Grafica Computerizzata

Corso di Laurea in Informatica

Docente: Giovanni Di Domenico

Modelli e Sistemi Grafici

Organizzazione della lezione

- Primi passi della CG
- Aspetti della generazione delle immagini
- Architetture grafiche
- La pipeline grafica
- Introduzione ad OpenGL
- Struttura di applicazione OpenGL
- WebGL, esempi, shaders

Computer Graphics

- La Grafica Computerizzata (CG) si occupa di tutti gli aspetti riguardanti la generazione di una immagine di sintesi mediante l'uso di un Personal Computer.
- Quindi un sistema grafico è costituito da:
  - Hardware
  - Software
  - Applicazioni

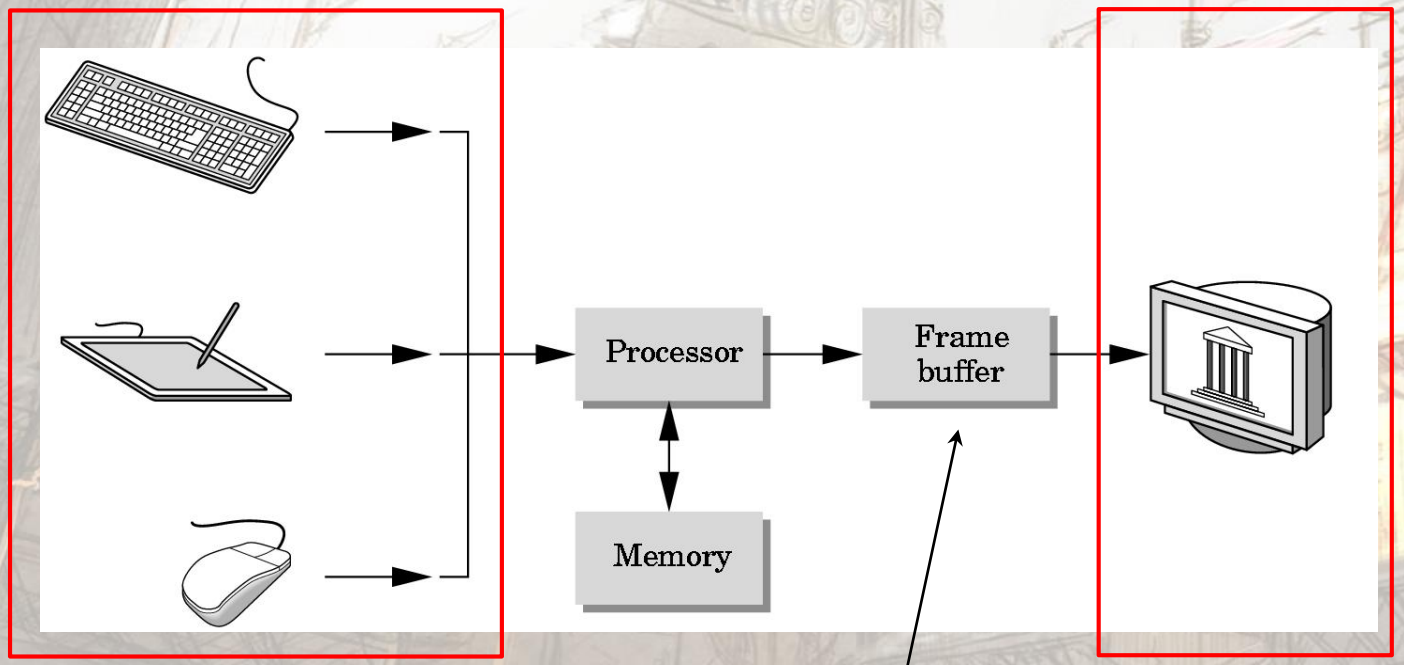
Un esempio

- Con quale software è stato modellato?
- **Software**: Maya per la modellazione ed il rendering ma Maya è sviluppato sopra OpenGL.

- Quale tipo di **hardware** è necessario?
- Un PC con una scheda grafica

- **Applicazione**: L'oggetto è una rappresentazione del sole che un'artista ha realizzato per un'animazione in un planetario.

Un sistema grafico di base



Dispositivi di input

Dispositivo di output

Buffer di memoria per l'immagine generata

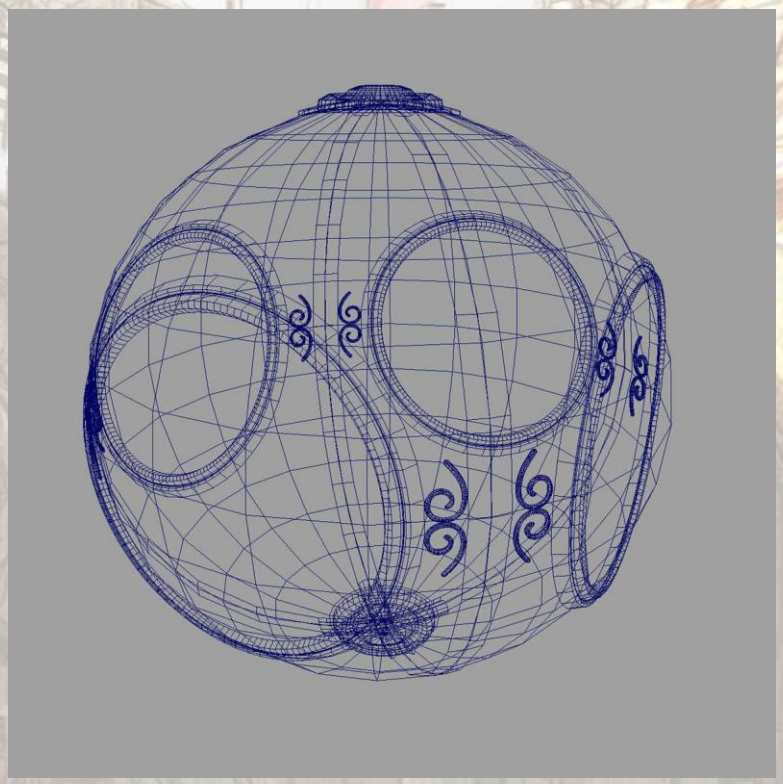
Computer Graphics: 1950-1960

- La grafica computerizzata risale alle origini dei primi dispositivi di calcolo automatico
  - Disegno di carte geografiche
  - Plotters

- La visualizzazione utilizzava un convertitore A/D collegato ad un monitor CRT di tipo vettoriale o calligraphic.
  - I costi per il refresh dei CRT erano troppo alti
  - Computer lenti, costosi e poco affidabili

Computer Graphics: 1960-1970

- *Wireframe* graphics
  - Si disegnano solo le linee
- Sketchpad
- Display Processors
- Introduzione dei tubi catodici



Rappresentazione wireframe

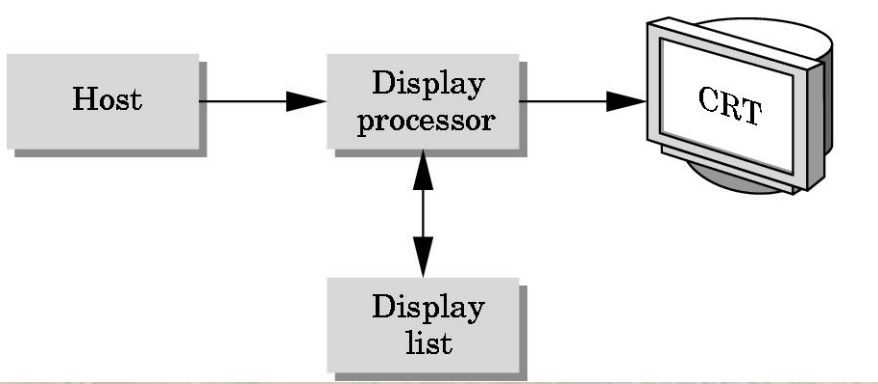
Sketchpad

- Sviluppato da un'idea di Ivan Sutherland durante il lavoro di dottorato al MIT
  - Riconosce il potenziale dell'interazione uomo-macchina
  - Idea del Loop
    - Disegna qualcosa
    - L'utente sposta la penna ottica
    - Il computer disegna una nuova immagine
- Sutherland sviluppò molti dei comuni algoritmi sulla computer graphics

- Il filmato Sketchpad sul sito mostra il lavoro di Sutherland.

Display Processor

- Invece di affidare ad un computer (CPU) il compito di aggiornare il display si impiega un processore specializzato chiamato display processor (DPU).



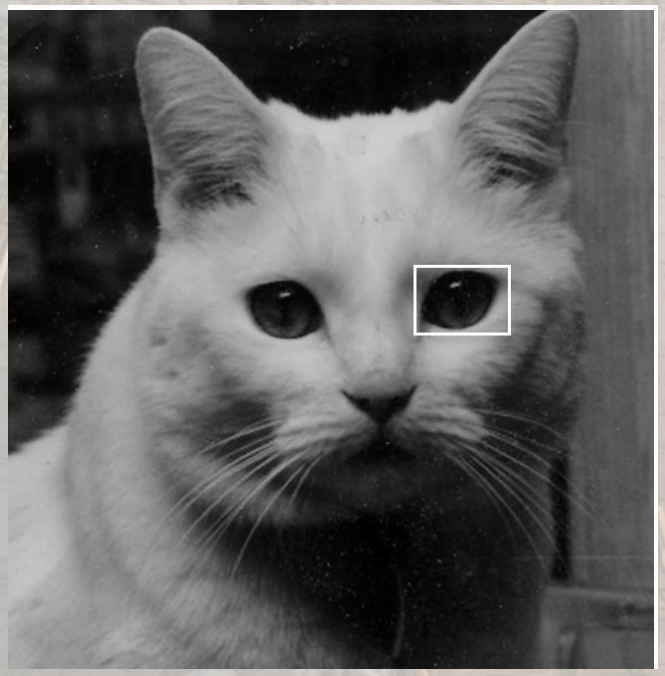
- La grafica è memorizzata in una display list all'interno del display processor
- La macchina ospite compila la display list e la invia al DPU

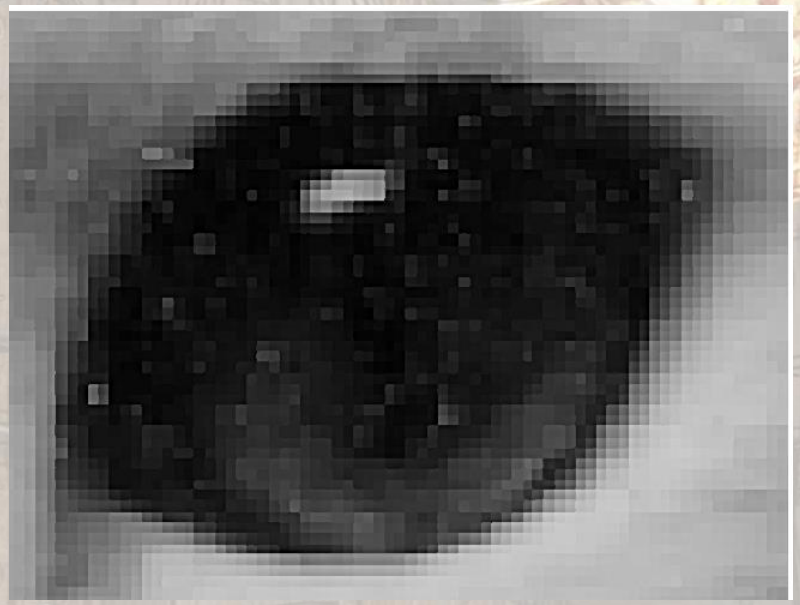
Computer Graphics: 1970-1980

- Grafica Raster
- Vengono proposti i primi standard grafici
  - ***IFIP***
    - **GKS** in Europa
      - **Diventa uno standard ISO per 2D**
    - **Core** in Nord america
      - **Non riesce ad imporsi come standard ISO per il 3D**
- Introduzione delle prime Workstation e dei primi PC

Grafica Raster

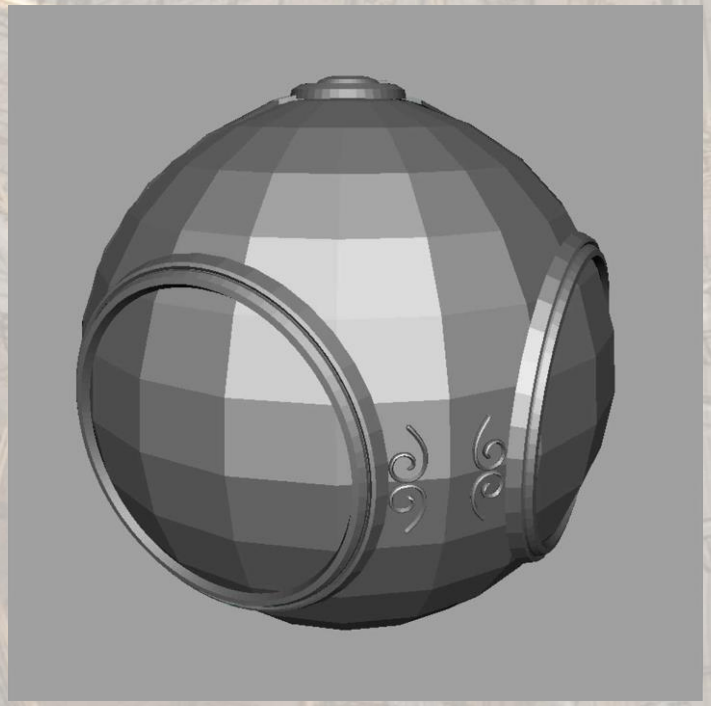
- L’immagine è prodotta da una matrice (*raster*) di singoli elementi dell’immagine chiamati *pixel* all’interno del *frame buffer*.





Grafica Raster

- La grafica raster permette di effettuare il paesaggio da immagini wireframe ad immagini in cui i poligoni sono riempiti.

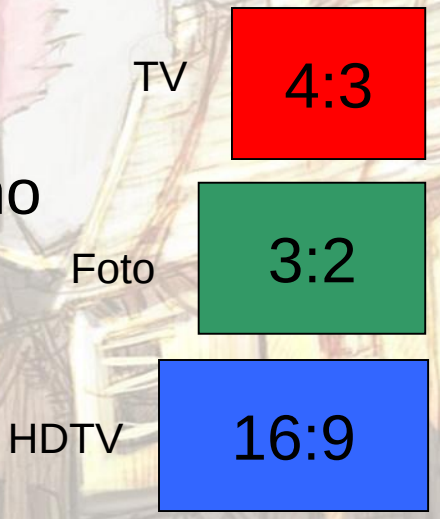


Frame buffer

- I pixel sono memorizzati in una parte della memoria di sistema chiamata **frame buffer.**
- La quantità di memoria dedicata al frame buffer dipende da:
  - Risoluzione
  - Profondità (depth): numero di bit per pixel
- La profondità determina il numero di colori che possiamo rappresentare
  - 1 bit (2 colori), 8 bit (256 colori)
- I sistemi true-color o RGB-color hanno una profondità di 24 bit per componente:
  - 8 bit per il rosso, 8 bit per il verde, 8 bit per il blu.

