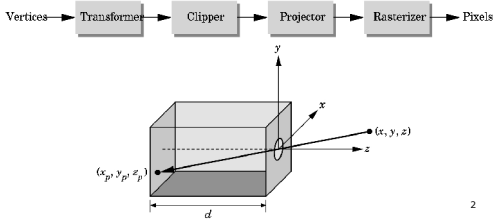
Costruzione di Interfacce
Lezione 4
Trasformazioni Affini

cignoni@isti.cnr.it
http://vcg.isti.cnr.it/~cignoni

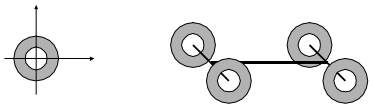
Cambi di Sistemi di riferimento

- Il primo step della pipeline di rendering è quello di trasformare la scena nel sistema di riferimento della camera



Object Frame

- Perché ogni oggetto ha il suo sistema di riferimento?
  - Uso Multiplo di uno stesso oggetto
  - Posizione parametrica



Coordinate Omogenee

- Si dice che un punto *P* è rappresentato dalla matrice colonna **p**

$$\mathbf{p} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ 1 \end{bmatrix}$$

- E un vettore *w* è rappresentato dalla matrice colonna **a**

$$\mathbf{a} = \begin{bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ 0 \end{bmatrix}$$

Trasformazioni Affini

- Notare che se u è un vettore solo 9 elementi di A sono usati nella trasformazione

$$\mathbf{Au} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ 0 \end{bmatrix}$$

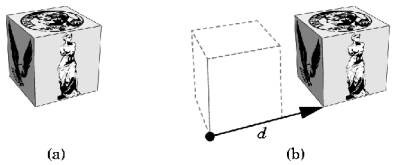
- La quarta colonna corrisponde alla quarta riga della matrice di cambiamento di frame, che conteneva il nuovo punto di origine del frame (che chiaramente non serve se si parla di vettori)

Traslazione

- modifica i punti di un frame sommando a tutti i punti un vettore di spostamento *d*

$$P' = P + d$$

$$\mathbf{p}' = \mathbf{p} + \mathbf{d}$$



Traslazione

$$\mathbf{p}' = \mathbf{p} + \mathbf{d} \qquad \mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \mathbf{p}' = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} \quad d = \begin{bmatrix} \alpha_x \\ \alpha_y \\ \alpha_z \\ 0 \end{bmatrix}$$

$$\mathbf{p}' = \mathbf{T}\mathbf{p} \qquad \mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & \alpha_x \\ 0 & 1 & 0 & \alpha_y \\ 0 & 0 & 1 & \alpha_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

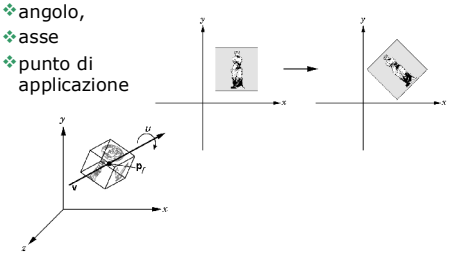
Traslazione

$$\mathbf{T}(\alpha_x, \alpha_y, \alpha_z) = \begin{bmatrix} 1 & 0 & 0 & \alpha_x \\ 0 & 1 & 0 & \alpha_y \\ 0 & 0 & 1 & \alpha_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{T}^{-1}(\alpha_x, \alpha_y, \alpha_z) = \mathbf{T}(-\alpha_x, -\alpha_y, -\alpha_z) \begin{bmatrix} 1 & 0 & 0 & -\alpha_x \\ 0 & 1 & 0 & -\alpha_y \\ 0 & 0 & 1 & -\alpha_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotazione

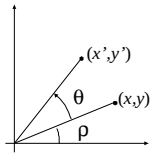
- Di una rotazione si deve specificare
  - angolo,
  - asse
  - punto di applicazione



