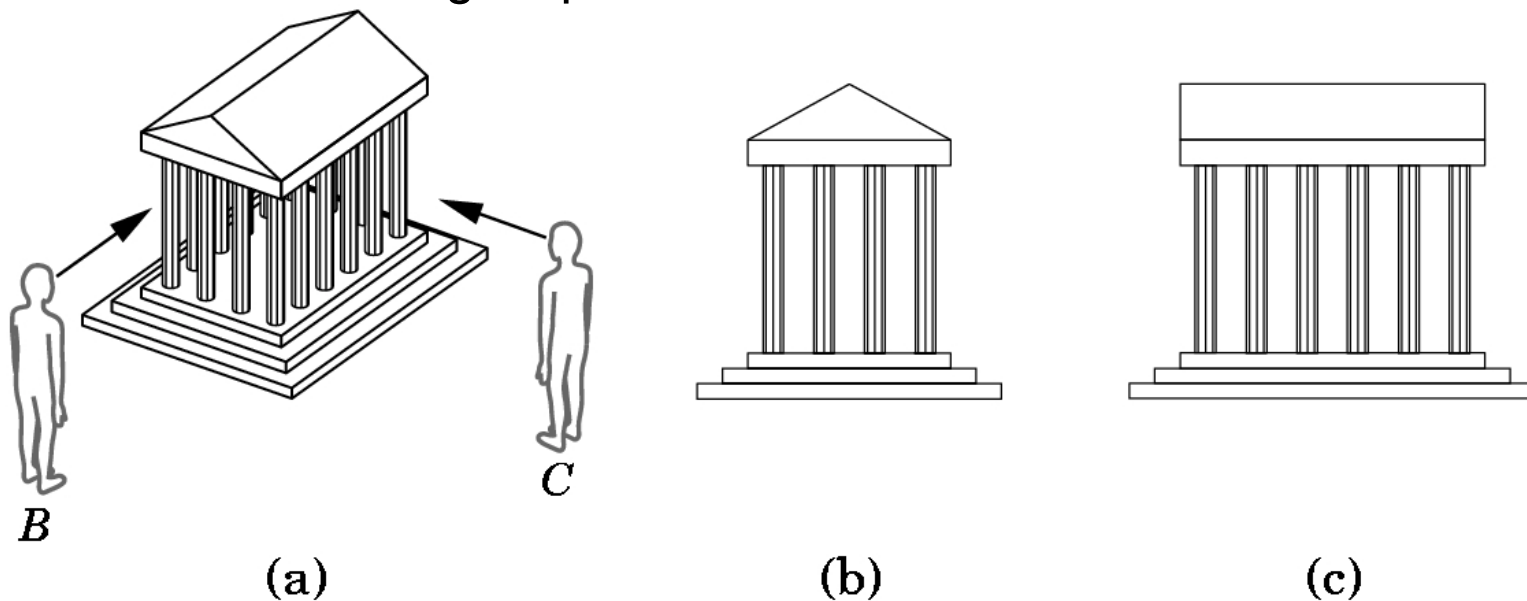


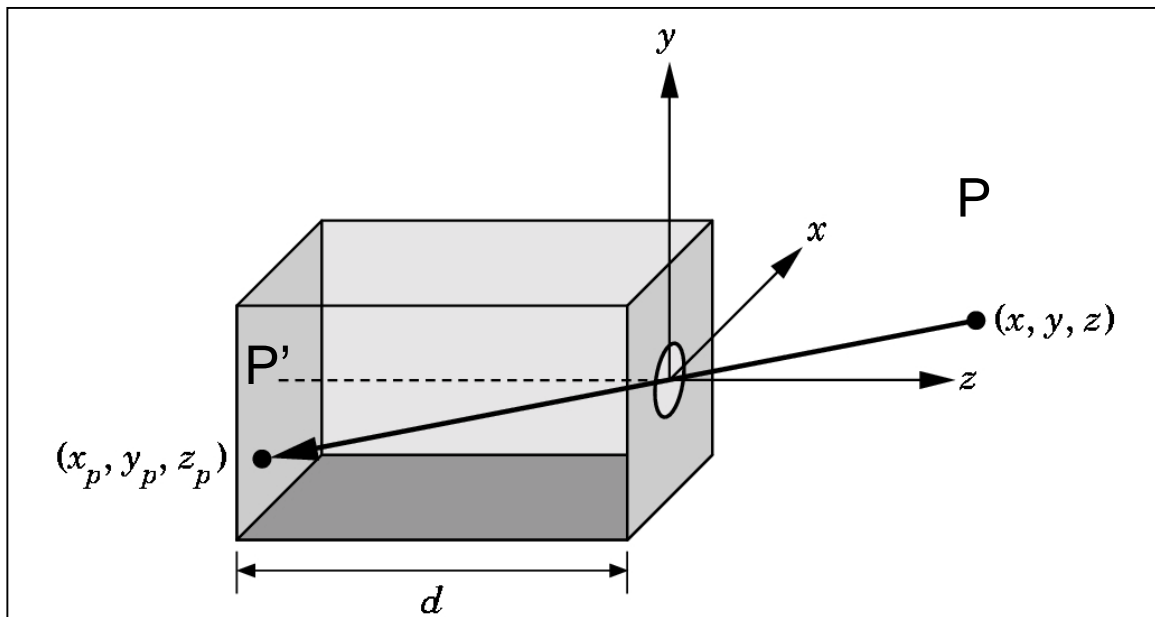
OpenGL: visualizzazione 3D

La visualizzazione di una scena avviene come se si stesse usando una macchina fotografica per la quale si può controllare la posizione nello spazio 3D; si può cambiare il tipo di lente dell'obiettivo e, in fase di sviluppo della fotografia, decidere le dimensioni dell'immagine prodotta.