Risoluzioni e aspect ratio

- L’aspect ratio indica il rapporto fra la larghezza e l'altezza di una immagine bidimensionale
  - Tipiche misure sono 4:3(TV) e 16:9(HDTV)
- I vecchi monitor per personal computer hanno risoluzioni:
  - 640 x 480 (VGA)
  - 1024 x 768 (XVGA)
  - 1280 x 1024 (SXGA)
- I monitor HDTV hanno risoluzioni verticali di 780 o
- 1080 sia progressivo (720p, 1080p) che interlacciato
- (720i, 1080i)
  - HDTV: 1920 x 1080 e 1280 x 720
  - PC: 1920 x 1024 e 1280 x 768

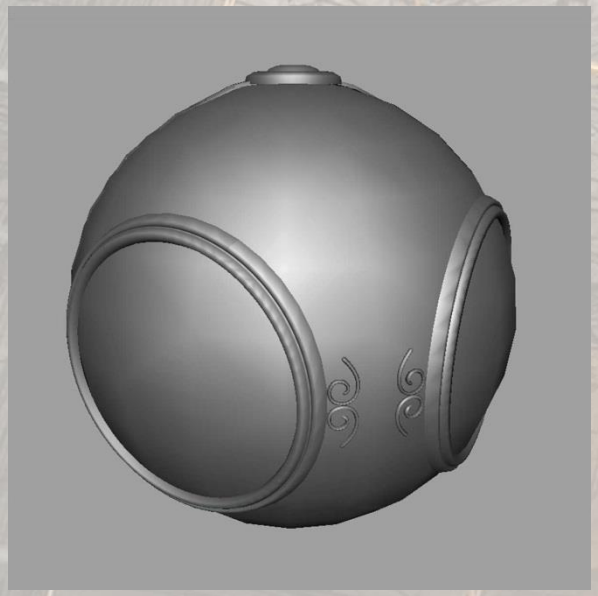


PCs and Workstations

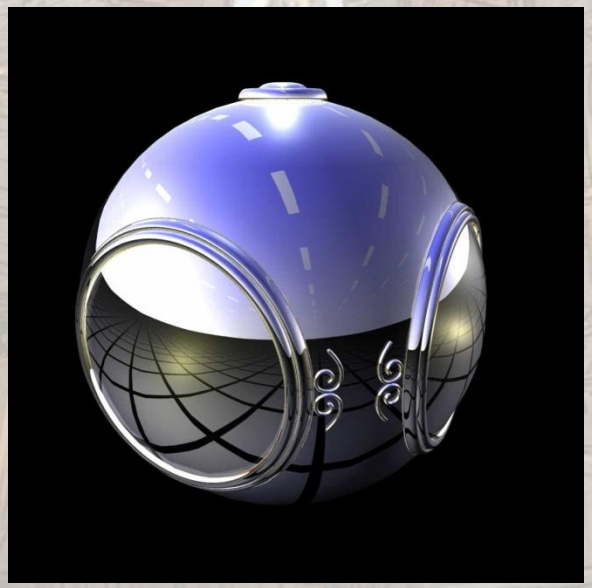
- Sebbene oggi non ci sia più una netta distinzione tra PC e workstation, storicamente hanno avuto un'evoluzione differente.
  - Le prime workstation erano caratterizzate da:
    - Collegamenti di rete utilizzando il modello client-server
    - Un alto grado di interattività
  - Nei primi personal computer parte della memoria di sistema era dedicata al frame buffer
    - Si potevano facilmente modificare le immagini cambiando il contenuto del buffer

Computer Graphics: 1980-1990

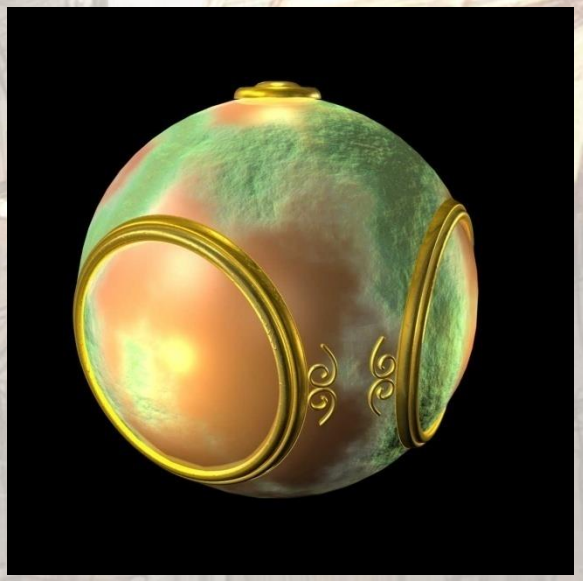
- La ricerca inizia a spingere verso il realismo



smooth shading



environment mapping



bump mapping

Computer Graphics: 1980-1990

- Special purpose hardware
  - Silicon Graphics geometry engine
    - La pipeline grafica è implementata tramite VLSI
- Industry-based standards
  - PHIGS
  - RenderMan
- Grafica in rete: X Window System
- Human-Computer Interface (HCI)

Computer Graphics: 1990-2000

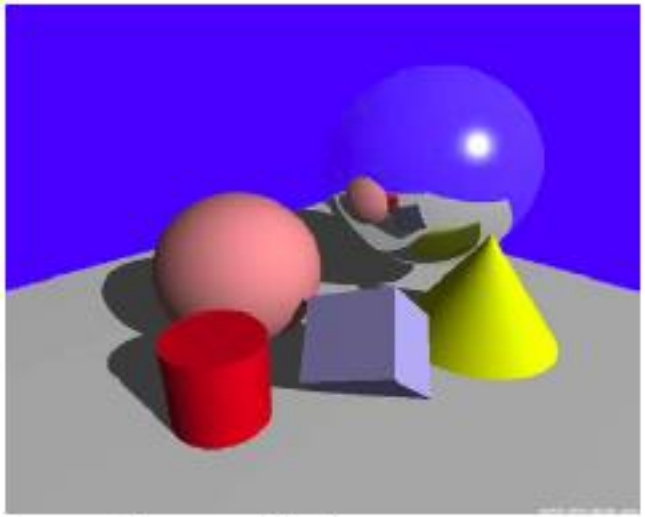
- OpenGL API
- L'hardware si evolve offrendo nuove caratteristiche
  - Texture mapping
  - Blending
  - Accumulation, stencil buffers
- Produzione dei primi film realizzati completamente in CG
  - Toy Story è il primo film in CG

Computer Graphics: 2000-oggi

- Nasce il concetto di fotorealismo
- Le schede grafiche per i PC dominano il mercato attraverso le GPU (Graphic Unit Processor)
  - Nvidia, ATI (ora AMD)
- Nasce una vera e propria industria basata sulla CG
- I video giochi e i video giocatori determinano la direzione in cui si muove il mercato
  - Indirettamente finanziano anche la ricerca
- Pipeline grafiche programmabili
- Sviluppi di progetti basati su AI per l'interazione avanzata uomo-elaboratore: vedi il filmato del progetto Milo

Fotorealismo

- Uno degli obiettivi della CG è produrre immagini indistinguibili dalla realtà
- Il termine fotorealismo è comunemente usato per indicare di quanto un'immagine digitale è simile alla realtà
  - Una formalizzazione del concetto non è semplice

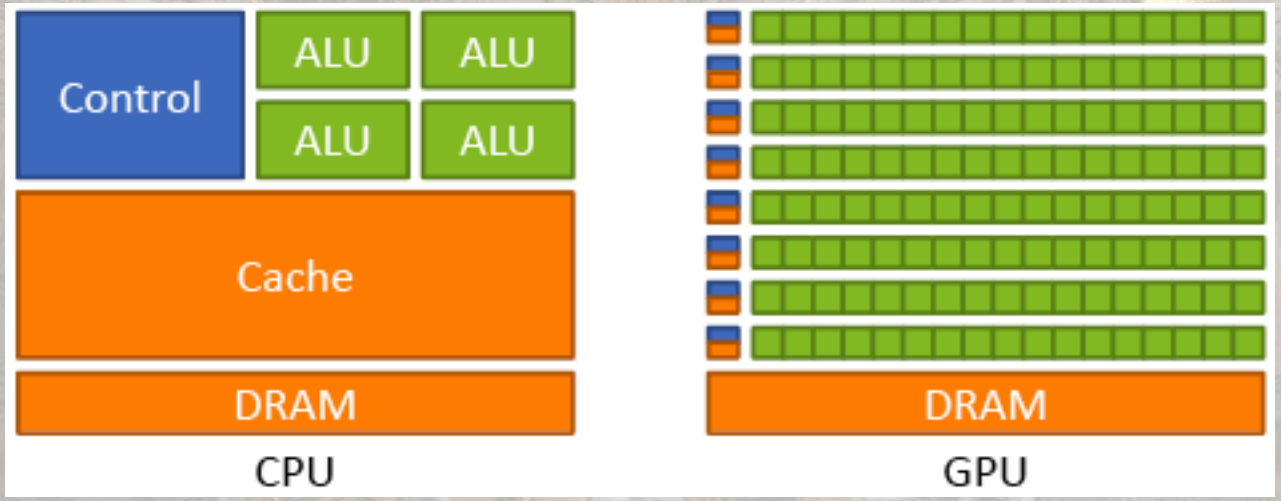


Poco fotorealistico

Molto fotorealistico

Graphics Processing Unit (GPU)

- La GPU è il microprocessore di una scheda video per computer o console
- Specializzate nell'eseguire elaborazioni 3D
  - Bandwith e velocità di elaborazione circa 10 volte una CPU
- Utilizzata anche per applicazioni general purpose (GPGPU)

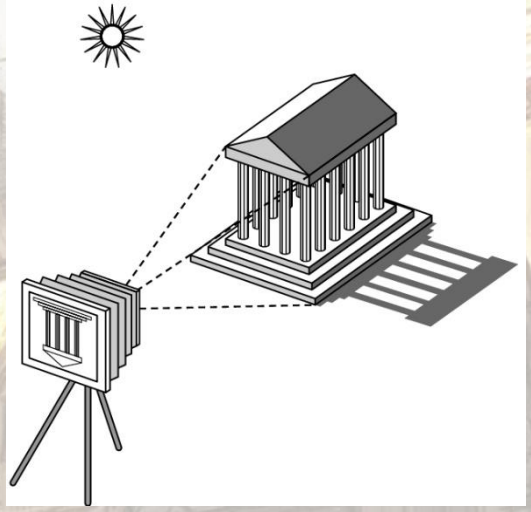


Formazione dell'immagine

- La CG genera immagini sintetiche utilizzando approcci inspirati alla realtà
  - Videocamere
  - Microscopi
  - Telescopi
  - Sistema visivo umano

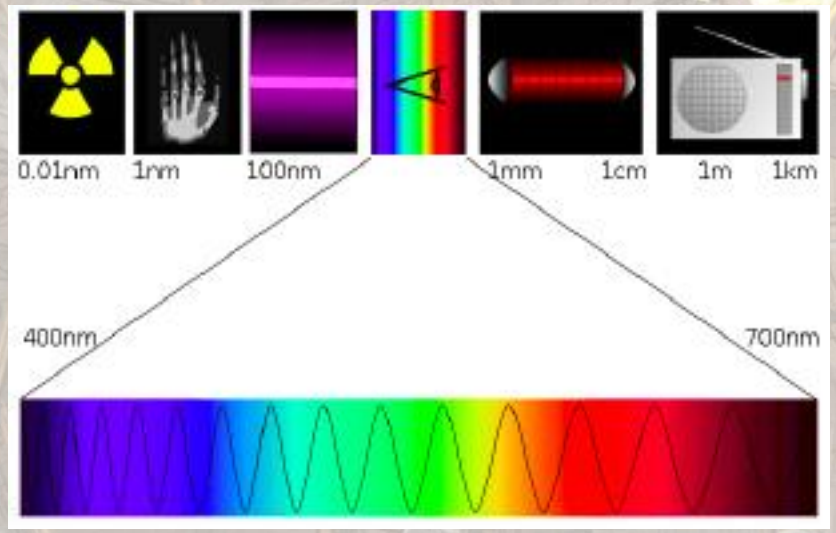
La scena

- Per poter generare un'immagine partiamo da una scena composta da
  - **- Oggetti**
  - **- Osservatore**
  - **- Sorgenti di luce**
- All'interno di una scena è necessario specificare i seguenti parametri
  - **- Forma, colore e tipo di luce**
  - **- Parametri che definiscono il tipo di materiale**
  - **- La posizione e la direzione dell'osservatore**
- Oggetti, sorgenti di luce e l'osservatore sono sempre indipendenti



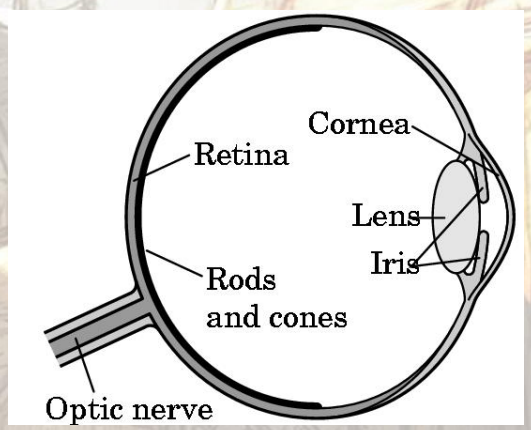
La luce

- La luce è la porzione dello spettro elettromagnetico visibile all'occhio umano
- La lunghezza d'onda è compresa tra 400 nm e 700 nm
- Le lunghezze d'onda maggiori appaiono rosse mentre le più minori appaiono blu



Sistema Visivo umano

- L'occhio umano possiede una lente che cattura l'immagine proiettandola sulla retina
- L'iride permette di variare la quantità di luce che colpisce la lente
- Sulla retina sono presenti due tipi di sensori chiamati coni e bastoncelli
  - **Coni**
    - Sono responsabili della visione a colori ma sensibili solo a luci piuttosto intense
    - Si suddividono in tre differenti tipologie responsabili della visione dei tre colori primari rosso, verde e blu
    - Operano soprattutto in condizione di luce piena
  - **Bastoncelli**
    - Permettono la visione anche quando la luce è scarsa
    - Bastoncelli sono particolarmente sensibili a basse intensità di luce ma non ai colori



Modello di colore

- Un modello di colore è un modello matematico astratto che permette di rappresentare i colori in forma numerica
- Un modello di colore permette ad una applicazione di associare ad un vettore numerico un elemento dello spazio dei colori
- Il gamut (o gamma) è un sottoinsieme limitato dello spazio dei colori rappresentabile con il modello di colore

