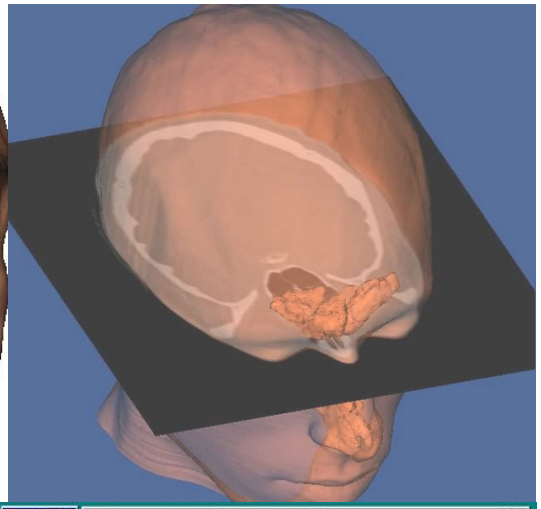
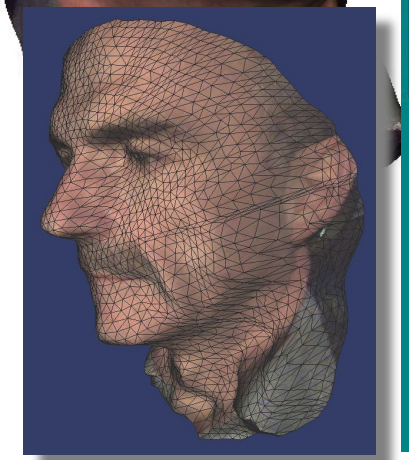
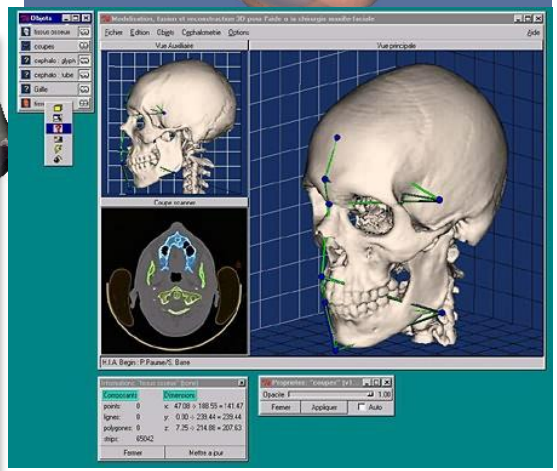
DII – Universitá di Siena

Introduzione alla Computer Grafica







la Torre Berardino

Mariottini Gian Luca

Dipartimento di Ingegneria dell'Informazione - **Siena 18-02-03 Ore 14.00-18.00**

Argomenti affrontati

1) Introduzione
   a) Applicazioni della CG
   b) Architettura hardware della CG
2) Il Processo di Visione in 3 Dimensioni
   a) Osservatore e scena
   b) Modelli di Telecamere
   c) Trasformazioni Geometriche
3) Algoritmi di Rendering non-geometrici
   a) Back-face culling
   b) Eliminazione delle superfici nascoste

Argomenti affrontati

4) Tecniche di Illuminazione & Ombreggiatura
   a) Le basi
   b) Rilflessione Diffusa e Speculare
   c) Flat Shading
   d) Gouraud Shading
   e) Phong Shading
5) Texture Mapping
   a) Texture e Bump Mapping
   b) Aliasing e anti-aliasing
6) OPENGL-VTK

Cos'è la Computer Grafica?

Confusione tra "imaging", "computer graphics" e "computer vision"!!

- L'***Imaging***, o Image Processing è lo studio delle immagini in 2D.

  Include le tecniche per ruotare, scalare, estrarre informazioni bidimensionali,... dalle immagini digitali.

- La ***Computer Graphics*** è il processo di creazione di immagini a partire da dati elaborati dal computer.

  Include sia il disegno 2D sia le più complesse tecniche 3D (rendering,texture mapping, shading etc.)

- La **Computer Vision** si occupa di analizzare le immagini per estrarne dati significativi (proprietà 3D, etc.).

La Computer Grafica...

Genera Immagini...

- approssimazione del mondo con funzioni discrete
- un insieme di punti colorati (PIXELS) approssimano l'immagine su un display grafico

Computer Graphics significa simulazione...

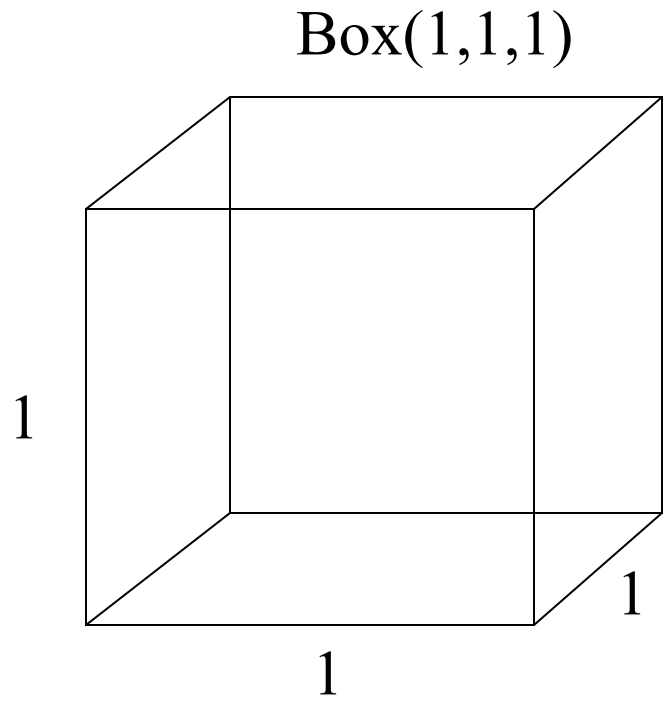
- di forma e posizione degli oggetti
- di aspetto e fisicità
- di movimento di oggetti

Computer Grafica interattiva...

- uno dei modi più naturali per comunicare con il computer!

Modello tridimensionale

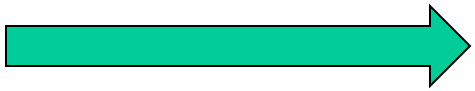
- Descrizione di un oggetto 3D!



Dal modello all'immagine

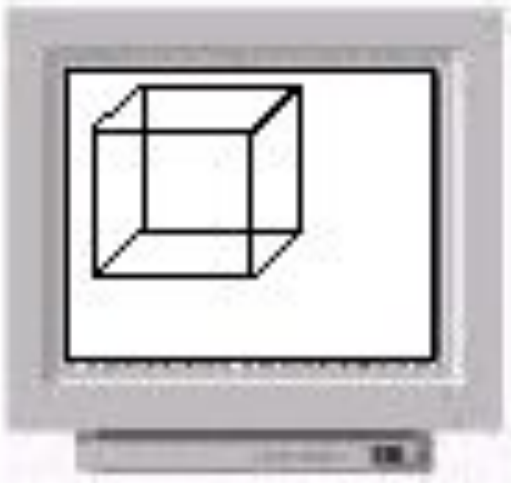
Modello 3D

Computer Graphics



Immagine

Box(1,1,1)



Dal modello all'immagine

Modello 3D

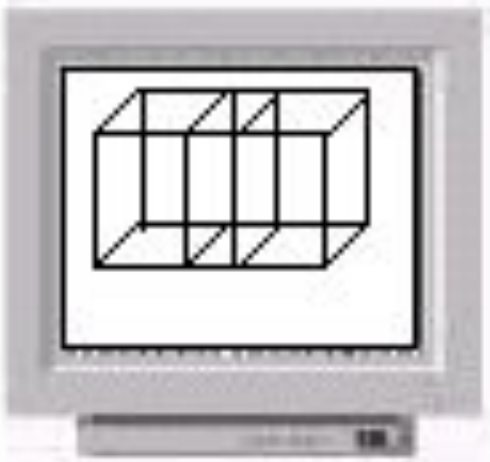
Computer Graphics

Immagine

Modellazione Geomerica

Box(1,1,1)

Translate(1,0,0)



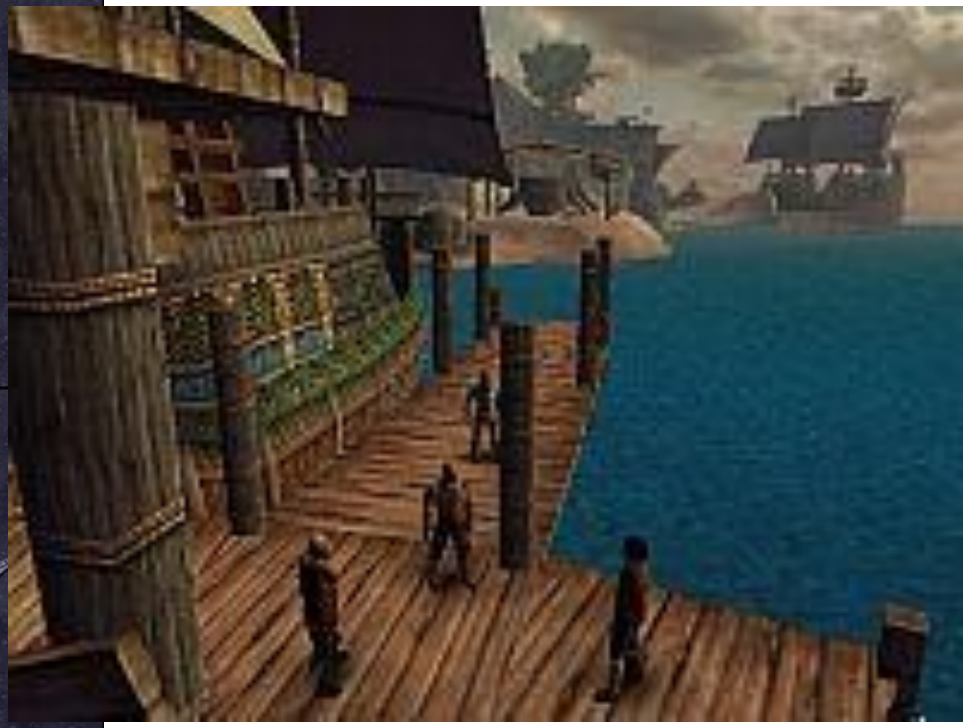
...le applicazioni...

Campi Applicativi della Computer Grafica

Il cinema!

Campi Applicativi della Computer Grafica

I Videogames!



Campi Applicativi della Computer Grafica

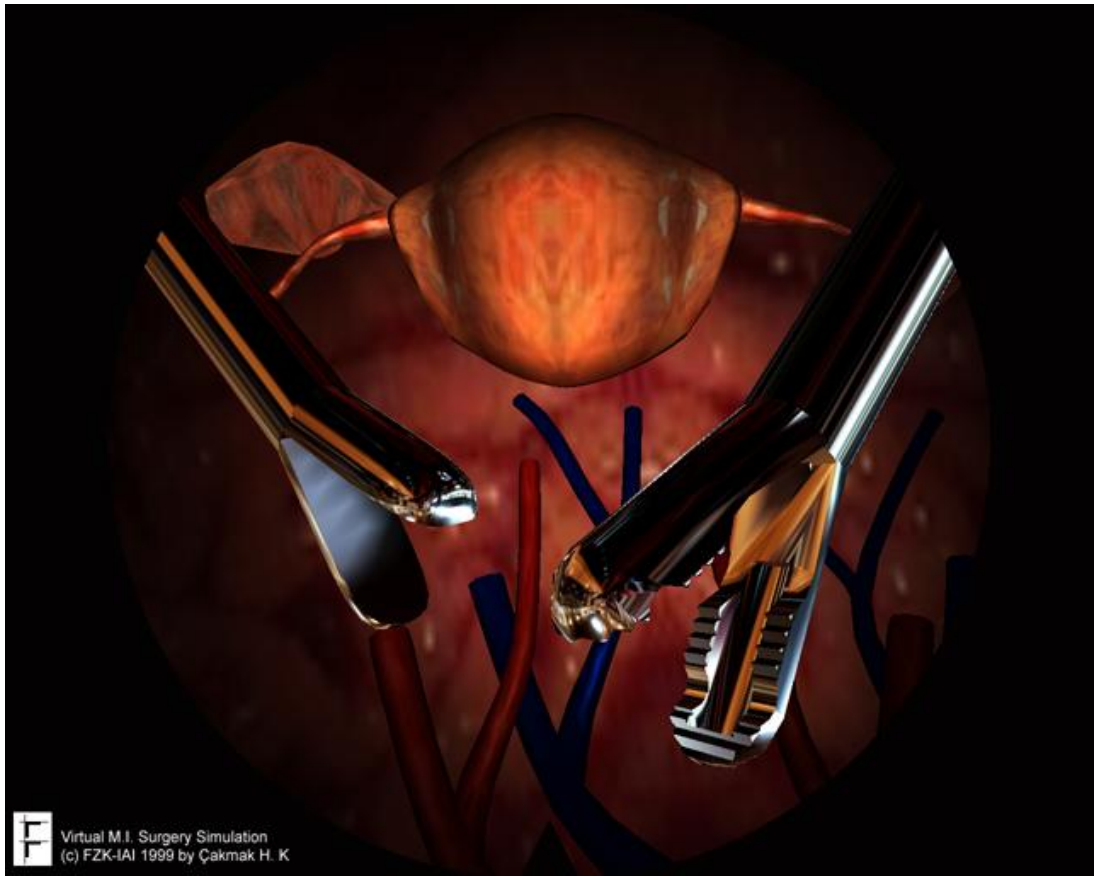
La Medicina!

From ...ultrasound images

...to 3D (4D) reconstruction !!

Campi Applicativi della Computer Grafica

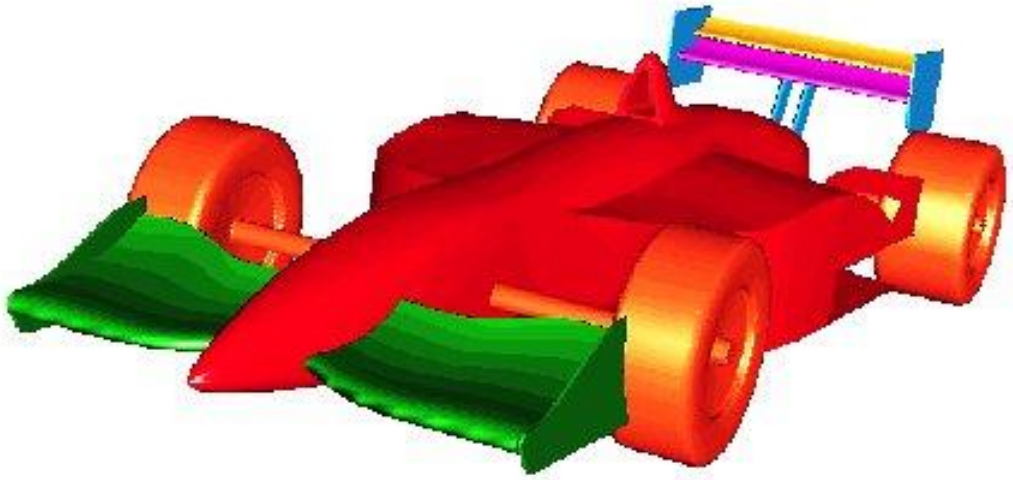
La Medicina!

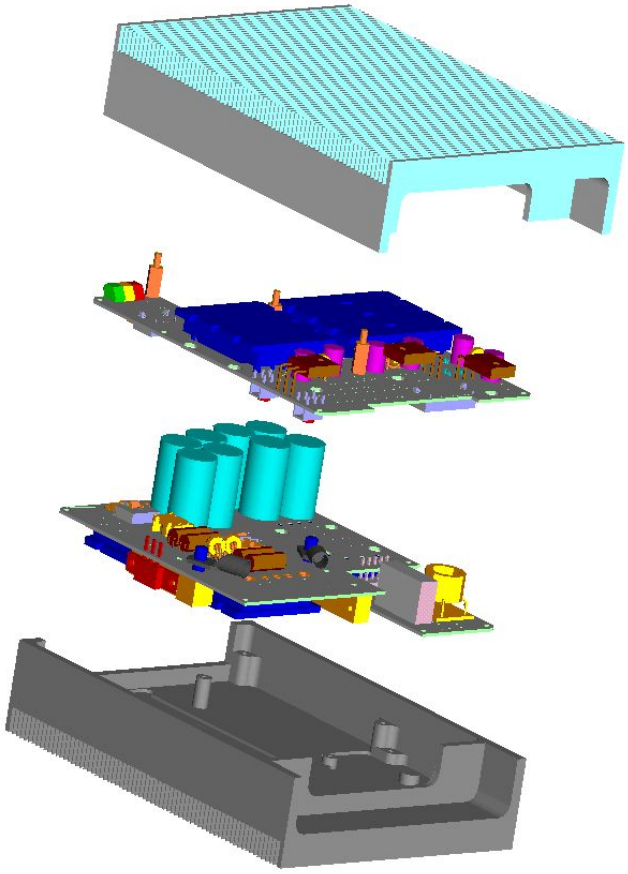


Simulazione chirurgica

Progettazione e manifattura assistita da PC

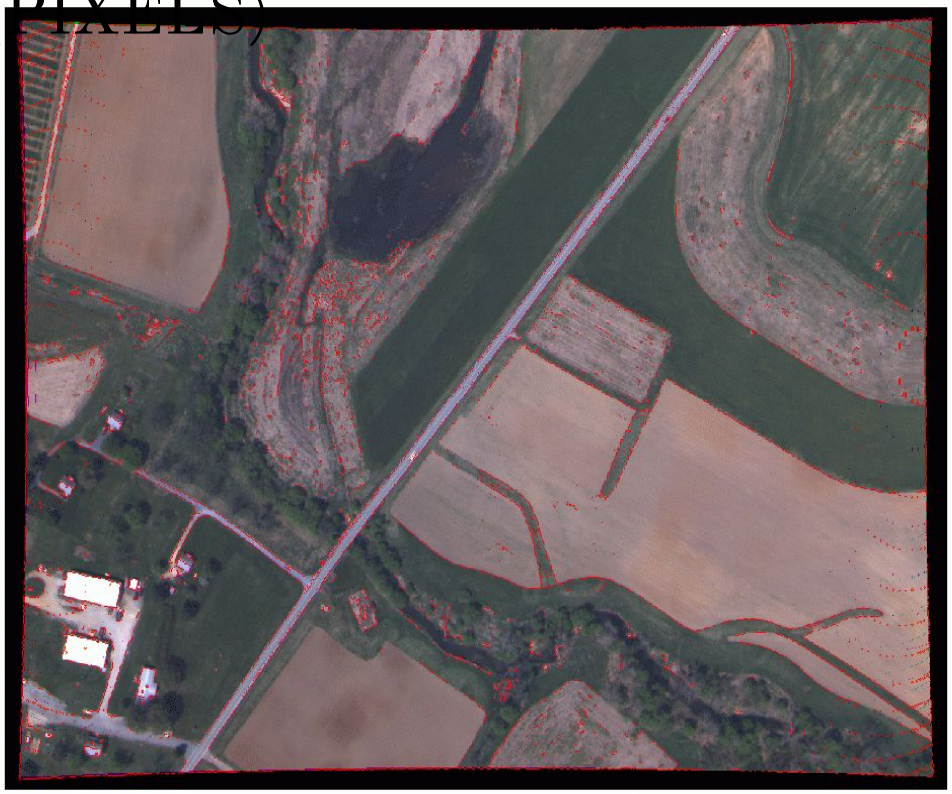
CAD e CAM





Le Immagini Digitali

Immagine = Matrice di punti colorati (PIXELS)