Pinhole camera model

Consiste in un box con una pellicola fotografica da un lato e un foro dall'altro lato. Idealmente si pensi che dal foro possa entrare solamente un raggio luminoso emesso dall'oggetto esterno $P=(x,y,z)$.



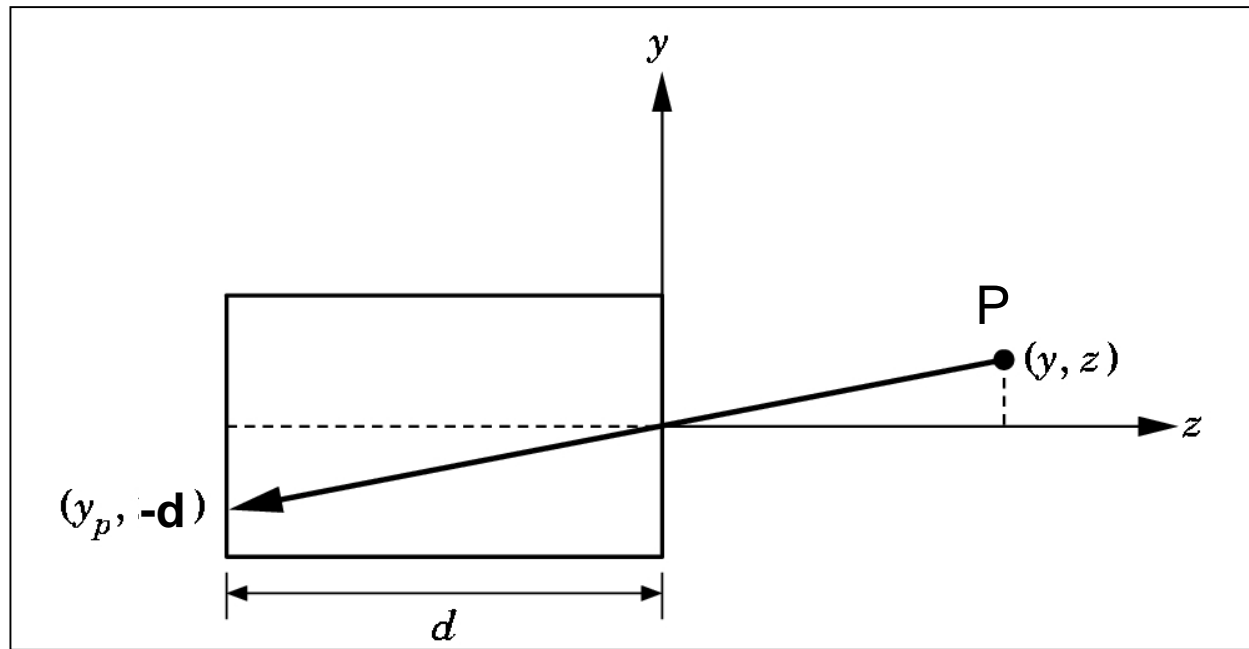
$$y_p = \frac{-y}{z/d}$$

$$x_p = \frac{-x}{z/d}$$

$$z_p = -d$$

Pinhole camera model (2)

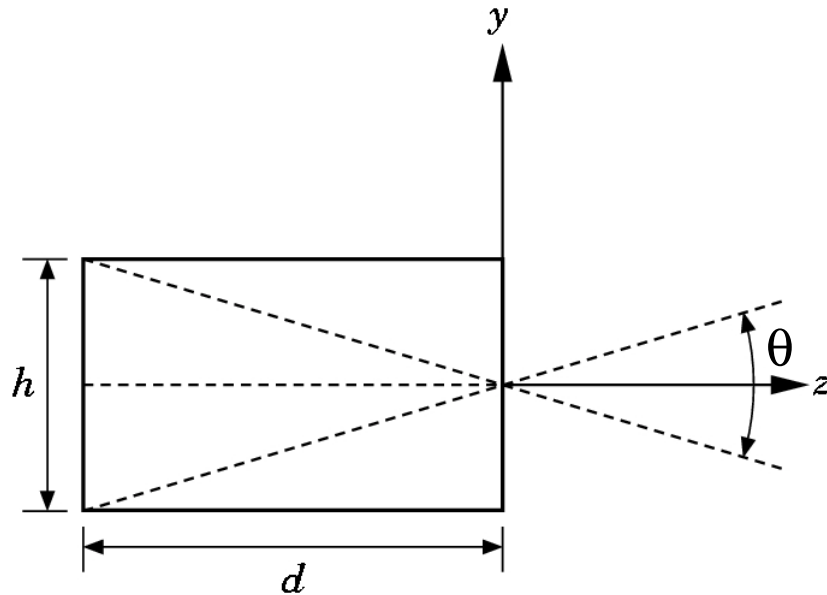
Si consideri ora la sezione trasversale della camera.