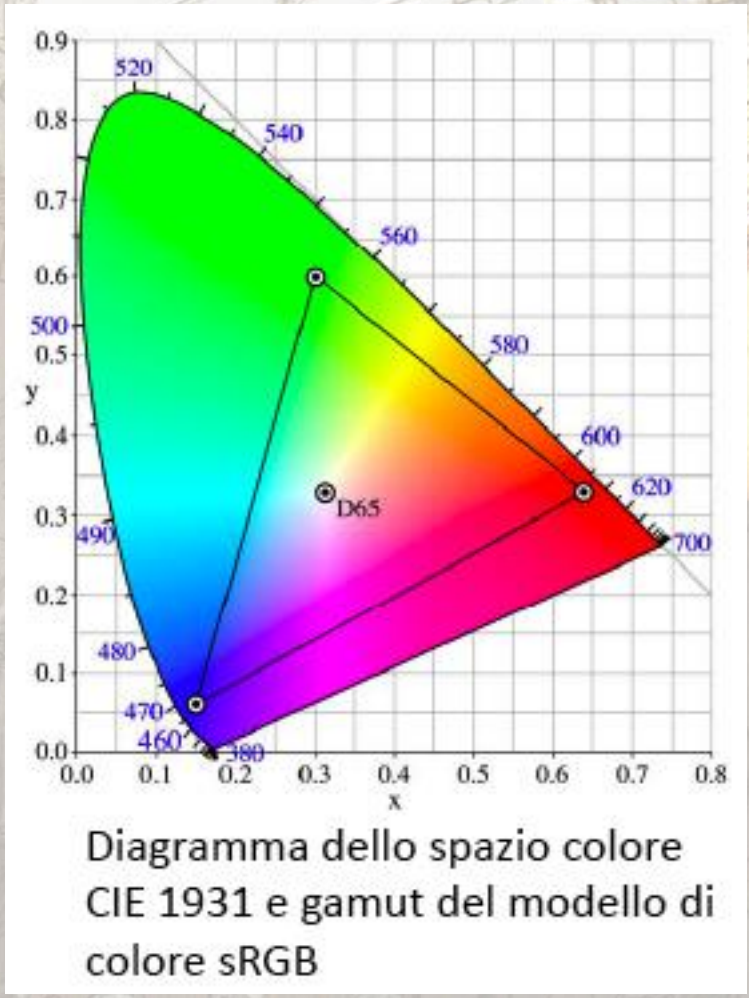


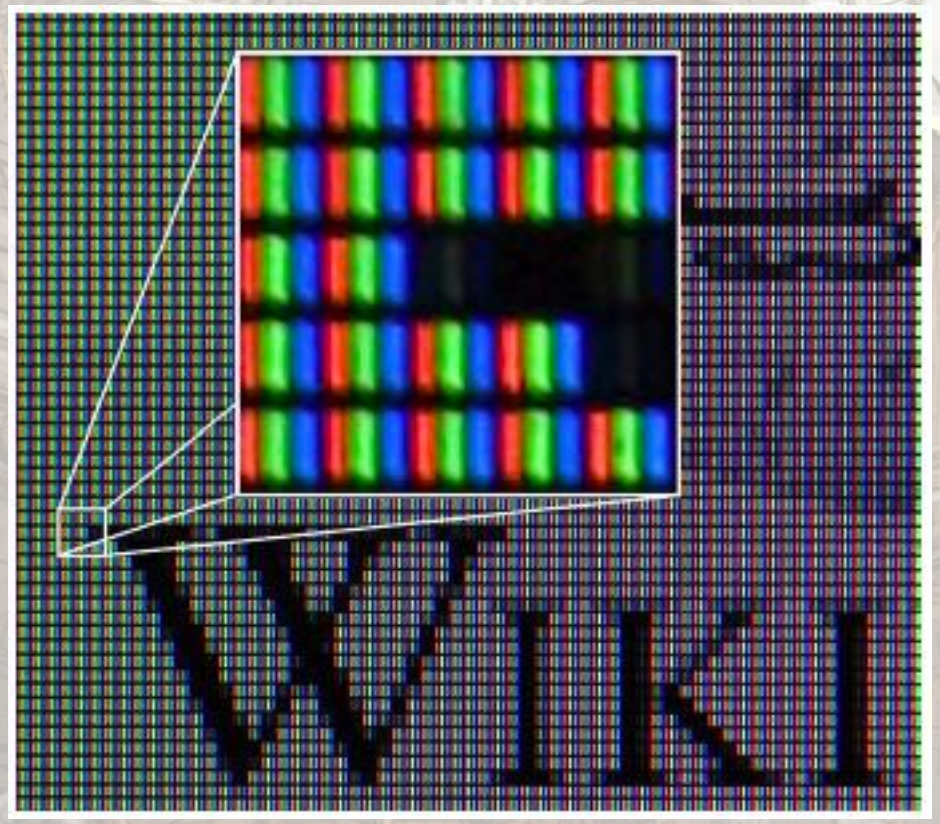
Diagramma dello spazio colore CIE 1931 e gamut del modello di colore sRGB

Due modelli di colore

- **Mescolanza additiva**
  - Si basa sul meccanismo di visione umana
  - Il colore è formato sommando tre colori primari
    - Usato da CRT, LCD, proiettori, pellicole positive
  - I colori primari sono Rosso(R), Verde (G) e Blue (B)
- **Mescolanza sottrattiva**
  - Il colore è formato filtrando la luce bianca attraverso il ciano (C), il magenta (M) ed il giallo (G)
    - Interazione luce-materia
    - Stampa
    - Pellicole negative

Modello di colore RGB nei monitor LCD

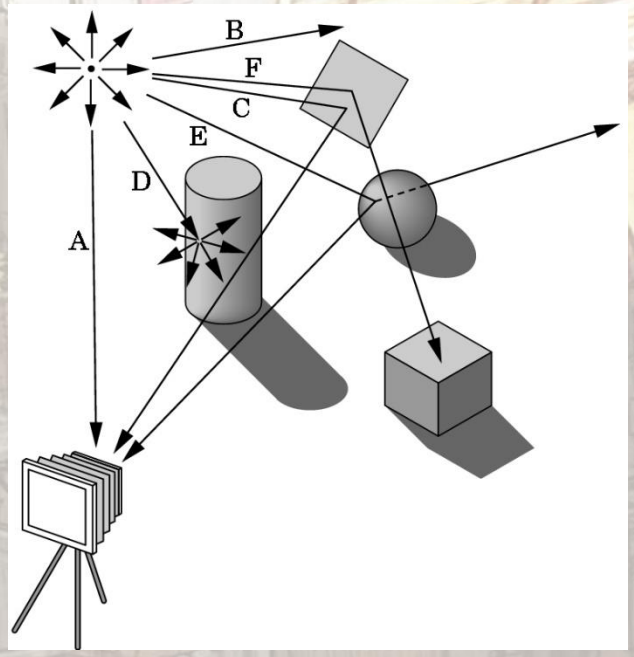
- In uno schermo a colori ogni pixel è in realtà suddiviso in 3 subpixel dotati di filtro rosso, verde e blu



Computer graphics e ottica geometrica

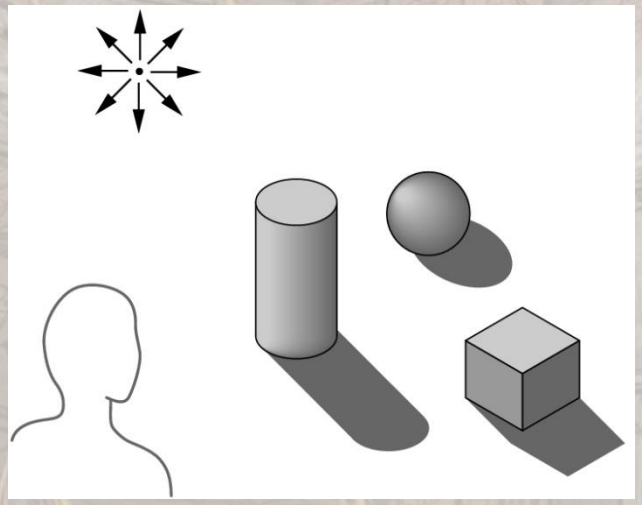
- La natura della luce come onda elettromagnetica non viene mai presa in considerazione nella CG
- Nella CG la luce è considerata attraverso il modello dell'ottica geometrica
- Possiamo creare un'immagine seguendo i raggi dalla sorgente di luce fino alla lente della camera (ray tracing)

Il percorso di un raggio può essere molto complesso da seguire



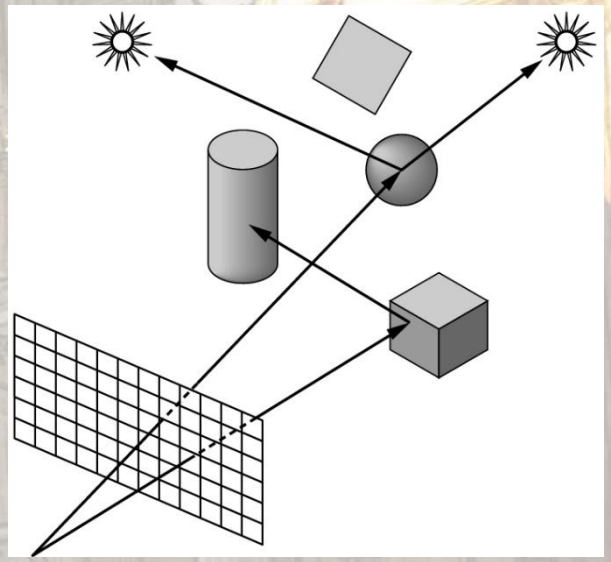
Illuminazione locale e globale

- Non è possibile calcolare indipendentemente il colore e le ombre degli oggetti
  - Alcuni oggetti potrebbero bloccare la luce
  - La luce si riflette da un'oggetto ad un'altro
  - Alcuni oggetti potrebbero essere translucenti



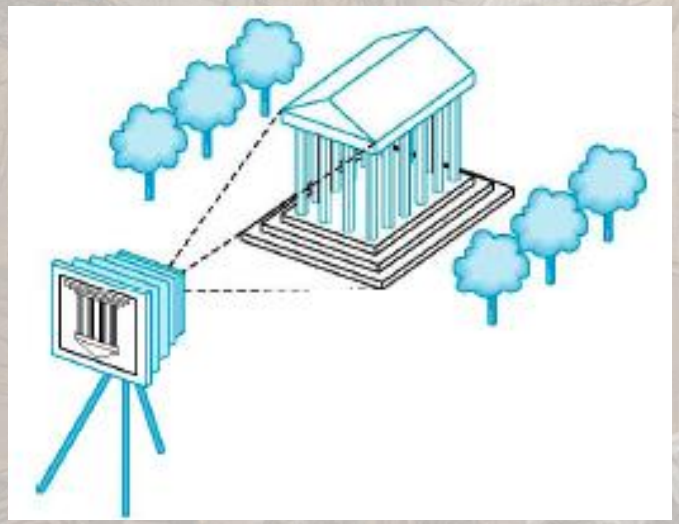
Approccio fisico

- **Ray Tracing**: simuliamo la propagazione dei raggi dal centro di proiezione fino a quando non sono assorbiti o vanno all'infinito
  - Gestisce gli effetti di illuminazione globale
  - Lento
- E' necessario tenere l'intera scena in memoria
- **Radiosity**: consideriamo la luce dal punto di vista energetico
  - Molto lento



Pinhole Camera

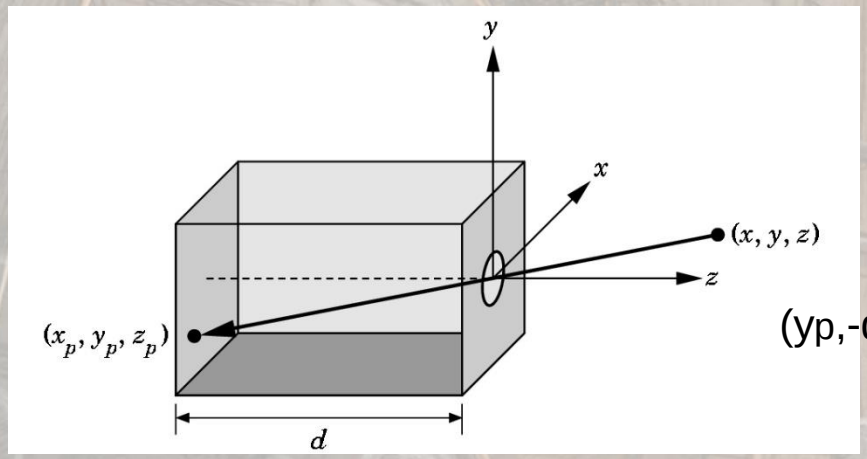
- Una pinhole camera (camera oscura) è una scatola chiusa con un piccolo foro su un lato che lasci entrare la luce
- La luce proietta sul lato opposto (una lastra) all'interno della scatola l'immagine capovolta di quanto si trova avanti al foro
  - Più il foro è piccolo e più l'immagine risulta nitida e definita

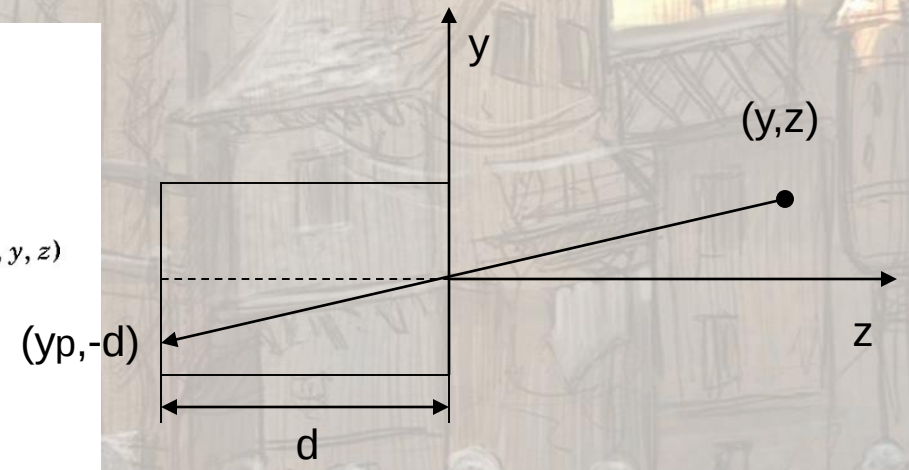


Pinhole Camera

- La lastra in cui si forma l'immagine è a distanza d dal foro
- Il punto (xp,yp,zp) sulla lastra è chiamato proiezione del punto (x,y,z) nello spazio, dove:

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

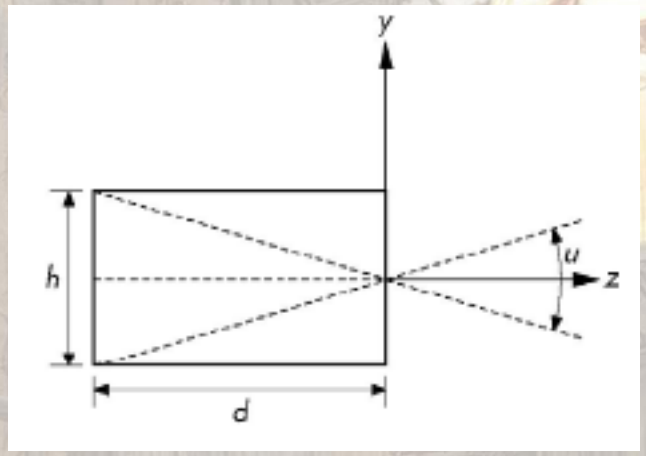




Field of view

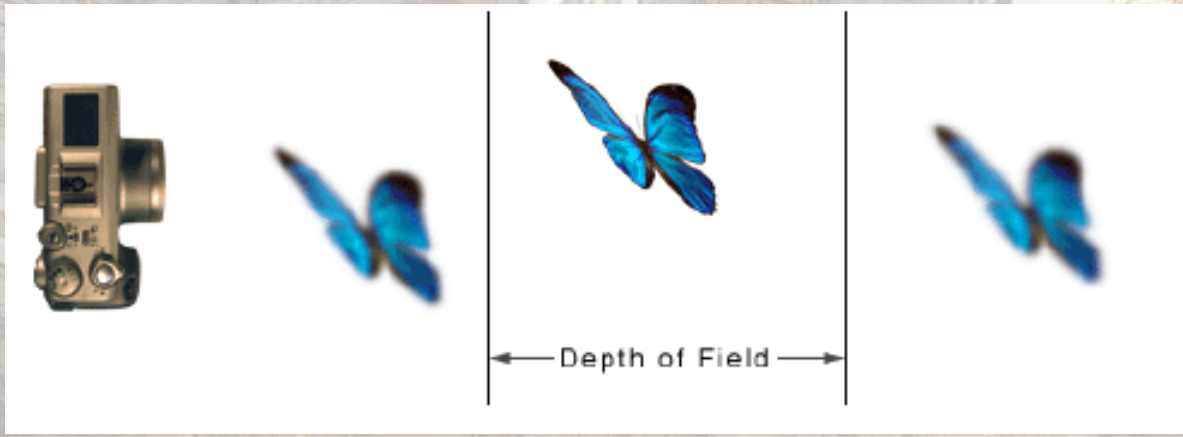
- Il field of view (campo di vista) è l'angolo formato dal più grande oggetto visibile sulla lastra
- Se h è l'altezza della camera allora l'angolo vi vista è pari a :

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



Profondità di campo

- Il termine depth of field (profondità di campo) è la distanza davanti e dietro al soggetto principale che appare nitida o a fuoco

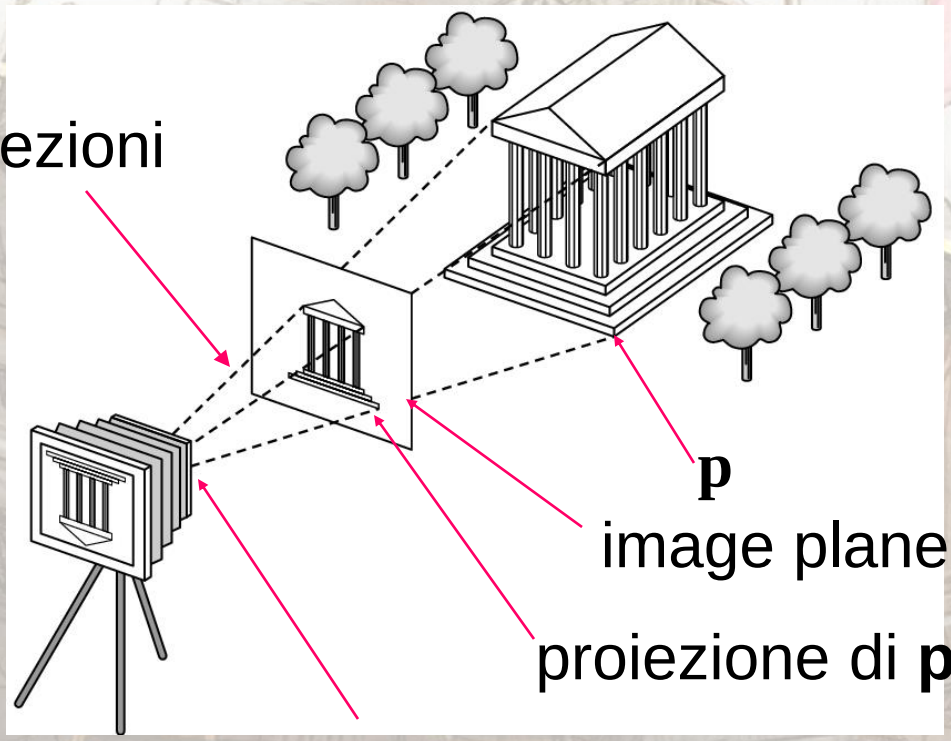


- La pinhole camera ha il vantaggio di avere una profondità di campo illimitata, ogni punto nel suo campo visivo apparirà a fuoco indipendentemente dalla distanza

Svantaggi della pinhole camera

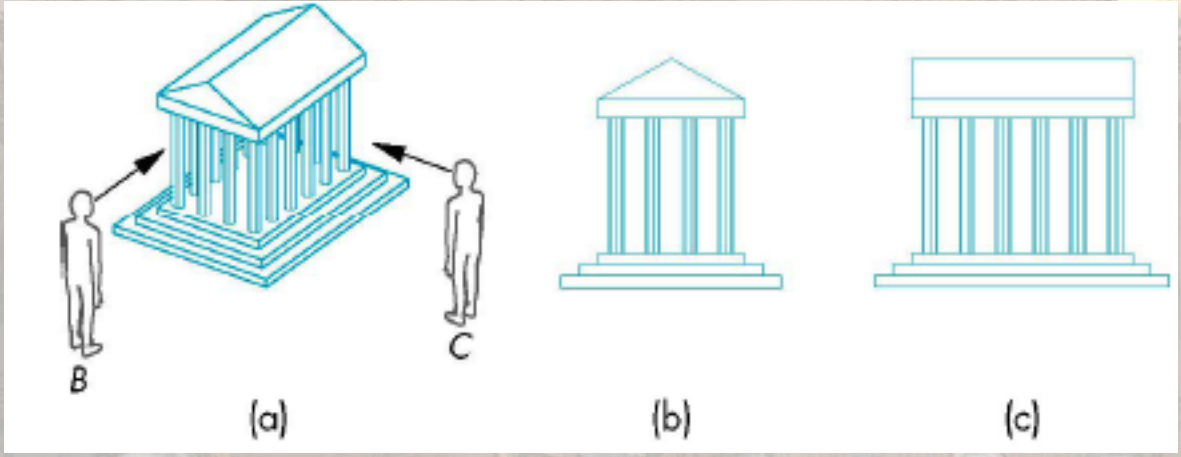
- Produce immagini poco nitide, perché i raggi luminosi provenienti dal soggetto divergono e creano piccoli cerchi
  - Per aumentare la nitidezza è necessaria una diminuzione del diametro e dello spessore del foro
  - Un foro troppo stretto comporta inoltre la comparsa di problemi di diffrazione
- Non è possibile modificare il field of view
- L'utilizzo delle lenti permette di risolvere questi problemi

Modello di camera sintetica



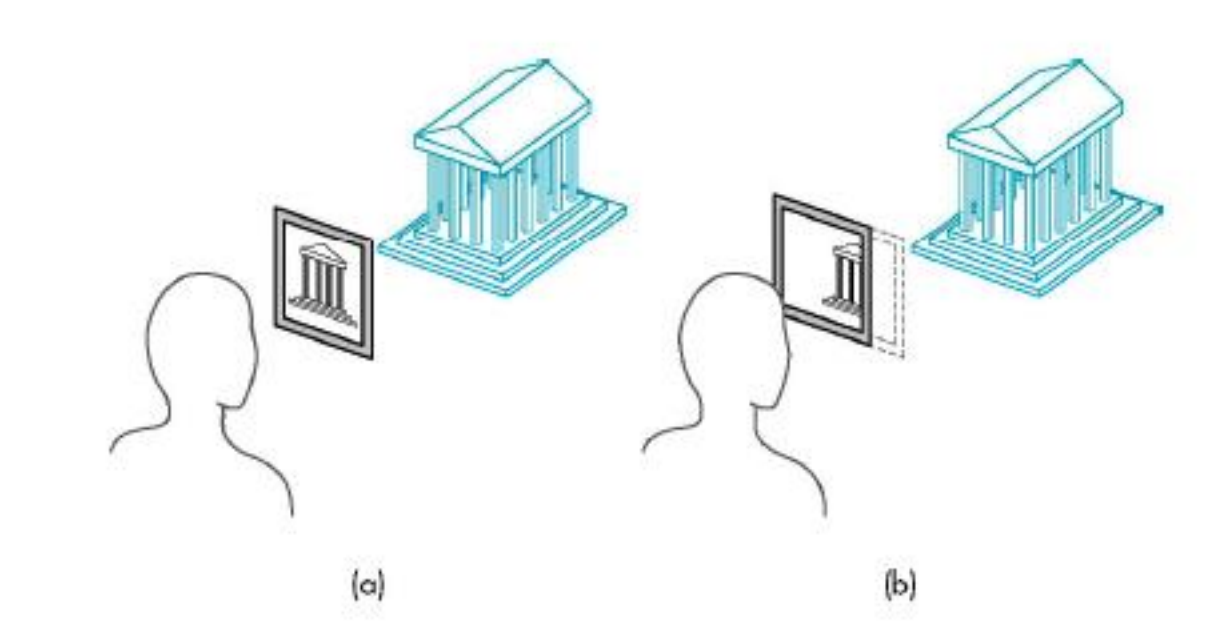
Vantaggi della camera sintetica

- L'osservatore, gli oggetti e le sorgenti di luce sono indipendenti
- La grafica 2D è un caso specifico della grafica 3D
- Permette di avere un API semplice
  - Specifichiamo separatamente oggetti, luci, camera e tutti gli attributi
  - L'immagine finale può essere prodotta in diversi modi
- L'approccio si presta ad essere implementato efficientemente in hardware



Clipping Window

- La clipping window sul piano di proiezione definisce la finestra attraverso la quale l'osservatore vede la scena



Un API per il modello di camera sintetica

- Il modello di camera sintetica si presta molto bene ad essere implementata via software
- Una API deve poter gestire
  - Oggetti
  - Materiali
  - Osservatore
  - Luci

- Come implementare una API di questo tipo?

La pipeline grafica

Il compito di un sistema grafico

Scena costruita dall'utilizzatore

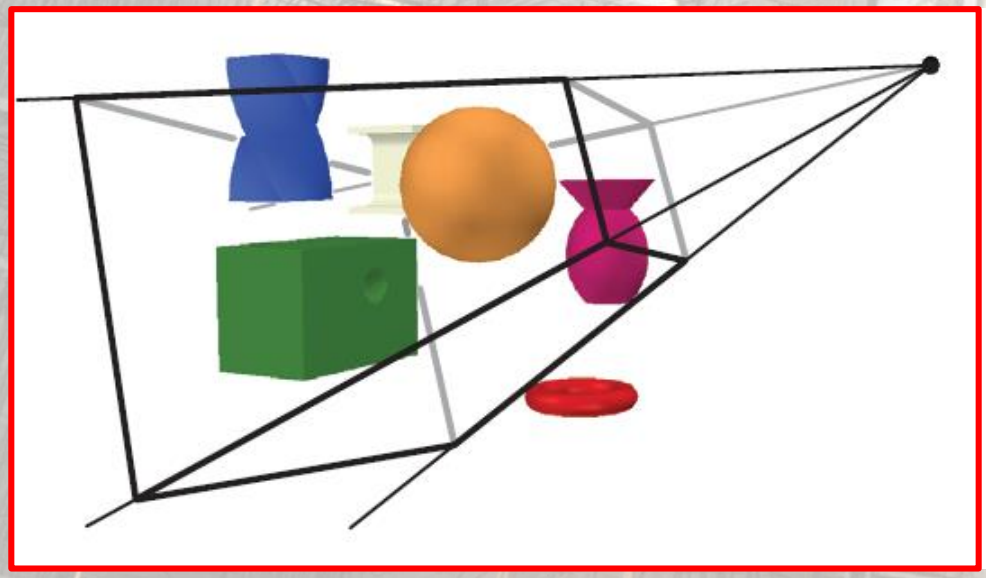
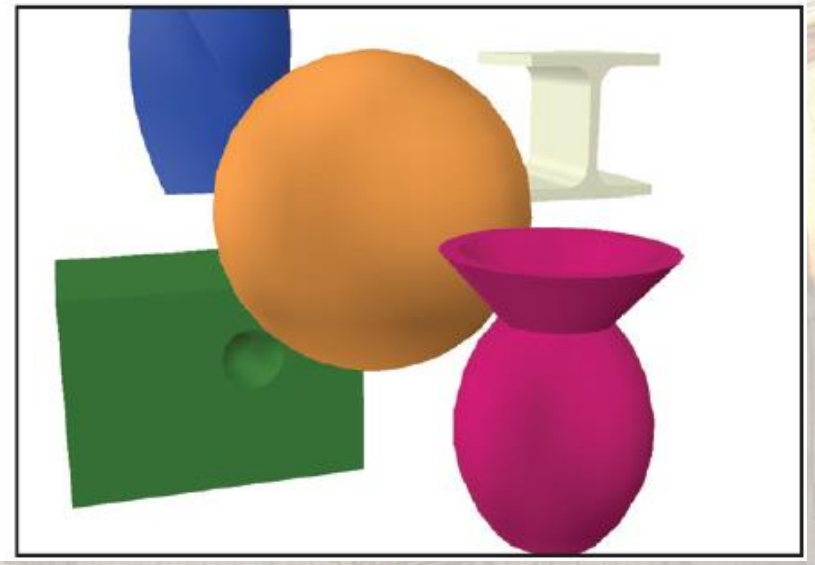
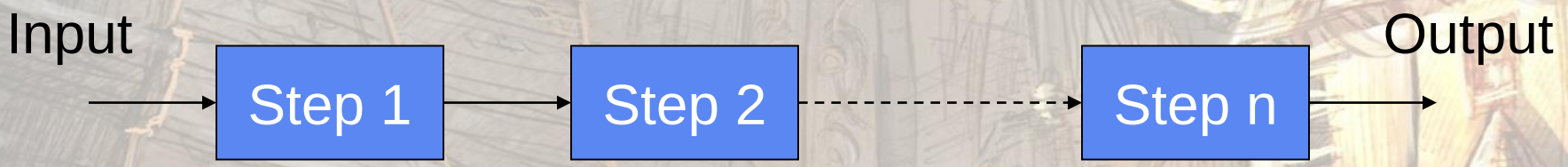


Immagine prodotta



Architettura a pipeline: perché?

