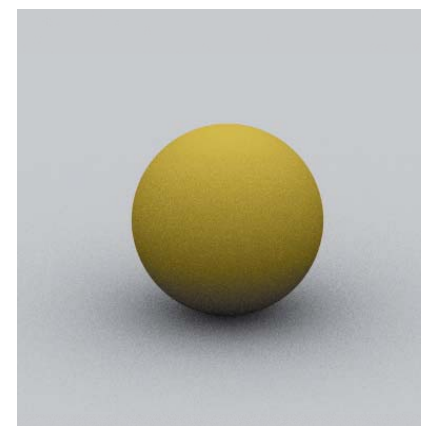
1 campione



16 campioni



64 campioni



256 campioni

# La classe AmbientOccluder

```
class AmbientOccluder: public Light {
public:

    AmbientOccluder(void);
    ...
    void
    set_sampler(Sampler* s_ptr);

    virtual Vector3D
    get_direction (ShadeRec& sr);

    virtual bool
    in_shadow(const Ray& ray, const ShadeRec& sr) const;

    virtual RGBColor
    L(ShadeRec& sr);

private:

    Vector3D u, v, w;
    Sampler* sampler_ptr;
    RGBColor min_amount;
};
```

# Esempio di uso di

```
void World::build(void) {
    vp.set_hres(400);
    vp.set_vres(400);
    vp.set_samples(1);

    tracer_ptr = new RayCast(this);
    MultiJittered* sampler_ptr = new MultiJittered(num_samples);
    AmbientOccluder* occluder_ptr = new AmbientOccluder;
    occluder_ptr->scale_radiance(1.0);
    occluder_ptr->set_min_amount(0.0);
    occluder_ptr->set_sampler(sampler_ptr);
    set_ambient_light(occluder_ptr); set_ambient_light(ambient_ptr);

    Pinhole* pinhole_ptr = new Pinhole;
    ...
    // sphere
    Matte* matte_ptr1 = new Matte;
    matte_ptr1->set_ka(0.75);
    matte_ptr1->set_kd(0);
    matte_ptr1->set_cd(1, 1, 0); // yellow
    Sphere* sphere_ptr = new Sphere(Point3D(0, 1, 0), 1);
    sphere_ptr->set_material(matte_ptr1);
    add_object(sphere_ptr);

    // ground plane
    Matte* matte_ptr2 = new Matte;
    matte_ptr2->set_ka(0.75);
    matte_ptr2->set_kd(0);
    matte_ptr2->set_cd(white);
    Plane* plane_ptr = new Plane(Point3D(0), Normal(0, 1, 0));
    plane_ptr->set_material(matte_ptr2);
    add_object(plane_ptr);
}
```