

# Introduzione alla Generazione di Immagini Fotorealistiche

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

Dott. Ugo Erra

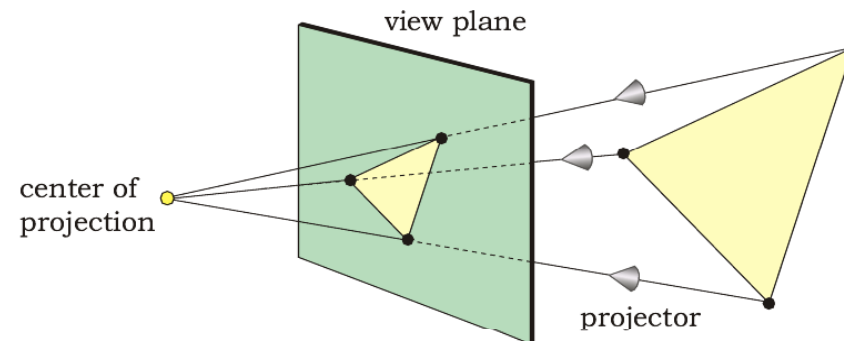
*6° Lezione – Proiezione prospettica*

# Sommario

- Proiezione prospettica
  - Proprietà della prospettiva
  - Una semplice implementazione
  - Distorsione prospettica
  - Un modello di pinhole camera
-

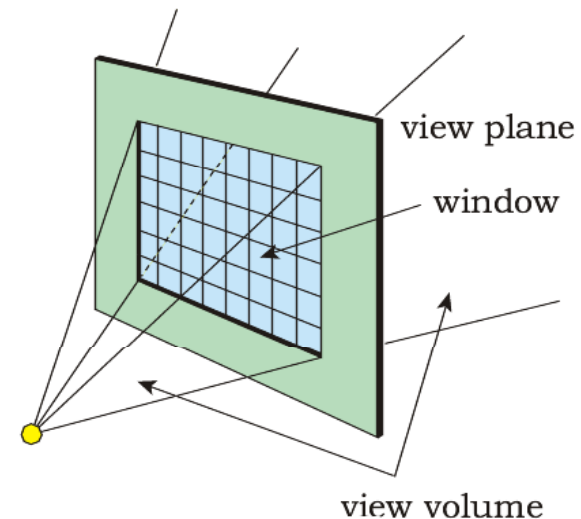
# Definizioni - 1

- Una *proiezione* trasforma un punto nello spazio 3D in un punto su di una superficie 2D chiamata *view plane*
- I punti sono trasformati lungo dei raggi *proiettori* che convergono in un punto chiamato *centro della proiezione*



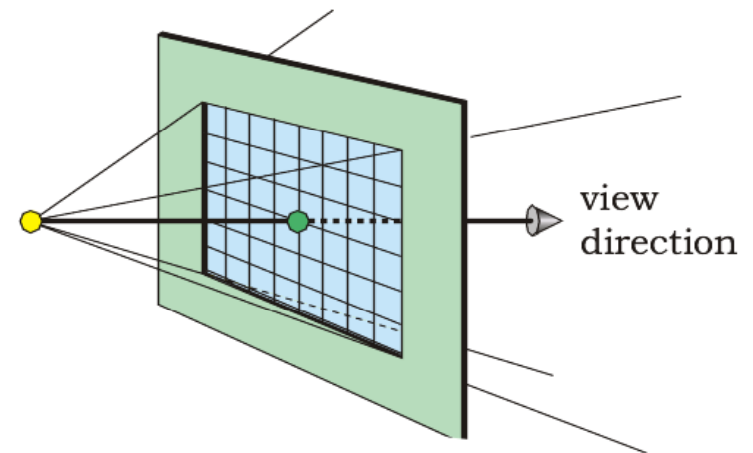
# Definizioni - 2

- Come nella proiezione ortografica definiamo una finestra di pixel sulla view plane
- La parte della scena che è visibile dalla finestra dipende dalla posizione del centro di proiezione e dall'orientamento della finestra
- Definiamo il *view volume* come una piramide il cui apice è il centro della proiezione



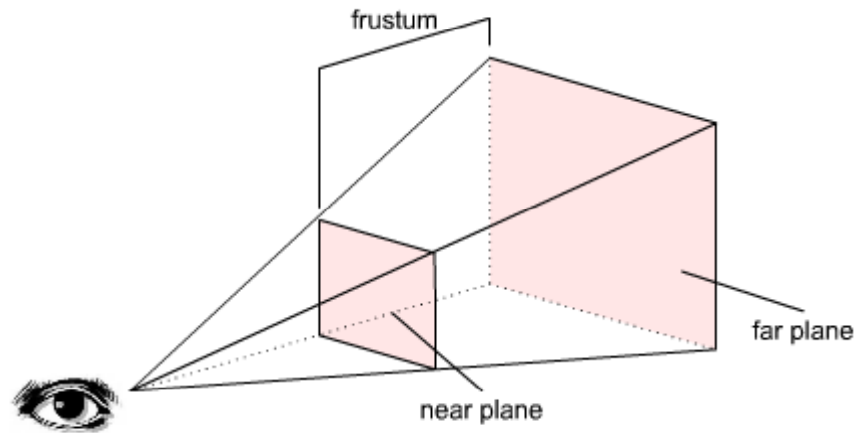
# Definizioni - 3

- In un generico sistema di proiezione prospettica definiamo
  - La posizione del centro di proiezione
  - La posizione e l'orientamento della finestra
- In un sistema di proiezione simmetrico la retta che definisce la *direzione dello sguardo* (view direction)
  - Passa dal centro della finestra
  - E' perpendicolare alla finestra
- Il view volume è una piramide infinita simmetrica



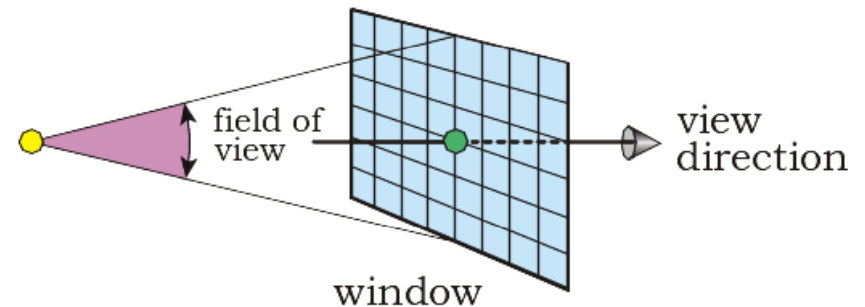
# Frustum

- Il *view frustum* è un view volume del piani di clipping definiti come *near clipping plane* e *far clipping plane*
  - Utilizzato nella grafica raster



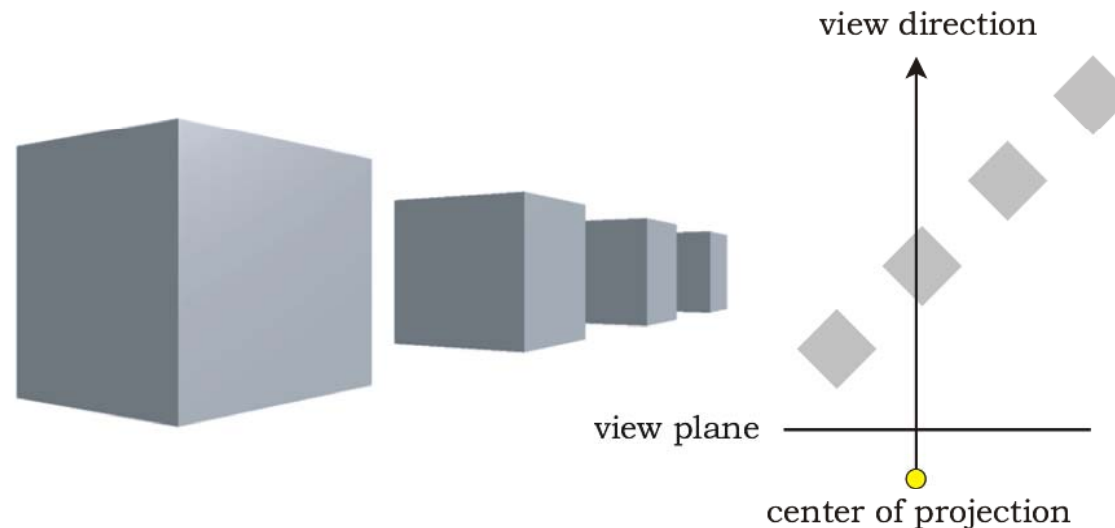
# Field of view

- Un'altra misura della quantità di scena visibile è il campo visivo (field of view)
- Per impostare un campo visivo è sufficiente specificare l'angolo sotteso dal centro della proiezione dal piano superiore ed inferiore della piramide