Rotazione

- Caso semplice asse z, intorno all'origine, di un angolo θ
- Possiamo considerare il problema in 2d

$$x = \rho\cos\phi$$
$$y = \rho\sin\phi$$
$$x' = \rho\cos(\phi + \theta)$$
$$y' = \rho\sin(\phi + \theta)$$



Rotazione

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \mathbf{R}(\theta)_z \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \\ z \\ 1 \end{bmatrix}$$

$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotazione inversa

- Le matrici di rotazione viste finora sono invertibili facilmente.

$$\mathbf{R}^{-1}(\theta) = \mathbf{R}\ (-\theta)$$
$$\sin(-\theta) = -\sin(\theta)$$
$$\cos(-\theta) = \cos(\theta)$$

- Quindi basta trasporre…

$$\mathbf{R}^{-1}(\theta) = \mathbf{R}^T(\theta)$$

Rotazioni

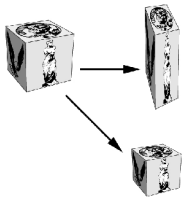
- Finora abbiamo visto solo rotazioni intorno all'origine e lungo gli assi.
- Una rotazione arbitraria si ottiene componendo piu' rotazioni e traslazioni

Scalatura

- Non rigida
- Non uniforme lungo gli assi
- Solo centrata all'origine

$$x' = \beta_x x$$
$$y' = \beta_y y$$
$$z' = \beta_z z$$

$$\mathbf{T}(\beta_x, \beta_y, \beta_z) = \begin{bmatrix} \beta_x & 0 & 0 & 0 \\ 0 & \beta_y & 0 & 0 \\ 0 & 0 & \beta_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Composizione di Trasformazioni

- Le trasformazioni sono matrici
- L'applicazione di trasformazione è la moltiplicazione di una matrice per un vettore.
- L'applicazione di una sequenza di trasformazioni ad un punto corrisponde ad una sequenza di moltiplicazioni di di matrici per vettori
- Associatività

$$\mathbf{q} = \mathbf{CBAp}$$
$$\mathbf{q} = (\mathbf{C}(\mathbf{B}(\mathbf{Ap})))$$
$$\mathbf{q} = (\mathbf{CBA})\mathbf{p}$$

Composizione di trasformazioni

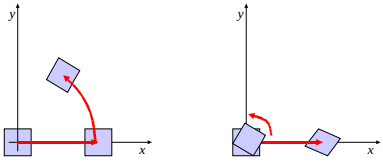
- Conviene se si deve trasformare un solo punto $\mathbf{q} = (\mathbf{C}(\mathbf{B}(\mathbf{Ap})))$
- Conviene se si deve trasformare molti punti $\mathbf{q} = (\mathbf{CBA})\mathbf{p}$

Composizione di Trasformazioni

- La moltiplicazione tra matrici NON è commutativa
- Quindi l'ordine delle trasformazioni È importante

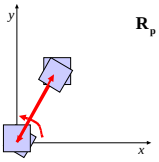
Composizione di Trasformazioni

$\mathbf{q} = \mathbf{RTp}$ $\quad$ $\mathbf{q} = \mathbf{TRp}$



Rotazione intorno ad un punto

- Si ottiene spostando tutto il sistema di riferimento nel punto, facendo la rotazione e rimettendo tutto a posto



$$\mathbf{R}_p = \mathbf{T(p)R}_0\mathbf{T(-p)}$$

Rotazione intorno ad un asse

- L'idea e' quella di fare :
  - due rotazioni che portino l'asse di rotazione specificato a coincidere con l'asse z,
  - la rotazione lungo quell'asse di quanto specificato
  - rimettere tutto a posto

$$\mathbf{R} = \mathbf{R}_x(-\theta_x)\mathbf{R}_y(-\theta_y)\ \mathbf{R}_z(\theta_z)\ \mathbf{R}_y(\theta_y)\mathbf{R}_x(\theta_x)$$