N pixels

M pixels

Dimensione dell'immagine = NxM pixels

Immagini: formati

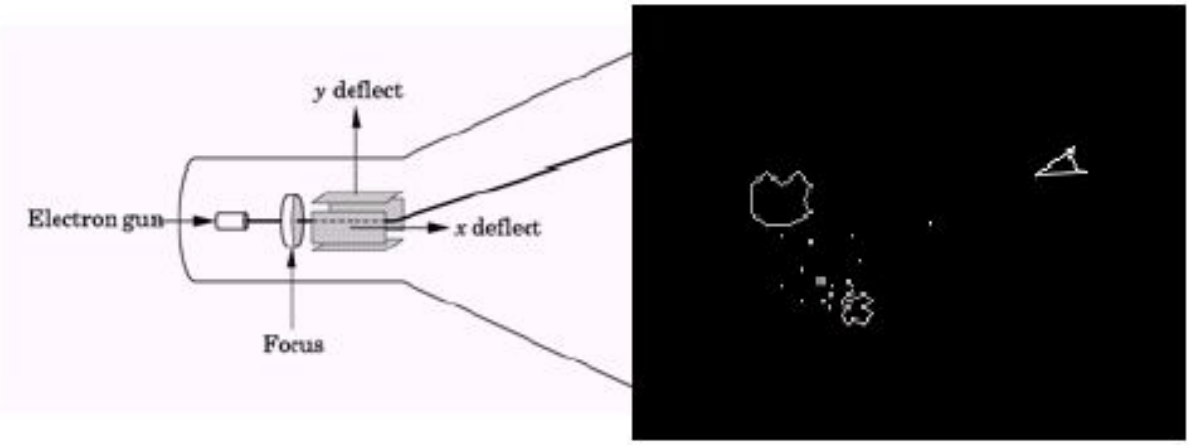
- **Proprietari**
  - PSD (Photoshop)
  - ...
- **Standard**
  - TIFF
  - EPS
  - JPG,GIF
  - BMP, PCX
  - ...

Evoluzione della Computer Graphics

Guidato dall'evoluzione dell'hardware:

(Grafica Vettoriale- ANNI '60-'70) :fascio di elettroni che colpisce la superficie del CRT viene guidato per muoversi da una parte all'altra....

Prime applicazioni ai videogames!

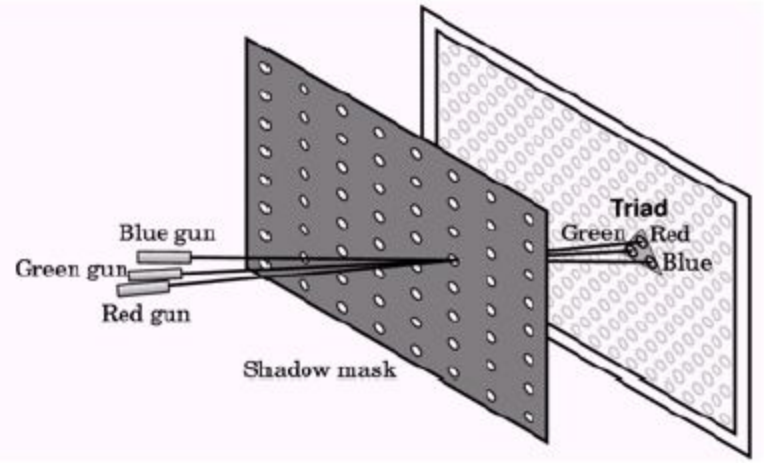


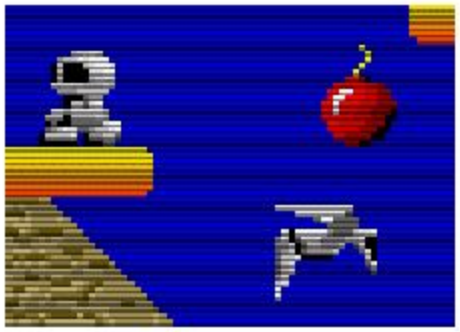
Asteroids

Evoluzione della Computer Graphics

Guidato dall'evoluzione dell'hardware:

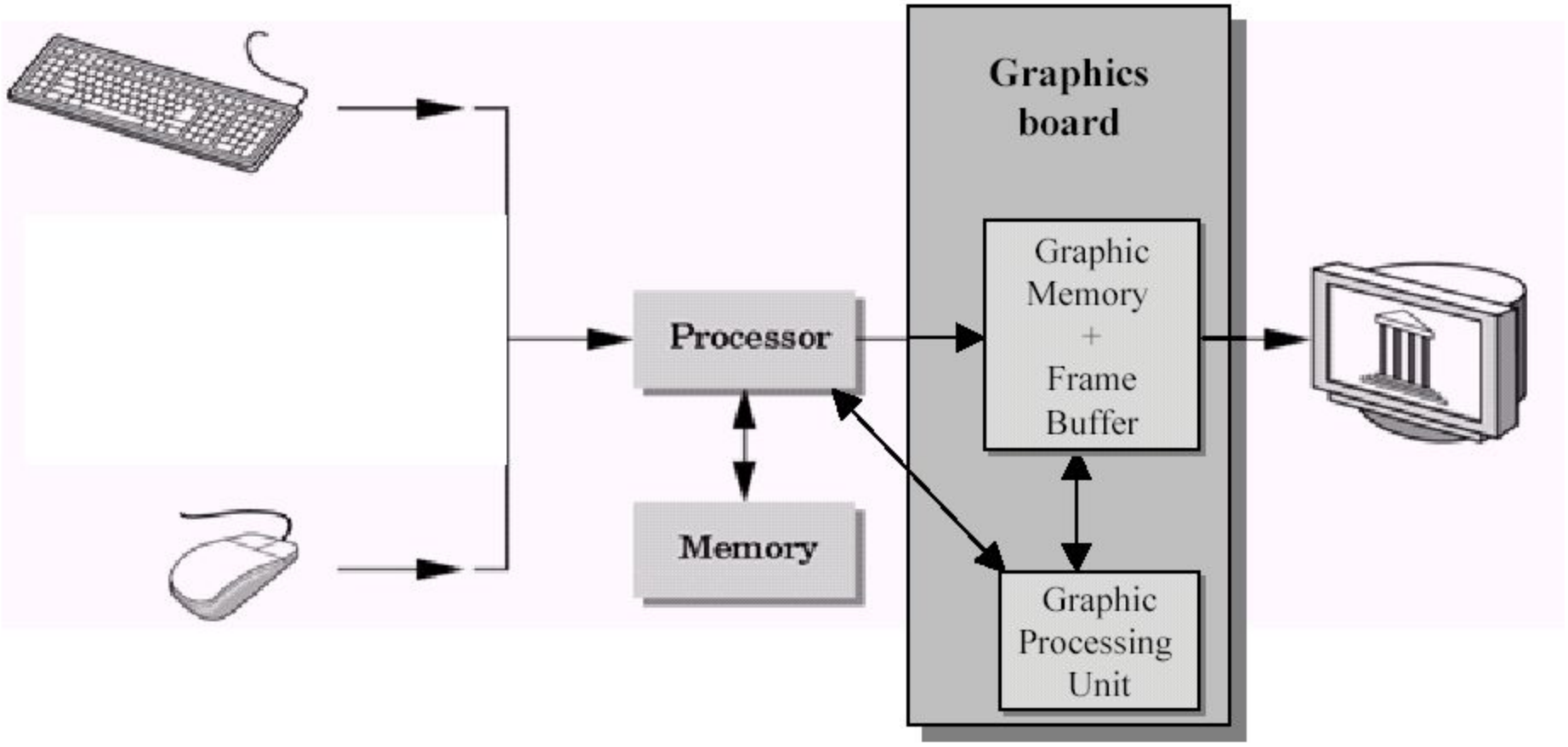
(Grafica Raster- ANNI '80□) :un'immagine è rappresentata da una Matrice rettangolare (*raster*) di elementi (*pixel,* abbreviazione di *picture element*)





Illuminare i pixels in modo opportuno!

Architettura HW per la Computer Grafica (I)



Architettura HW per la Computer Grafica (II)

FRAME BUFFER

Una porzione di memoria dedicata alla memorizzazione dell'immagine come insieme di pixel da mostrare a video.

GRAPHIC BOARD

Una scheda hw dedicata ad elaborare dati grafici e a pilotare il display raster

1) GPU (Graphic Processing Unit) per compiere elaborazioni specifiche necessarie per formare l'immagine.
2) Graphic Memory memoria ausiliaria utilizzata dalla GPU per le sue operazioni.

Il Processo di Visione in 3 dimensioni

Fare Grafica 3D???

Non significa programmare sul Frame Buffer!

Creare scene in un mondo virtuale 3D composte da oggetti geometrici con attributi pittorici + immagini e testo!

Interagire virtualmente con gli oggetti che compongono la scena!

Guardare le scene da un determinato punto di vista e visualizzare l'immagine corrisipondente! (telecamera)

Animare le scene in conseguenza di cambiamenti di punti di vista, cono di vista, posizione e/o deformazione degli oggetti.

Paradigma per la sintesi di immagini 3D

Due entità fondamentali:

1- **Oggetti** (sintetici)

↓

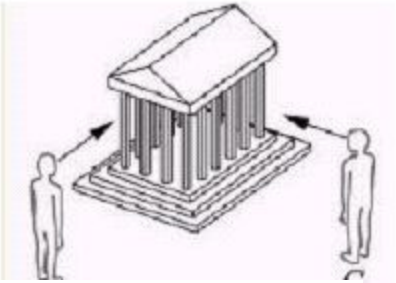
"Rappresentazione digitale di forma e caratteristiche di un oggetto reale tridimensionale"

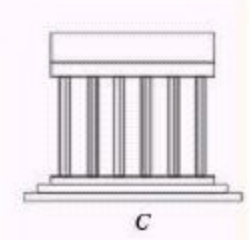
↓

...posizionati nello spazio 3D...

↓

...generano scene.





2- **Osservatore** (virtuale)

↓

Permette di osservare la scena da un certo punto di vista.

↓

Rendering

=

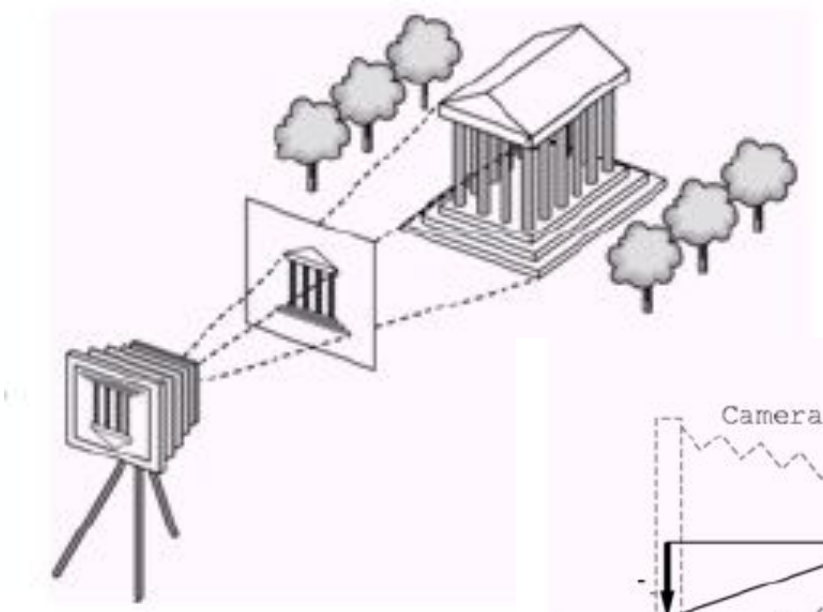
"Processo grazie al quale l'osservatore genera un'immagine a partire da una scena"

Per definire un osservatore...

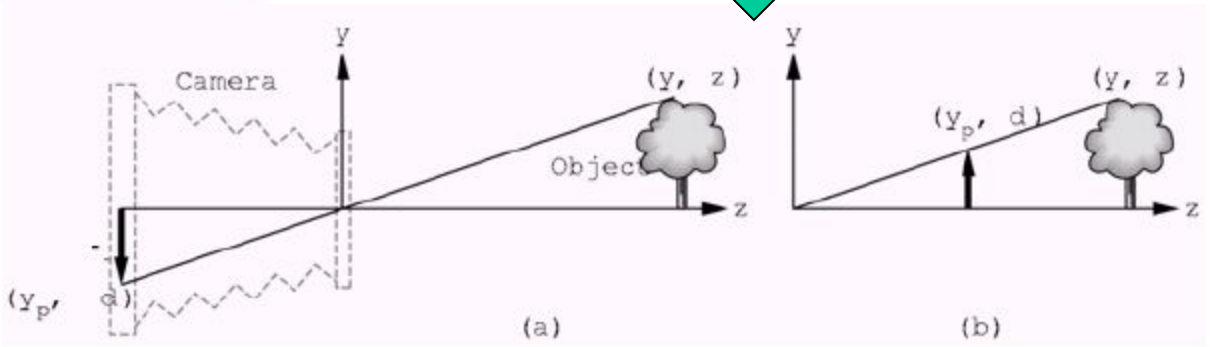
Punto di vista: punto 3D da cui l'osservatore guarda la scena.

Direzione di vista: la direzione verso cui guarda.

Distanza focale: dà informazioni sulla porzione di scena osservata.



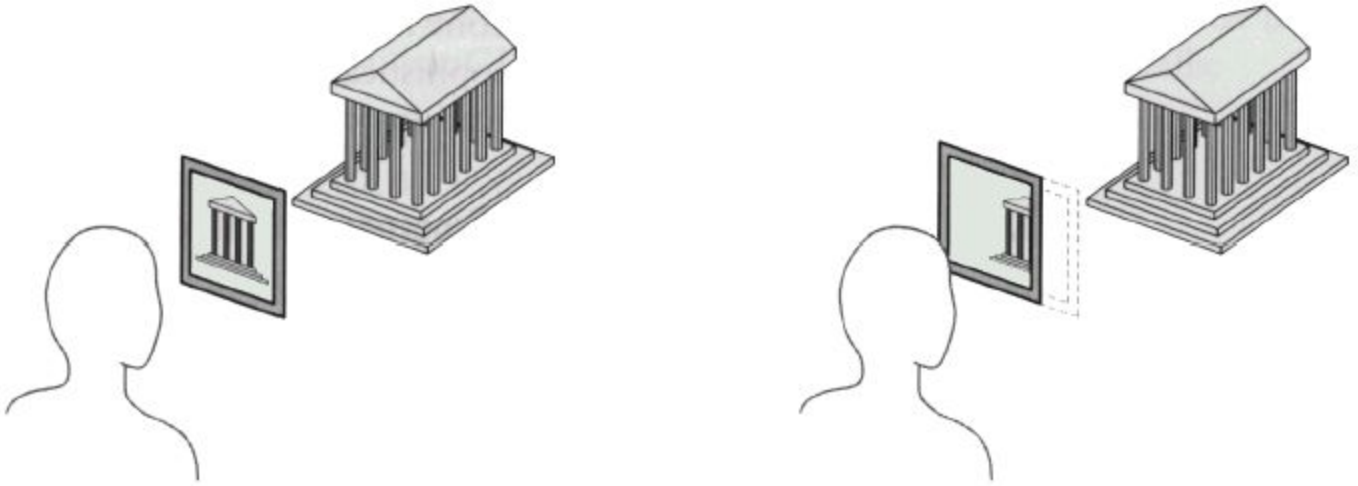
PIN-HOLE CAMERA MODEL



...operazione di clipping

Posizionamento di una finestra (window) sul piano immagine.