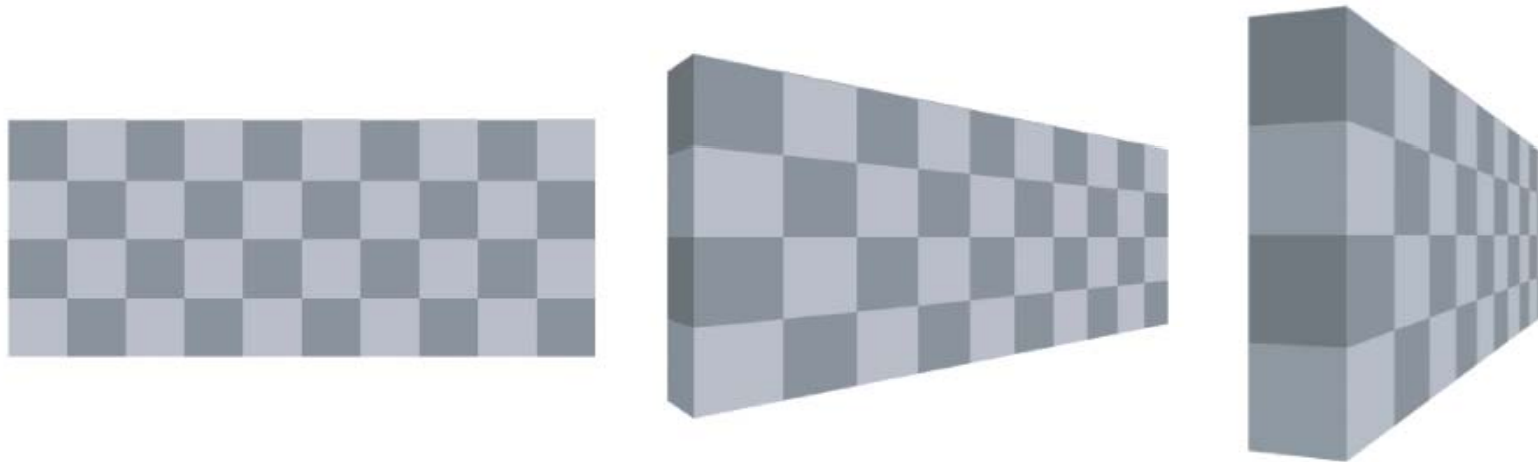
# Proprietà della proiezione prospettica

- Proprietà 1: Nella proiezioni prospettica la dimensione di un oggetto diminuisce all'allontanarsi dal centro della proiezione



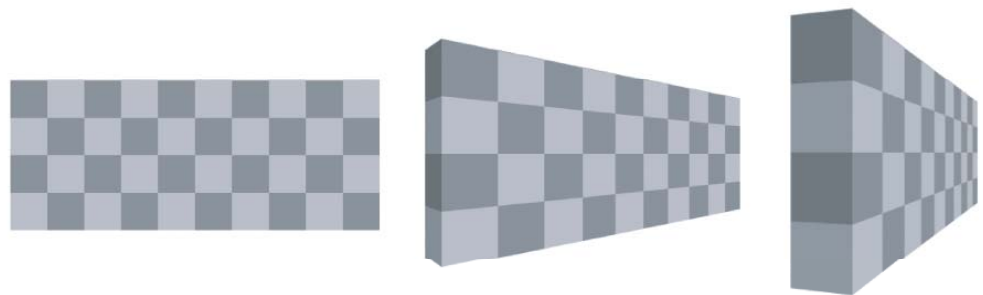
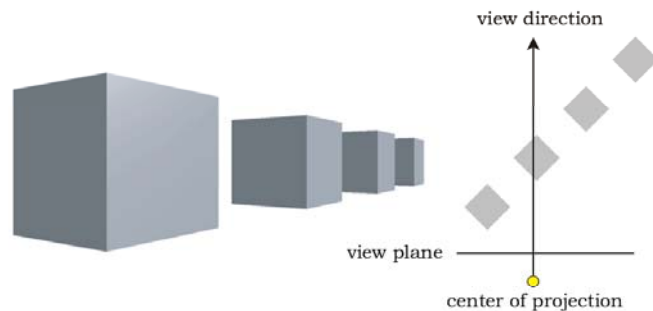
# Proprietà della proiezione prospettica

- Proprietà 2: Nella proiezione prospettica la dimensione della proiezione orizzontale diminuisce al ruotare dell'oggetto



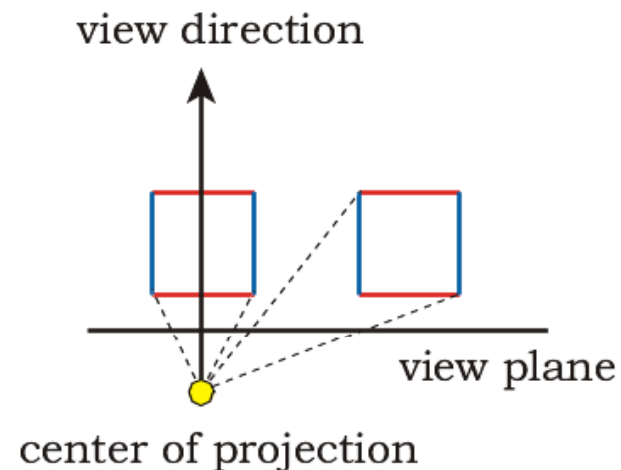
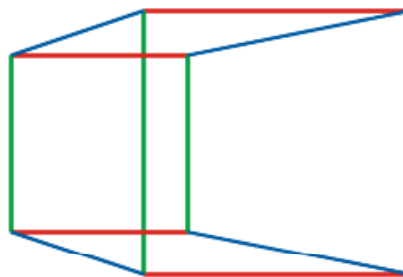
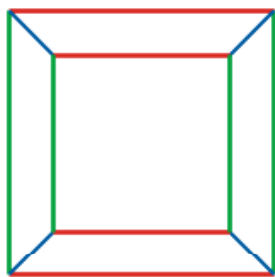
# Proprietà della proiezione prospettica