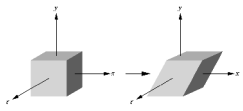
- trovare θ_y θ_x non è semplice…

Shearing

- Sarebbe derivabile dalle altre…
- Lo spostamento e' proporzionale alla coord y;

$$x' = x + y\cot\theta$$
$$y' = y$$
$$z' = z$$

$$H_{xy}(\theta) = \begin{bmatrix} 1 & \cot\theta & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



OpenGL

- Nello Stato di OpenGL ci sono tre matrici 4x4 di trasformazioni
  - ModelView
  - Project
  - Texture
- Una di queste e' sempre la matrice corrente
- Tutte i comandi che modificano matrici fanno sempre riferimento alla matrice corrente

Opengl

- Per cambiare matrice Corrente
- glMatrixMode(***)
  - GL_MODELVIEW
  - GL_PROJECTION
  - GL_TEXTURE
- Per rimpiazzare la matrice
  - glLoadIdentity()
  - glLoadMatrix(Pointer to a matrix);
- Tutti gli altri comandi modificano (moltiplicano per un altra matrice) la matrice corrente.

OpenGl

- Leggere una matrice
- glGetDoublev (GL_MODELVIEW_MATRIX,pointer_to_matrix)
- Nota: nello scambio diretto di matrici OpenGL assume che siano memorizzate in column major order (e.g. per colonne)

$$\begin{bmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{bmatrix}$$

OpenGL

in gradi!

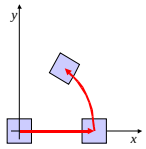
- Rotazioni
  - glRotatef(angle, ax,ay,az);
- Traslazioni
  - glTranslatef(dx,dy,dz);
- Scalature
  - glScalef(sx,sy,sz)
- Generica
  - glMultMatrix(matrix_pointer)

Composizione di Matrici in Opengl

- I comandi opengl postmoltiplicano la matrice corrente per quella specificata

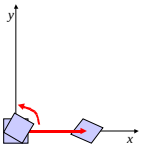
$\mathbf{q} = \mathbf{RTp}$ (Stato della matrice corrente dopo il comando a sinistra)

	I
glRotate	R
glTranslate	RT



$\mathbf{q} = \mathbf{TRp}$

	I
glTranslate	T
glRotate	TR



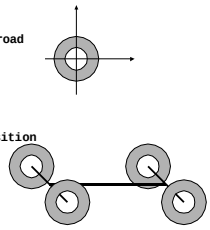
Stack di matrici

- La matrice corrente puo' essere salvata in uno stack e recuperata successivamente
- glPushMatrix()
- glPopMatrix()

Stack di Matrici

- Il Matrix Stack e' particolarmente utile quando si disegnano scene strutturate

```
// Whole car transformation
// eg position of the car on the road
glTranslate(CarPos)
glRotate(CarDirection,0,1,0)
For(i=0;i<4;++i)
    {
      glPushMatrix();
      // relative single wheel position
      glTranslate(wheelpos[i]);
      DrawWheel(i);
      glPopMatrix();
    }
```

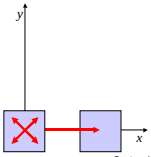


Composizione di Matrici in Opengl

- I comandi opengl postmoltiplicano la matrice corrente per quella specificata

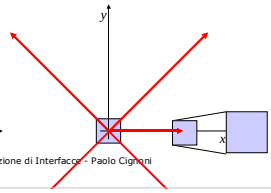
q=TSp (Stato della matrice corrente dopo il comando a sinistra)

	I
glTranslate	T
glScale	TS



q=STp

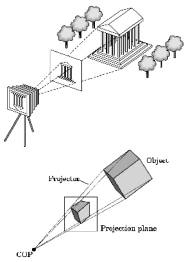
	I
glScale	S
glTranslate	ST



Trasformazioni

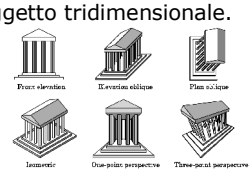
- Due trasformazioni vengono applicate ai vertici della geometria che voglio disegnare
  - Trasformazione di Modellazione
    - Porta la geometria nel sistema di riferimento standard della camera
  - Proiezione di vista
    - Proietta la geometria dal sistema di riferimento della camera sul piano di proiezione.