Il punto $P'=(x_p, y_p, -d)$ si chiama PROIEZIONE di P sul piano a distanza d .

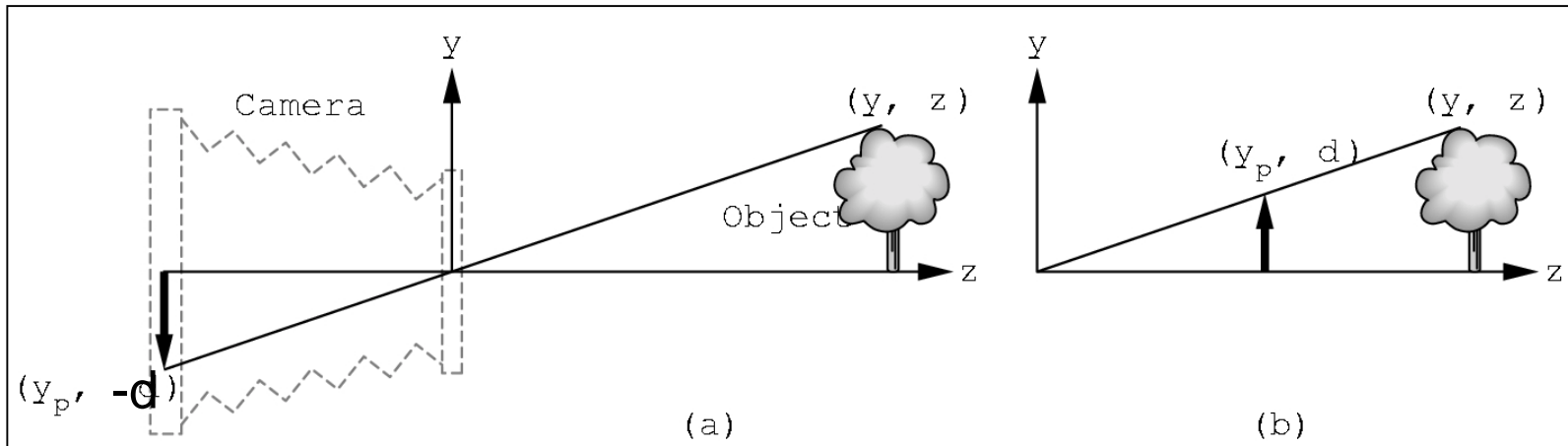
Pinhole camera model (3)

θ si chiama **ANGOLO di vista** della camera (fovy). Considerando un piano di proiezione di altezza h e distante d dall'origine del piano (y,z) si ha:



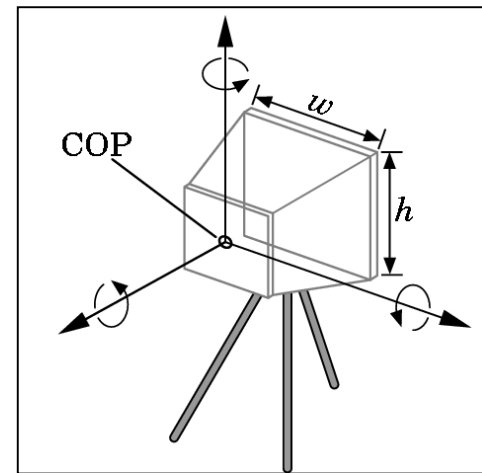
$$\theta = 2 \tan^{-1} \frac{h}{2d}$$

Pinhole camera model (4)