- Proprietà 3: La proiezione prospettica preserva le linee rette



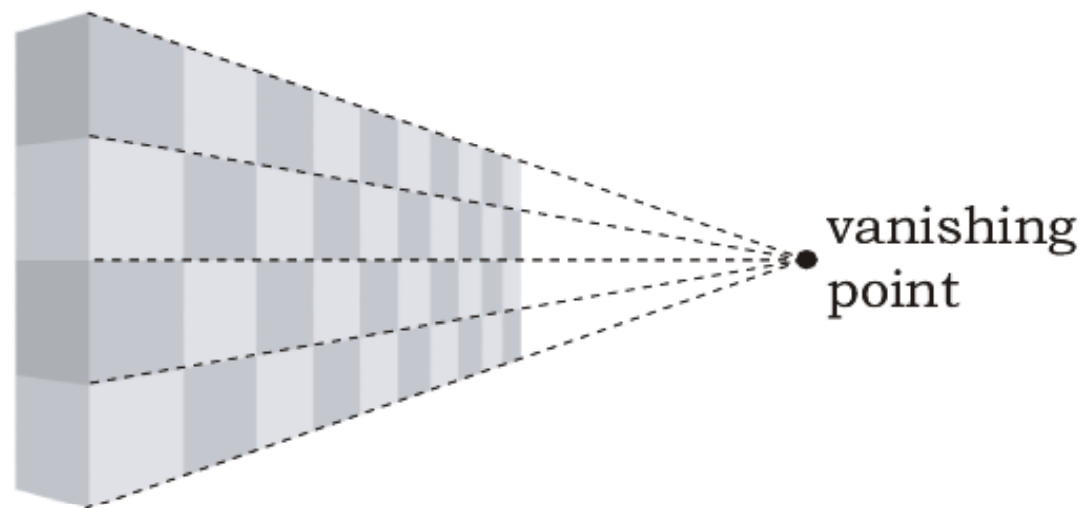
# Proprietà della proiezione prospettica

- Proprietà 4: L'insieme delle linee parallele alla view plane si mantengono parallele alla view plane sul piano di proiezione



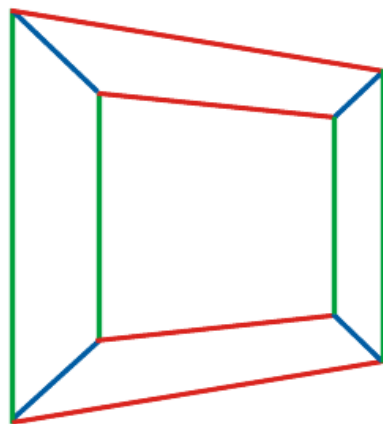
# Proprietà della proiezione prospettica

- Proprietà 5: L'insieme delle linee parallele che non sono parallele alla view plane convergono in un *punto di fuga* sul piano della proiezione

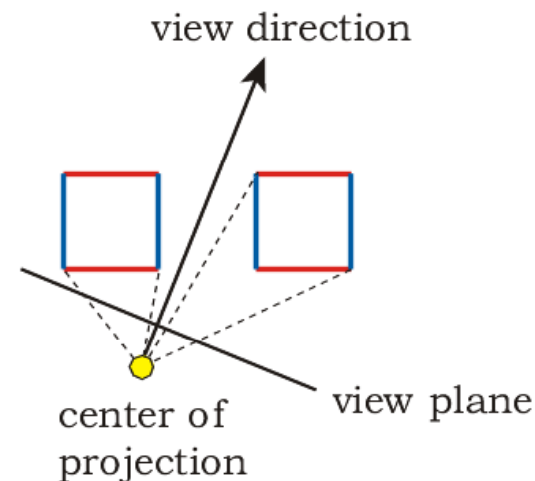
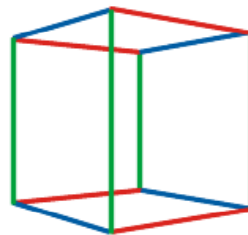


# Punti di fuga

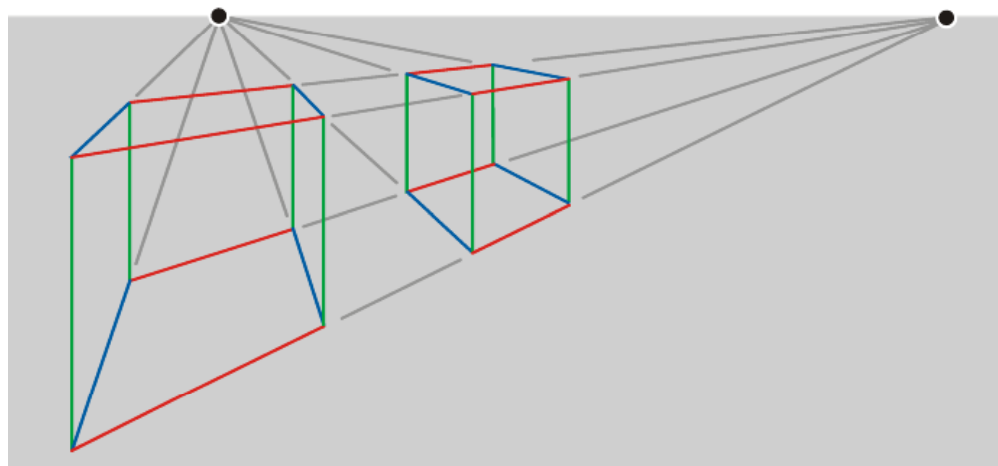
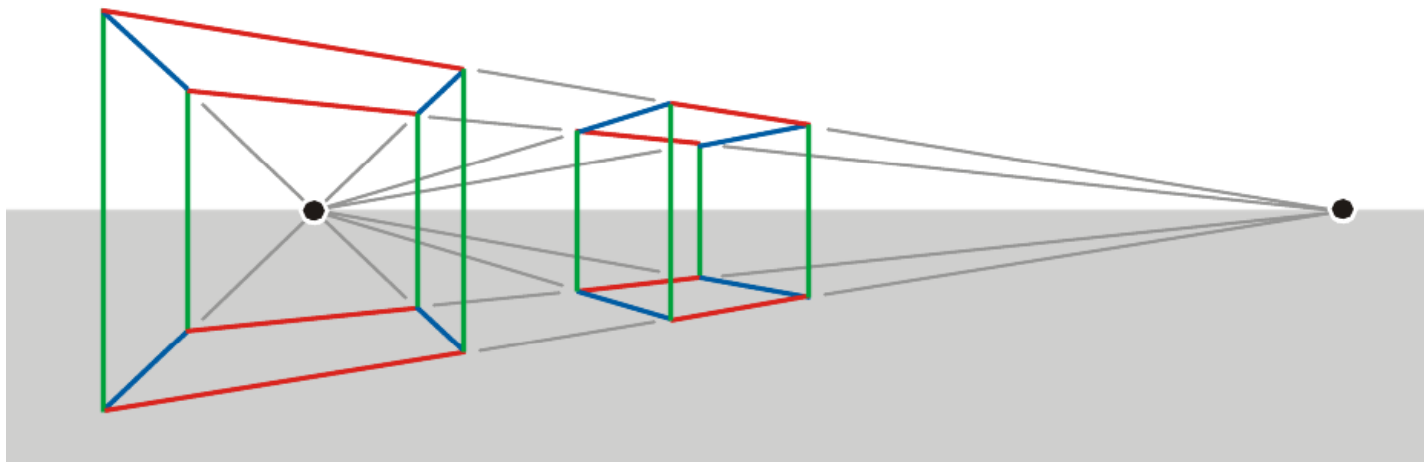
- La proiezioni prospettica è caratterizzata in base al numero di punti di fuga
  - Sono possibili uno, due e tre punti di fuga nella proiezione prospettica