Elementi di base



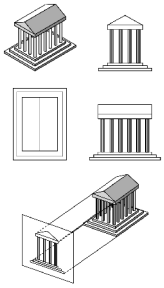
Prospettiva Classica

- Il problema è quello classico, da sempre affrontato, nell'arte, in architettura e della progettazione, di riuscire a riportare su di un piano, in maniera rigorosa un oggetto tridimensionale.



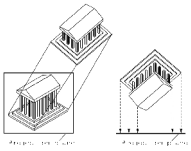
Proiezioni ortografiche classiche

- L'oggetto ha una struttura regolare, ben assimilabile ad un parallelepipedo;
- Proietto sui lati del box.
  - Centro di proiezione, all'infinito,
  - proiettori paralleli e perpendicolari al piano di proiezione
  - Piano di proiezione perpendicolare agli assi del sistema di riferimento



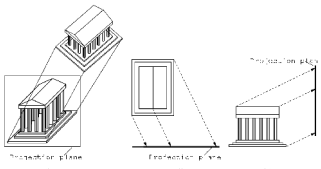
Proiezioni Assonometriche

- Piani di proiezione in posizione non vincolata:
  - Centro di proiezione, all'infinito,
  - proiettori paralleli e perpendicolari al piano di proiezione
  - Piano di proiezione:
    - Simmetrico ai 3 assi (isometrica)
    - Simmetrico rispetto a 2 assi (dimetrico)
    - Posizione qualsiasi (trimetrica)

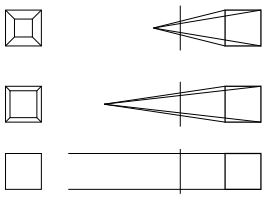


Proiezioni Oblique

- Proiettori paralleli ma non perpendicolari al piano di proiezione



Ortografica



Proiezioni prospettiche

- Caratteristica principale:
  - diminuzione della dimensione apparente degli oggetti all'aumentare della distanza dall COP.
  - I proiettori passano tutti per il COP
- Caso Classico
  - Il centro di proiezione simmetrico rispetto alla finestra nel piano di proiezione
- Caso generico,
  - COP svincolato dal piano di proiezione

Prospettiva classica

- Al solito nella visione tradizionale del disegno tecnico si è soliti distinguere vari tipi di proiezione prospettica, indicando il *numero di punti all'infinito* (vanishing points);
  - a) three vanishing points
  - b) two vanishing points
  - c) one vanishing points

(a) (b) (c)

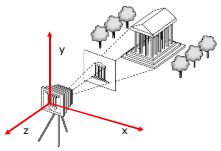
Prospettiva Classica

- In realtà questi, come nel caso ortogonali sono solo vincoli sul piazzamento del piano di proiezione
  - One point Persp: Piano di proiezione parallelo ad uno dei piani del sistema di riferimento
  - Two point Persp: Piano di proiezione perpendicolare ad uno dei piani del sistema di riferimento
  - Three Point Persp: Piano di proiezione Libero

(a) (b) (c)

Camera Frame

- Si assume che la camera sia piazzata con il centro di proiezione sull'origine e diretta verso l'asse z negativo.
- È compito della matrice di Modellazione portare gli oggetti in questo frame.
- Questo sistema di riferimento è detto camera frame, o eye frame