- Un intero processo/sistema può essere suddiviso in una pipeline con numero N di stadi allo scopo di aumentarne le prestazioni.



- Esempi:
  - Catena di montaggio di un automobile.

  - Impianto di risalita di una stazione sciistica.

- La velocità della pipeline è determinata dallo stadio più lento.

Esempio:

Dobbiamo eseguire la seguente operazione su una serie di numeri:**(a+b)*c+d**

…	…	…	
a3	b3	c3	d3
a2	b2	c2	d2
a1	b1	c1	d1

3 cicli di clock per la singola operazione

No-pipeline

Pipeline

1	a1+b1		
2	a2+b2	(a1+b1)*c1	
3	a3+b3	(a2+b2)*c2	(a1+b1)*c1+d1
4	a4+b4	(a3+b3)*c3	(a2+b2)*c2+d2
5	a5+b5	(a4+b4)*c4	(a3+b3)*c3+d3
6	a6+b6	(a5+b5)*c5	(a4+b4)*c4+d4

dopo 6 cicli di clock ho eseguito 3 operazioni utilizzando la pipeline, altrimenti avrei avuto bisogno di 9 cicli di clock

(a1+b1)*c1+d1

(a2+b2)*c2+d2

(a3+b3)*c3+d3

Graphics Pipeline

- Una graphics pipeline realizza il processo di rendering della scena a partire da un insieme di oggetti

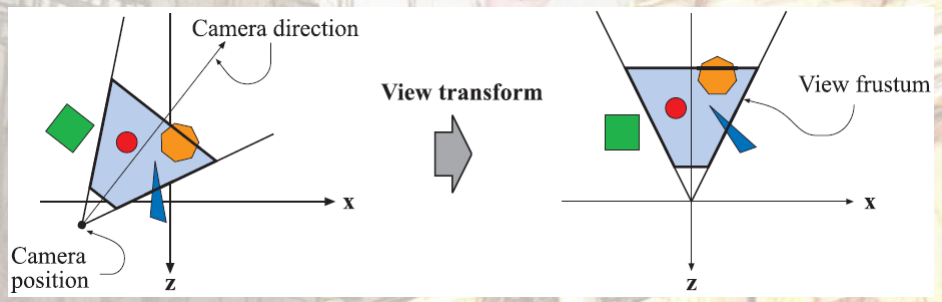
applicazione

display

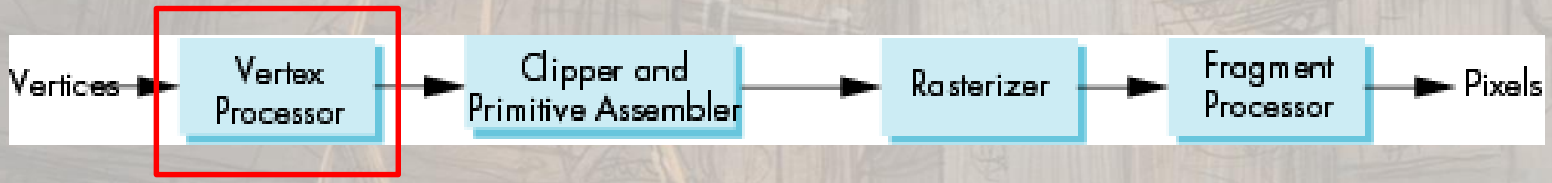
- Gli oggetti sono composti di vertici (anche milioni) i quali definiscono oggetti complessi
- La pipeline processa gli oggetti seguendo l'ordine imposto dall'applicazione
- Ogni oggetto potrà influire sulla formazione dell'immagine finale all'interno del frame buffer

Vertex Processor

- Gli oggetti sono formati da vertici, questo modulo si occupa di convertire gli oggetti da un sistema di coordinate ad un altro sistema:
  - Object coordinates
  - Camera (eye) coordinates
  - Screen coordinates

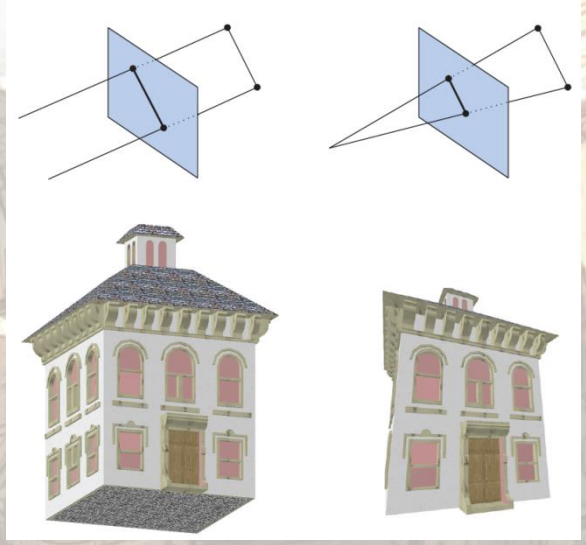


- Ogni cambio di coordinate è equivalente ad matrice di trasformazione
- Il vertex processor si occupa anche di calcolare il colore dei vertici (**vertex shading**)

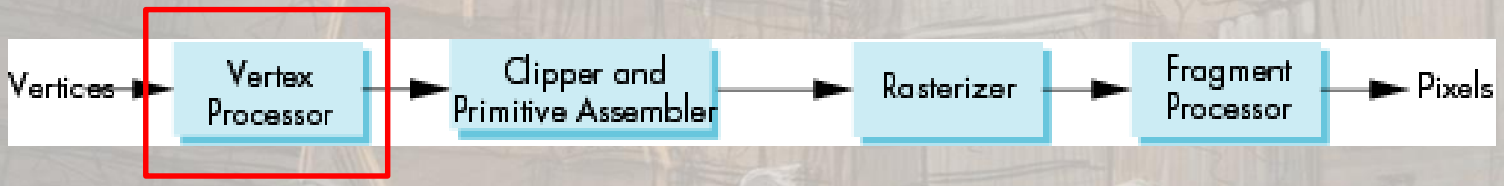


Projection

- Projection è il processo che si occupa di utilizzare i parametri dell'osservatore nello spazio per produrre una immagine 2D dagli oggetti 3D

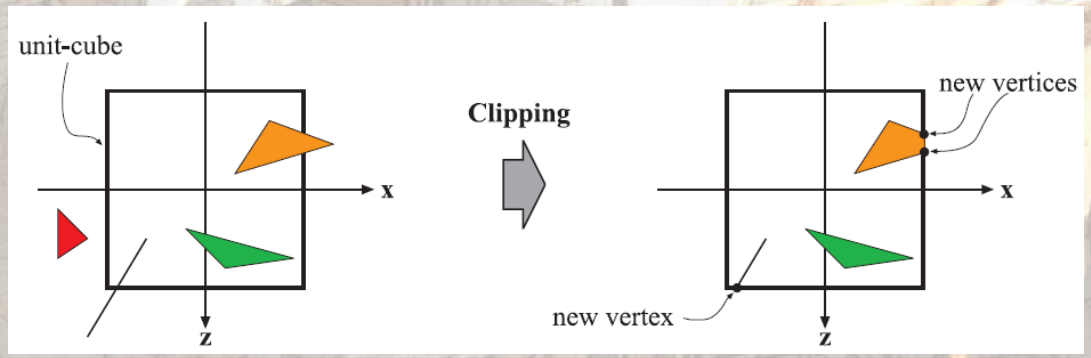


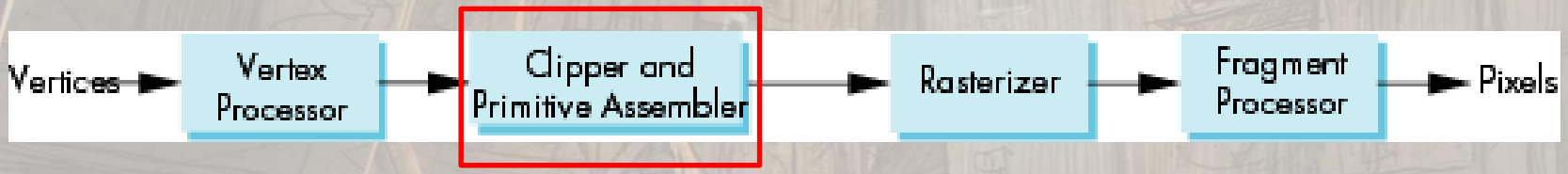
  - Proiezione prospettica: tutti i raggi di proiezione terminano nel centro di proiezione
  - Proiezione parallela: tutti i raggi di proiezione sono paralleli, il centro di proiezione è posto all'infinito



Primitive Assembly

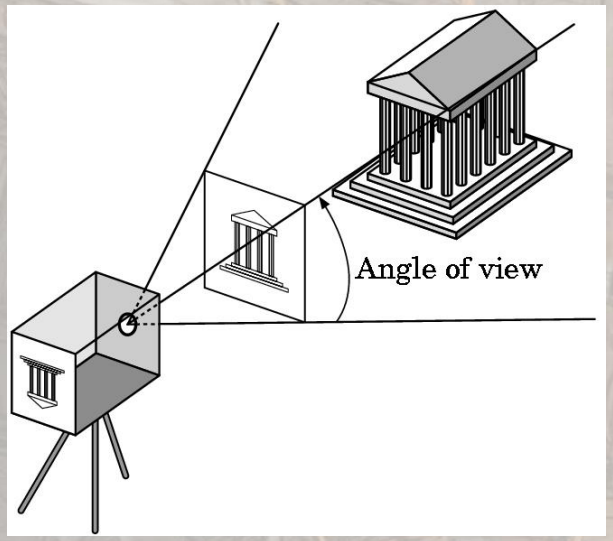
- I vertici processati sono considerati come parte di oggetti (modelli 3D) prima che la fase di clipping e rasterizzazione possa avvenire:
  - Line segments
  - Polygons
  - Curves and surfaces

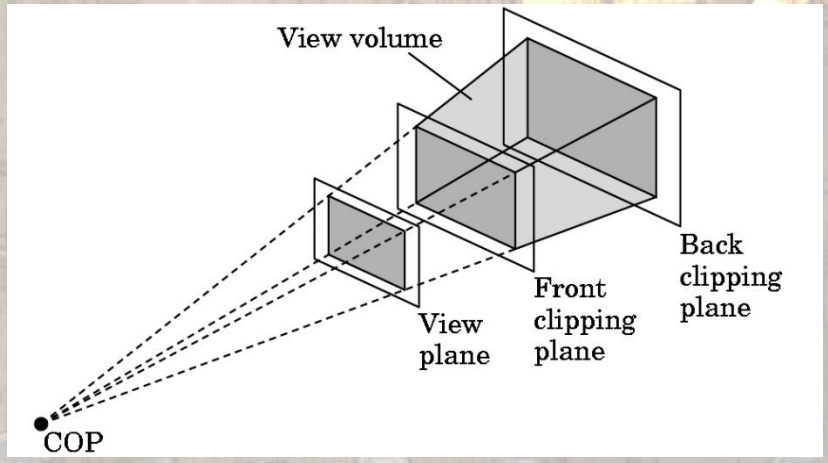




Clipping

- Così come una videocamera non può vedere tutto, la camera virtuale deve escludere quella parti della scena non visibili
- Tutti gli oggetti non presenti nel campo visivo devono essere tagliati fuori dalla scena attraverso il processo di **clipping**





Rasterization

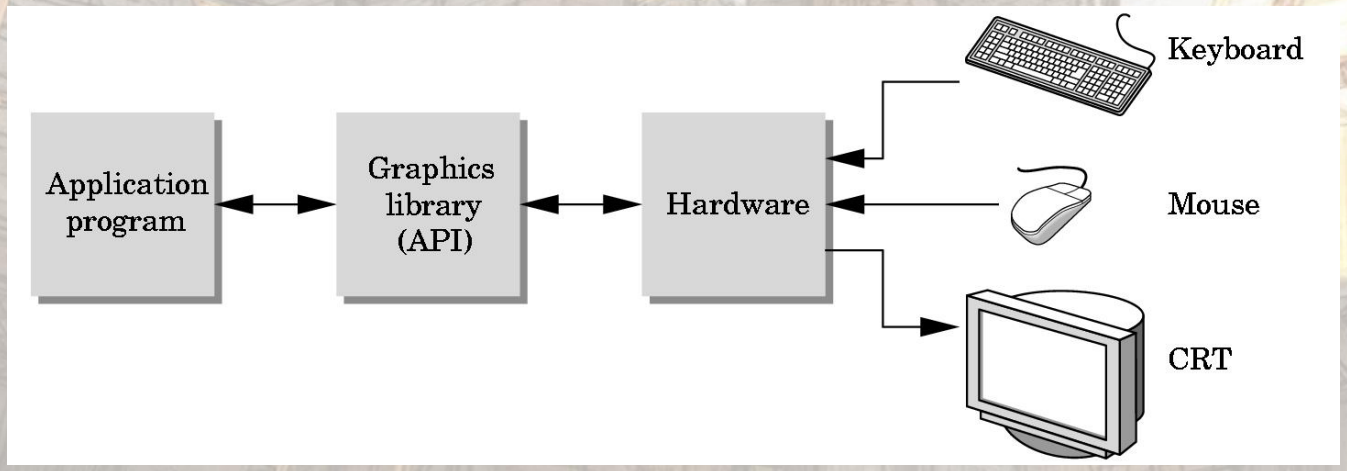
- Se un oggetto non è “tagliato” fuori allora è necessario determinare il colore appropriato dei pixel
- Rasterizer produce un insieme di frammenti per ogni pixel
- I frammenti sono dei “potenziali pixel”
  - Hanno una posizione nel frame buffer
  - Hanno un colore ed una profondità
- Gli attributi dei vertici degli oggetti sono interpolati dal rasterizer per ottenere i frammenti

Fragment Processing

- I frammenti sono processati per determinare il colore finale all'interno del frame buffer
- Il colore può essere determinato utilizzando il texture mapping oppure interpolando il colore dei vertici
- I frammenti possono essere scartati da altri frammenti più vicini alla camera
  - Algoritmi di rimozione delle superfici nascoste

The Programmer's Interface

- Il programmatore vede il sistema grafico attraverso una interfaccia software: API



Contenuto delle API

- Funzioni che permettono di assemblare la scena per produrre l'immagine finale
  - **Oggetti**
  - **Osservatore**
  - **Sorgenti di luce**
  - **Materiali**
- Altre informazioni
  - Prelievo dell'input da periferiche come mouse e tastiera