Solo le linee verdi sono ancora parallele al view plane

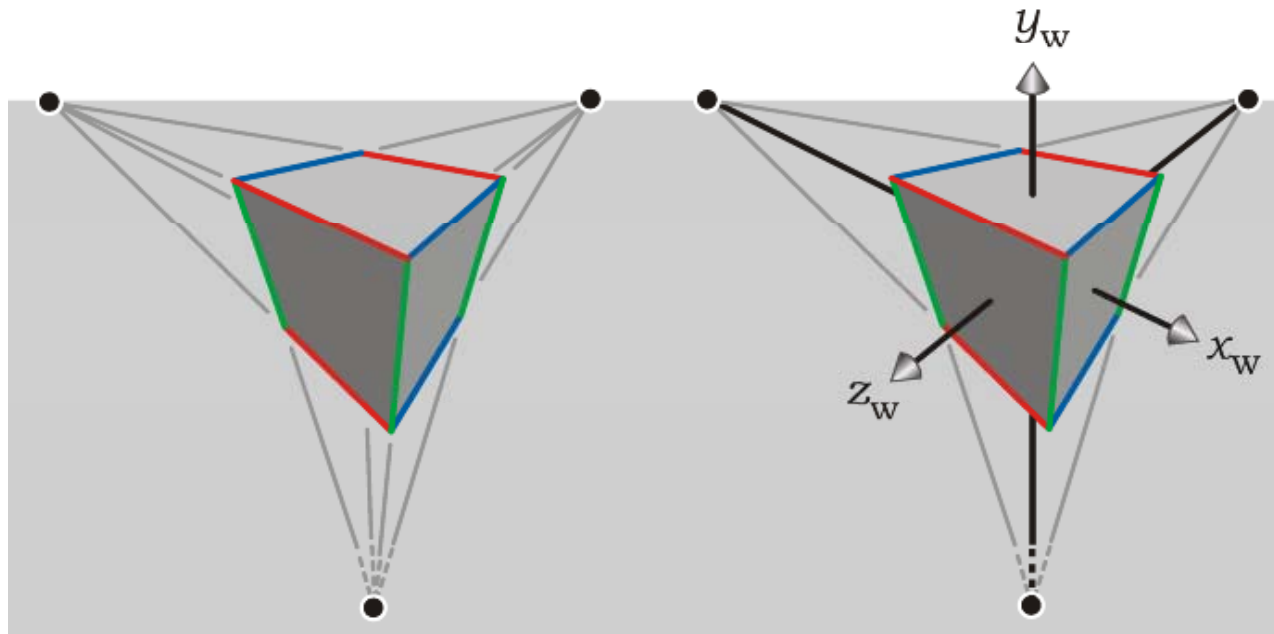


# Punti di fuga



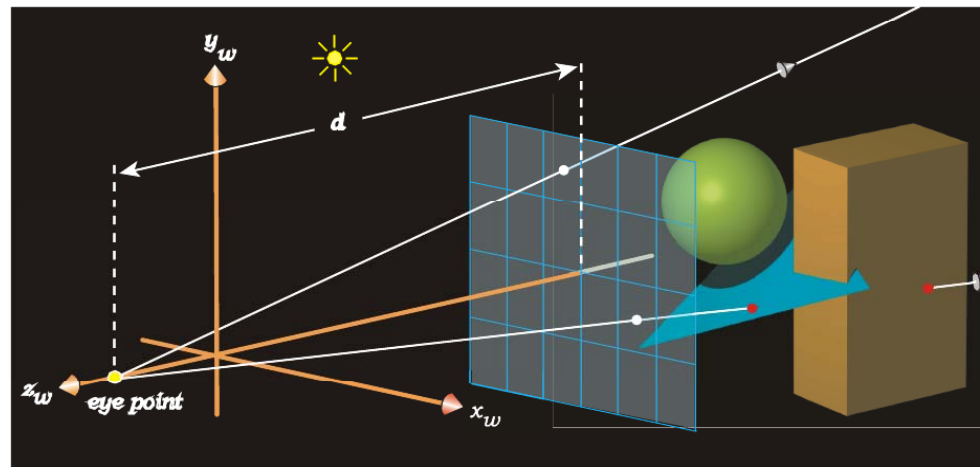
# Proprietà della proiezione prospettica

- Proprietà 6: Se un insieme di linee sono parallele agli assi della scena i punti di fuga sono sugli assi e sono noti come punti di fuga principali



# Vista prospettica allineata agli assi

- Consideriamo un caso semplice di vista prospettica
  - Il centro della proiezione è posizionato sull'asse  $z_w = e$
  - La direzione dello sguardo è verso  $-z_w$
  - La distanza dal view plane  $d$  è misurata a partire dal centro della proiezione



# Vista prospettica allineata agli assi

- Sia  $h_{res}$  e  $v_{res}$  la risoluzione orizzontale e verticale e sia  $s$  la dimensione del pixel
- Nella proiezione prospettica tutti raggi hanno la medesima origine

$$\mathbf{o} = (0, 0, e)$$

- La direzione di un raggio che passa per il centro di un pixel nella riga  $r$  e colonna  $c$  è data

$$\mathbf{d} = (s(c - h_{res}/2 + 0.5), s(r - v_{res}/2 + 0.5), -d)$$

---

# Metodo World:render\_perspective

```
void World::render_perspective(void) const {
    RGBColor      pixel_color;
    Ray           ray;

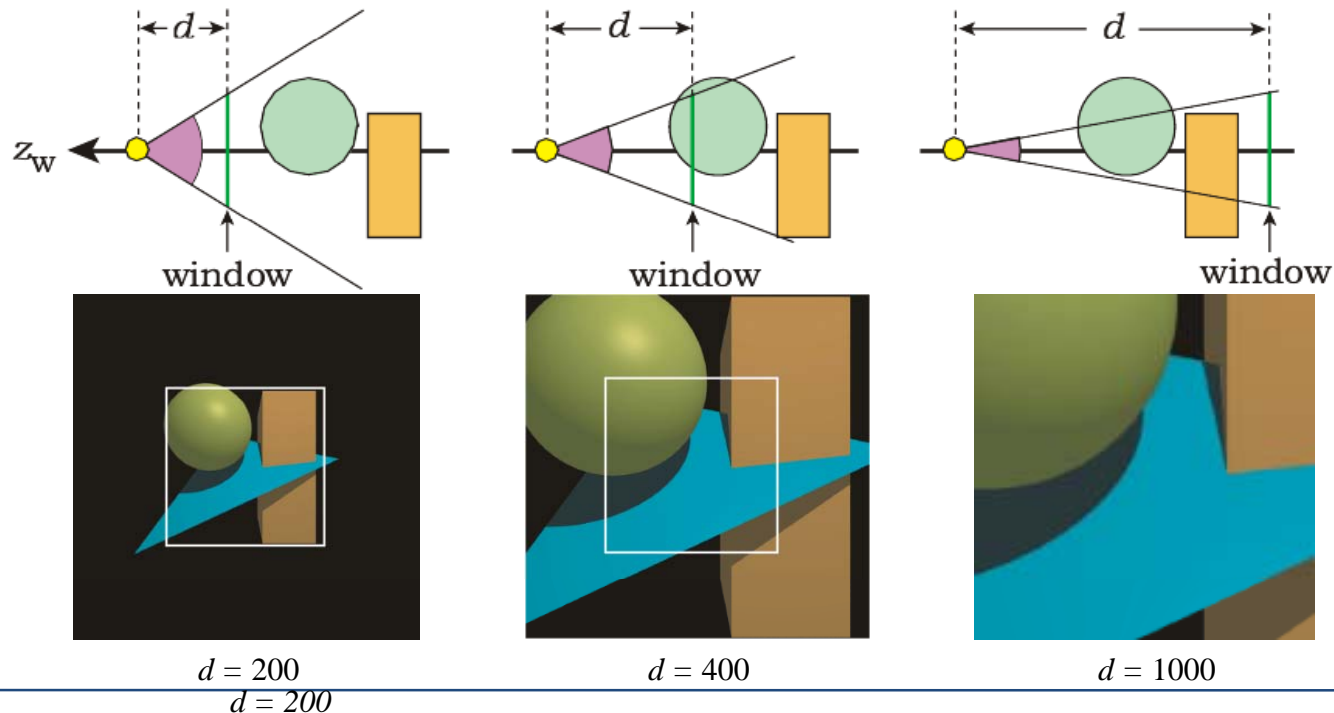
    open.window(hres, vres);
    ray.o = Point3D(0.0, 0.0, eye);

    for (int r = 0; r < vres; r++)           // up
        for (int c = 0; c <= hres; c++) {    // across
            ray.d = Point3D(s * (c - hres / 2.0 + 0.5), s * (r - vres / 2.0 + 0.5), zw);

            ray.d.normalize();
            pixel_color = tracer_ptr->trace_ray(ray);
            display_pixel(r, c, pixel_color);
        }
}
```