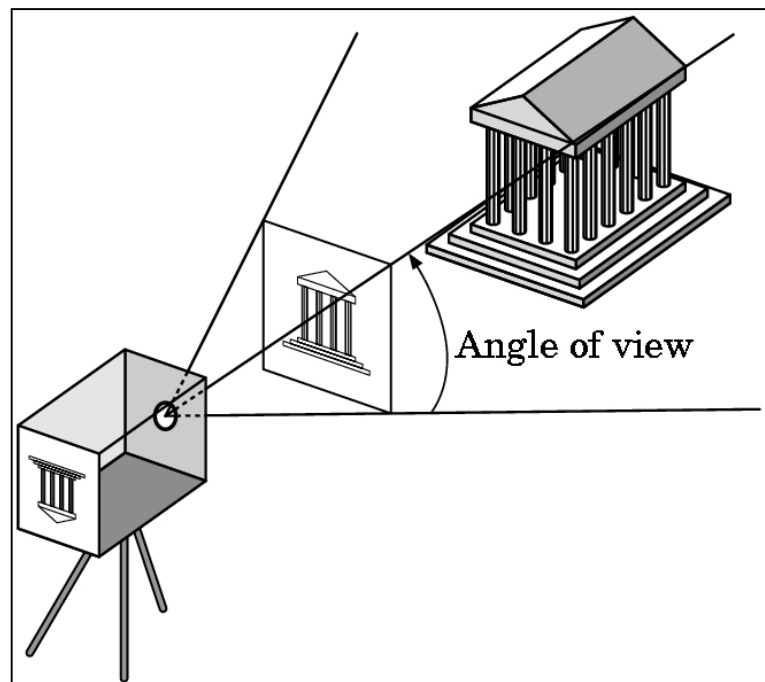
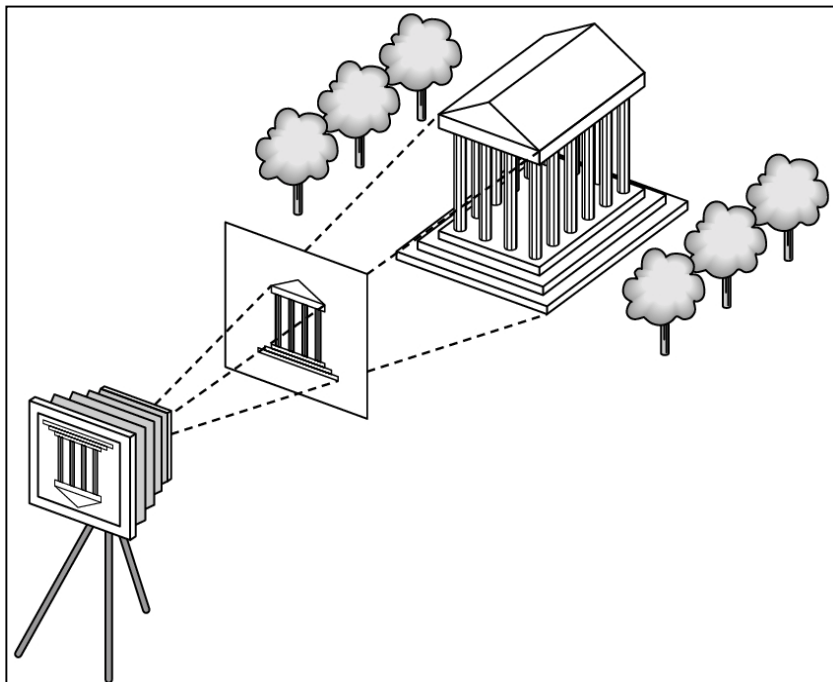


Si supponga quindi di portare il piano di proiezione davanti rispetto alla lente della camera.

Il centro della “lente” si chiama COP (Center of Projection).



Pinhole camera model (5)

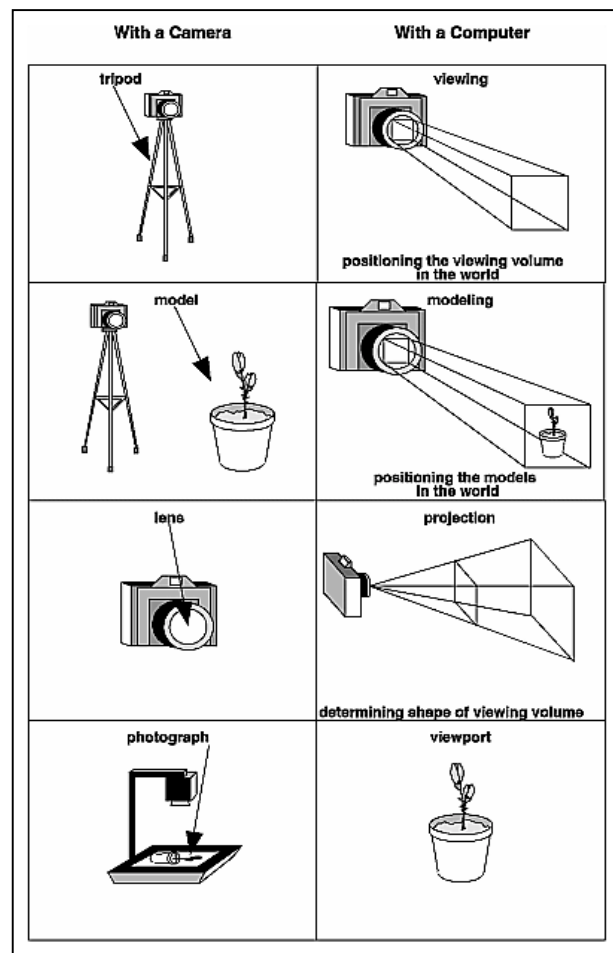


Proiezione prospettica sul piano viewport di visualizzazione.

OpenGL viewing transformation (2)