Primitive di disegno

- L'API normalmente supporta un numero limitato di primitive di disegno, come:
  - Punti (0D object)
  - Segmenti (1D objects)
  - Poligoni (2D objects)
  - Curve e superfici
- Tutte le primitive sono definite attraverso locazioni nello spazio o vertici

Esempio

Tipo di oggetto

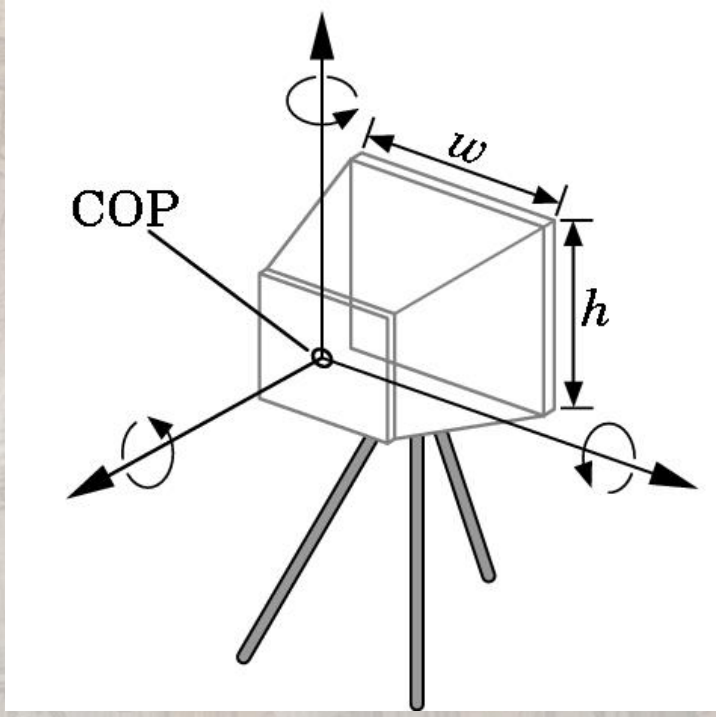
Posizione dei vertici

```
glBegin(GL_TRIANGLE)
  glVertex3f(0.0, 0.0, 0.0);
  glVertex3f(0.0, 1.0, 0.0);
  glVertex3f(0.0, 0.0, 1.0);
glEnd( );
```

Fine della definizione

Camera Specification

- Nove gradi di libertà
  - Posizione del centro delle lenti
  - Orientazione
- Lenti
- Dimensione del film
- Orientazione del piano di ripresa



Luci e Materiali

- Tipi di luce
  - Sorgenti puntiformi o sorgenti distribuite. Spot lights
  - Sorgenti vicine o distanti
  - Colore
- Proprietà dei Materiali
  - Assorbimento: color properties
  - Scattering
    - Diffuse
    - Specular

Rendering

- Il processo di generazione di una immagine digitale a partire da una descrizione della scena è chiamata rendering
- La complessità della scena ed il modello di rendering influenza direttamente il tempo necessario
- Tipi di rendering:
  - Real Time: OpenGL/DirectX
  - Off-Line: Ray Tracing/Radiosity/Photon Mapping

Introduzione a OpenGL

IRIS GL

- Agli inizi degli anni 90 Silicon Graphics era leader indiscusso nella progettazione di architetture per la visualizzazione 3D
  - IRIS GL era la libreria grafica fornita con le sue workstation.
- La richiesta di hardware specializzato spinse i produttori a progettare hardware ad-hoc
  - Ogni produttore forniva hardware con proprie specifiche
  - Ogni produttore spingeva per una propria libreria grafica
- IRIS GL era una buona libreria ma fortemente dipendente dall'hardware
  - Esistevano diverse varianti
- Silicon Graphics ebbe l'idea vincente mettendo d'accordo tutti

OpenGL

- Silicon Graphics propose nel 1992 uno standard aperto e compatibile rivolto a qualunque produttore hardware
- Obiettivi dell'API OpenGL
  - Facile impiego
  - Implementazione vicino all'hardware per garantire ottime prestazioni
  - Focus on rendering
  - Dedicata solo alla visualizzazione e indipendente dal Sistema Operativo

Evoluzione di OpenGL

- Inizialmente lo standard OpenGL era gestito dall'OpenGL Architectural Review Board (OpenGL ARB)
  - Alcuni membri erano SGI, Microsoft, Nvidia, HP, 3DLabs, IBM,…….
  - Relatively stable (present version 2.1)
    - Nel corso degli anni l'evoluzione si è legata con le capacità dell'hardware
      - **3D texture mapping and texture objects**
      - **Vertex programs**
  - Ogni produttore può aggiungere nuove caratteristiche attraverso l'impiego delle estensioni
  - OpenGL ARB è stata rimpiazzata da Khronos
    - http://www.khronos.org/

Cronologia di OpenGL

- Versione 1.0 (1992)
  - 1995: OpenGL 1.1 (vertex array, texture object)
  - 1997: OpenGL + Direct3D = Fahrenheit
  - 1998: OpenGL 1.2 (3D textures, separate specular, imaging)
  - 2001: OpenGL 1.3 (compressed texture, cube maps, dot3)
  - 2002: OpenGL 1.4 (mipmap, shadow)
  - 2003: OpenGL 1.5 (vertex buffer object, occlusion query)
- Versione 2.0 (2004)
  - 2005: OpenGL Embedded Systems (OpenGL ES) per dispositivi mobile
  - 2006: OpenGL 2.1 (GLSL - OpenGL shading language)
- Versione 3.0 (2008)
  - 2009 OpenGL 3.1 (texture objects)
  - 2009 OpenGL 3.2 (Geometry shader)
- Versione 4.0
  - Marzo 2010 (accesso alle GPU di ultima generazione)
  - Luglio 2010 OpenGL 4.1 (tessellation-control & tessellation-evaluation shaders)
  - Agosto 2011 OpenGL 4.2
  - Agosto 2012 OpenGL 4.3 (versione 4.3 di GLSL, nuovo metodo di compressione texture)
  - Luglio 2013 OpenGL 4.4
  - Agosto 2014 OpenGL 4.5

OpenGL sui personal computer

- Nel 1997 la 3dFX propose una scheda accellerata per PC
  - Molti ingegneri della 3dFX provenivano dalla Silicon Graphics
- Portare l'innovazione delle schede grafiche dalle workstation ai PC
- La 3dFX implementò una versione preliminare di OpenGL
  - La prima applicazione fu Quake

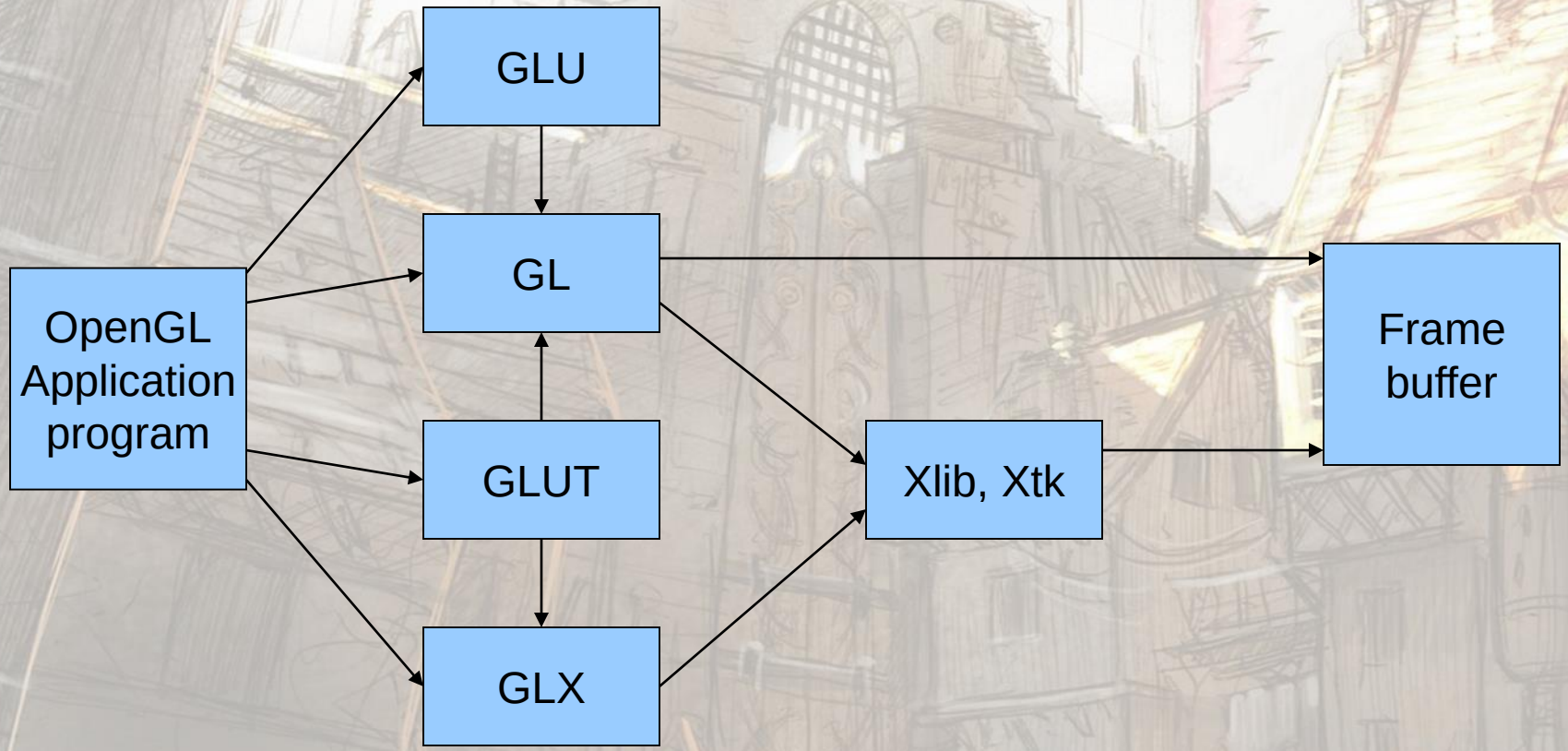
Le librerie OpenGL

- OpenGL core library
  - OpenGL32 su Windows
  - GL sui sistemi unix/linux (libGL.a)
- OpenGL Utility Library (GLU)
  - Fornisce funzionalità utilizzate di frequente evitando la riscrittura di codice
  - Glu32.lib su Windows
  - GLU sui sistemi unix/linux (libGLU.a)
- Librerie necessarie a scrivere applicazioni OpenGL su sistemi operativi
  - GLX per X window systems
  - WGL per Windows
  - AGL per Macintosh

GLUT

- OpenGL Utility Toolkit (GLUT)
  - Fornisce una astrazione per interfacciarsi con tutti i sistemi
    - Apertura della finestra di rendering
    - Gestione degli eventi generati da mouse e tastiera
    - Gestione dei menu
  - Permette la portabilità del codice ma non supporta tutti i widget tipici dei moderni sistemi operativi
    - No slide bars

Organizzazione delle librerie OpenGL



Le estensioni come evoluzione

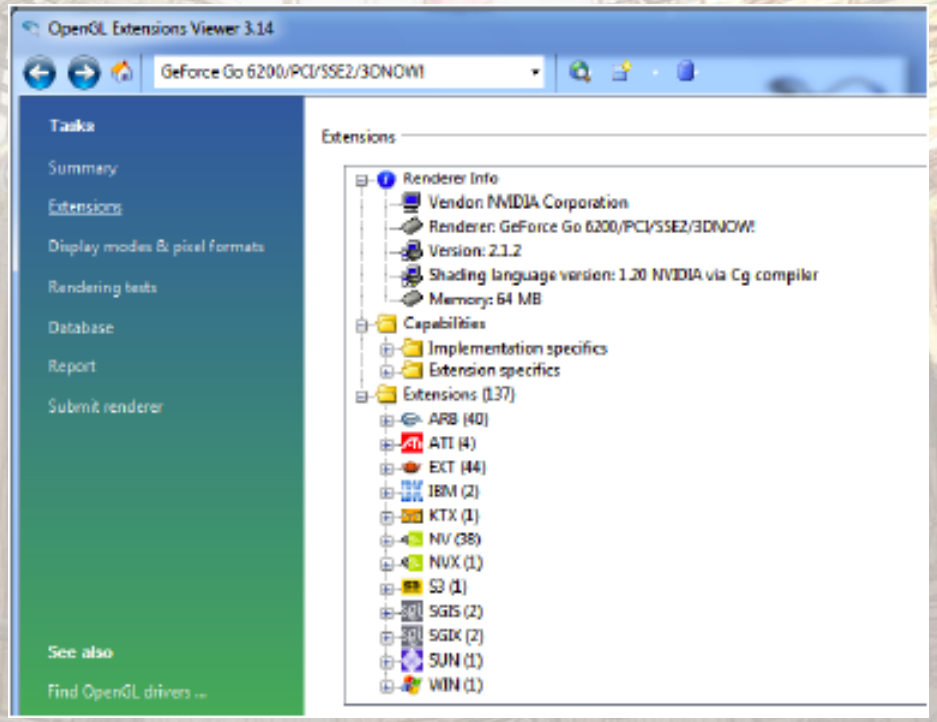
- L’evoluzione di OpenGL si misura principalmente dalle estensioni
- Una estensione indica un insieme di comandi per implementare nuove features
  - Ogni produttore hardware può implementare indipendentemente le proprie estensioni
  - Se l’estensione trova accoglienza verso gli sviluppatori la si promuove a standard
- Dal 1992 al 2009 sono stati introdotti dai produttori di schede video più di 100 estensioni, molte delle quali oggi appartengono al core di OpenGL

Esempi di estensioni

- GL_ARB_shadow
  - Semplifica la creazione delle ombre
- GL_EXT_frame_buffer_object
  - Permette il rendering off-screen (non in finestra)
- GL_ATI_float_texture
  - Amplia la gestione delle texture a 256 bit (32 bit per componente)
- GL_NV_vertex_program
  - Introduce la programmabilità dei vertici sulla GPU

Uso delle estensioni

- Interpretazioni delle estensioni
  - **EXT**: estensioni sperimentali adottate da almeno due produttori
  - **ARB**: estensioni approvate ma non ancora inserite nel core
  - **ATI**: estensioni proprietarie ATI
  - **NV**: estensioni proprietarie NVIDIA
- Le estensioni non inserite nel core devono essere inizializzate esplicitamente
  - OpenGL Extension Wrangler Library (GLEW) è una libreria open-source per inizializzare le estensioni
- La documentazione di nuove estensioni è argomento per esperti



http://www.realtech-vr.com/glview/index.html

OpenGL come macchina a stati

- Una macchina a stati è un modello di funzionamento di un sistema composto da stati ed azioni
  - Lo stato memorizza il comportamento della macchina fino a quel momento
  - L'azione indica il tipo di attività che la macchina è in grado di compiere in un determinato stato
- OpenGL è paragonabile ad una macchina a stati
- Le funzioni di OpenGL sono di due tipi
  - Funzioni di generazioni di primitive grafiche
    - Possono produrre un output se la primitiva è visibile
    - In che modo sono eleborati i vertici e il rendering delle primitive sono controllate dallo stato
  - Modifica dello stato
    - Funzioni di trasformazione
    - Funzioni di impostazioni degli attributi