Sistemi di coordinate in OpenGL

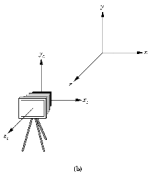
- Object:
  - la ruota con l'origine nel centro.
- World:
  - la ruota piazzata nel sistema di riferimento del mondo (e.g. quello classico con y = alto del mondo, ecc,).
  - Cambia quando si muove la macchina
- Eye:
  - Il sistema di riferimento in cui l'origine e' il centro di proiezione, la z è la direzione di vista ecc.
  - Cambia quando muovo l'osservatore.
- Clip
- Normalized device
- Window

Sistema di riferimento della Camera

- Come si specifica il sistema di riferimento della camera?
  - Matrice di trasformazione che fa parte delle matrici di modellazione
  - Passa dalle coordinate di mondo alle coordinate di occhio

Piazzare la camera

- Caso semplice,
- Voglio piazzare la camera in modo che inquadri oggetti centrati sull'origine (del sistema di riferimento mondo), guardando lungo la z negativa (come di default)
- E` solo una traslazione sull'asse z:



$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & ? \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Piazzare la camera

- Caso Generale
- Definire una camera significa definire una proiezione (prospettica)
  - La camera (il centro di proiezione) è centrata in un punto detto VRP (view Reference Point)
  - Il Piano di proiezione è perpendicolare ad un vettore VPN (view plane normal)
- Del piano e' necessario specificare anche l'orientamento, quindi
  - Si specifica VUP (view up vector)

Piazzare la camera

- La trasformazione non e' altro che un cambio di sistemi di riferimento.
- Basta definire gli assi e l'origine del un sistema di riferimento in termini dell'altro.
- Noi abbiamo la posizione della camera nel sistema di riferimento world
- Dobbiamo ancora esprimere gli assi del sistema di riferimento camera in sr World.

Camera axis in world space

- Uno alla volta.
- L'asse z della camera e' semplicemnte la direzione di proiezione, cioe' la normale al piano di proiezione
- L'asse x della camera deve essere perpendicolare al vettore up e all'asse z appena trovato

$$x = up \times z$$

- L'asse y e' semplicemente il prodotto vettore tra gli altri due assi che ho trovato (attenti al segno)

$$y = z \times x$$

Cambio di Frame

- Dati due sistemi di riferimento.

$$\{v_1, v_2, v_3, P_0\} \quad \{u_1, u_2, u_3, Q_0\}$$

- Esprimiamo uno in termini dell'altro:

$$\begin{aligned} u_1 &= \gamma_{11}v_1 + \gamma_{12}v_2 + \gamma_{13}v_3 \\ u_2 &= \gamma_{21}v_1 + \gamma_{22}v_2 + \gamma_{23}v_3 \\ u_3 &= \gamma_{31}v_1 + \gamma_{32}v_2 + \gamma_{33}v_3 \\ Q_0 &= \gamma_{41}v_1 + \gamma_{42}v_2 + \gamma_{43}v_3 + P_0 \end{aligned}$$

- Questo definisce la matrice 4x4 di cambiamento di frame

$$\mathbf{M} = \begin{bmatrix} \gamma_{11} & \gamma_{12} & \gamma_{13} & 0 \\ \gamma_{21} & \gamma_{22} & \gamma_{23} & 0 \\ \gamma_{31} & \gamma_{32} & \gamma_{33} & 0 \\ \gamma_{41} & \gamma_{42} & \gamma_{43} & 1 \end{bmatrix}$$

Cambio di Frame

- La matrice di cambiamento di frame

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ Q_0 \end{bmatrix} = \mathbf{M} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix}$$

- Date le due rappresentazioni **a,b** in coordinate omogenee in differenti frame (sia di un vettore che di un punto), vale:

$$\mathbf{b}^T \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ Q_0 \end{bmatrix} = \mathbf{b}^T M \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix} = \mathbf{a}^T \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix} \Rightarrow \mathbf{a} = \mathbf{M}^T \mathbf{b}$$