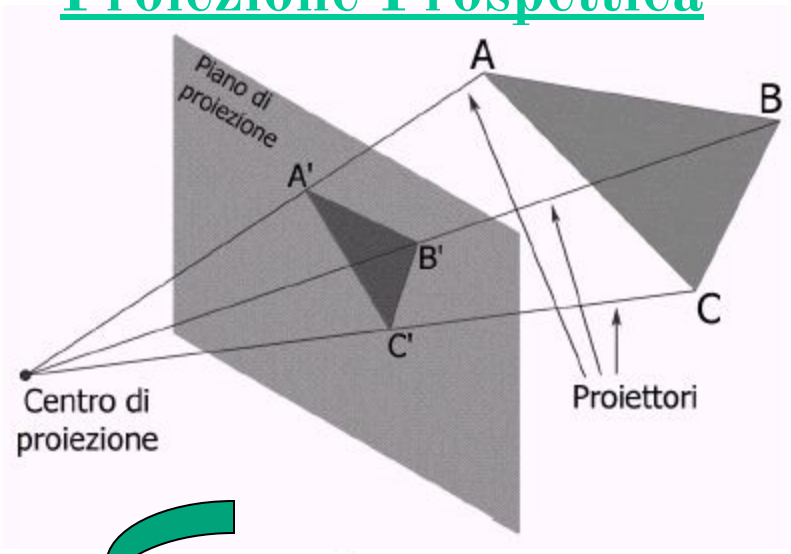
Mantenere solo le parti visibili della scena

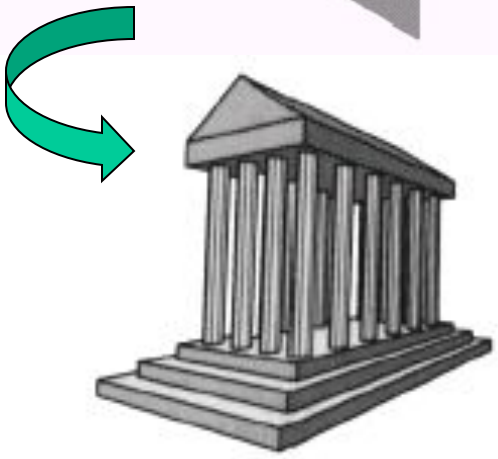


Fase di clipping con due diverse finestre

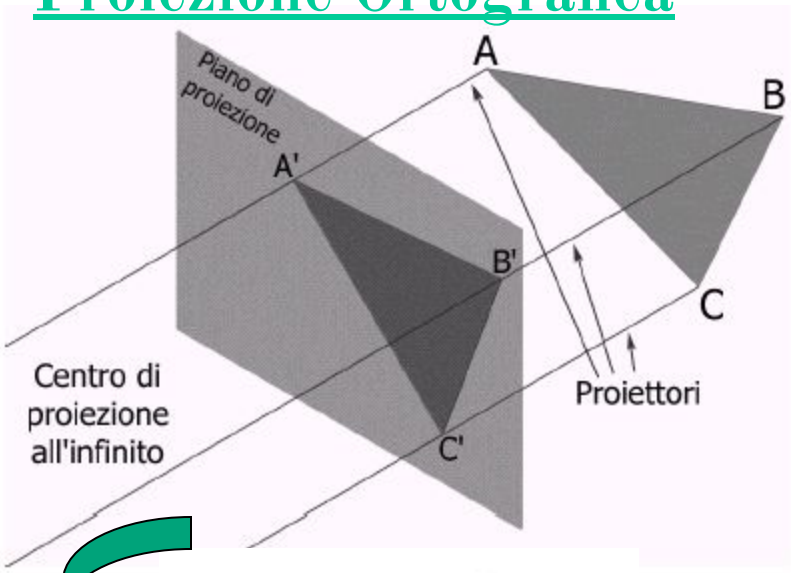
...tipi di proiezioni

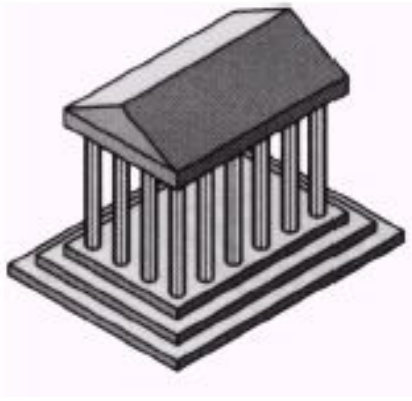
Proiezione Prospettica





Proiezione Ortografica





Parametri del modello pin-hole

Si devono specificare:

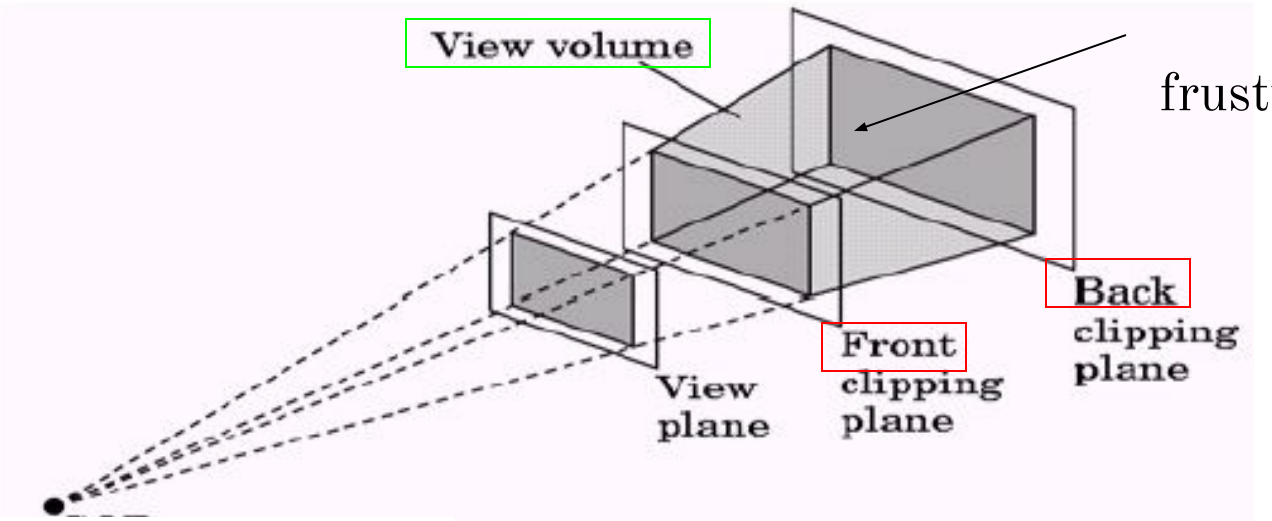
1-Posizione;

2-Orientamento;

3-Distanza Focale.

Si definisce un *volume di vista* (FOV) che racchiude tutti gli oggetti visibili dalla telecamera.

Volume di vista ⟹ Tronco di Piramide



Trasformazioni Geometriche

1) Posizionamento degli oggetti nella scena

   Sono necessarie...

   Traslazioni, Rotazioni, Scalature

2) Scena da inquadrare

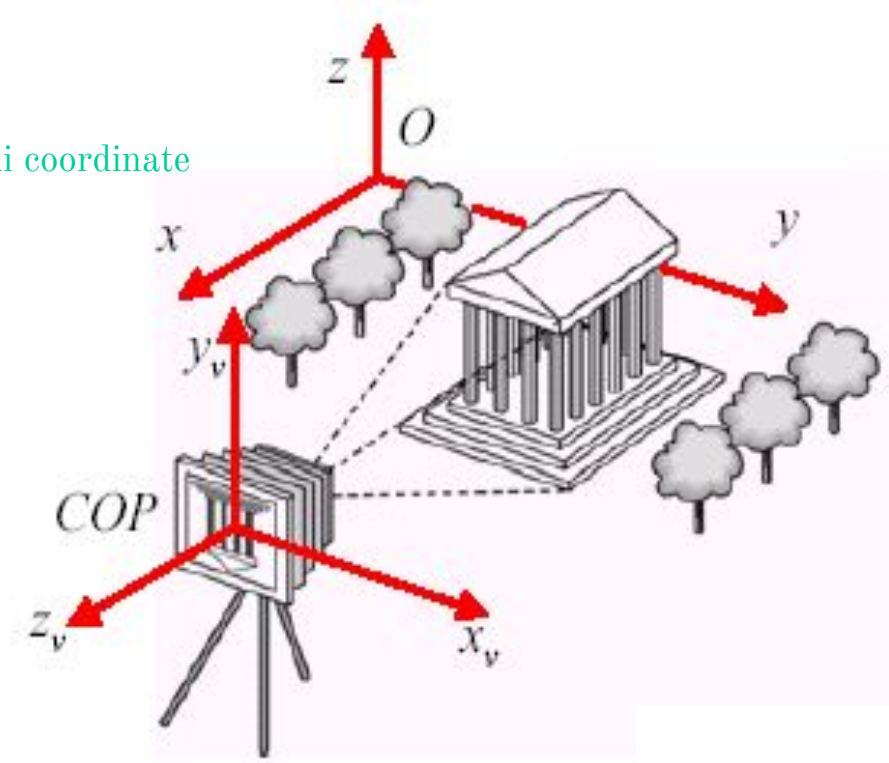
   ...esprimere oggetti scena nel sistema di coordinate di vista...

   Trasformazioni Rigide

3) Scena da visualizzare

   ...modello di telecamera

   Proiezione ortografica o prospettica



Trasformazioni Geometriche

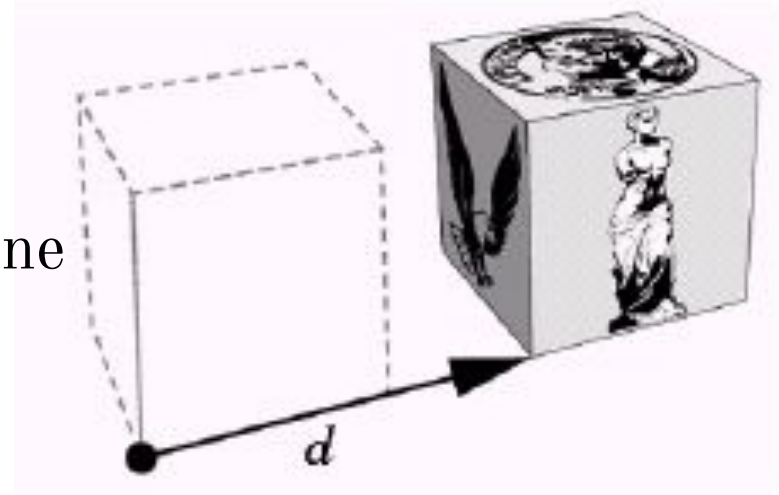
Traslazione

Vettori in coordinate omogenee

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

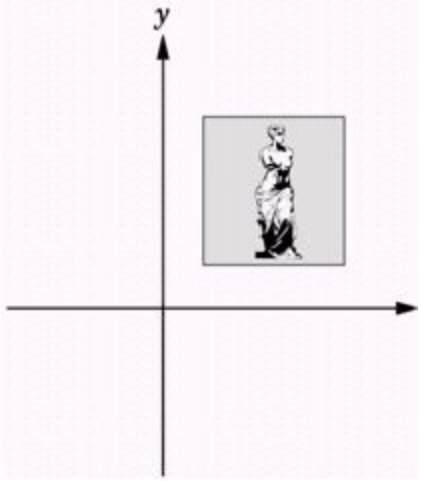
$$\mathbf{p}' = \mathbf{T}\mathbf{p} \qquad \mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & \alpha_x \\ 0 & 1 & 0 & \alpha_y \\ 0 & 0 & 1 & \alpha_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

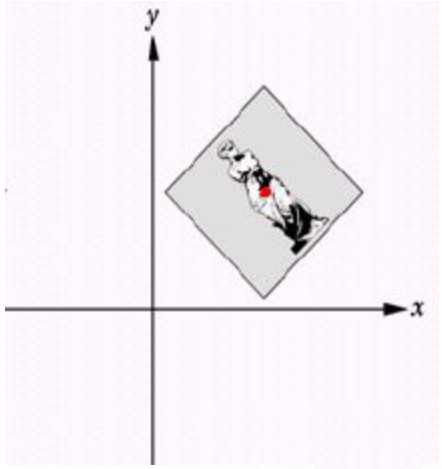
Traslazione



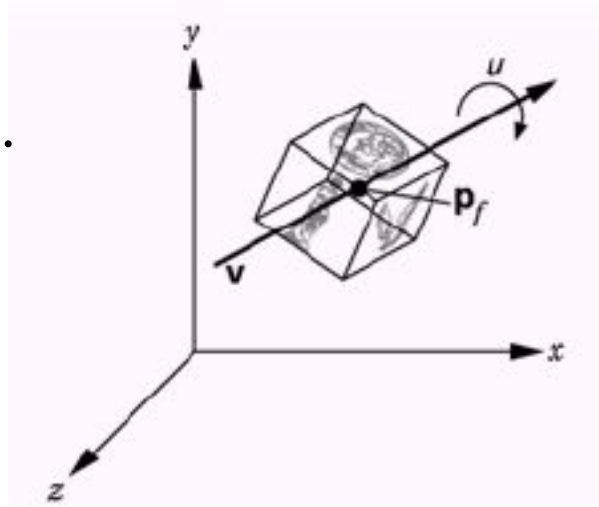
Trasformazioni Geometriche

Rotazione





….2D e 3D…

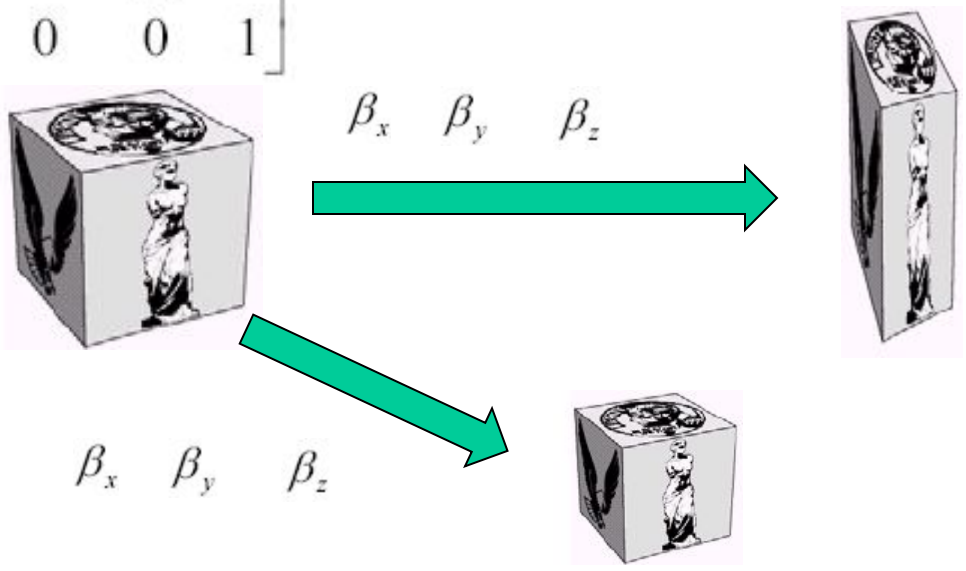


$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \quad \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \ \ldots$$

Trasformazioni Geometriche

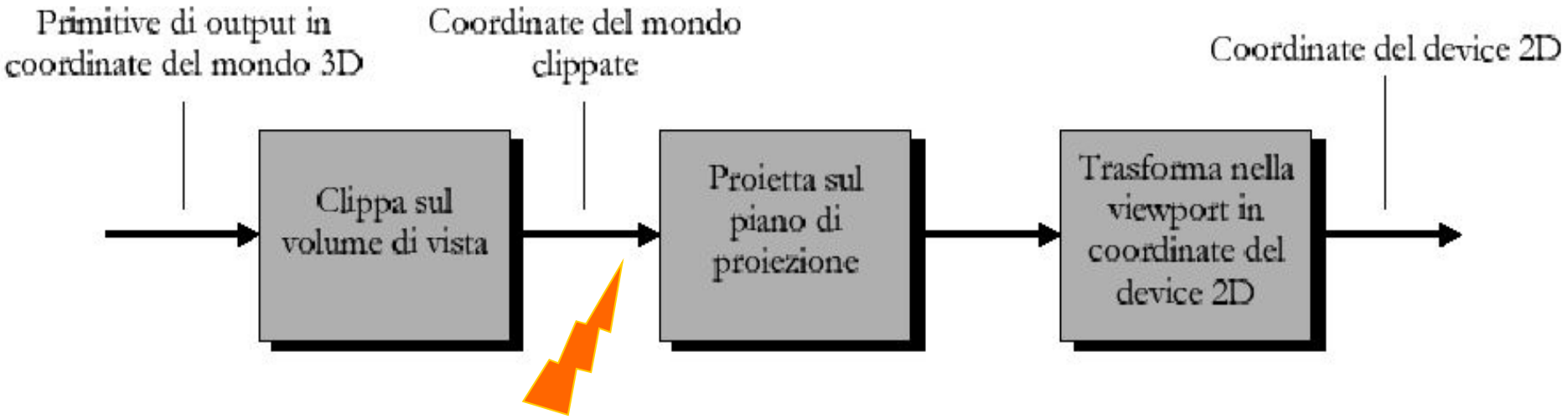
Scalatura

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



Modello di Visione 3D

...ricapitolando...



Eliminare oggetti non visibili...algoritmi non geometrici

Algoritmi di Rendering non Geometrici

Algoritmi di Rendering non geometrici

Scopo: Determinare gli oggetti visibili presenti nella scena per semplificare e accelerare le operazioni di proiezione.

Algoritmi *non-geometrici*

Non compiono trasformazioni sugli oggetti interni alla scena !

Sviluppo di nuovi algoritmi

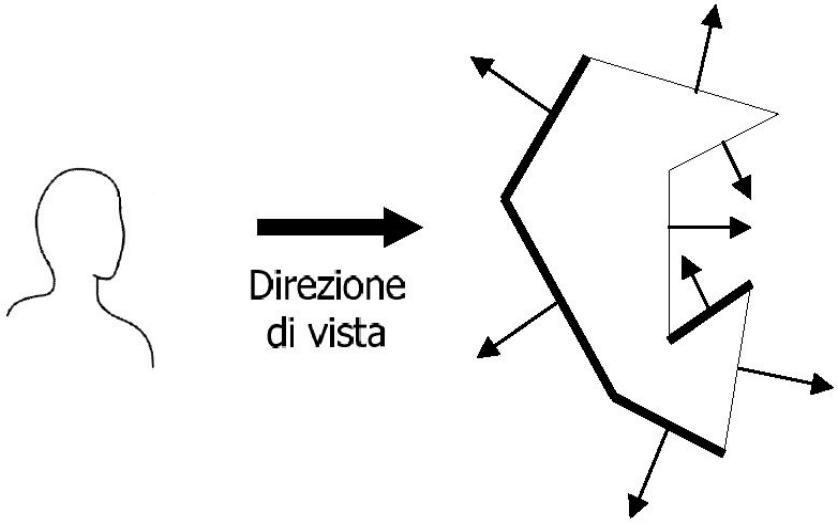
Progetto di speciali architetture hardware

Back-face culling

IPOTESI: Normali alle superfici dei poligoni sono dirette verso l'esterno del poligono.

normale che punta verso l'osservatore ⇨ superficie visibile

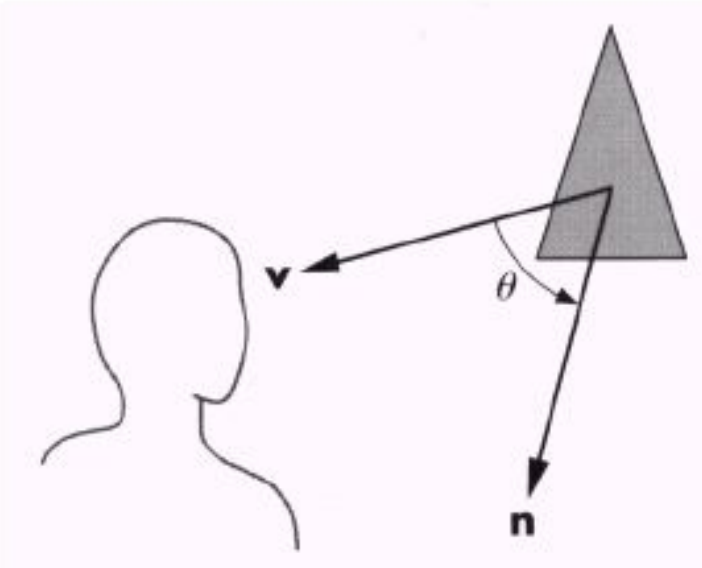
normale che punta via dall'osservatore ⇨ superficie invisibile



Back-face culling

"Rimozione delle superfici nascoste"

Back-face culling



Superficie visibile: $-90° \le \theta \le 90°$

$\Downarrow$

$\cos\theta \ge 0$

$\Downarrow$

$n \cdot v \ge 0$

Dimezzamento dei tempi necessari per il rendering

Eliminazione delle superfici nascoste

Scopo: Determinare se un oggetto è oscurato o no da altri oggetti.