Esempio di OpenGL come macchina a stati

- Le funzioni glEnable/glDisable modificano lo stato di OpenGL relativamente agli effetti

```
glDisable(GL_FOG);
glEnable(GL_LIGHTING);
```

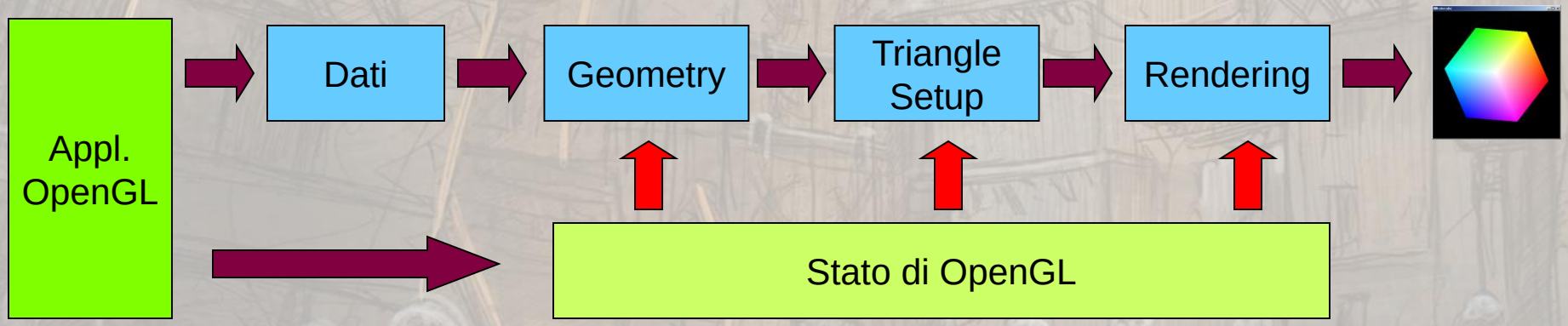
- Altre funzioni impostano i valori correnti da usare fino ad una nuova chiamata della stessa funzione

```
glColor3f(1.0,1.0,0.0);
glTranslate3f(1.0,0.0,0.0);
```

- Di default OpenGL ha uno stato predefinito. Controllate la documentazione: non date nulla per scontato.
  - Ad esempio la nebbia è disabilitata di default

Pipeline di OpenGL

- I dati di input vengono elaborati, trasformati e combinati in base ai valori dei parametri forniti ad OpenGL attraverso la pipeline
- La rendering pipeline non è unica, ogni produttore implementa la propria basandosi sulle performance, sui costi e sui consumi
  - Per ragioni commerciali non sempre un produttore rilascia pubblicamente tutte le specifiche relativamente alla sua implementazione
- Possiamo comunque ritrovare all'interno di ogni pipeline i seguenti stadi



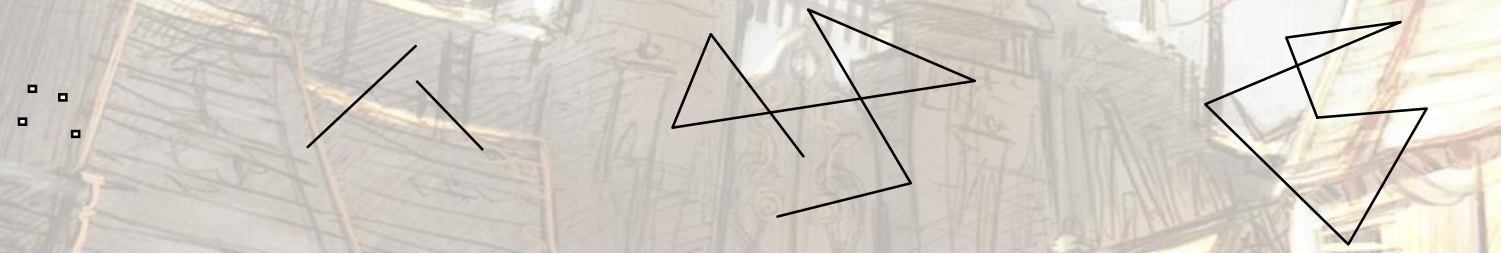
Una pipeline smemorata

- La rendering pipeline di OpenGL non ricorda **MAI** i dati dell'ultimo rendering
  - Per ogni frame i vertici dovrebbero essere passati di nuovo in input
- E' possibile mantenere in memoria texture e vertici per evitare ripetitivi trasferimenti di dati
  - Al termine di ogni rendering bisogna nuovamente "indicare" i dati necessari al prossimo rendering
- La filosofia di OpenGL è:

  Visualizzare i dati in input non gestirli
- Lasciare la gestione dei dati in input alla CPU permette ai produttori di concentrarsi principalmente sulle prestazioni.

Primitive: punti e linee

`glBegin(mode)`



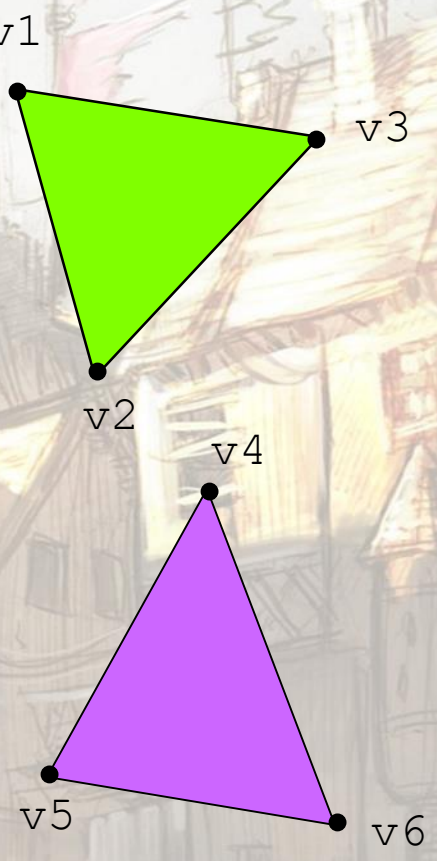
`GL_POINTS` **`GL_LINES`** **`GL_LINE_STRIP`** **`GL_LINE_LOOP`**

GL_TRIANGLES

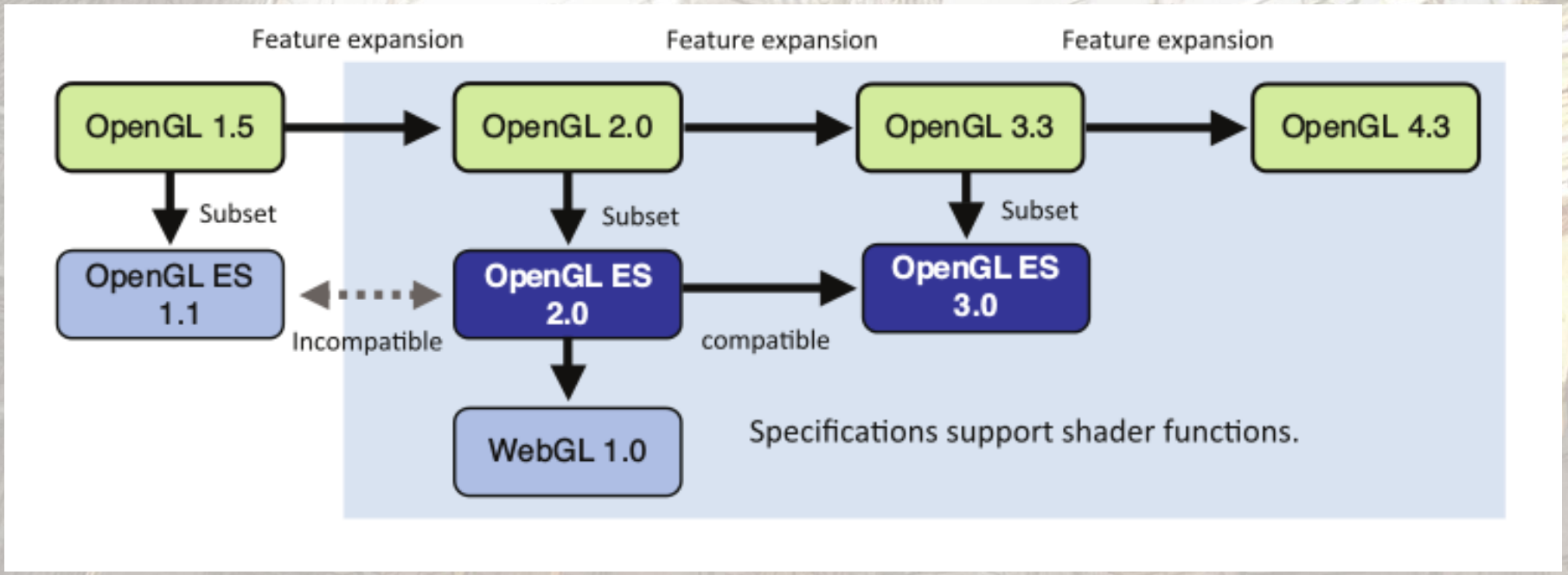
```
// Inizio passaggio dei vertici
glBegin(GL_TRIANGLES)

   //Primo triangolo
   glVertex(v1); /*1*/
   glVertex(v2); /*2*/
   glVertex(v3); /*3*/
   //Secondo triangolo
   glVertex(v4); /*4*/
   glVertex(v5); /*5*/
   glVertex(v6); /*6*/

  // Termina di passare i vertici
glEnd();
```

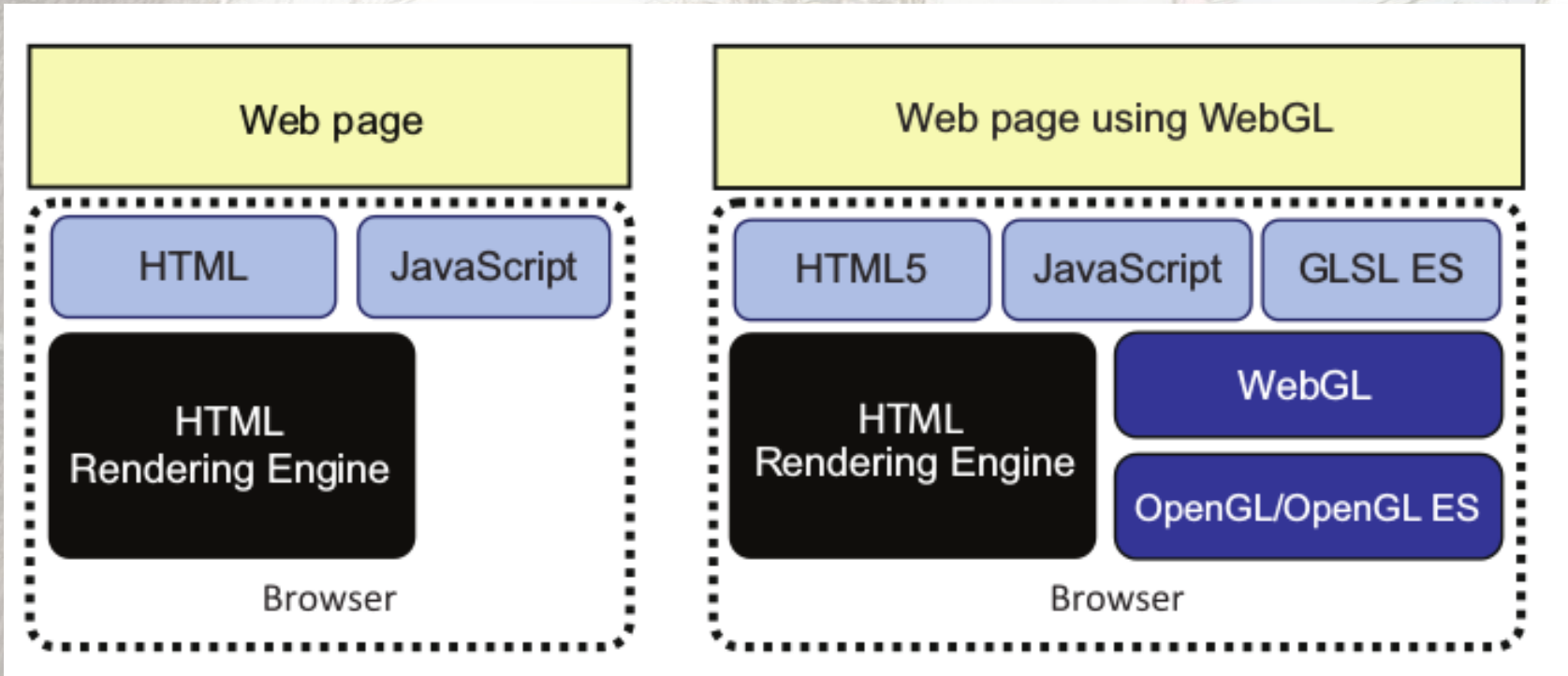


Le origini di WebGL



WebGL è stata rilasciata nel 2011, ed è derivata da OpenGL ES 2.0 che introduce la programmazione degli shader (linguaggio di shading GLSL).

La struttura di un applicazione WebGL



Una pagina web di tipo dinamico che impiega WebGL richiede l'uso di HTML5, Javascript come linguaggi di programmazione ed usa GLSL + un subset di funzionalità di OpenGL.