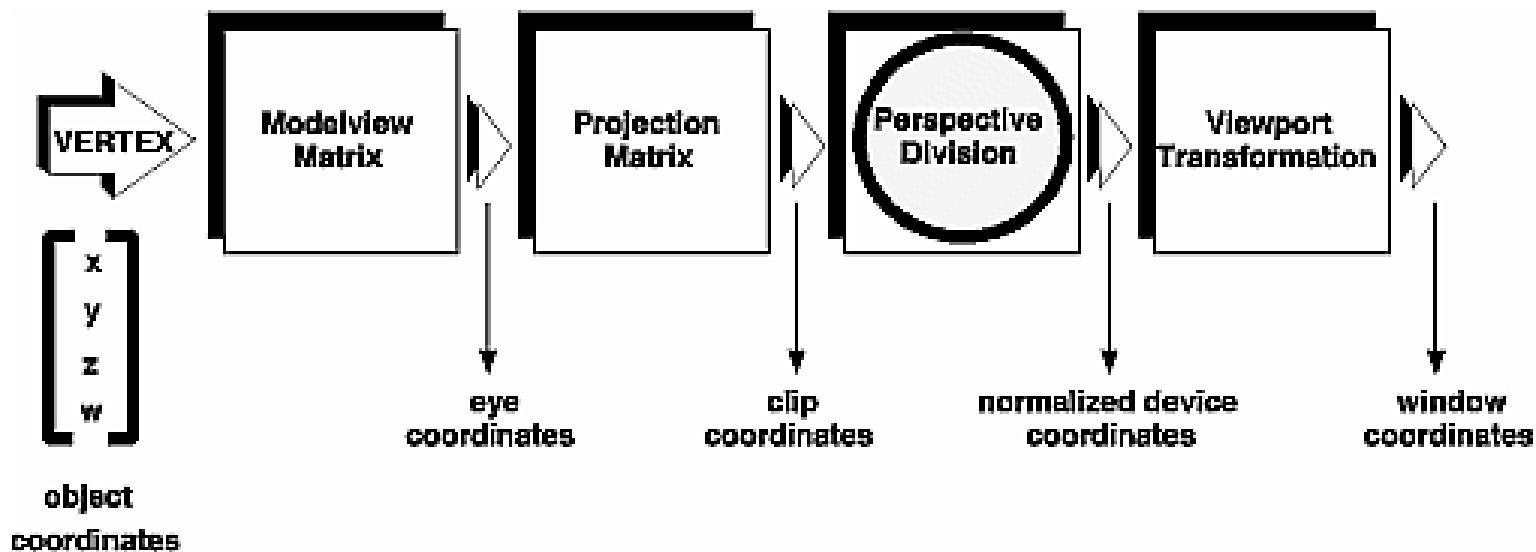
Quindi le fasi per riuscire a produrre l'immagine desiderata sono:

1. Posizionamento della camera nello spazio 3D (`gluLookat`)(`MODEL_VIEW`).
2. Modellare la scena (display degli oggetti) (`MODEL_VIEW`).
3. Scelta del tipo di proiezione e della lente (`PROJECTION`).
4. Dimensionamento dell'immagine finale (`glViewport`)(`MODEL_VIEW`).



OpenGL viewing transformation (3)

In OpenGL tutte le trasformazioni necessarie a produrre la scena desiderata avvengono tramite l'applicazione di matrici agli oggetti che compongono la scena.



OpenGL viewing transformation (4)

Le principali matrici sono: **MODEL_VIEW MATRIX** che deve essere abilitata nel momento in cui si agisce sulla scena (oggetti, punto di vista (posizionamento camera), VIEWPORT, colori, luci, rotazioni...), e **PROJECTION_MATRIX** per definire i tipi di proiezione.

Per abilitare l'uno o l'altro stack di matrici si chiama

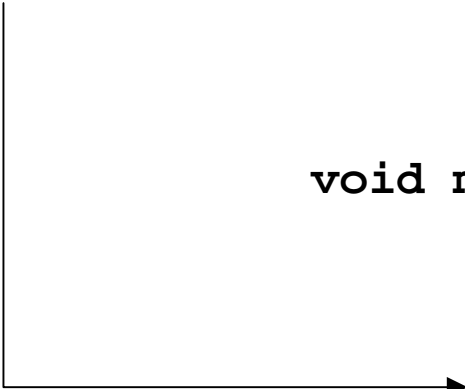
`glMatrixMode(GL_MODELVIEW)` oppure
`glMatrixMode(GL_PROJECTION)`.

In fase di `INITGL()` nel `main.cpp` è quindi necessario specificare la posizione della camera, tipo di proiezione, eventuale presenza di luci, ...

Esempio:

Impostazioni iniziali della scena

```
void main(int argc, char *argv[]){  
    glutInit(&argc, argv);  
    InitGLUT();  
    InitGL();  
    glutMainLoop();  
    exit(0);  
}
```



Esempio (2):

```
void InitGL(){
glClearColor( .0f, .0f, .5f, 1.0f ); //Colore di sfondo
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glMatrixMode(GL_PROJECTION); ←
glLoadIdentity();
    glOrtho(-1.0,1.0,-1.0,1.0,1.0,2.0);
glMatrixMode(GL_MODELVIEW); ←
glLoadIdentity();
::::::::::::::::::::::::::::::::::::
::::::::::::::::::::::::::::::::::::
::::::::::::::::::::::::::::::::::::
}
```

Attivo
PROJECTION_MATRIX
sto specificando il
tipo di proiezione
quindi il sistema di
riferimento

Attivo **MODELVIEW**
da qui in poi posso
specificare la
struttura della mia
scena

OpenGL: Modelview & Projection

In OpenGL dopo la chiamata di `glMatrixMode(GL_MODELVIEW)` è possibile chiamare tutte le funzioni che:

1. disegnano la mia scena;
2. posizionano i miei oggetti nello spazio;
3. posizionano il punto di vista (`glLookAt(...)`);
4. effettuare trasformazioni di rotazione, traslazione e scala;
5. altro che riguarda la composizione della scena.

Con l'abilitazione di `GL_PROJECTION` vado a modificare la struttura di quello che si chiama il **volume di vista**, cioè quello spazio all'interno del quale i miei oggetti saranno visualizzati e proiettati sulla mia viewport secondo la particolare proiezione definita.