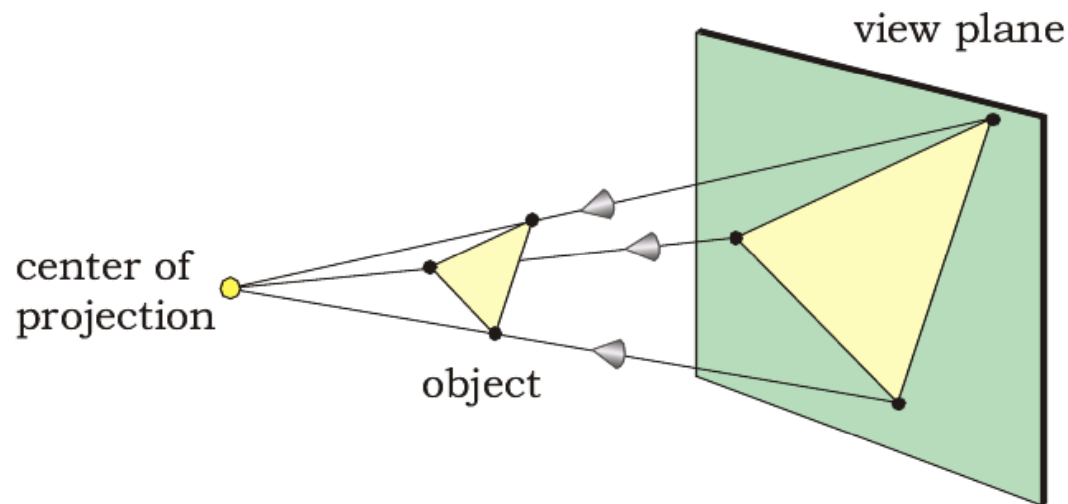
# Spostare il viewplane

- Supponiamo che:
  - La risoluzione e la dimensione del pixel è fissata
  - Il centro di proiezione  $e$  immobile
- Incrementando  $d$  riduciamo il campo visivo permettendo di zoomare all'interno della scena



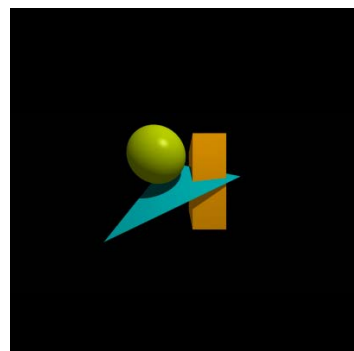
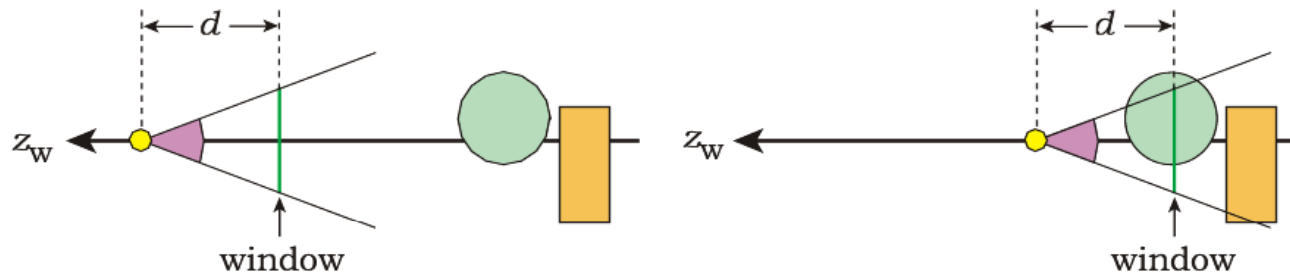
# Zoom

- In base a dove è posizionato il view plane la dimensione della proiezione degli oggetti sarà differente
  - Se il view plane si trova tra il centro di proiezione ed un oggetto questo apparirà più piccolo della sua dimensione reale
  - Se il view plane si trova ben oltre un oggetto questo apparirà più grande della sua dimensione reale

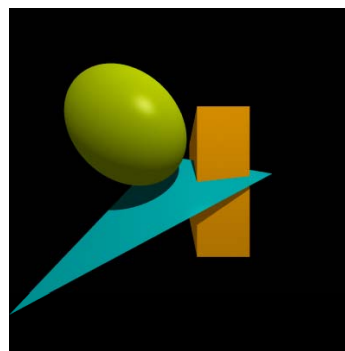


# Spostare il centro di proiezione

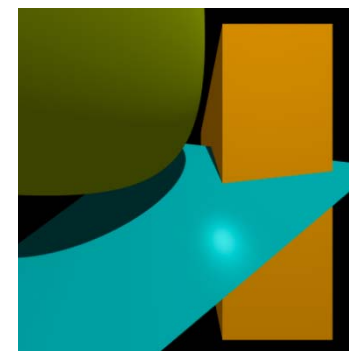
- Supponiamo di mantenere  $d$  fissato e variare il centro di proiezione
  - La dimensione della view plane non varia
- La dimensione degli oggetti cambia (zoom) ma cambia anche la proiezione prospettica
- La forma degli oggetti cambia fino ad ottenere una *distorsione prospettica*



$e=(0,0,200)$



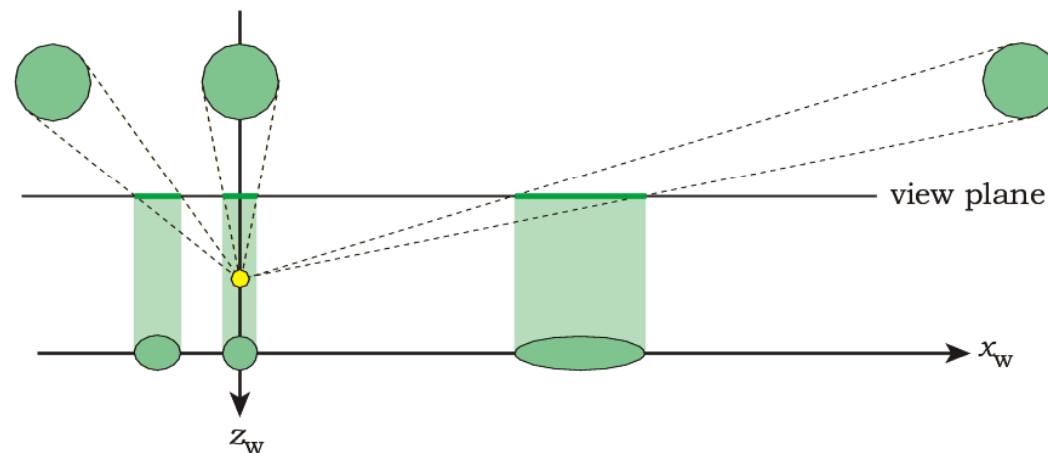
$e=(0,0,150)$



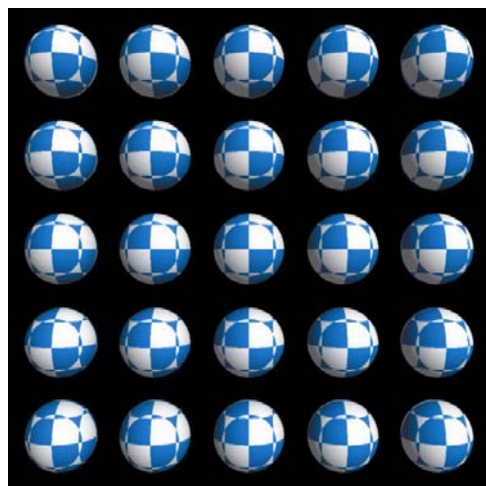
$e=(0,0,100)$

# Distorsione prospettica

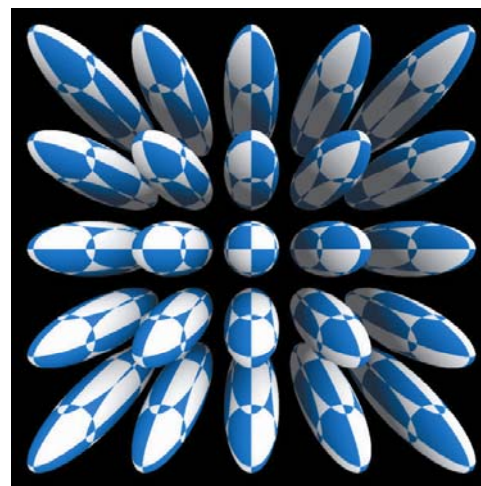
- La distorsione prospettica è causata dal fatto che il view plane è piano
  - La retina dell'occhio umano è sferica
- La distorsione è particolarmente accentuata per oggetti sferici