- Noi abbiamo
- Camera e suoi parametri in coordinate di mondo.
- Vogliamo trovare la trasf **M** che prenda la rappresentazione di un punto p_w in coordinate omogenee spazio di mondo e ne dia la sua rappresentazione in coordinate di camera p_e.

$$\mathbf{p}_e = \mathbf{M}\mathbf{p}_w$$

Eye to World

- Esprimo il sistema di riferimento della camera in termini dello spazio di mondo

$$\mathbf{p}_w = \mathbf{M}\mathbf{p}_e \qquad \{u,v,n,P_0\} \quad \{x,y,z,0\}$$

$$\begin{aligned} u &= u_x x + u_y y + u_z z \\ v &= v_x x + v_y y + v_z z \\ n &= n_x x + n_y y + n_z z \\ P_0 &= p_x x + p_y y + p_z z + 0 \end{aligned} \qquad \mathbf{M} = \begin{bmatrix} u_x & v_x & n_x & p_x \\ u_y & v_y & n_y & p_y \\ u_z & v_z & n_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

World to eye

- A noi serve la matrice inversa:
  - da world a eye
- Per invertirla si nota che la possiamo scrivere come un prodotto tra Rot e Tras

$$\mathbf{M} = \begin{bmatrix} u_x & v_x & n_x & p_x \\ u_y & v_y & n_y & p_y \\ u_z & v_z & n_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \mathbf{TR} = \begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_x & v_y & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

World to Eye

- Invertire Traslazioni e rotazioni è facile…

$$\mathbf{M}^{-1} = (\mathbf{TR})^{-1} = \mathbf{R}^{-1}\mathbf{T}^{-1} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -p_x \\ 0 & 1 & 0 & -p_y \\ 0 & 0 & 1 & -p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} =$$

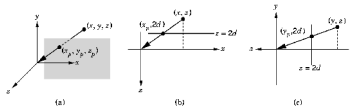
$$\begin{bmatrix} u_x & u_y & u_z & -p_x u_x - p_y u_y - p_z u_z \\ v_x & v_y & v_z & -p_x v_x - p_y v_y - p_z v_z \\ n_x & n_y & n_z & -p_x n_x - p_y n_y - p_z n_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In opengl

- Grazie al cielo in opengl tutto ciò è ben semplice
- gluLookAt(eyex, eyex, eyex,
  atx,aty,atz,
  upx,upy,upz);
- Definisce la trasformazione che porta dallo spazio di mondo allo spazio di eye;
- NOTA bene la gluLookAt va nella ModelView Matrix

Matrici di Proiezione Prospettica

- Assunto che siamo nel sistema di riferimento della camera con il centro di proiezione nell'origine, e il piano di proiezione a distanza d lungo l'asse -z
- Vogliamo trovare la proiezione (x_p, y_p, z_p) sul piano di proiezione di un punto (x, y, z)



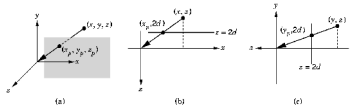
Proiezione Prospettica

- Si ottiene che:

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

$$x_p = \frac{x}{z/d} \qquad y_p = \frac{y}{z/d}$$

- Nota che questa trasformazione non è lineare, né affine, né reversibile.



Coordinate Omogenee

- Estendiamo la nostra def di coordinate omogenee dicendo che un punto p può essere rappresentato come

$$\mathbf{p} = \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix} \quad \text{con} \quad w \neq 0$$

- Un punto in 3D corrisponde ad una linea in 4d.
- Posso Sempre recuperare la forma con 1 come quarto elemento
- Posso fare matrici che modificano il quarto elemento.