Modelview & Projection

Consideriamo parte dell'esempio precedente:

```
glMatrixMode(GL_PROJECTION);  
glLoadIdentity();  
glOrtho(-1.0,1.0,-1.0,1.0,1.0,2.0);
```

La `glOrtho` permette di definire le dimensioni del volume di vista e quindi lo spazio tridimensionale all'interno del quale saranno collocati gli oggetti della scena.

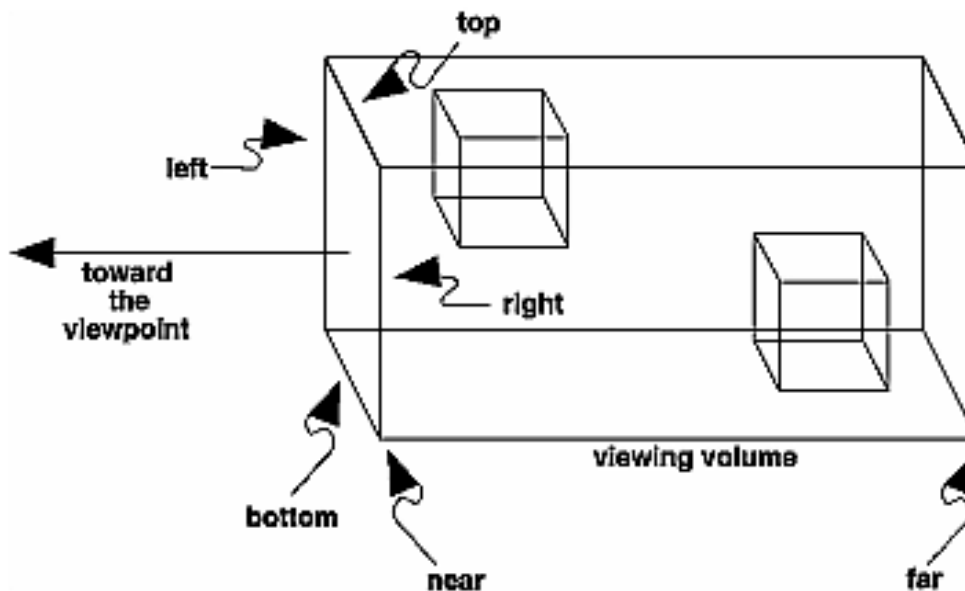
In particolare:

```
void glOrtho(left, right, bottom, top, near, far)
```

definisce una proiezione ortografica parallela al volume di viewing.

Modelview & Projection

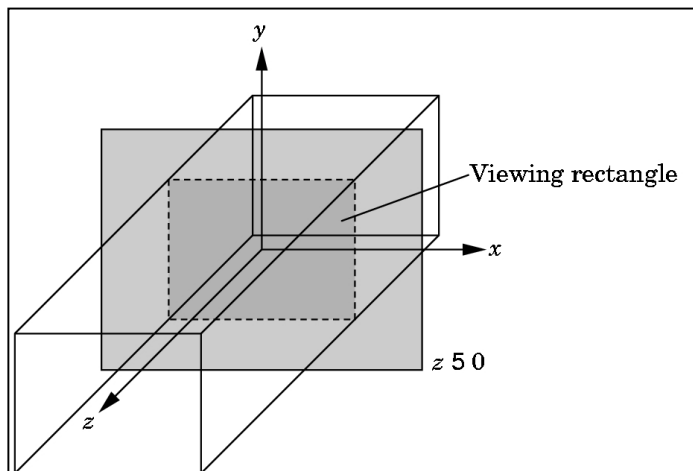
Quindi specificare le dimensioni di un sistema di coordinate significa definire un volume di vista nel quale costruire la scena. Tutti gli oggetti con coordinate esterne a tale volume, saranno tagliati (clipping).



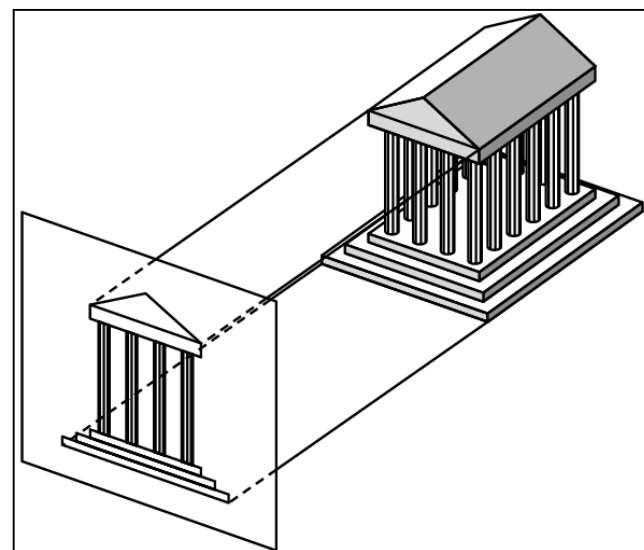
Volume di vista
definito dalla
`glOrtho(...)`.

Proiezione Prospettica

La proiezione prospettica è quella più utilizzata e realistica, mentre la proiezione parallela (glOrtho...) viene principalmente utilizzata per la visualizzazione di architetture edili (CAD architettonici). Con la glOrtho il volume di vista è definito come un parallelepipedo, nella proiezione prospettica invece abbiamo che il volume di vista è un tronco di piramide.