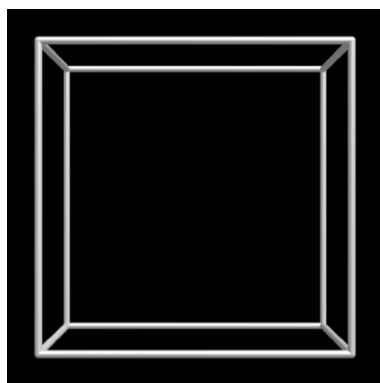
# Distorsione prospettica



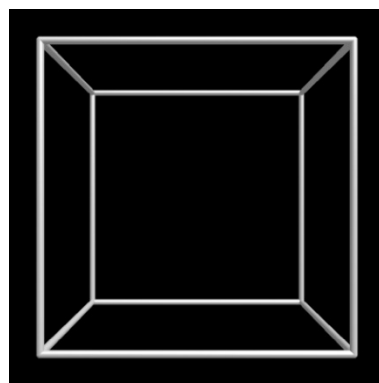
$e = 15, d = 875$



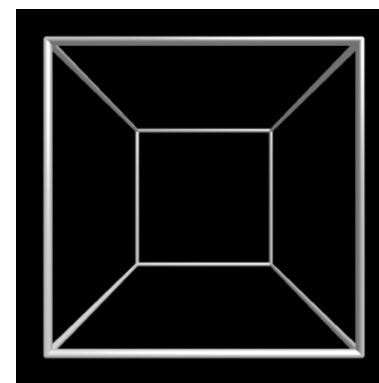
$e = 2, d = 85$



$e = 10, d = 1687$



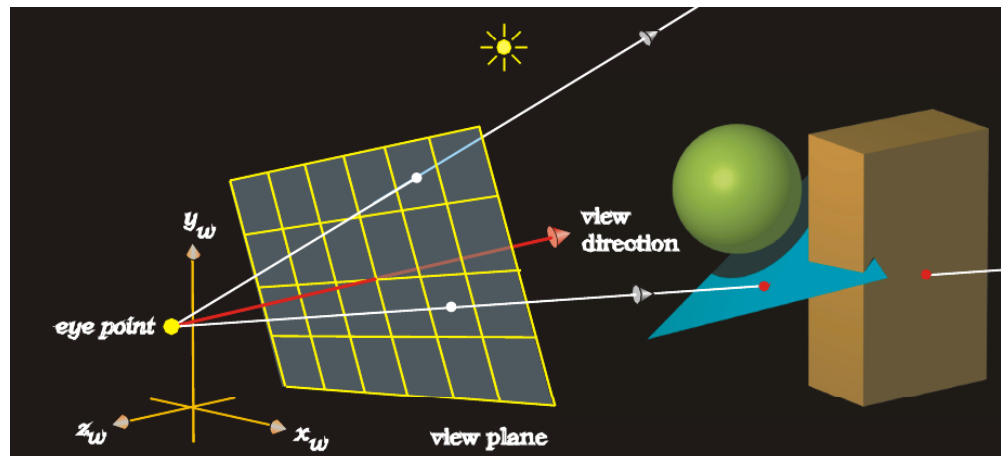
$e = 5, d = 750$



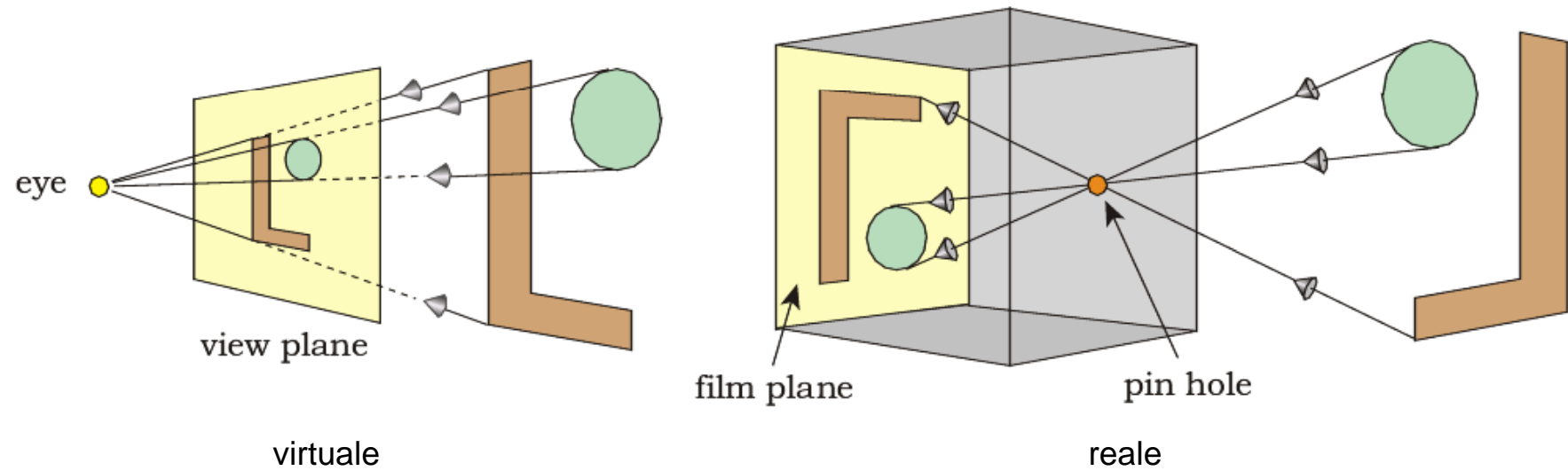
$e = 2.5, d = 280$

# Un modello di pinhole camera

- Un modello di pinhole camera implementa la visione prospettica utilizzando le seguenti informazioni
  - La posizione dell'osservatore (eye point)
  - La direzione dello sguardo (view direction)
  - Un orientamento attorno alla direzione dello sguardo
  - Una distanza tra l'osservatore ed il view plane