Coordinate Omogenee

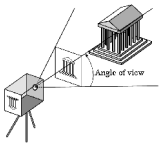
- In particolare possiamo definire la matrice

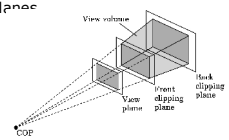
$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \quad \mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \mathbf{q} = \mathbf{Mp} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix} \approx \begin{bmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ d \\ 1 \end{bmatrix}$$

Che effettua la trasformazione prospettica, purchè si normalizzi dividendo per la quarta componente

Proiezione prospettica in opengl

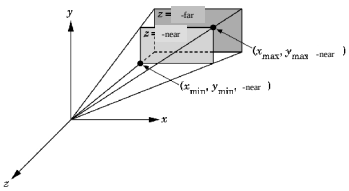
- Finora abbiamo definito solo l'operazione di proiezione.
- Per definire una camera dobbiamo anche definire il view volume
  - Angle of view
  - Near and far clipping planes





Proiezione prospettica in opengl

- glFrustum(xmin,xmax,ymin,ymax,near,far);
- gluPerspective(fov,aspect,near,far)



Proiezioni Ortogonali

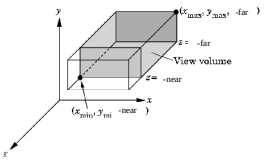
- Caso particolare di proiezione parallela in cui le linee di proiezione sono perpendicolari al view plane
- La proiezione è semplicemente

$$x_p = x \qquad y_p = y \qquad z_p = 0$$

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

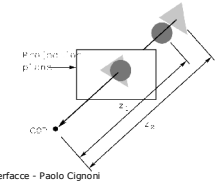
Proiezioni Ortogonali in OpenGL

- glOrtho(xmin,xmax,ymin,ymax,near,far);
- I clipping planes sono a z= -near e z=-far



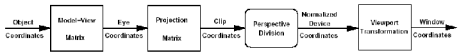
Hidden Surface Removal

- Si deve disegnare sul frame buffer solo quelle porzioni di primitive che sono davanti a tutte le altre; esistono numerosi algoritmi
- Tecnica Zbuffer, per ogni pixel dello schermo memorizzo la minima distanza dal centro di proiezione disegnata in quel pixel.
- Quando rasterizzo, disegno (e aggiorno lo zbuffer, solo se davanti.



Window e Device coords

- In Opengl si distingue tra
- Normalized Device (screen) Coords
  - Sono 3d e mantengono la depth
- Window Coord
  - Sono 2d.



Normalized Device Coord

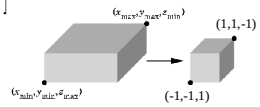
- In Opengl Perspective Transf + division Convertono a Normalized Device Coord
- La geometria è quindi clippata sul
- Canonical View Volume

$$x = \pm 1$$
$$y = \pm 1$$
$$z = \pm 1$$

Mapping the view volume

- Sul Canonical view Volume,
- In questo caso la proiezione ortografica diventa

$$\begin{bmatrix} \frac{2}{x_{max}-x_{min}} & 0 & 0 & -\frac{x_{max}+x_{min}}{x_{max}-x_{min}} \\ 0 & \frac{2}{y_{max}-y_{min}} & 0 & -\frac{y_{max}+y_{min}}{y_{max}-y_{min}} \\ 0 & 0 & \frac{2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Esempio tipico

- Un app carica/genera un oggetto in un suo sistema di riferimento e lo vuole far vedere.
  - Calcolare il bbox dell'oggetto.
  - Spostare l'oggetto nell'origine
  - Scalare l'oggetto fino alla dimensione desiderata che sia compatibile con le dim del mio volume di vista
  - traslare l'oggetto nel volume di vista

Esempio tipico

- L'ordine delle cose deve essere quello giusto

```
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
gluPerspective(ViewAngle,1,.1,10);
glMatrixMode (GL_MODELVIEW);
glLoadIdentity ();
glTranslatef(0,0,-4);  // anche una glulookat
                       // andava bene
float d = 2.0/m.bbox.Diag();
glScalef(d, d, d);
glTranslate(-m.bbox.Center());
DrawMyObject();
```