2 tipi di algoritmi:

- Algoritmi in *object-precision (o object-space)*
- Algoritmi in *image-precision (o image-space)*

a) object-space:

1) A oscura B: visualizza A;
2) B oscura A: visualizza B;
3) A e B completamente visibili: visualizza A e B;
4) A e B si oscurano parzialmente:calcola le parti visibili.

Eliminazione delle superfici nascoste

Si prende uno dei k poligoni e lo si confronta con restanti k-1:

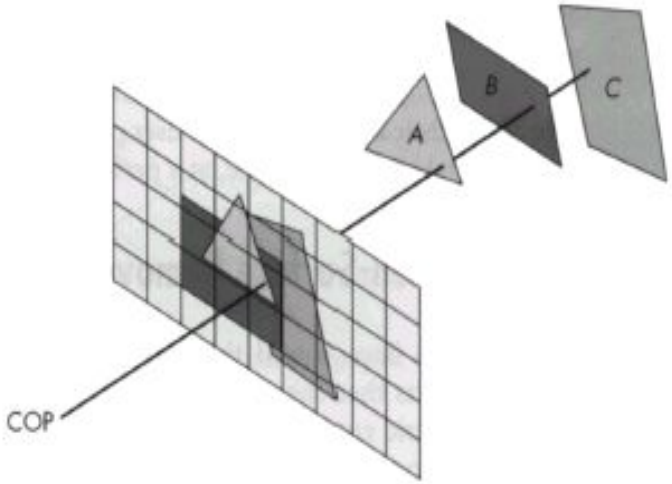
COMPLESSITA'= $O(k^2)$

b) image-space:

Per ogni pixel si considera il raggio che parte dal centro di proiezione e passa per quel pixel.

Quale oggetto incontra per primo?

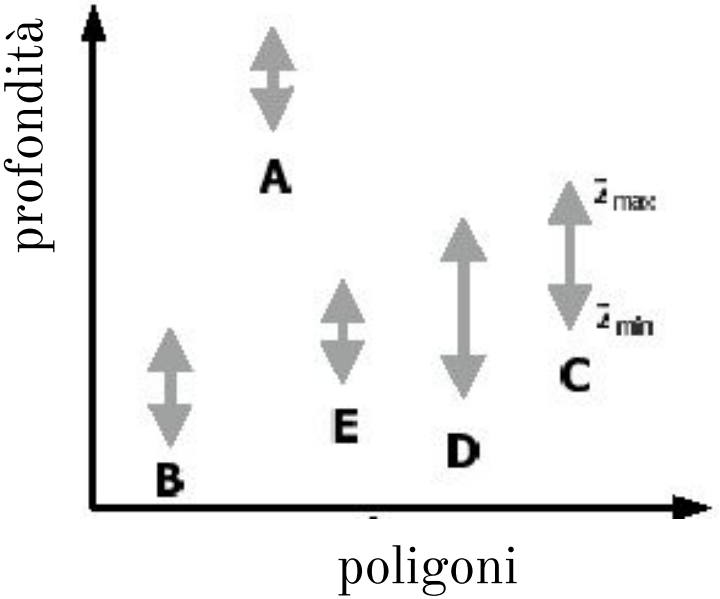
Lo si disegna!



Eliminazione delle superfici nascoste

Algoritmo Depth-Sort= "Algoritmo del pittore"

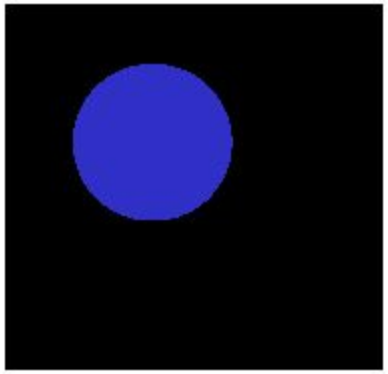
IDEA = visualizzare gli elementi in modo inverso rispetto all'ordine di profondità (gli oggetti + lontani sono oscurati da quelli più vicini)



Tecniche di Illuminazione & Ombreggiatura (shading)

Illuminazione & Ombreggiatura

Incrementare il realismo della scena !!



Scena poco realistica ☹

Scena + realistica !
☺

Illuminazione & Ombreggiatura

1. Come la luce incide sull'oggetto?
2. Come l'oggetto riflette la luce (superf. lucida/opaca)?

Per simulare quello che avviene nel normale processo di illuminazione abbiamo bisogno di un modello di ombreggiatura e di lighting.

Illuminazione

Ogni punto dell'oggetto nella scena è illuminato in funzione di:

1-Sorgenti di luce presenti sulla scena;

2-Posizione del punto;

3-Orientamento della superficie nel punto;

4-Proprietà riflettenti del materiale.

LIGHTING="Calcolo dell'equazione di illuminazione in uno o più punti dell'oggetto da illuminare".

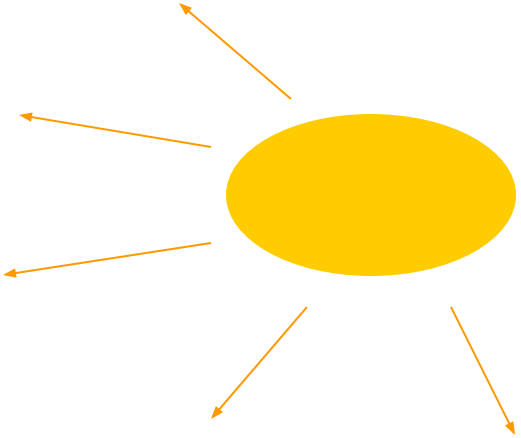
Illuminazione

SHADING="Modo con cui il lighting viene utilizzato per simulare
l'illuminazione durante il rendering".

Componenti dell'illuminazione

Componente emissiva

Oggetto=Sorgente da cui la luce scaturisce in modo uniforme.



$$I = k_i$$

I =Intensità risultante

k_i =Luminosità intrinseca dell'oggetto

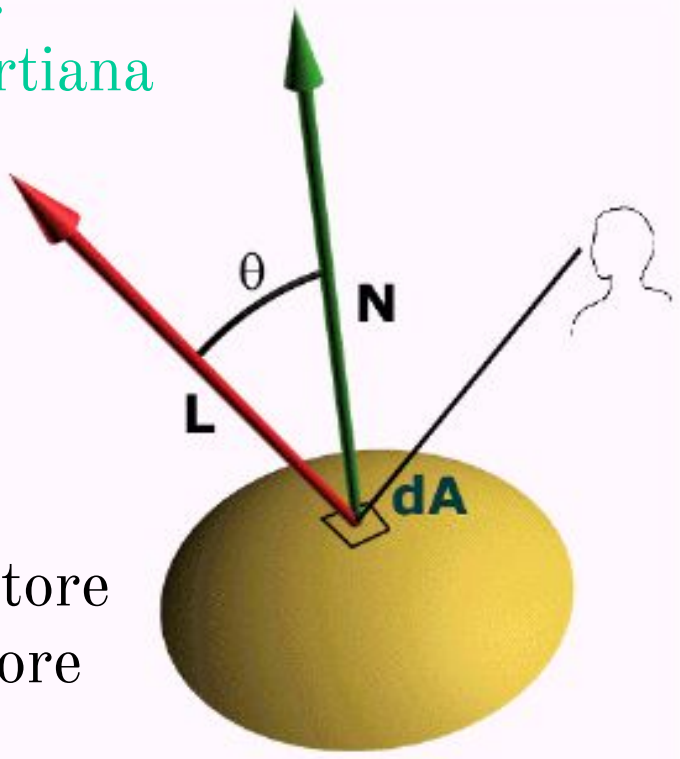
Usata solo per le fonti di luce presenti nel campo di vista

Riflessione Diffusa

detta anche...

Riflessione Lambertiana

La luminosità dipende dall'angolo tra
La direzione del raggio luminoso **L** e
la normale **N** alla superficie nel punto.

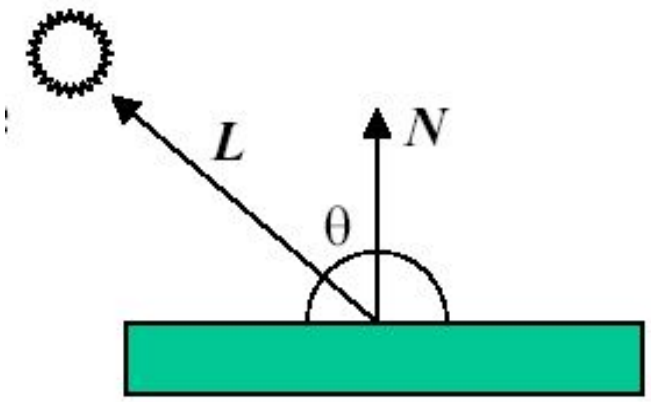


La quantità di luce che arriva all'osservatore non dipende dalla posizione dell'osservatore stesso.

Riflessione Diffusa

L'equazione di illuminazione per l'apporto dato dalla riflessione diffusa, è data dalla Legge di Lambert:

$$I = I_p k_d \cos\theta$$



I_p =Intensità luminosa

k_d =*Coefficiente di riflessione diffusa* del materiale (tra 0 e 1)

Angolo compreso tra 0° e 90° ⇒ Punto non illuminato da sorgenti dietro al punto stesso.

Se i vettori N e L sono normalizzati ⇒ $I = I_p k_d (\overline{N} \cdot \overline{L})$

Riflessione Diffusa

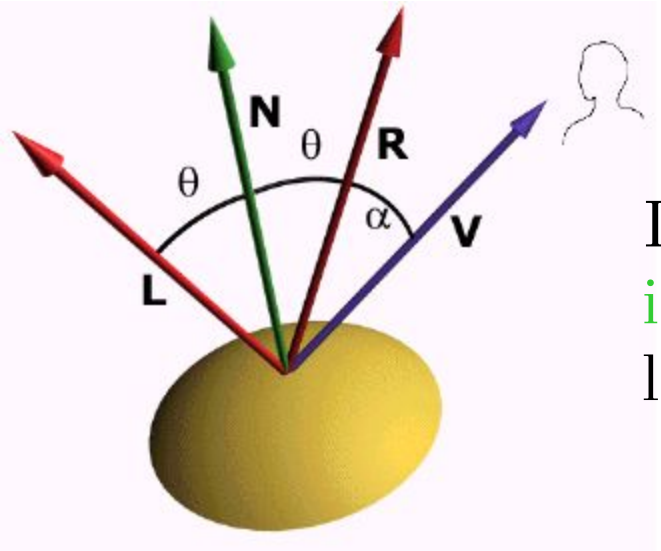
Sorgente luminosa posta all'infinito ( )

L è costante per tutta la superficie

Non deve essere calcolato ad ogni passo!

Riflessione diffusa

Riflessione Speculare

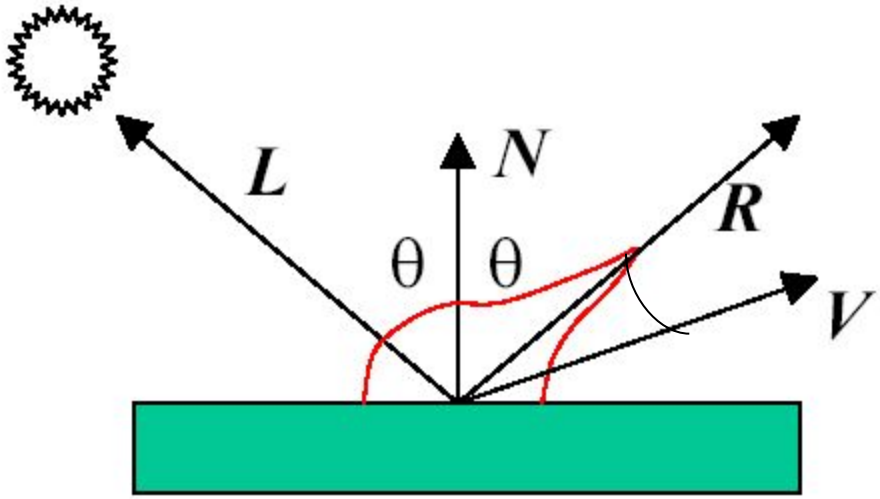


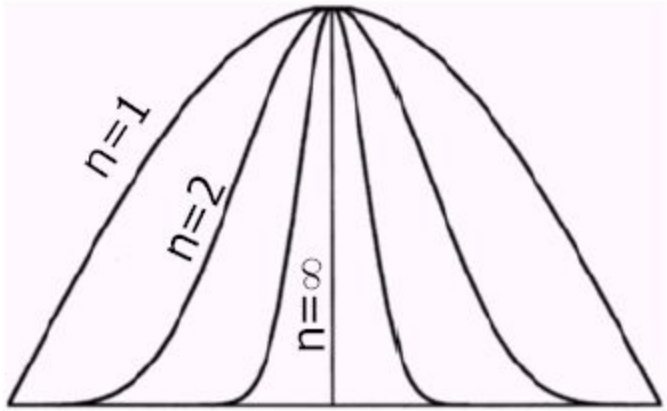
La luce non viene riflessa in maniera uguale in tutte le direzioni!

Riflessione speculare

Solo nella direzione di riflessione **R,** ottenuta riflettendo **L** rispetto a **N**

Riflessione Speculare





$$I = f_{att} I_p k_s \cos^n \alpha$$

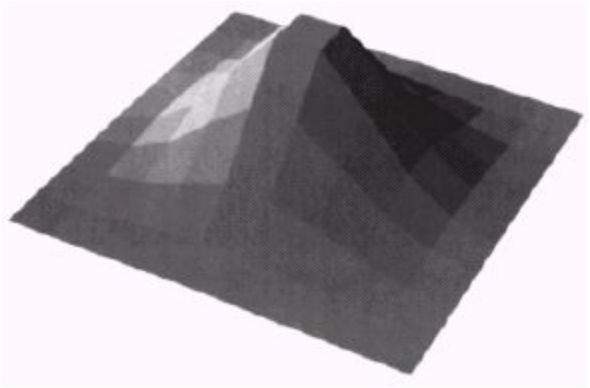
Equazione Completa

$$I = k_i + I_a k_a + f_{att} I_p \left[k_d \left(\overline{\mathrm{N}} \cdot \overline{\mathrm{L}} \right) + k_s \left(\overline{\mathrm{R}} \cdot \overline{\mathrm{V}} \right)^n \right]$$

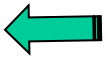
Emissiva Ambientale Diffusa Speculare

Flat Shading

Consiste nell'applicare il modello di illuminazione scelto una volta per ogni poligono costituente la scena.

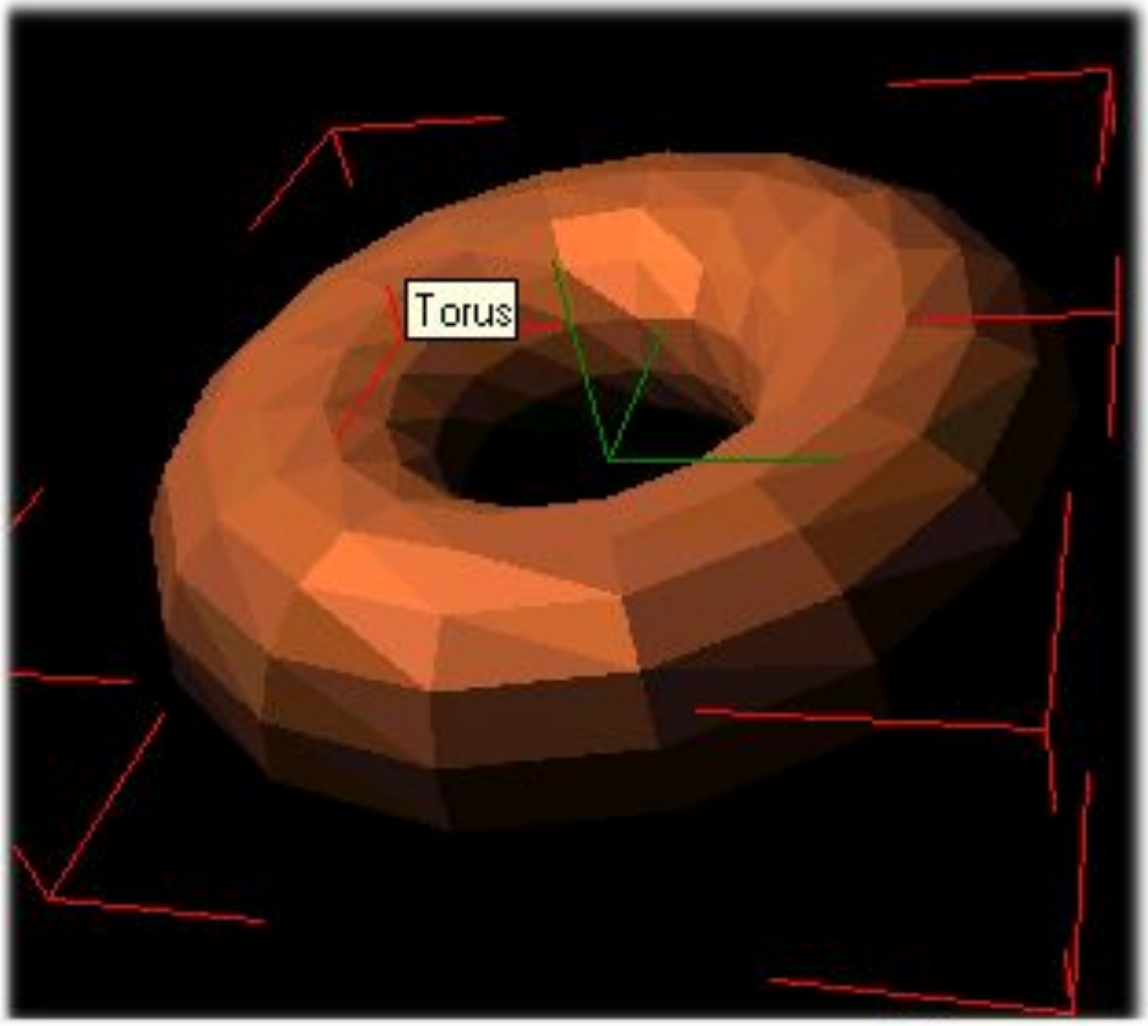


Equazione di illuminazione calcolata una volta per ogni poligono



EFFICIENTE

Flat Shading:esempio



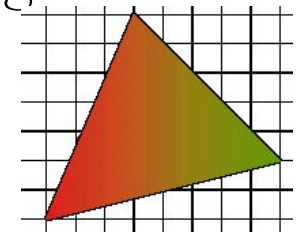
Gouraud Shading

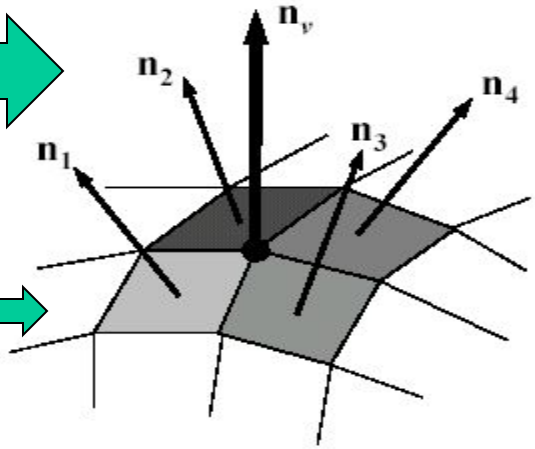
Valutare l'equazione di illuminazione sui vertici del poligono!