<Canvas>

- WebGL impiega l'elemento **<canvas>** , introdotto dal HTML5, per disegnare all'interno di una pagina web.
- Il linguaggio Javascript ci fornisce gli strumenti per disegnare punti, linee, rettangoli, cerchi,…
- Prendiamo in considerazione il file DrawRectangle.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8"/>
    <title>Draw a blue rectangle (canvas version)</title>
    <style type="text/css">
      canvas {border: 2px dotted blue;}
    </style>
  </head>
  <body onload="main()">
    <canvas id="example" width="400" height="400">
      Please use a browser that support "canvas"
    </canvas>
    <script src="DrawRectangle.js"></script>
  </body>
</html>
```

<Canvas> - DrawRectangle.js

```
// DrawRectangle.js
function main() {
  // retrieve <canvas> element
  var canvas = document.getElementById('example');
  if(!canvas) {
      console.log('failed to retrieve <canvas> element');
      return;
  }
  // Get the rendering context for 2DCG
  var ctx = canvas.getContext('2d');
  // draw blue rectangle
  ctx.fillStyle='rgba(0,0,255,1.0)'; // set blue color
  ctx.fillRect(120,10,150,150); // Fill the rectangle with blue color
}
```

Step #1

Step #2

Step #3

Il programma WebGL più breve

- Il programma WebGL più breve ci permette di cancellare l'area di disegno <canvas> riempiendo l'area con il colore che decidiamo.
- In questo caso abbiamo due file
  - HelloCanvas.html
  - HelloCanvas.js
- Il file html differisce dall'esempio precedente per il fatto che carichiamo dei file Javascript aggiuntivi:

```
<script src="../lib/webgl-utils.js"></script>
<script src="../lib/webgl-debug.js"></script>
<script src="../lib/cuon-utils.js"></script>
<script src="HelloCanvas.js"></script>
```

HelloCanvas.js

```
//function main() {
  // retrieve <canvas> element
  var canvas = document.getElementById('webgl');
  // get the rendering context for WebGL
  var gl = getWebGLContext(canvas);
  if(!gl) {
      console.log('Failed to get the rendering context for WebGL');
      return;
  }
  // specify the color for clearing <canvas>
  gl.clearColor(0.0,0.0,0.0,1.0);    // clear <canvas>
  gl.clear(gl.COLOR_BUFFER_BIT);
}
```

HelloPoint1

- Il prossimo passo è sviluppare un'applicazione WebGL che disegni un punto al centro dell'area <canvas>.
- Potrei pensare che basti procedere come in DrawRectangle.js:
  - Seleziono il colore **gl.drawColor(1.0,0.0,0.0,1.0);**
  - Disegno il punto **gl.drawPoint(0,0,0,1.0);**
- Invece bisogna procedere in modo diverso e far ricorso ad un nuovo meccanismo chiamato **shader**.

HelloPoint1.js

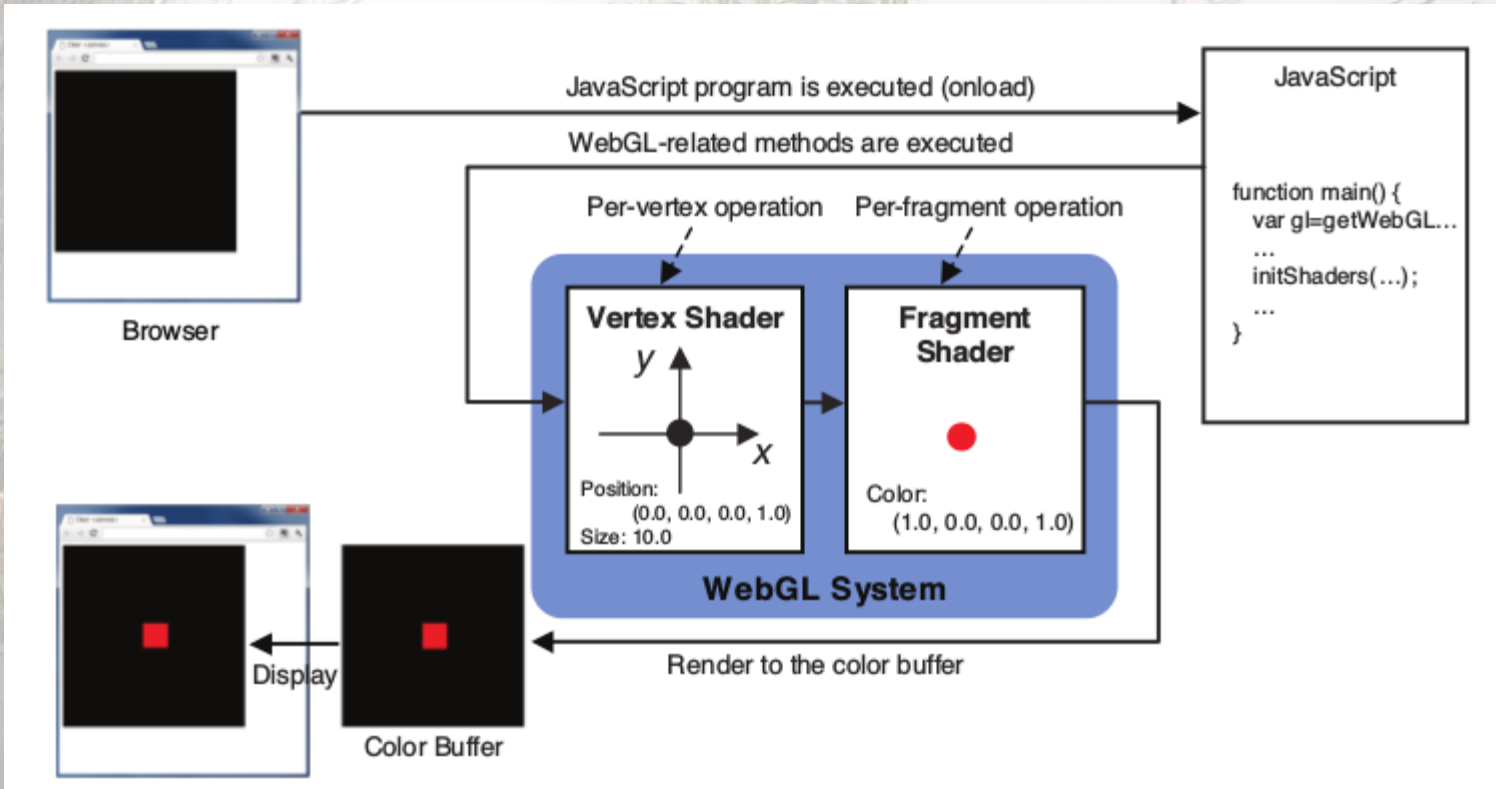
```
 1 // HelloPint1.js
 2 // Vertex shader program
 3 var VSHADER_SOURCE =
 4   'void main() {\n' +
 5   '  gl_Position = vec4(0.0, 0.0, 0.0, 1.0);\n' +
 6   '  gl_PointSize = 10.0;\n' +
 7   '}\n';
 8
 9 // Fragment shader program
10 var FSHADER_SOURCE =
11   'void main() {\n' +
12   '  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);\n' +
13   '}\n';
14
15 function main() {
16   // Retrieve <canvas> element
17   var canvas = document.getElementById('webgl');
18
19   // Get the rendering context for WebGL
20   var gl = getWebGLContext(canvas);
                    ...
26   // Initialize shaders
27   if (!initShaders(gl, VSHADER_SOURCE, ...
                    ...
32   // Set the color for clearing <canvas>
33   gl.clearColor(0.0, 0.0, 0.0, 1.0);
                    ...
35   // Clear <canvas>
36   gl.clear(gl.COLOR_BUFFER_BIT);
37
38   // Draw a point
39   gl.drawArrays(gl.POINTS, 0, 1);
40 }
```

Vertex shader program
(Language: GLSL ES)

Fragment shader program
(Language: GLSL ES)

Main program
(Language: JavaScript)

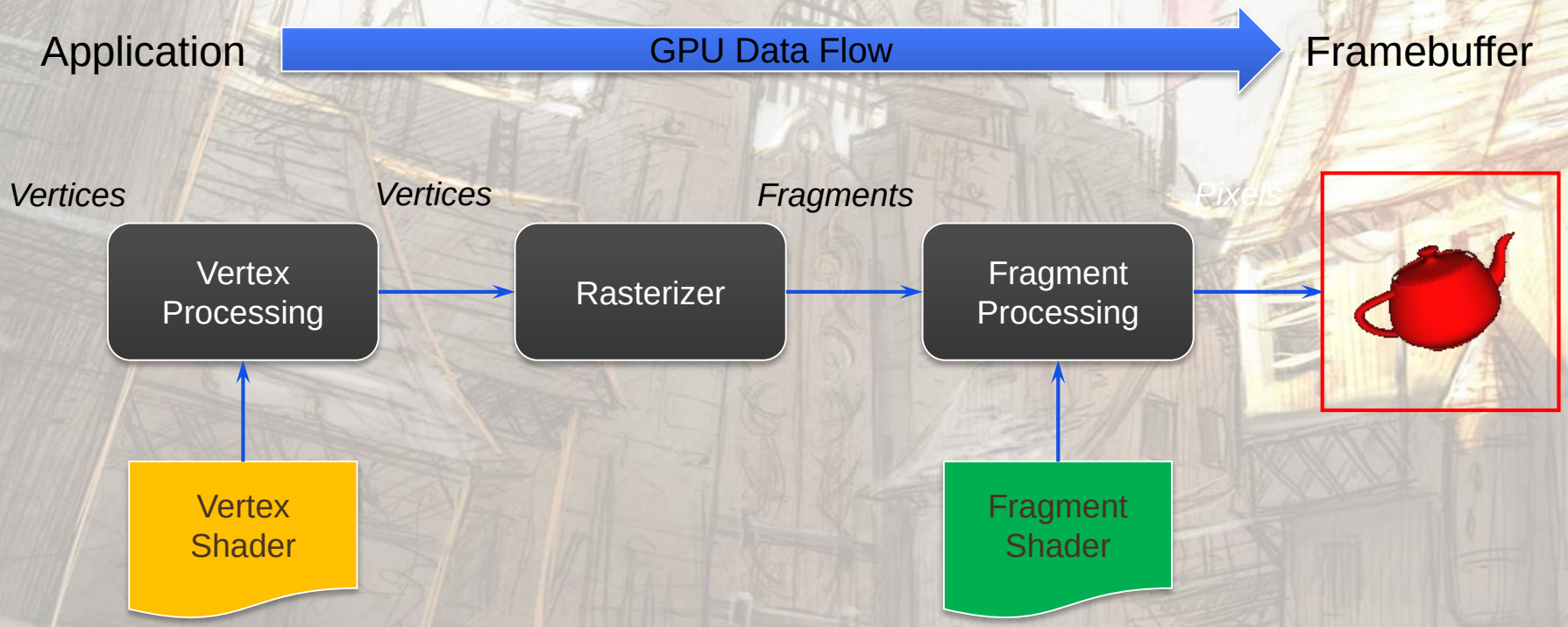
Cosa è uno shader



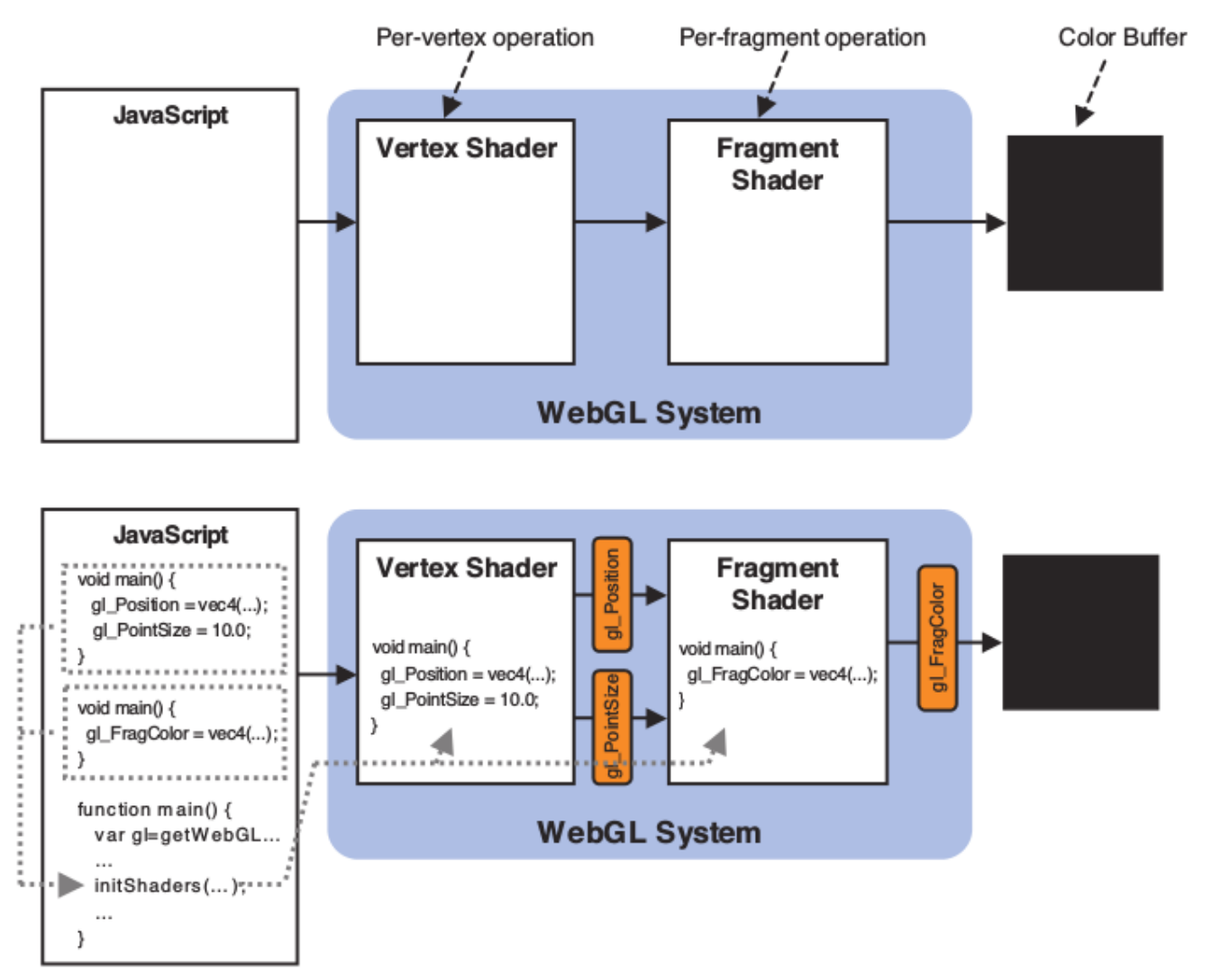
Esistono due tipi di shader in WebGL:

- Vertex shader: descrive ciascun vertice e le relative proprietà (colore,normale,...)
- Fragment shader: elaborazione per-frammento come il calcolo dell'illuminazione

Versione semplificata della pipeline grafica



InitShaders()



gl.drawArrays()

gl.drawArrays(gl.POINTS,0,1);

`gl.drawArrays(mode, first, count)`

Execute a vertex shader to draw shapes specified by the *mode* parameter.

Parameters	mode	Specifies the type of shape to be drawn. The following symbolic constants are accepted: `gl.POINTS`, `gl.LINES`, `gl.LINE_STRIP`, `gl.LINE_LOOP`, `gl.TRIANGLES`, `gl.TRIANGLE_STRIP`, and `gl.TRIANGLE_FAN`.
	first	Specifies which vertex to start drawing from (integer).
	count	Specifies the number of vertices to be used (integer).
Return value	None	
Errors	INVALID_ENUM *mode* is none of the preceding values.	
	INVALID_VALUE *first* is negative or *count* is negative.	