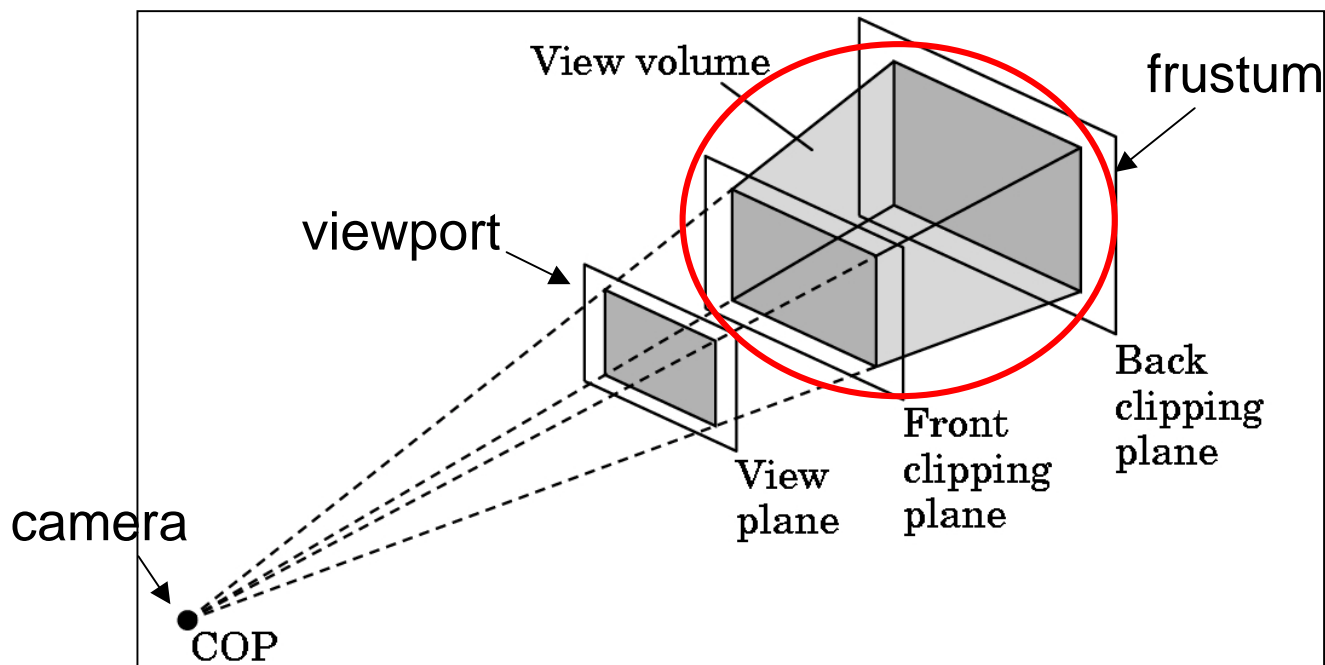
Proiezione ortografica parallela glOrtho



Proiezione Prospettica (2)

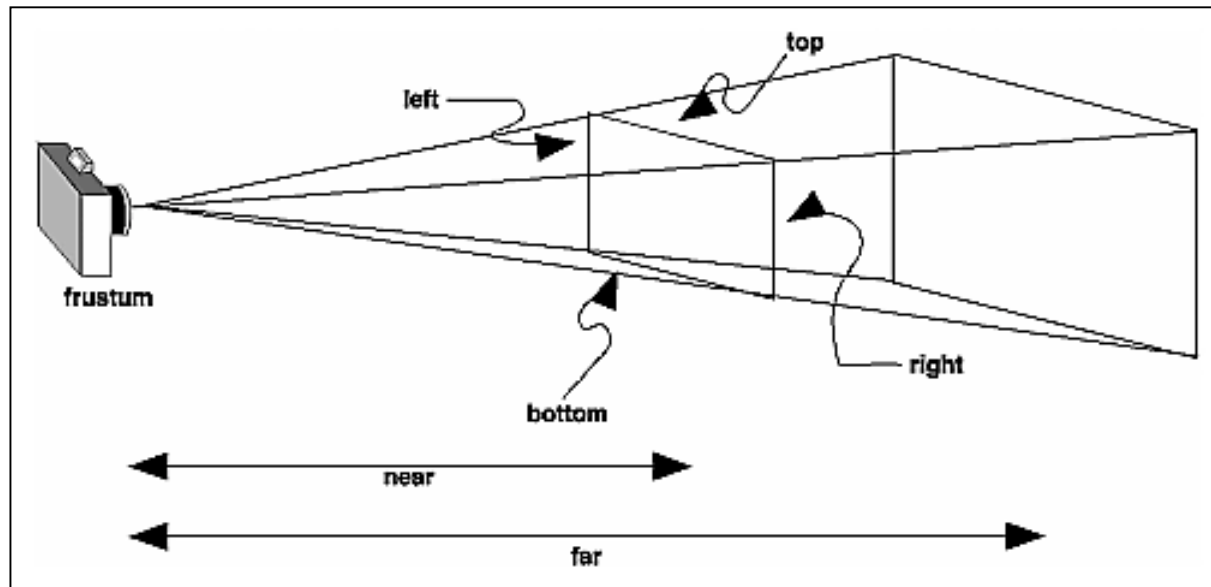
```
void glFrustum(left, right, bottom, top, near, far)
```

Definisce la proiezione prospettica di un oggetto sul piano di vista.