Si eliminano le discontinuità

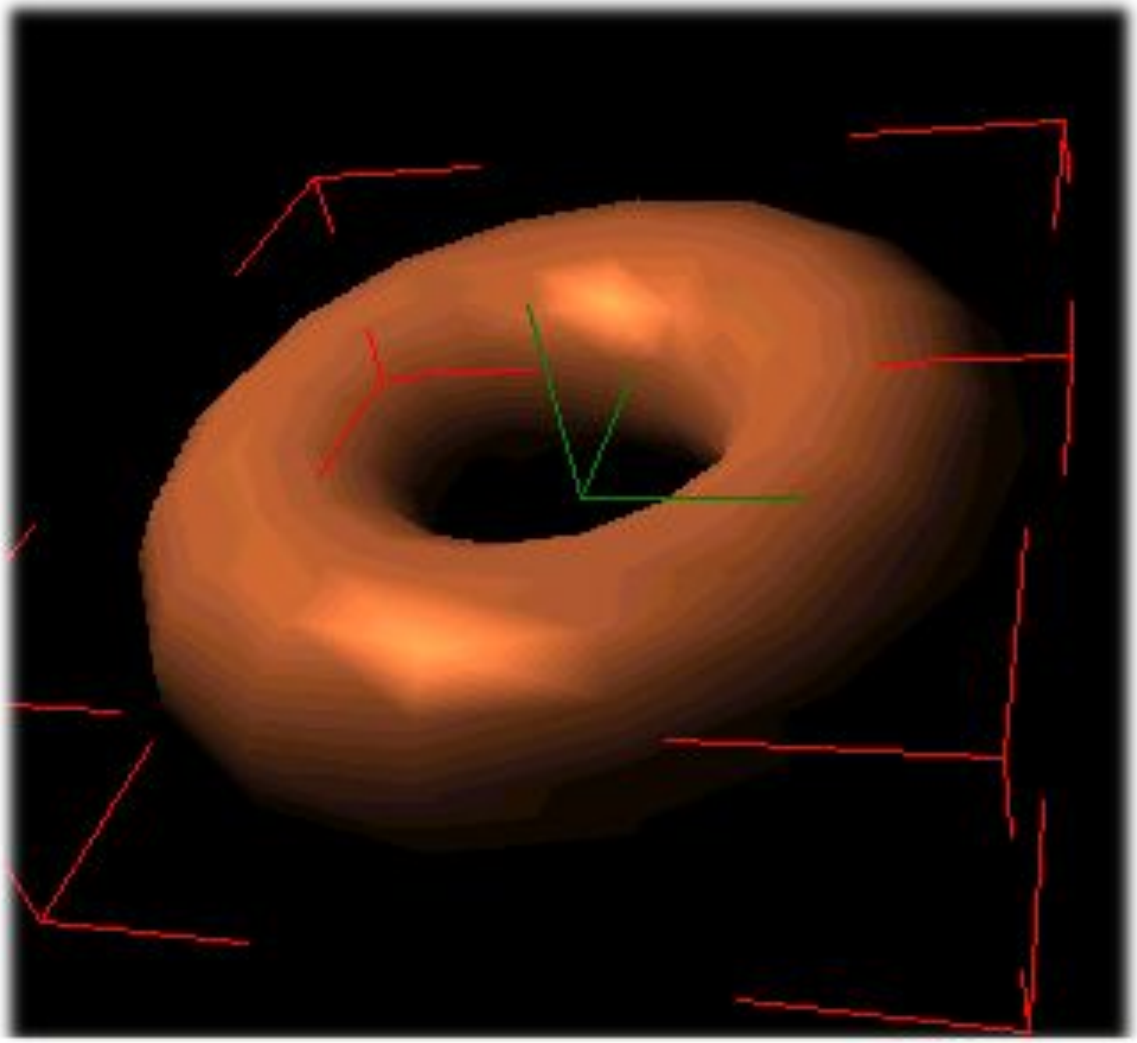
1. La normale di ciascun vertice viene valutata con la media delle normali dei poligoni adiacenti!

2. Poi ogni poligono è ombreggiato tramite interpolazione lineare.





Gouraud Shading:esempio



Phong Shading

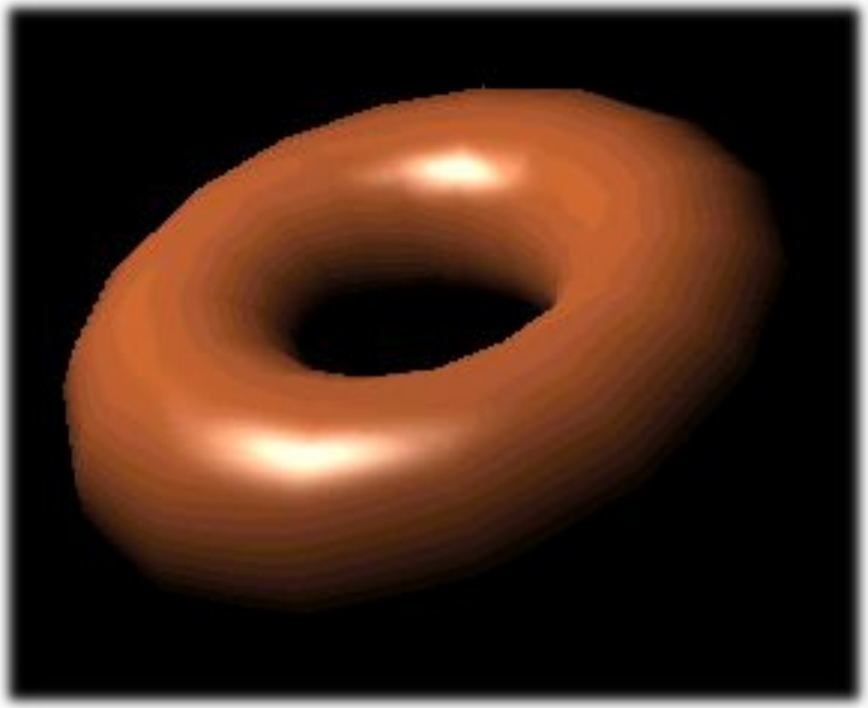
Metodo computazionalmente + oneroso!

Maggior realismo alla fine del rendering!

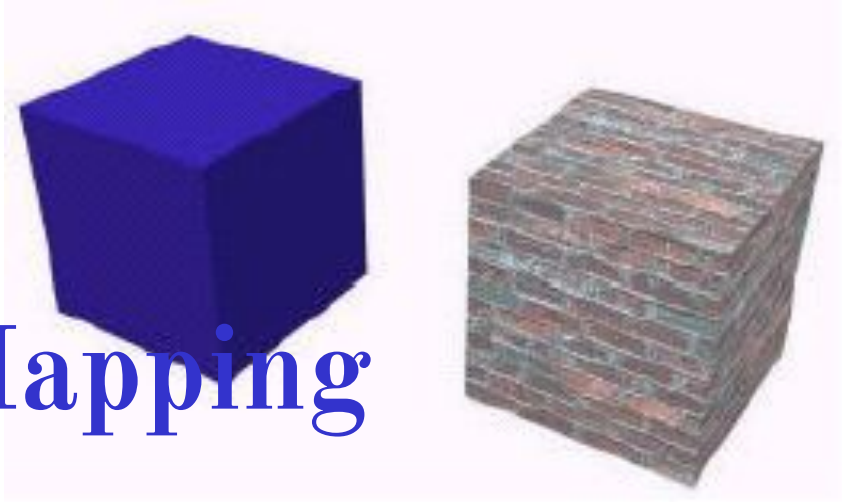
Si interpolano i vettori normali anziché i colori.

+

Equazione di illuminazione calcolata per ogni pixel.



Texture Mapping



Il Texture Mapping

Il Texture Mapping

Scopo= Modellare superfici non lisce o non semplicemente colorate.

Il Texture Mapping

Si mappa un'immagine (texture) sulla superficie di un oggetto.

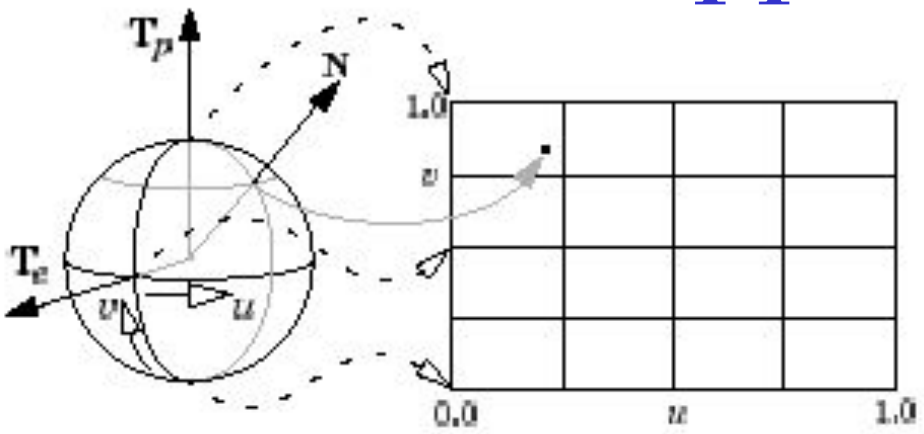
Spazio della texture: (u, v)

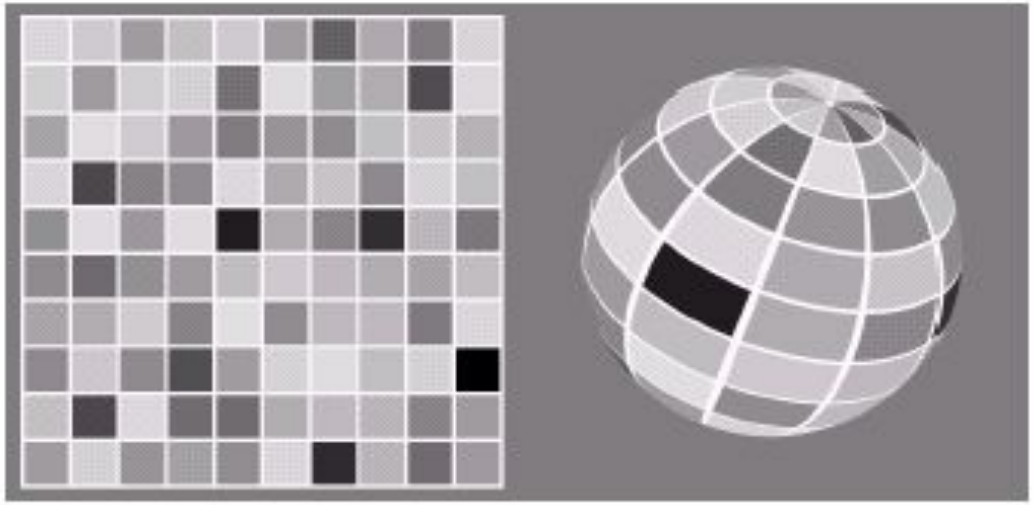
Spazio dell'oggetto: (θ, ϕ)

Si deve determinare la funzione di mappatura:

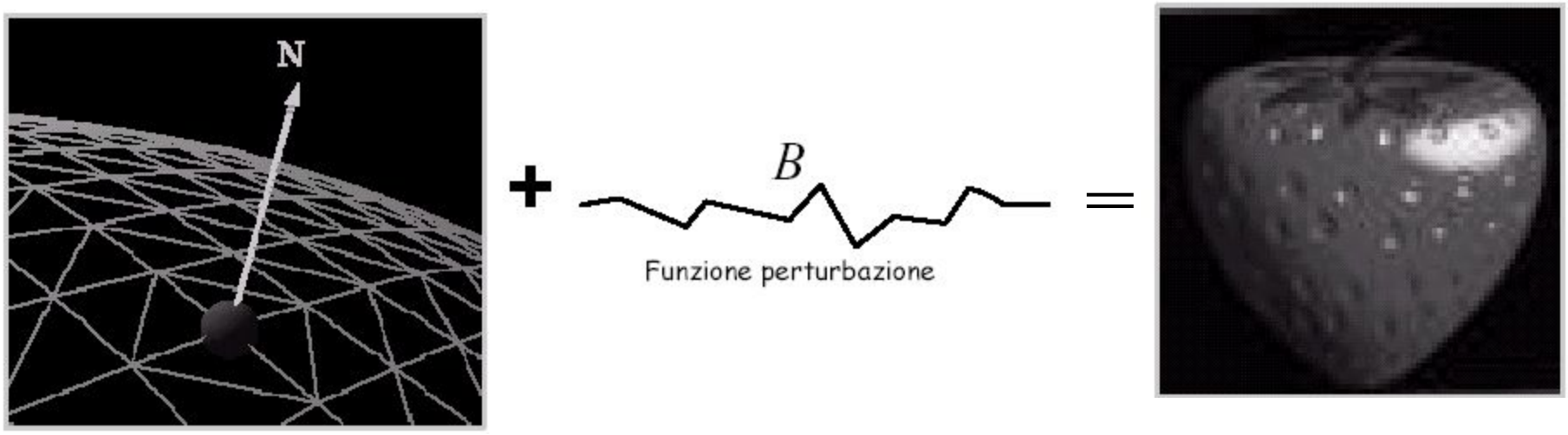
$$u = f(\theta, \phi) \qquad v = g(\theta, \phi)$$

Il Texture Mapping





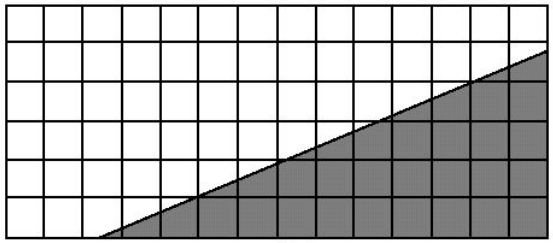
Il Bump Mapping



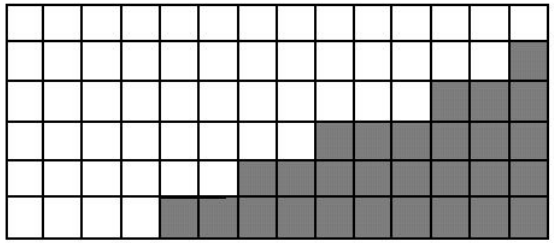
L'Aliasing

Cos'è l'aliasing?

a) Rappresentazione discreta è approssimativa!
b) Se si vuole ricostruire la distribuzione della luce si ha distorsione.



reality



image

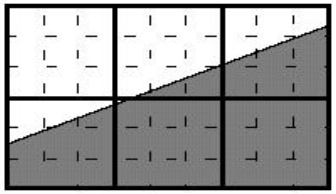
L'Anti-Aliasing

Cos'è l' anti-aliasing?

Metodo per ridurre gli effetti del campionamento!

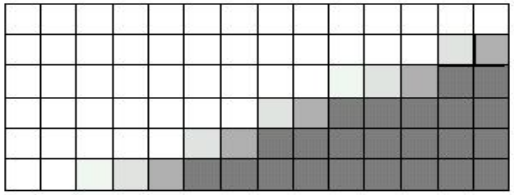
Filtraggio:

1) **box** –media delle intensità;
2) **gaussiano** – media pesata in fz. della distanza del pixel dal centro.

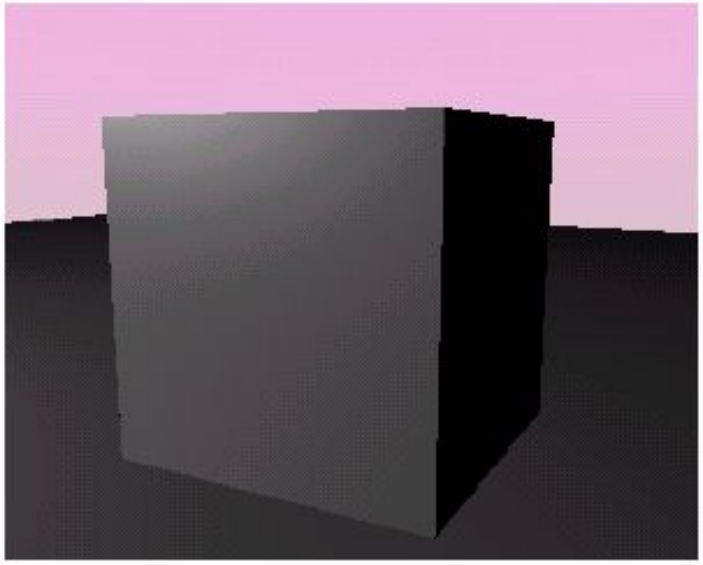


Pixel values

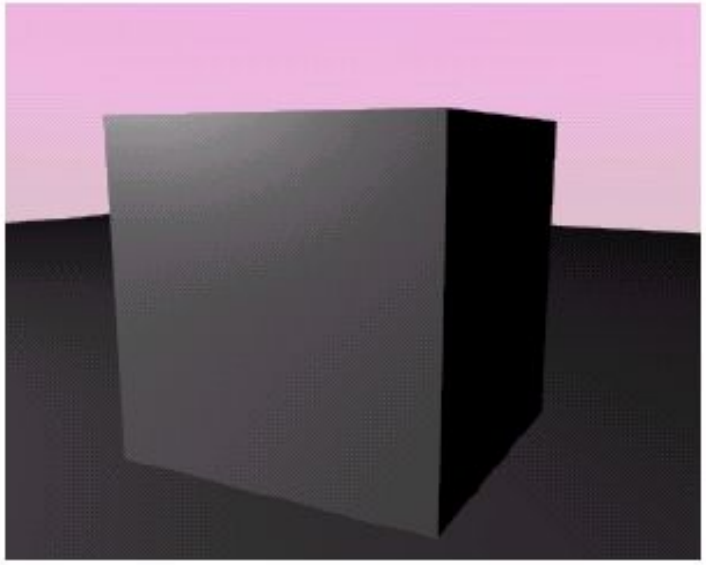
0	1/9	5/9
7/9	1	1



L'Anti-Aliasing



(1) Aliasing

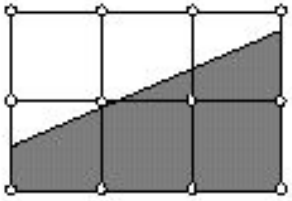


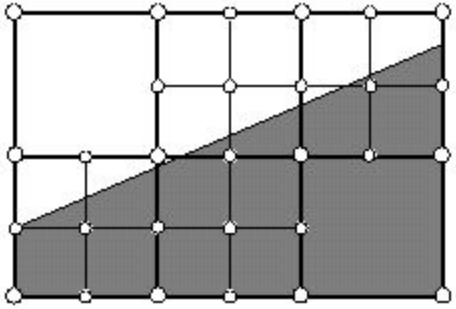
(2) Anti aliasing

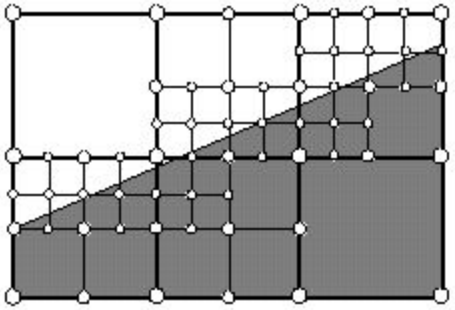
L'anti-aliasing

Altri metodi:

Supercampionamento adattativo

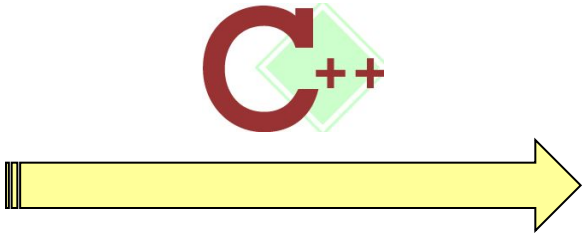






Outline

- **OpenGL** (Open Graphics Library www.opegl.org)
  - Esempi
  - Applicazione OpenGL in Microsoft Visual Studio 6.0
- **VTK** (Visualization ToolKit www.kitware.com)
  - Applicazione VTK in Microsoft Visual Studio 6.0

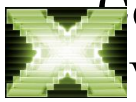


Esistono diverse librerie avanzate per fare grafica 3D interattiva. Le due piu' utilizzate sono :

OpenGL (introdotta dalla Silicon Graphics IRIS GL)

- Basso livello
- Sviluppo semi-aperto
- Multipiattaforma
- Semplici
- Standard scelto dall'industria della computer graphics
- Non Object-Oriented

Direct3D (Microsoft)



- Alto livello
- Componente della librerie multimediale DirectX
- Sviluppo proprietario
- Supporto della sola piattaforma Windows
- Object-Oriented (COM component object model)

- ✔ OpenGL e' una "**rendering library**" (pipeline di visualizzazione).
- ✔ Non prevede strutture dati per la gestione degli oggetti.
- ✔ Lavora in "**immediate mode**" (non ha una pipeline di elaborazione).
- ✔ Se si vuole costruire e modificare oggetti complessi si devono usare librerie ad alto livello sviluppate sopra OpenGL (Es. OpenInventor, VTK,...).

Generalmente si possono fare due cose con OpenGL:

1. Disegnare qualcosa.
2. Modificare il modo con cui OpenGL disegna.

OpenGL e' una macchina a stati !

le *variabili di stato* guidano il processo di rendering.

Colore
Tipo di rendering (flat,phong,..)
Luci, etc...

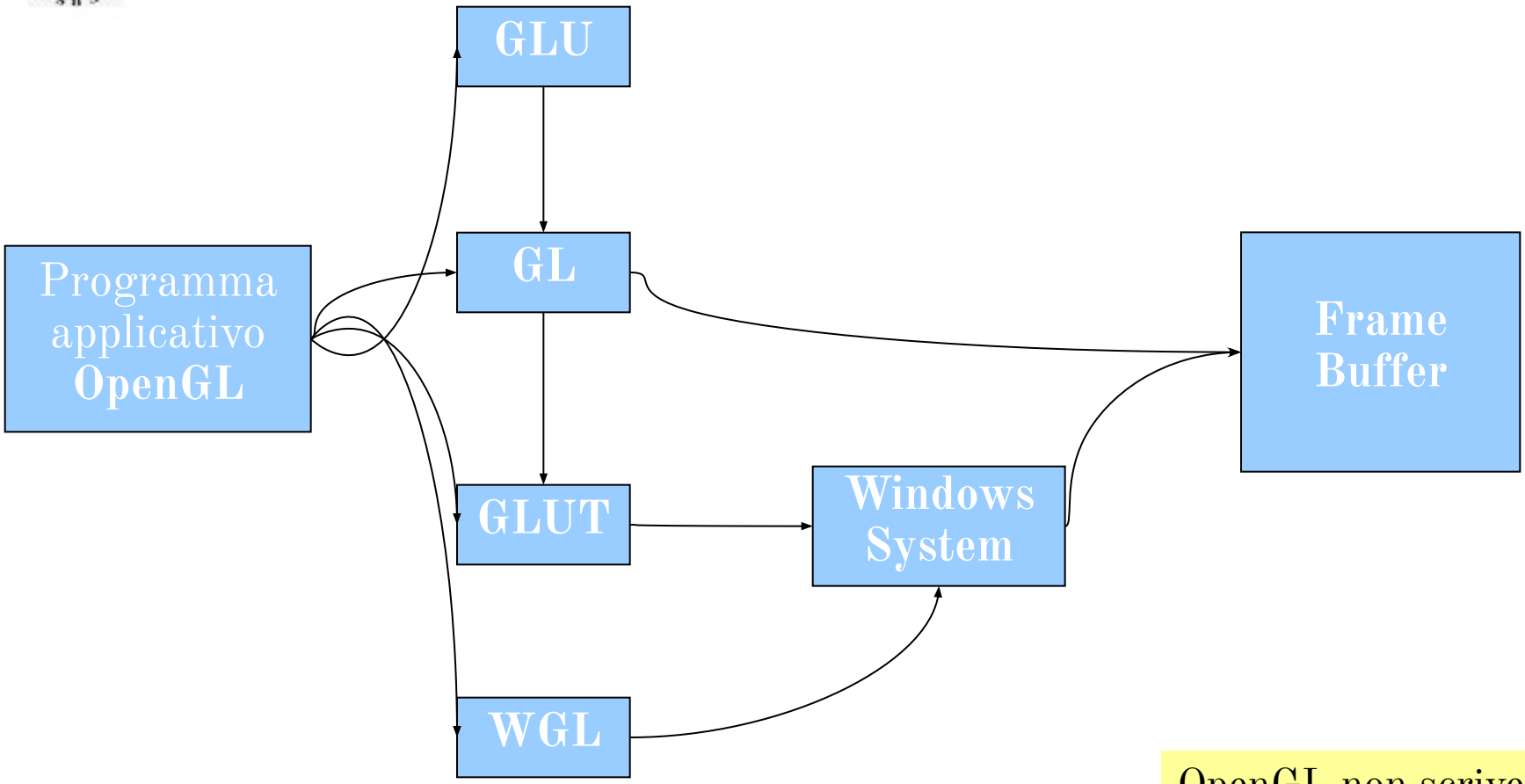
E' stata progettata per essere hardware&windowing&platform independent.

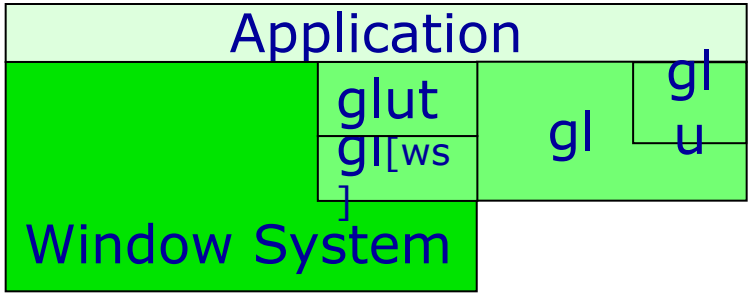
- ✔non ha accesso diretto alla memoria video (**frame buffer**)
- ✔non ha istruzioni per gestire finestre e input (**GLUT**)

L'ambiente di sviluppo per OpenGL e' costituito da un insieme di librerie:

- ✔**gl** (OpenGL)
- ✔**glu** (Utility)
- ✔**gl[ws]** (Funzioni dedicate per il particolare ws:Window System)
- ✔**glut** (Interfaccia OpenGL con il Window System del S.O)

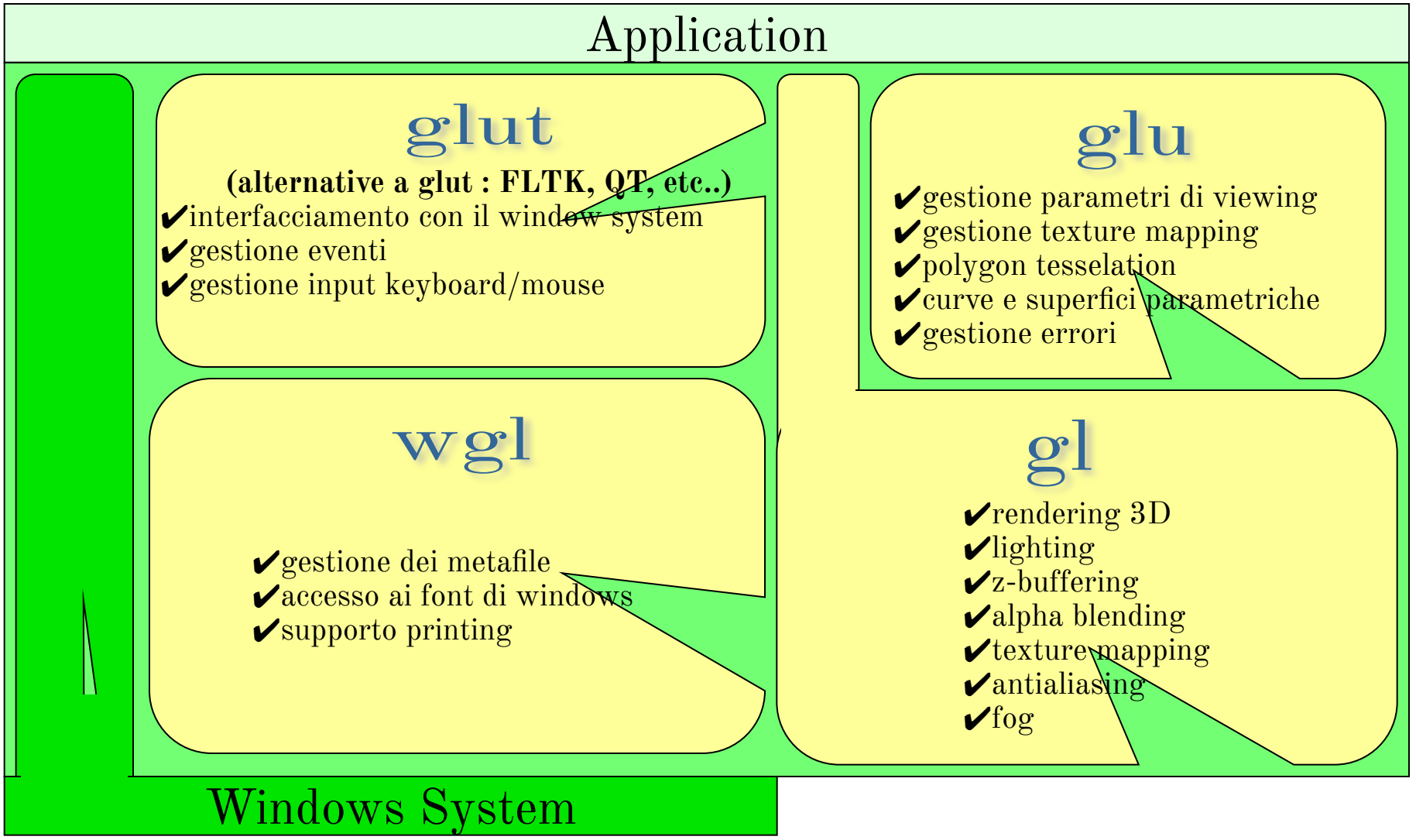
Librerie





OpenGL non scrive direttamente sullo schermo ma accede ad un frame buffer.

Piu' nel dettaglio...

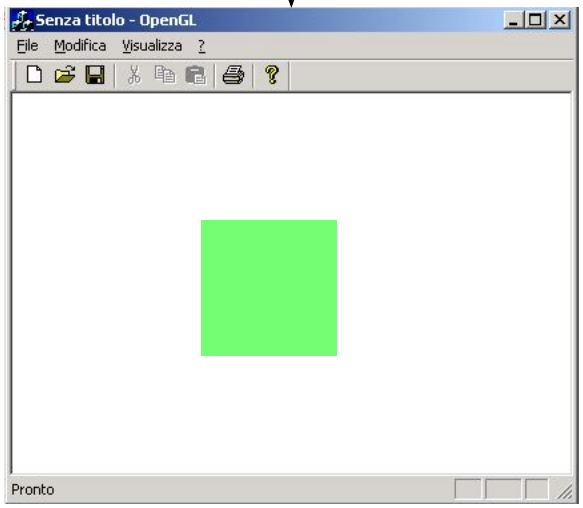


I/O

L'uso di librerie come GLUT per la gestione della GUI e' da preferire quando si vogliono scrivere applicazioni con interfaccia grafica multipiattaforma.

In generale librerie dedicate al window system del particolare SO (es: MFC per Windows) permettono una maggiore integrazione con il SO.

glut/wgl



3D API

GLUT 2/2


```
int main(int argc, char** argv)

glutInit(&argc, argv);            /* Inizializzazione GLUT */
glutInitDisplayMode (GLUT_RGBA| GLUT_DOUBLE );

glutInitWindowSize (500, 500);          /* Creazione della finestra */
glutCreateWindow ("OpenGL Test");

init ();                          /* Inizializzazione applicazione */

glutReshapeFunc(resize);                /* Specifica la funzione di rendering */
glutDisplayFunc(display);

glutKeyboardFunc(keyboard);         /* Specifica la funzione di input da tastiera */

glutIdleFunc(idle);                /* Specifica la funzione di idle */

glutMainLoop();                   /* Entra nel loop principale dell'applicazione */

return 0;
}
```

C:\Documents and Settings\Administrator\Desktop\esempi\GLUTWindow.exe

Convenzioni

- Tutti i **comandi** hanno il prefisso **gl.**

- **Costanti** e **variabili di stato** sono scritte tutte in maiuscolo, iniziano con **GL** e le parole sono separate da '_'.
  Es: GL_COLOR_BUFFER_BIT.

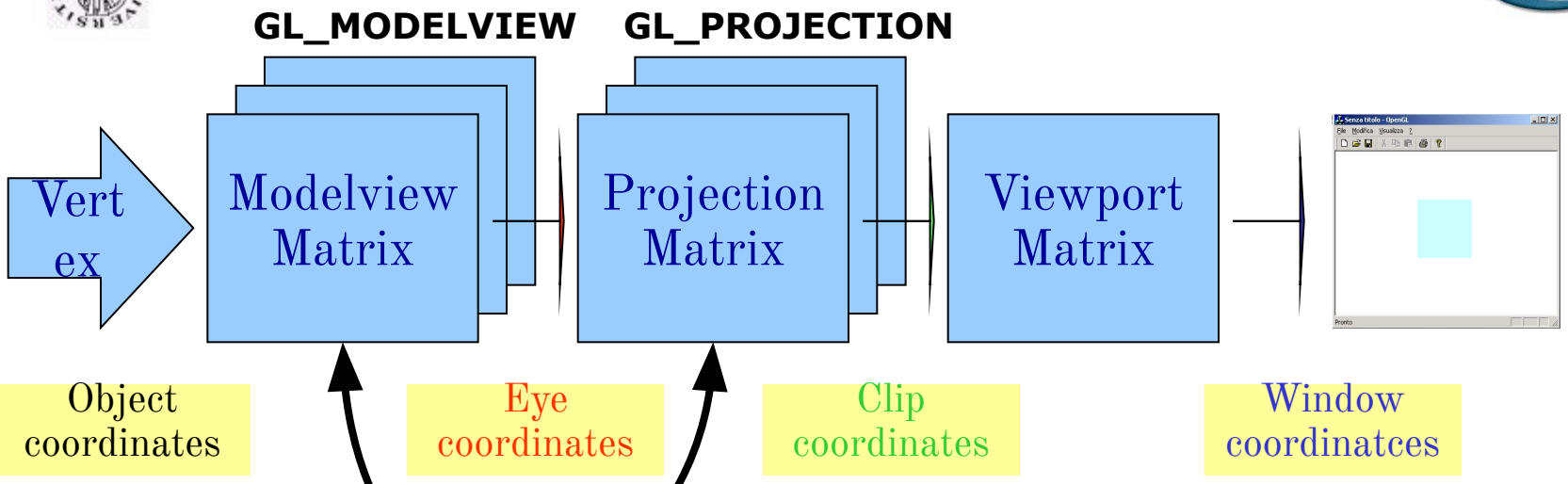
- I **tipi** in OpenGL hanno dei nomi interni che incominciano per GL, del tipo: GLbyte, GLshort, GLint, GLfloat, GLdouble

- Per operare un overloading di funzioni in C, le OpenGL dichiarano la stessa funzione con nome diverso per indicare il tipo ed il numero di argomenti.

Es: 1. glVertex3f(GLfloat x, GLfloat y, GLfloat z)
definisce un vertice (punto) con tre coordinate float

2. glVertex2i(GLint x, GLint y)
definisce un vertice (punto) con due coordinate intere

Stack di matrici



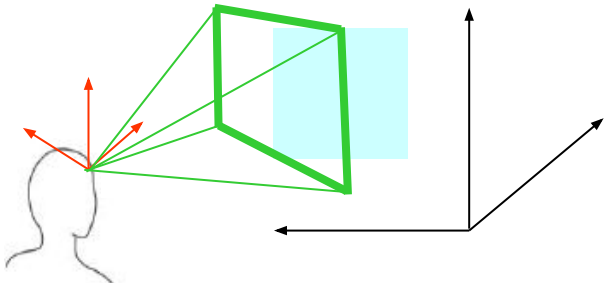
$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

La matrice corrente e' quella in cima allo stack.