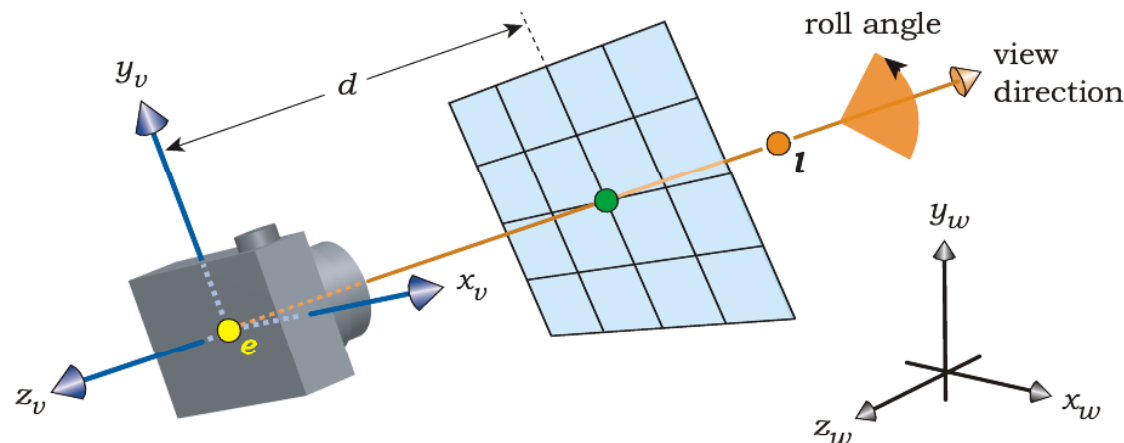


# Pinhole camera reale e virtuale



# Interfaccia utente alla pinhole camera

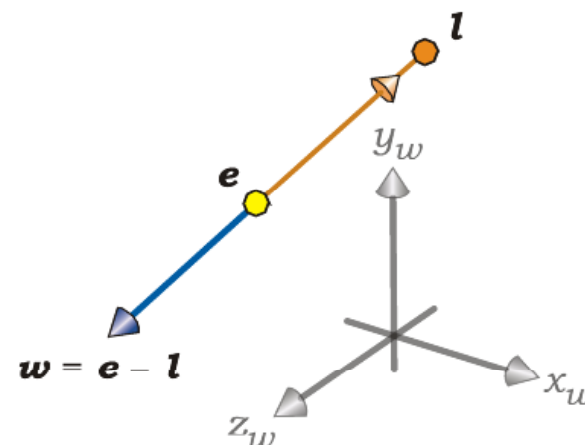
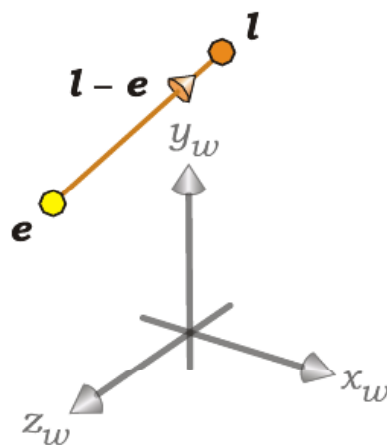
- Per costruire un sistema di riferimento per la camera utilizzeremo i seguenti parametri:
  - La posizione dell'osservatore  $e$
  - La direzione dello sguardo  $l$
  - Un vettore  $up$
  - La distanza dal view plane  $d$
- Tutti i parametri sono definiti all'interno delle world coordinates



# Viewing coordinates - 1

- Lo scopo è definire un sistema di coordinate per la camera, chiamato *viewing system*, costruendo una base ortonormale  $(u, v, w)$
- Il vettore  $w$  sarà dato da:

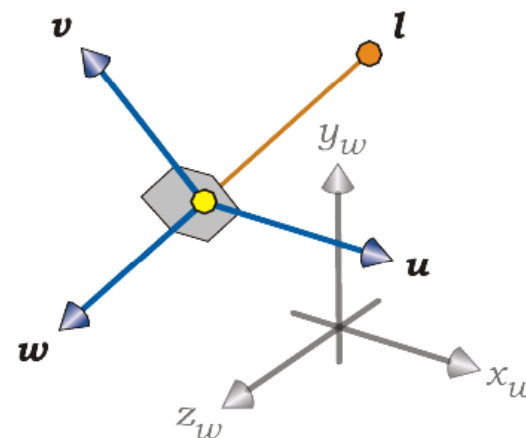
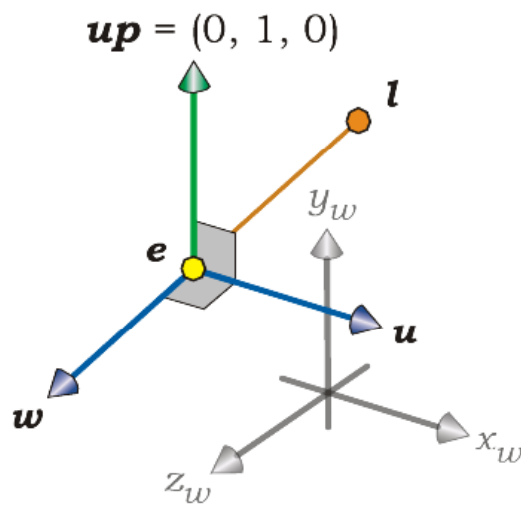
$$w = (e - l) / \|e - l\|$$



# Viewing coordinates - 2

- Supponendo il vettore  $up = (0, 1, 0)$  calcoliamo il vettore  $u$  come:

$$u = up \times w / \|up \times w\|$$

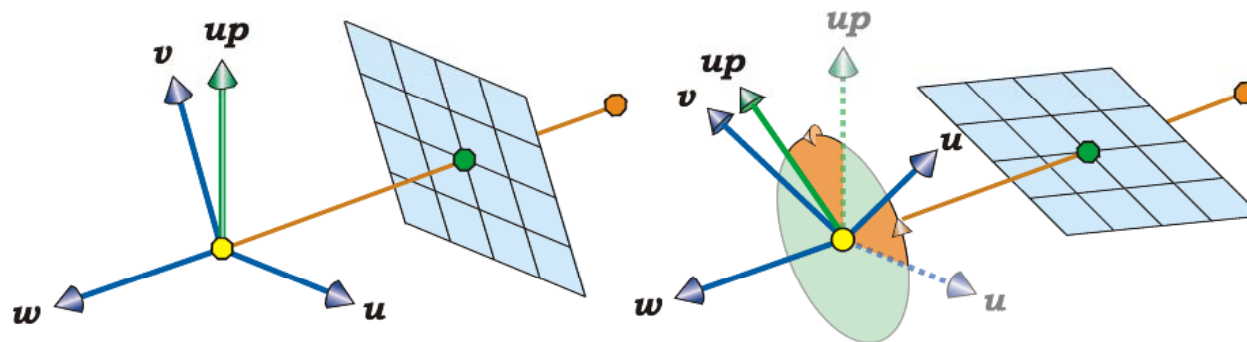


# Viewing coordinates - 3

- Il vettore  $v$  è dato da:

$$v = w \times u$$

- Il sistema di riferimento costruito fornisce anche la possibilità di ruotare la camera rispetto alla direzione dello sguardo
- Due osservazioni
  - La riga dei pixel orizzontali del view plane è parallela a  $u$
  - La colonna dei pixel verticali del view plane è parallela a  $v$



# Come calcolare i raggi primari?

- La base ortonormale ( $\mathbf{u}$ ,  $\mathbf{v}$ ,  $\mathbf{w}$ ) e la posizione dell'osservatore  $e$  formano un sistema di riferimento con il quale calcolare i raggi primari
- Dato un pixel di coordinate  $(r, c)$ , la posizione di un punto  $\mathbf{p}=(x_v, y_v)$  all'interno del pixel sarà:

$$x_v = s(c - h_{res}/2 + p_x)$$

$$y_v = s(c - v_{res}/2 + p_y)$$

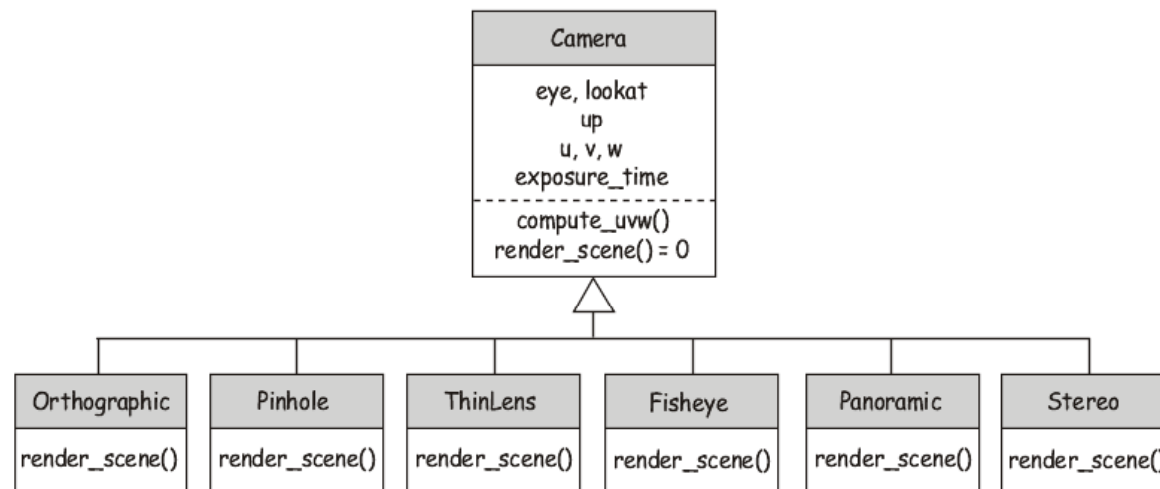
- Un raggio primario  $\mathbf{d}$  può essere espresso come combinazione lineare dei vettori della base

$$\mathbf{d} = x_v \mathbf{u} + y_v \mathbf{v} - d \mathbf{w}$$

- Poiché  $\mathbf{u}$ ,  $\mathbf{v}$  e  $\mathbf{w}$  sono rappresentati mediante world coordinate anche  $\mathbf{d}$  lo sarà
  - Il vettore  $\mathbf{d}$  deve essere normalizzato
-

# Implementazione

- Poiché saranno possibili diversi tipi di camere prevediamo una classe Camera generica che andremo successivamente a specializzare



# La classe Camera

```
class Camera {
public:

    virtual void render_scene(const World& w) = 0;

    void compute_uvw(void);

protected:

    Point3D      eye;           // eye point
    Point3D      lookat;       // lookat point
    Vector3D     u, v, w;      // orthonormal basis vectors
    Vector3D     up;           // up vector
    float        exposure_time;

};
```

# Il metodo Camera::compute\_uvw

```
void Camera::compute_uvw(void) {  
  
    w = eye - lookat;  
    w.normalize();  
    u = up ^ w;  
    u.normalize();  
    v = w ^ u;  
  
}
```

---

# La sottoclasse Camera:pinhole

```
class Pinhole: public Camera {
public:

    Vector3D get_direction(const Point2D& p) const;

    virtual void render_scene(const World& w);

private:

    float d;    // view plane distance
    float zoom; // zoom factor

};
```

```
Vector3D Pinhole::get_direction(const Point2D& p) const {

    Vector3D dir = p.x * u + p.y * v - d * w;
    dir.normalize();

    return(dir);
}
```

# Il metodo Pinhole::render\_scene

```
void Pinhole::render_scene(const World& w) {
    RGBColor          L;
    ViewPlane         vp(w.vp);
    Ray ray;
    int depth = 0;
    Point2D pp; // sample point on a pixel
    int n = (int)sqrt((float)vp.num_samples);

    vp.s /= zoom;
    ray.o = eye;

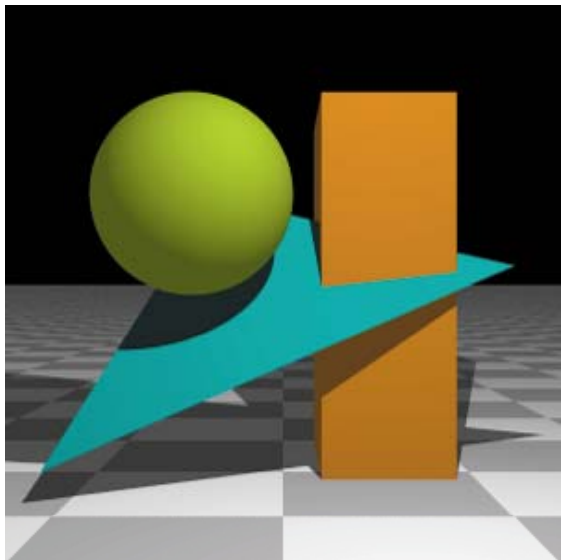
    for (int r = 0; r < vp.vres; r++) // up
        for (int c = 0; c < vp.hres; c++) { // across
            L = black;
            for (int j = 0; j < vp.num_samples; j++) { // across pixel
                sp = vp.sampler_ptr->sample_unit_square();
                pp.x = vp.s * (c - 0.5 * vp.hres + sp.x);
                pp.y = vp.s * (r - 0.5 * vp.vres + sp.y);
                ray.d = get_direction(pp);
                L += w.tracer_ptr->trace_ray(ray, depth);
            }
            L /= vp.num_samples;
            L *= exposure_time;
            w.display_pixel(r, c, L);
        }
}
```

# Il metodo World::build

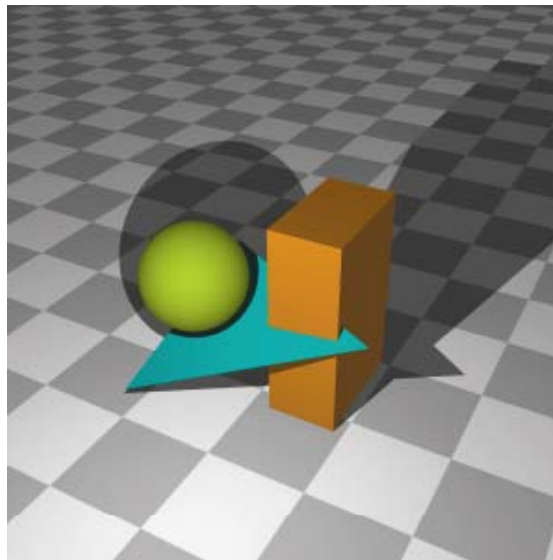
```
void World::build(void) {  
    ...  
    Pinhole* pinhole_ptr = new Pinhole();  
  
    pinhole_ptr->set_eye(300, 400, 500);  
    pinhole_ptr->set_lookat(0, 0, -50);  
    pinhole_ptr->set_viewdistance(400);  
    pinhole_ptr->compute_uvw();  
  
    set_camera(pinhole_ptr);  
};
```

---

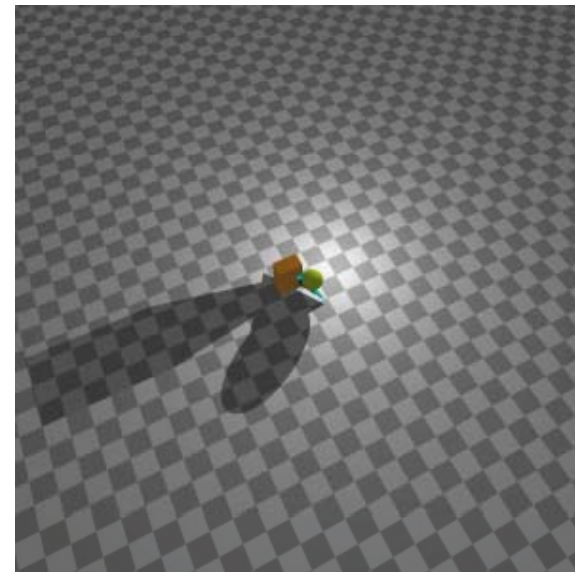
# Esempi



$$\begin{aligned}e &= (0,0,50) \\l &= (0,0,0) \\d &= 500\end{aligned}$$



$$\begin{aligned}e &= (300,400,50) \\l &= (0,0,-50) \\d &= 400\end{aligned}$$



$$\begin{aligned}e &= (-1000,2000,500) \\l &= (0,-100,0) \\d &= 250\end{aligned}$$