Proiezione Prospettica (3)

Definizione del frustum:



left = xmin
right = xmax

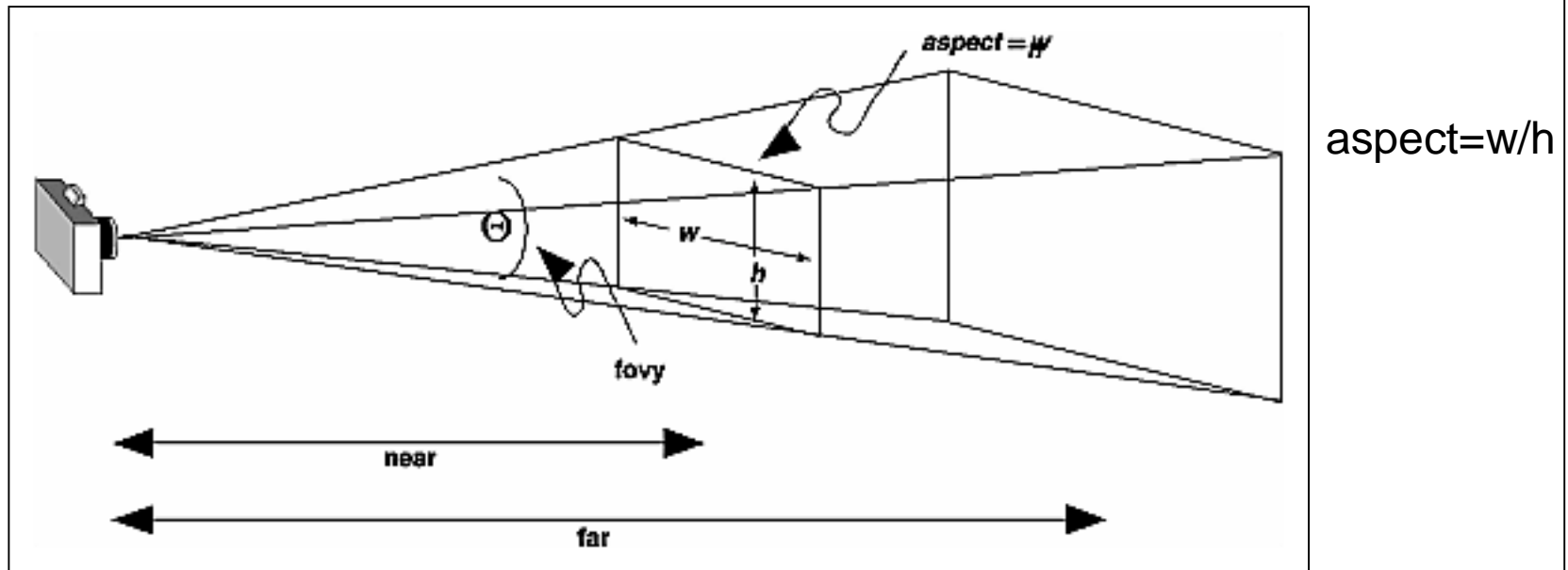
bottom=ymin
top = ymax

near < z < far

Proiezione Prospettica (4)

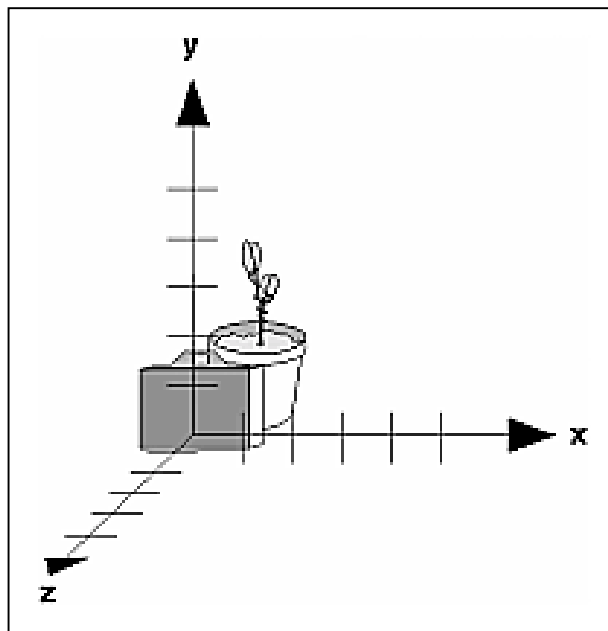
Un altro modo di definire il volume di visualizzazione prospettica è quello tramite l'uso di:

```
glPerspective(fovy, aspect, near, far);
```



Posizionamento Punto di Vista

Una volta definita la proiezione che permette di creare l'immagine sulla viewport, è possibile modificare la posizione del punto di vista. Per default, il punto di vista è posizionato nell'origine del sistema 3D definito dalla `glOrtho` o dal `glFrustum`, e punta nella direzione della `z` in senso negativo.



Posizionamento punto di Vista (2)

Il cambiamento di un punto di vista è particolarmente utile quanto si vuole dare l'impressione di muoversi all'interno di una scena 3D.

```
void gluLookat(eyex, eyey, eyez, atx, aty, atz,  
              upx, upy, upz);
```