- Le OpenGL introducono tre stack di matrici
- Gli stack si possono manipolare uno alla volta; per selezionare lo stack su cui si vuole lavorare si usa :

void **glMatrixMode**(GLenum *mode*)

mode = [**GL_MODELVIEW, GL_PROJECTION**]



glLoadIdentity() : Inizializza all'identita'

glLoadMatrix(matrice): Carica una matrice 4x4

glMultMatrix(matrice) : Moltiplica, a destra, la matrice corrente per la matrice specificata

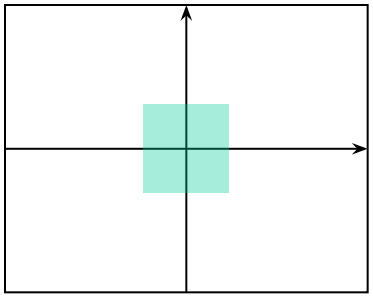
Trasformazioni (Modelview matrix)

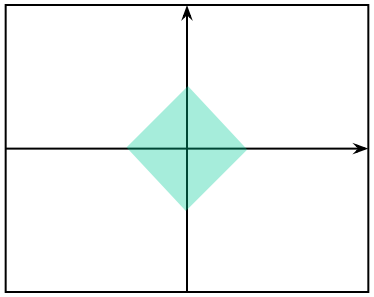
- **glTranslate(Dx, Dy, Dz) :** Trasla di (Dx, Dy, Dz)
- **glRotate(angle, x, y, z)** : Ruota di **angle** (antiorario) rispetto al vettore (x,y,z).
- **glScale(Sx, Sy, Sz)** : Scala sui tre assi di (Sx,Sy,Sz).

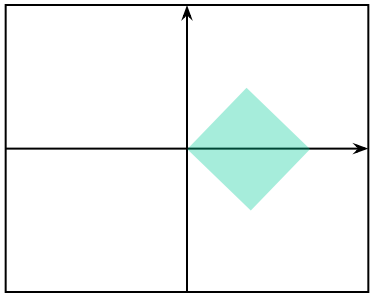
```
void main()
{
  DrawSquare();
}
```

```
void main()
{
glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glRotatef(45.0, 0.0, 0.0, 1.0);
   DrawSquare();
}
```

```
void main()
{
glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glRotatef(45.0, 0.0, 0.0, 1.0);
   glTranslate(1.0, 0.0, 0.0);
   DrawSquare();
}
```







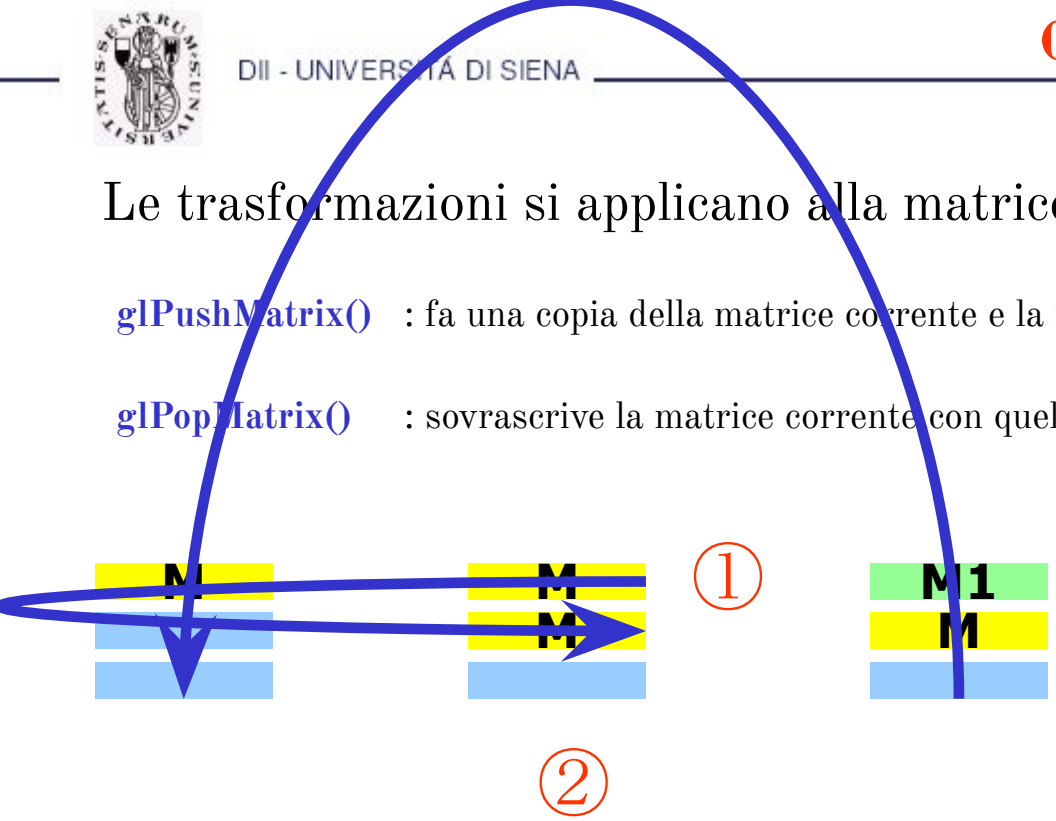
transformation.exe

Operazioni sugli stack

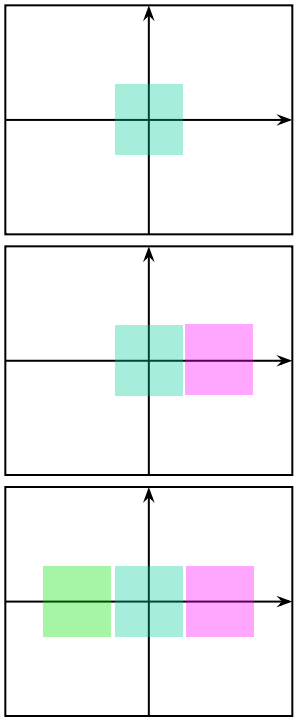
Le trasformazioni si applicano alla matrice in cima allo stack corrente.

glPushMatrix() : fa una copia della matrice corrente e la inserisce nello stack.

glPopMatrix() : sovrascrive la matrice corrente con quella che segue nello stack.




```
void main()
{
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();                  // M
  DrawSquare();
  ① glPushMatrix();
    glTranslatef(2.0, 0.0, 0.0); //M1
    DrawSquare();
  ② glPopMatrix();
  glTranslatef(-2.0,0.0, 0.0);
  DrawSquare();
}
```



Proiezioni (Projection matrix)

- Si modifica la matrice di proiezione.

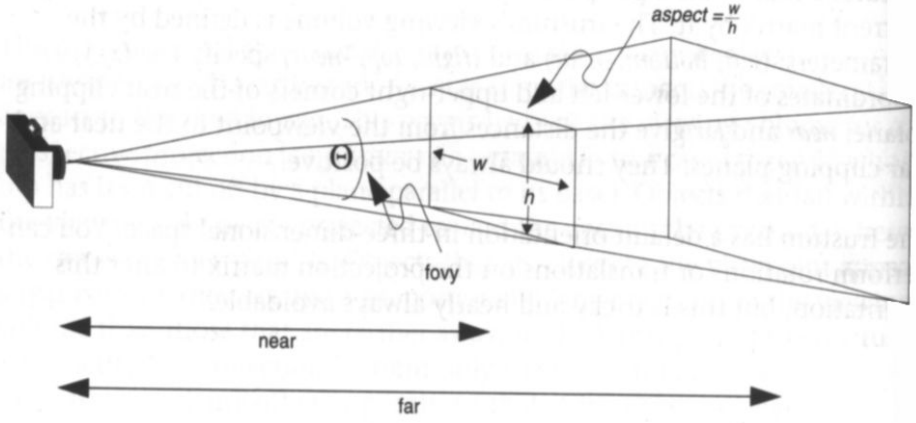
glMatrixMode(GL_PROJECTION)

Prospettiche

- **glFrustum**(left,right,bottom,top,near,far)
  definisce una matrice di proiezione dando i clipping planes (near e far) e la finestra (sul piano near).

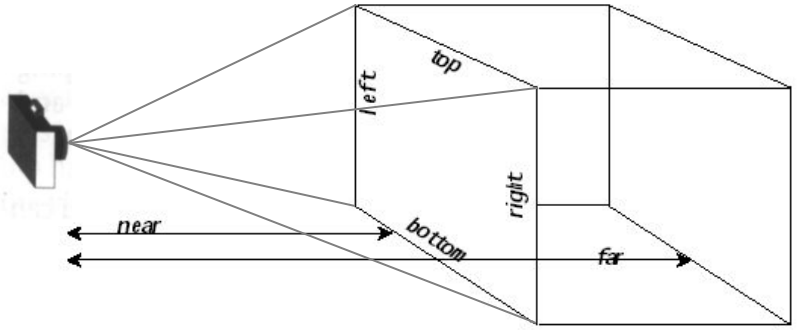
In alternativa usando la libreria GLU:

- **gluPerspective**(fovy,aspect,near,far)
  definisce una matrice di proiezione dando l'angolo di apertura (fovy) sul piano xz, il rapporto della finestra w/h (aspect ratio) i clipping planes (near e far).



Parallele

- **glOrtho**(left,right,bottom,top,near,far)
  definisce una matrice di proiezione dando i clipping planes (near e far) e la finestra (sul piano near).

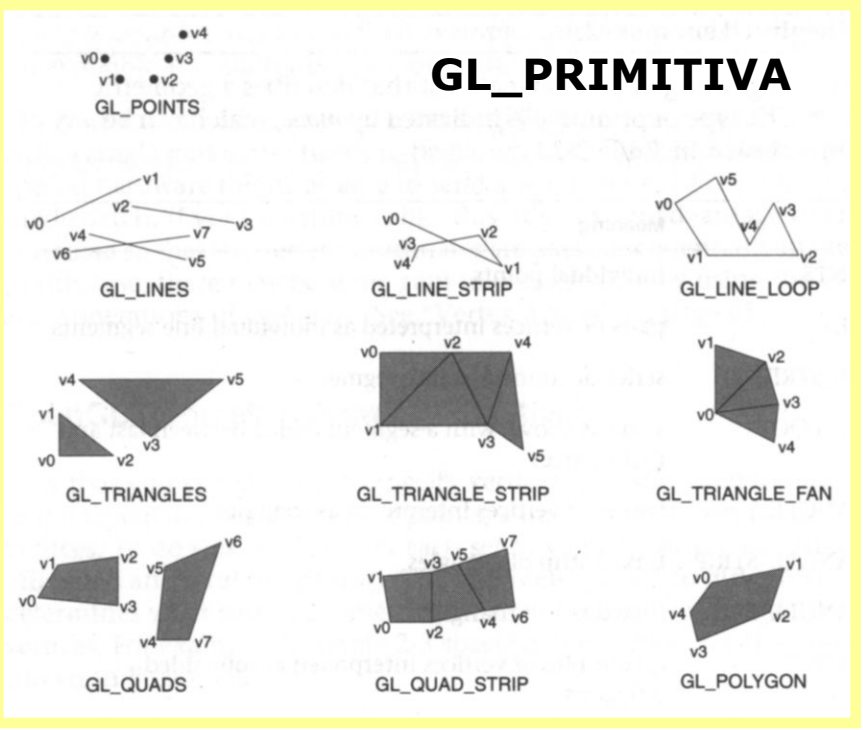


projection.exe (f,o,p)

Primitive grafiche

- Le primitive grafiche in OpenGL sono tutte specificate da una lista di vertici.
- Ogni primitiva viene disegnata in modo diverso.

```
glBegin(GL_PRIMITIVA)
    glVertex3f(x,y,z)
    <lista dei vertici>
glEnd()
```



POINTS: ogni vertice passato rappresenta un punto nello spazio;

LINES: ogni coppia di vertici consecutivi definisce un segmento passato all pipeline grafica. Se i vertici sono dispari, l'ultimo non viene considerato

LINE STRIP: e' come il punto precedente, ma i segmenti si considerano collegati,quindi ogni vertice costituisce un segmento con il vertice precedente.

LINE LOOP: come il precedente, ma l'ultimo vertice ed il primo sono connessi e formano un segmento.

TRIANGLES: questa primitiva costruisce un triangolo per ogni terna di vertici consecutivi che contiene. Se il numero di vertici non `e multiplo di tre gli ultimi vengono ignorati.

TRIANGLE STRIP: come la precedente, ma i primi tre vertici definiscono il primo triangolo, ogni vertice successivo definisce un triangolo con i due vertici precedenti

TRIANGLE FAN: come la precedente, ma con un vertice a comune a tutti i triangoli

QUADS: costruisce un quadrilatero con quattro vertici consecutivi.

QUAD STRIP: come TRIANGLE STRIP (vengono utilizzati quadrilateri)

POLYGON : i vertici vengono uniti secondo l'ordine con cui vengono definiti nella lista vertici.

shapes.exe

Frame Buffer

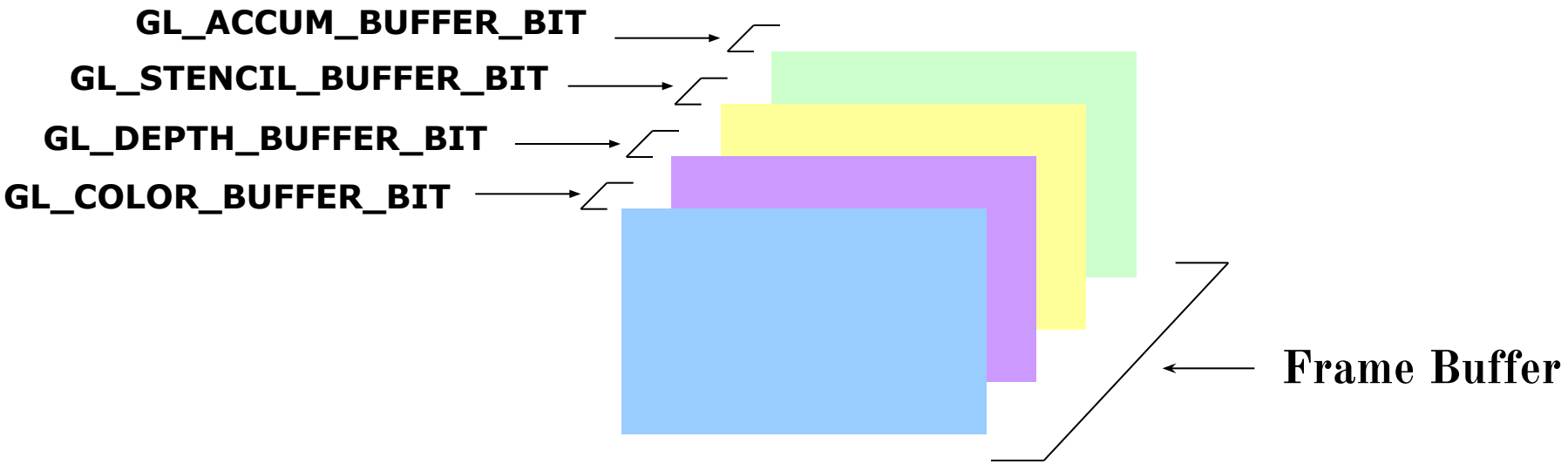
Il framebuffer di OpenGL e' composto in realta' da un insieme di buffers, che sono:

Color buffers : Viene resa l'immagine (almeno 2, anche di piu' per il rendering stereo.

Depth buffer : Usato per eliminare le parti nascoste.

Stencil buffer : Usato per "ritagliare" aree dello schermo.

Accumulation buffer : Accumula i risultati per effetti speciali come Motion Blur, simulare il fuoco, ombre, ...



Double Buffering

- OpenGL viene utilizzata in modo intensivo per fare delle animazioni.
- Rendere fluida una animazione richiede di disegnare sullo schermo molte immagini in sequenza ad un ritmo di almeno 30 al secondo.

! Si nota che lo schermo viene pulito.

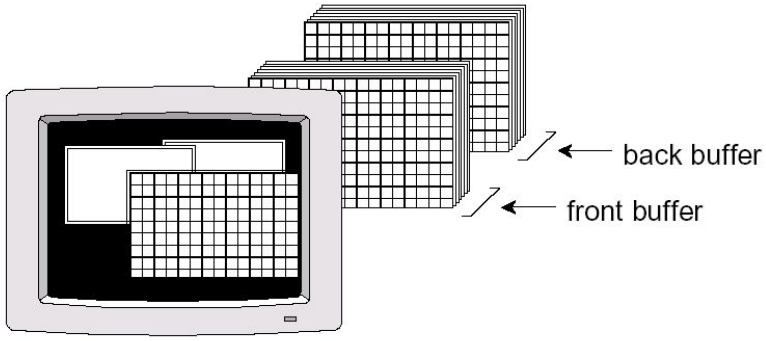
! Si vede mentre l'immagine viene disegnata.

! L'immagine non e' stabile poiche' lo schermo viene continuamente aggiornato.

→ **FLICKERING (SFARFALLIO)**

Il double buffering risolve il problema dello sfarfallio con la seguente tecnica:

- *Usa un secondo buffer dove viene effettuato il disegno.*
- *Disegna sul secondo buffer.*
- *Sostituisci il secondo buffer al primo solo dopo che e' stato completamente disegnato.*



In OpenGL il double buffering viene gestito da GLUT (o dal sistema operativo). In GLUT abbiamo i comandi:

glutInitDisplayMode(GLUT_DOUBLE)

Dichiara che si useranno due buffers.

glutSwapBuffers()

Scambia i due buffers, presentando il secondo sullo schermo.

C:\Documents and Settings\Administrator\Desktop\esempi\no-doublebuffer.exe

Z-Buffering (Depth Buffer)

Per default, OpenGL non elimina le parti nascoste, ma disegna gli oggetti nell'ordine in cui li incontra. Per eliminare le parti nascoste bisogna :

Inizializzare la finestra predisponendola per fare il controllo sulla profondita':

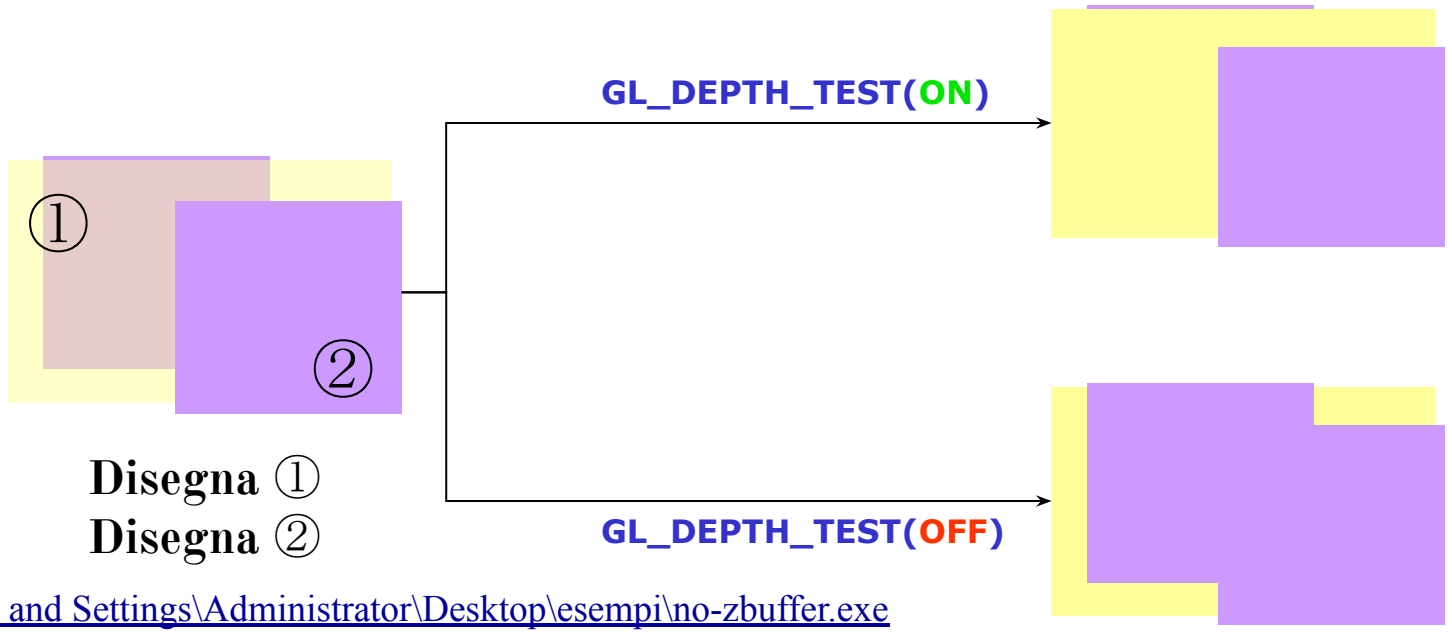
glutInitDisplayMode(GLUT_DEPTH | .....)

rendere attivo il Depth Buffer:

glEnable(GL_DEPTH_TEST)

Prima di usare il buffer bisogna pero' inizializzarlo con:

glClear(GL_DEPTH_BUFFER_BIT)



C:\Documents and Settings\Administrator\Desktop\esempi\no-zbuffer.exe
C:\Documents and Settings\Administrator\Desktop\esempi\si-zbuffer.exe

Inizializzazione dei Buffer

Inizializza il buffer (o i buffers) indicati in *mask* usando i clear values correnti.

void **glClear**(GLbitfield *mask*)

GL_STENCIL_BUFFER_BIT
GL_DEPTH_BUFFER_BIT
GL_COLOR_BUFFER_BIT
GL_ACCUM_BUFFER_BIT

Es. glClear(**GL_STENCIL_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);**

Si possono settare i clear values correnti con :

- **glClearColor** setta il colore RGBA da usare per l'operazione di glClear del Color Buffer.
- **glClearDepth** setta il valore di profondità da usare per l'operazione di glClear del Depth Buffer.

OpenGL possiede due modalita' di gestione del colore

INDICIZZATA

(Si definisce una Look Up Table di colori)

- selezionare la modalita'
  `glutInitDisplayMode(GLUT_INDEX);`
- definire le entry della LUT
  `glutSetColor(index, r, g, b);`
- settare il colore per il buffer clearing
  `glClearIndex(index);`
- settare il colore corrente per disegnare
  `glIndexi(index);`

index	R	G	B
0	1	0	0
1	0.1	0.5	1
2	1	1	0.5
	:		

RGB

- selezionare la modalita'
  `glutInitDisplayMode(GLUT_RGB);`
- settare il colore per il buffer clearing
  `glClearColor(r, g, b,1.0);`
- settare il colore corrente
  `glColor3f(r, g, b);`

Permette di visualizzare la scena in modo realistico.
Per definire l'illuminazione e' necessario:

- fornire le normali sui vertici (o su poligoni)
- creare, selezionare e posizionare le sorgenti luminose
- definire le proprieta' del materiale associato agli oggetti
- creare e selezionare un modello di illuminazione

Per default l'illuminazione non e' attiva. Per renderla attiva bisogna dare i comandi:

glEnable(GL_LIGHTING); *Attiva l ' illuminazione.*
glEnable(GL_LIGHT0); *Accende la luce 0. Necessario affinche' le luci illuminino la scena.*

Quando e' attiva l'illuminazione il modello di disegno deve essere **GL_SMOOTH**:

glShadeModel(GL_SMOOTH);

..\doc-opengl\opengl\redbook\redbook\Debug\material.exe

Normali

Le normali uscenti dai vertici servono per calcolare quanta luce colpisce il vertice.

Tutte le primitive glut solide (non wire) hanno gia' definite queste normali (Es. glutSolidCube()).

```
glBegin(GL_POLYGON);
    glNormal3f(0.0 ,0.0, 1.0);glVertex3f(0.25, -0.25, 0.0);
    glNormal3f(1.0 ,0.0, 1.0); glVertex3f(0.75, 0.25, 0.0);
    glNormal3f(0.0 ,1.0, 1.0); glVertex3f(0.75, 0.75, 0.0);
    glNormal3f(1.0 ,1.0, 1.0); glVertex3f(0.25, 0.75, 0.0);
glEnd();
```

```
glBegin(GL_POLYGON);
    glNormal3f(0.0 ,0.0, 1.0);
    glVertex3f(0.25, -0.25, 0.0);
    glVertex3f(0.75, 0.25, 0.0);
    glVertex3f(0.75, 0.75, 0.0);
    glVertex3f(0.25, 0.75, 0.0);
glEnd();
```

Sorgenti luminose 1/2

Una sorgente luminose viene definita da :

glLightfv(Glenum light, Glenum pname, TYPE *param)

dove :

- light e' il nome della luce (GL_LIGHT0 .. GL_LIGHT7)
- pname e' il parametro da settare
- param e' un vettore di valori che caratterizza il parametro pname

pname = [GL_AMBIENT | GL_DIFFUSE | GL_SPECULAR]

definiscono le caratteristiche della luce

pname = [GL_POSITION | GL_SPOT_DIRECTION | GL_SPOT_CUTOFF]

definiscono posizione (direzione se all'infinito) e orientamento (solo se al finito) della luce.

Ambiente : Luce dispersa dalla riflessione sulle superfici tale che non ha una direzione, ma illumina uniformemente tutti gli oggetti.

Diffusa : Luce che colpisce un oggetto e viene diffusa in tutte le direzioni.

Speculare : Luce che colpisce un oggetto e viene diffusa principalmente in una direzione.

Sorgenti luminose 2/2

Definire i parametri di una sorgente luminosa:

```
GLfloat Light0Position[]   = { 0.0, 200.0, 0.0, 1.0 };
GLfloat Light0Direction[]  = { 0.0, -200.0, 0.0, 1.0 };
GLfloat Light0Cut[]        = { 30.0 };
GLfloat Light0Ambient[]    = { 0.0, 1.0, 0.0, 1.0 };
GLfloat Light0Diffuse[]    = { 0.0, 1.0, 0.0, 1.0 };
GLfloat Light0Specular[]   = { 0.0, 1.0, 0.0, 1.0 };


glLightfv(GL_LIGHT0,GL_POSITION,       Light0Position );
glLightfv(GL_LIGHT0,GL_SPOT_DIRECTION,Light0Direction );
glLightfv(GL_LIGHT0,GL_SPOT_CUTOFF,    Light0Cut );
glLightfv(GL_LIGHT0,GL_AMBIENT,        Light0Ambient );
glLightfv(GL_LIGHT0,GL_DIFFUSE,        Light0Diffuse);
glLightfv(GL_LIGHT0,GL_SPECULAR,       Light0Specular);

glEnable (GL_LIGHT0 );
```

Materiali 1/2

Specificare il materiale di un oggetto significa definirne il comportamento quando viene illuminato. Viene usato il comando:

glMaterialfv(Glenum face, Glenum pname, TYPE *param)

dove :

- face e' la faccia (front, back o entrambe) alla quale viene associato il materiale.
- pname e' il parametro da settare.
- param e' un vettore di valori che caratterizza il parametro pname

face = [GL_FRONT | GL_BACK | GL_FRONT_AND_BACK]

pname = [GL_AMBIENT | GL_DIFFUSE | GL_SPECULAR | GL_SHININESS | GL_EMISSION]

Materiali 2/2

```
GLfloat MaterialAmbient[]   = { 0.0, 0.3, 0.0, 1.0 };
GLfloat MaterialDiffuse[]   = { 0.0, 0.5, 0.0, 1.0 };
GLfloat MaterialSpecular[]  = { 0.0, 1.0, 0.0, 1.0 };
GLfloat MaterialShininess[] = { 80.0 };

glMaterialfv(GL_FRONT,GL_AMBIENT,   MaterialAmbient);
glMaterialfv(GL_FRONT,GL_DIFFUSE,   MaterialDiffuse);
glMaterialfv(GL_FRONT,GL_SPECULAR,  MaterialSpecular);
glMaterialfv(GL_FRONT,GL_SHININESS, MaterialShininess);

Draw();
```

lightmaterial.exe

Es. GLUTWindow.dsw

OpenGL in Microsoft Visual Studio 6.0

Introduzione

Il VTK e' un software :

- object-oriented
- open-source
- multipiattaforma (Windows, Unix-like)
- usa OpenGL

orientato alla:

- computer graphics
- visualizzazione
- elaborazione delle immagini

Il VTK e' costituito da due sottosistemi di base :

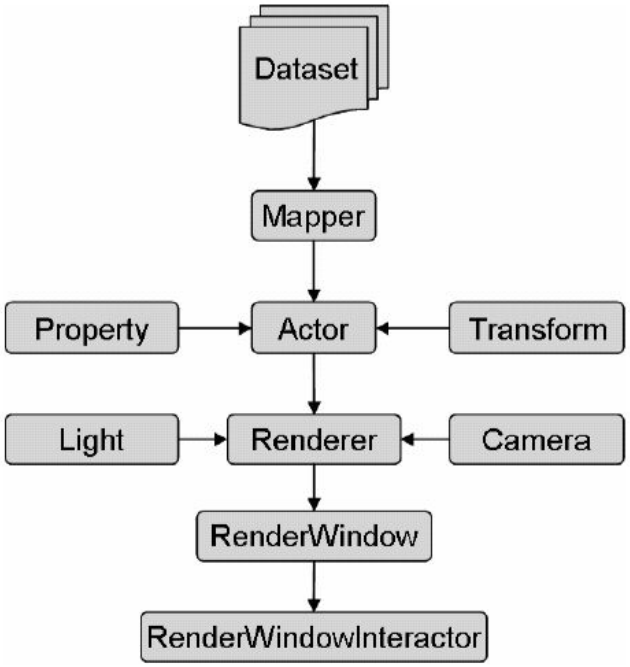
- un insieme di librerie compilate in C++, che costituiscono il cuore del sistema.
- ed un livello piu' alto,che ne permette l'utilizzo anche con linguaggi interpretati, come : Java, Tcl, Python.

Le classi sono organizzate in due categorie, che si occupano di :

- graphics model
- visualization model

Pipeline grafica

- Il paradigma utilizzato per descrivere una scena tridimensionale e' quello del set cinematografico, cosi in VTK troviamo gli oggetti: Actor, Light e Camera.
- Gli attori rappresentano le entita' visibili nella scena e sono associati ai mappers che ne forniscono la rappresentazione grafica (trasforma i dati da VTK a OpenGL).



Dataset : Sorgente dei dati (lettura da file, generazione dati)

Mapper : Mappa i dati da VTK a OpenGL

RenderWindow : Interfaccia VTK con il window system del SO

RenderWindowInteractor : Abilita l'interazione utente/scena VR

Actor : Entita' presente nella scena VR

- **Property** : propieta' dell'Actor (shading, color, etc..)
- **Transform** : rotazioni, traslazioni, resize dell'Actor.

Renderer : Esegue il rendering della scena

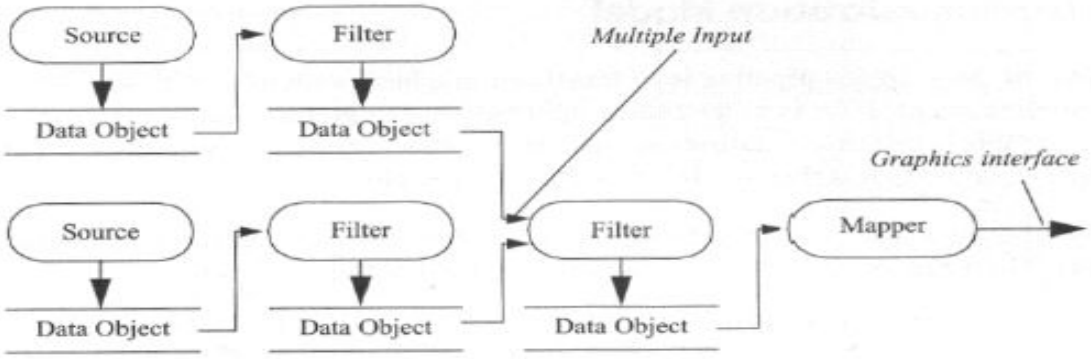
- **Light** : luci di scena
- **Camera** : gestione della telecamera (prospettica, parallela)

Pipeline visualizzazione

La pipeline di visualizzazione si occupa di costruire una rappresentazione geometrica dei dati, che viene poi resa attraverso la pipeline grafica.

- L' approccio usato e' quello data flow che coinvolge due oggetti di base :
  - **vtkDataObject**
  - **vtkProcessObject**
- Si e' scelto di separare le classi che contengono i dati, da quelle che elaborano.
- I vtkProcessObject costituiscono, dei generici filtri che operano sui vtkDataObject. Queste due classi di oggetti vengono interconnesse, cosi' da formare la pipeline di visualizzazione (data-flow network).
- La connessione tra due process object avviene di solito tramite l'istruzione

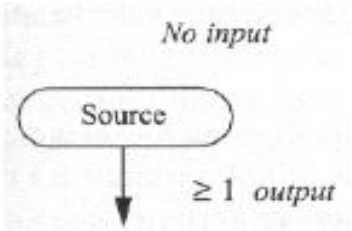
FilterA □ **SetInput(**FilterB □ **GetOutput());**



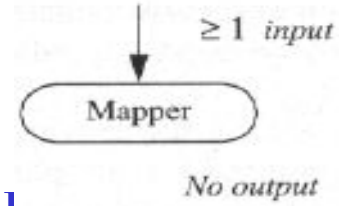
Process Object

Vi sono diversi tipi di process-object.

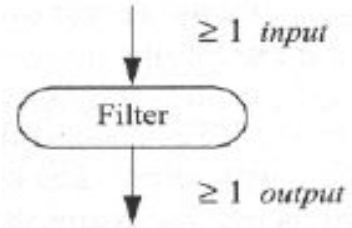
Source-Process : producono dati, leggendoli da file, o generandone di nuovi.

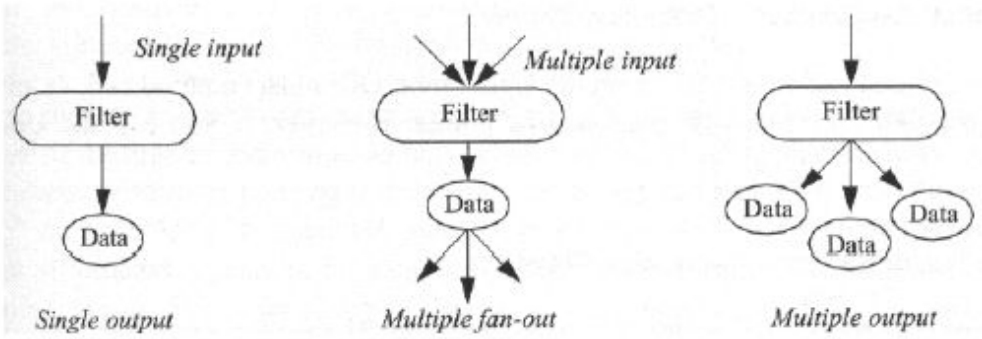


Mappers : trasformano i data objects in graphics data che sono poi resi dal renderer.



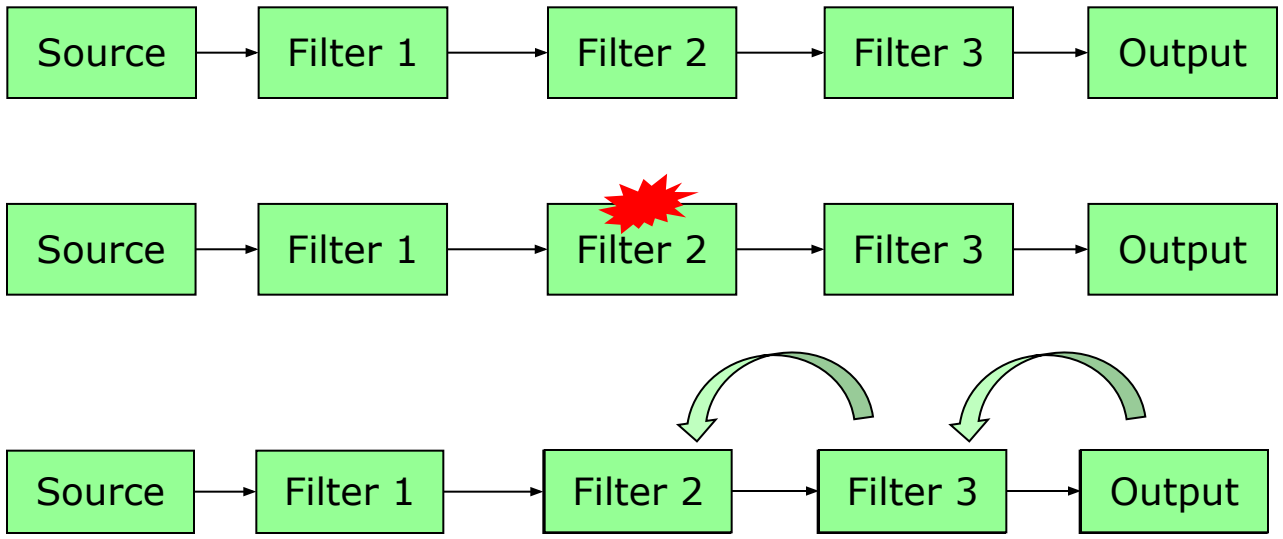
Filter : che necessariamente hanno un numero di ingressi ed uscite ≥ 1





Data-flow network

- Una volta che la data-flow network e' stata costruita e' necessario mandarla in esecuzione, per avviare l'elaborazione e quindi la visualizzazione dei risultati.
- Per fare questo nel VTK e' stato implementato un meccanismo di sincronizzazione basato su time-stamps. La richiesta di dati in uscita, attraverso il metodo **Update()**, provoca una richiesta a catena a tutti gli altri pocess object a monte, la pipeline viene cosi' ripercorsa a ritroso fino a tovare il primo elemento utile alla rielaborazione dei dati. Infatti, per evitare elaborazioni inutili, vengono attivati solo quegli oggetti che hanno visto una modifica dei parametri interni oppure dei dati in ingresso.



Es. Esempiovtk.dsw

VTK in Microsoft Visual Studio 6.0